Semantic Web 1 (0) 1–5
IOS Press

Ontology-Driven Modeling Framework for SOA Security Patterns

Editor(s): First Editor, University or Company name, Country; Second Editor, University or Company name, Country
Solicited review(s): First Solicited Reviewer, University or Company name, Country; Second Solicited Reviewer, University or Company name, Country
Open review(s): First Open Reviewer, University or Company name, Country; Second Open Reviewer, University or Company name, Country

Ashish Kumar Dwivedi [a,*,**] and Santanu Kumar Rath [b]

[a] *Department of Computer Science and Engineering, National Institute of Technology Rourkela, Odisha 769008 India*
E-mail: shil2007@gmail.com

[b] *Department of Computer Science and Engineering, National Institute of Technology Rourkela, Odisha 769008 India*
E-mail: skrath@nitrkl.ac.in

Abstract. Securing an application based on Service Oriented Architecture provides defenses against a number of threats arising from exposing applications and data to the Internet. A good number of security guidelines are available to apply security in web applications. But these guidelines are sometimes difficult to understand and generate inconsistencies. Security guidelines are often represented as security patterns to build and test new security mechanism. These patterns are nothing but design guidelines, but they have certain limitations in terms of consistency and usability. Hence, application of security patterns may be even insecure. To resolve this problem, a suitable modeling and analysis technique need to be required. In study, an ontology-based modeling and refinement framework is proposed for the web service security. In order to maximize comprehensibility, UML (Unified Modeling Language) notations are used to represent structural and behavioral aspects of a SOA-based system. Subsequently, a Web Ontology Language (OWL) is considered to model SOA security patterns. For analyzing security requirements, description logic is used. The proposed approach is evaluated in the context of e-Health-Care system by applying the modeling framework to provide the semantic infrastructure for SOA-based security critical system.

Keywords: Ontology-Driven Security Framework, SOA Security Patterns, Web Ontology Language, WS-Security, UML

1. Introduction

Service Oriented Architecture (SOA) is a special form of distributed systems, sharing business logics, data through a programmatic interface across the Internet makes them vulnerable to different security threats. Those security threats mostly arise as a result of poor software analysis and design practices. Incorporating security features in a SOA based system is a challenging task that can be achieved by considering a systematic and structured approach, combining principles of software and security engineering. In the present-day scenario, a good number of security standards for web services are available, such as World Wide Web Consortium (W3C), Advancing Open Standards for the Information Society (OASIS), Internet Engineering Task Force (IETF), etc., [1]. These standards are complex and sometimes overlapping in nature. As a result they are difficult to implement and prone to generate inconsistencies.

To overcome these problems, a good number of software design solutions are available which may reuse available security solutions by using security patterns.

*Corresponding author. E-mail: shil2007@gmail.com.
**Do not use capitals for the author's surname.

Software patterns are reusable documents that incorporate expert knowledge, represent recurring structures, activities, behavior, processes, or things during implementation phase [2]. Security standards can be represented as security patterns for making them easier to understand, to discover inconsistencies, to build a secure web application, and to abstract essential aspects of security mechanism.

In the past two decades, a number of software patterns have been proposed [3] [4] [5] [6] [7] [8]. These system patterns facilitate the understandability and construction of systems that provide predictable uninterrupted use of the services and resources for users. Security patterns extend the concept of design patterns to represent security mechanism as well as security standard. In the modern era, web security is different from end-to-end security requirements of an application. To protect web service infrastructure security policies need to be considered, which are mostly high level guidelines to represent the states of a system in a secure manner. Security patterns provide well proven generic solution for the web services at different level of abstraction ranging from architectural level patterns involving high-level design of the system to implementation level patterns [7]. It also provide guidance how to implement portion of functions in the system.

A good number of SOA security patterns have been already proposed for different requirements, such as SOA design patterns [9], access control pattern [10], firewall pattern [11], WS-Policy pattern [12], WS-Trust pattern [13], Misuse pattern [14], WS-SecureConversation pattern [15], patterns for distributed system [16], patterns for cloud [17], etc. These security patterns directly do not provide systematic guidelines with respect to current heterogeneous web application. They need to be verified and validated by using a suitable modeling techniques. In a pattern oriented software development, a number of patterns are specified using informal and semi-formal (natural languages and other graphical notations) approaches, which lead to ambiguities and inconsistencies. Checking the consistency and completeness of patterns and their composition helps in detecting problems in early stages of software development life cycle, mostly using the concept of formal modeling which is nothing but a set of mathematical based techniques for the specification, development and verification of software and hardware systems. The main aim of formal methods are to describe the software requirements precisely and unambiguously using certain tools and techniques that can capture the abstract features of a system.

In this study a composition of WS-Policy pattern [12], WS-Trust pattern [13], and WS-Federation pattern [18] are presented at the higher level of abstraction. The composition of these patterns are specified using UML class diagram and sequence diagram. In order to semantically specify these SOA patterns, the Web Ontology Language (OWL) [19] is used, which is an axiom-based language to model the problem domain as well as solution domain. In this study an Ontology Driven Security Framework (ODSF) is presented, which is an extension of traditional Model Driven Security Framework (MDSF) by refining SOA security patterns. The proposed modeling framework is based on the concept of metamodels, which help for mapping UML-based security notations into formal representation. Subsequently, an ontology is presented for the web service security patterns that is reusable and extendable, as well as deployable in web server. For the evaluation of this approach, a case study on e-Health-Care system has been taken into consideration. The ODSF offers a good number of rigorous modeling services, which represented as follows.

- ODSF is an extension of traditional Model-Driven Security (MDS) by supporting formal refinement of SOA security patterns.
- It enables the transformation process for UML-based security patterns into a formal representation.
- It presents ontology of SOA security patterns that helps to understand the semantic definition of patterns.
- ODSF offers an automated reasoning process by using an ontology editor that heps to perform an automated formal verification of SOA security patterns.

For analyzing security concepts, Model Driven Security (MDS) has been emerged in the early of 2000 as a specialized Model Driven Engineering (MDE) technique for supporting the development of a secure system [20]. Over the last decade, metamodels and ontologies are developed in parallel isolation. A metamodel is also known as model of models having an important role in standards community such as Object management group (OMG) [21]. Ontologies support an explicit formal construction for domain formalization by incorporating mathematical logics. A good number of literatures are available, which link ontologies and metamodels [22] [23] [24] [25].

In this study, metamodels and ontologies are combined to achieve semantic interoperability for to-

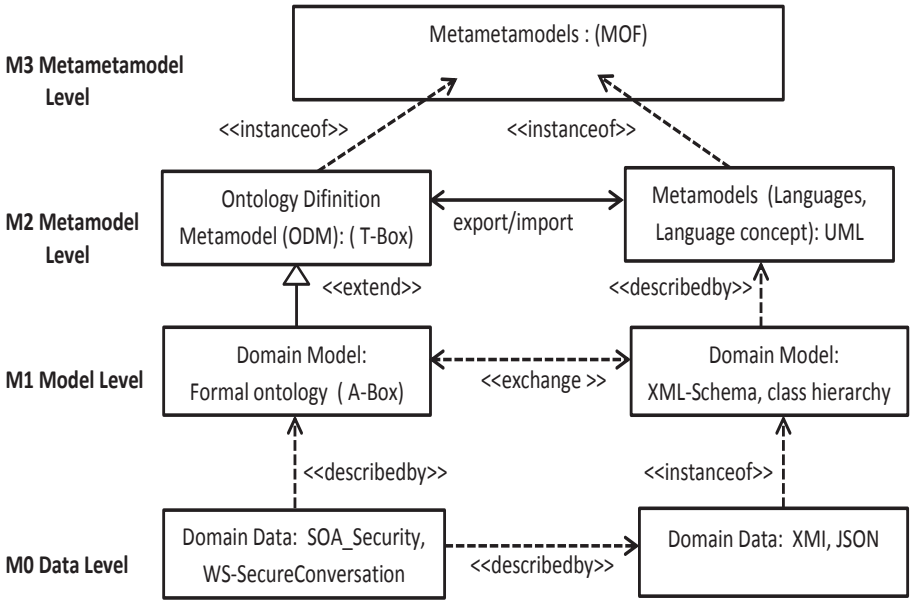

Fig. 1. Hierarchical organization of Metamodels

days complex systems. Developing a metamodel using Meta Object Facility (MOF) for a particular domain, such as SOA security pattern is a difficult task, for defining syntax and semantics of the new entities. To resolve this problem, UML abstraction has been considered, where UML metamodel elements such as class, attributes, relationship, etc. can be extended to build domain specific metamodel. Figure 1 shows the layered organization of OMG ODM approach, which presents a multilevel ontology architecture. According to OMG, an object in level M0 is an instance of a model in level M1; a model in level M1 is an instance of a metamodel in level M2; a metamodel in M2 is an instance of a metametamodel in level M3. ODM and OMG can be differentiated as descriptive and prescriptive models. The M1 level domain ontology can be exchanged with M1 level domain model. The M2 level can be used to develop a domain model (M1) that can be applied at level M0. Bridging the ontology with metamodel means creating definitions of ontology modeling languages in terms of OMG's MOF model. MOF is a simplified version of UML meta-meta-modeling, which helps to transform one model into another.

2. Related Work

In the area of SOA, a good number of security patterns have been proposed [1]. But these patterns are modeled using semi-formal notation such as UML notation. Formal modeling of the available SOA design patterns need to be required. Some of the related security modeling techniques are represented as follows.

A number of MDS approaches are available to handle security requirements using UML. UMLsec [26] is a UML profile extension for analyzing security-based systems, where Jürjens added stereotypes and tags in UML profile to model security requirements in a system. UMLsec is applied to model web applications, distributed systems, and embedded systems. SecureUML [27] is a modeling language based on Role-Based Access Control (RBAC) for specifying authorization constraints. SecureUML allows for weaving system models with security concerns. Alam et al. [28] presented an extensible model-driven security framework for enabling the design and implementation of secure work flows for various domains such as health, government, and education. Sánchez et al. [29] presented a MDS approach i.e., ModelSec that supports a generative architecture for handling security requirements. ModelSec is applied to model web application for the management of medical patients. The presented architecture automatically generates security

artifacts by using model transformation approach. Deveci and Caglayan [30] proposed a model driven security framework for the analysis, design, and evaluation of security properties of an information system.

Kobashi et al. [31] proposed an extended security pattern, which include requirement and design level patterns as well as a new model testing process. They proposed a tool i.e., test-driven secure modeling tool, which verifies as to whether the security patterns are properly applied and security vulnerabilities are resolved or not. Katt et al. [32] proposed a security framework that integrates pattern refinement to model driven security approach. They added a security pattern refinement layer, which supports the configuration of one security service with other different patterns. Uzunov et al. [16] presented a pattern-driven security methodology for distributed systems. They have presented a pattern-based model and meta-model for engineering a methodology of conceptual security framework.

Delessy and Fernandez [33] presented two approaches to secure SOA applications, which are based on model-driven development and the use of security patterns. Kou et al. [34] presented a metamodel called SoaML4Security, which introduces QoS concepts into SoaML in order to support the modeling of security aspect. Alam et al. [35] presented an interface model for web services, which is based on model-driven technique. They have performed by extending object constraint language (OCL) to define access control policy. Memon et al. [36] proposed security patterns refinement approach for model driven security. Their approach relies on the UML notation for Security modeling. Basin et al. [37] formalized the properties of security design models and their instances. They have analyzed security properties by using OCL queries and evaluated the queries on models or model instances. OWL-S (Web Ontology Language for Services) is mainly used to formally specify Web Services [38]. OWL-S supports semantic description model for the realization of invocation, interoperation, and composition of web services. But in OWL-S specification layers are not precisely separated using MOF technique.

Most of the above OCL based approaches have certain limitations for the properties that cannot be specified by simply evaluating OCL notation over the metamodel. Other MDA based approaches are not based on Semantic Secure Service Oriented Architecture (SSSOA), where an unambiguous (semantic or formal) representation of essential properties of a system required, which can be performed by automated ontology-based reasoning. Some of the related works are based on semantic representations of WS-Security and WS-Policy.

Kim et al. [39] presented security ontology for making security annotations. They have described how the ontology can be applied to the web services in SOA to present security requirements and capabilities. Garcia and Toledo [40] presented an approach that combines WS-BPEL, WS-Policy, and OWL for building secure business processes. They have considered policies to model service security capabilities and security requirements in business processes. Agostini et al. [41] presented an approach based on the use of ontologies to support the description of content of security certificates for services. Yu et al. [42] proposed a rule-based scheme to check whether security capabilities match security requirements. They have performed semantic modeling of WS-Security and applied inference rules for security capabilities. Brahim et al. [43] presented a semantic approach for specifying and matching web service security policies. They performed a transformation of WS-Security-Policy into an OWL-DL ontology.

Dietrich and Elgar [44] proposed a novel approach to the formal definition of design patterns. Authors formally defined design patterns and some related concepts such as pattern participant, pattern refinement, and pattern instance using OWL. Boaro et al. [45] have presented an integration of model checking and semantic reasoning technologies. They have presented services as state transition system and annotating them by means of description logic assertions. Modica and Tomarchio [46] presented a semantic framework capable of matchmaking in a smart way for security capabilities of providers and security requirements of customers and tested it on use-case scenario. Parreiras and Staab [47] proposed TwoUse approach, which enables UML modeling with semantic expressiveness of OWL-DL. They presented bridges based on a metamodel, library extensions and model transformations.

Katasonov [48] propsed an ontology based modeling framework. He claimed that the framework is to be implemented as a part of model driven engineering tools to support software engineers. Hästbacka and Kuikka [49] presented an application of OWL semantics and reasoning to models for developing control applications. To demonstrate their scheme, OWL based modeling method is considered, where models are transformed and combined with other engineering knowledge of a generic nature. Maged et al. [50] pre-

sented an approach i.e., Query/View/Transformation (QVT) to find problems in domain specific model. They have defined detection semantics and can be used in any MOF-based model.

A good number of formalization techniques are also available for analyzing SOA design patterns. But these techniques do not cover security features of a system. Tounsi et al. [51] presented a formal refinement-based approach for the modeling of message oriented SOA design patterns using SoaML (Service Oriented Architecture Markup Language) and Event-B. Kim and Carrington [52] formalized design patterns using formal modeling language Object-Z. They developed a role metamodel using an existing modeling framework, Eclipse Modeling Framework (EMF) and transformed the metamodel to Object-Z using model transformation techniques. Brown and Capretz [53] proposed the ODEP-DPS development process for the development of Data Providing Services (DPS).

From the above literature it is clear that a very limited number of modeling techniques are available for SOA security patterns. Few of them are not based on ontology, which lack proper semantic notation, interoperability, and scalability. A good number of SOA security patterns are available, which require to analyze the consistency of patterns composition. This study provides the modeling of web service security patterns using semantic notations. These notations within SOA allow reasoning tools to automate tasks, resolve data and process mismatches, and improves interoperability. Semantic SOA also helps automated discovery, ranking, negotiation, contracting, and composition of services [54]. Most of the existing ontology-based techniques do not represent behavioral aspects of system. Our modeling framework is helpful for both structural as well as behavioral aspect of SOA security patterns.

3. Proposed Work

Securing web services requires a set of security solutions to be applied during the web service communication life cycle. In the presence of different solutions, security patterns are the most widely used approach for providing guidance and representative architectural models for developers to use in order to realize more specific security policies as well as security capabilities. Web Service policies are considered to enhance communication mechanism by enabling quality parameters and secure service capabilities are used for making SOA-based business processes that satisfy user security requirements. But the patterns directly do not offer systematic guidelines to the system, instead it requires proper analysis to apply them in a particular scenario. This study does not provide a pattern composition approach, whereas it offers a SOA security pattern modeling approach. In the following subsection, the composition of web service security patterns is presented for a given context. Further it is analyzed by using an ontology-based approach.

3.1. SOA Security Patterns

In order to conduct business processes, a number of web services interact with each other. These interactions occur by exchanging a large number of SOAP (Simple Object Access Protocol) messages. In order to protect these messages (stored messages or messages in transit), a number of web service security standards are available in the form of patterns. But these patterns provide security mechanism in terms of their context. For providing security mechanism to SOA, there is a need to compose these patterns. In this study, a composition of WS-Policy pattern, WS-Trust pattern, and WS-Federation pattern is presented. The structural aspect of the composition of patterns is shown in Figure 2. The composition of these web service patterns is described by using four pattern template elements such as *context, problem, solution*, and *forces*. Context describe on what situation the problem occur. Problem describes when to apply patterns and describe specific design problems. Solution represents the elements that prepare design, their relationships, responsibilities, and collaborations. Forces are the results and trade-offs of applying patterns. The following pattern template elements describe about the composition of patterns, WS-Policy, WS-trust, and WS-Federation in terms of context, problem, solution, and forces.

Context: Web services communicate through the Internet, which is an insecure medium. Web services require to maintain secure and trusted relationships between them during communication process. They also require to leverage identity management for enabling cross-domain interactions between web services and users.

Problem: During the communication process of distributed applications, a number of malicious users and services may try to access stored information or information in transit. Without using WS-Policies, web services cannot preserve reliability, availability, and security in their interactions. If trust relationships are

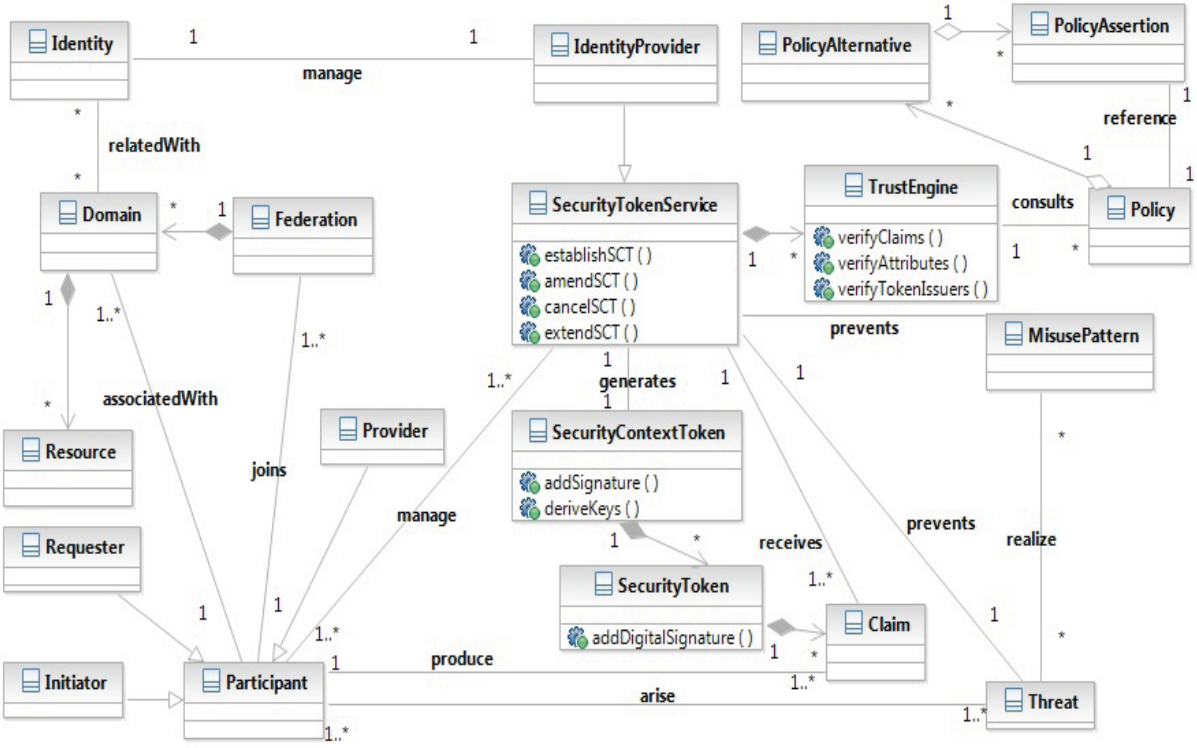

Fig. 2. Structural Aspect of SOA Security Patterns

not maintained between communicating parties, they have no means to preserve security and interoperability in their composition. In a cross-domain network, partners, applications, and business processes separately controls identity information about its users. Users may have multiple identities for accessing various accounts in different domains. They need to spare from giving their identities many times within a federation.

Solution: A number of solutions are available for the above defined problems occurred in a particular context. In this context solution can be provided by composing policy, trust, and federation patterns. Policies help to assure security, availability, and reliability by applying security assertions, which represent a capability and constraints of the behavior of web services. But there is also a need of trust mechanism for sharing information with each other. Trust mechanism can be achieved by using security tokens which become a proof to maintain a trust relationship between them. The identity problem can be solved by sharing an identity information i.e., federation metadata between the participants in a federation. An Identity information provides information about policies, federated services, and brokering of trust.

Forces: The above solutions are constrained by the number of the following forces:

- Malicious users and services can modify and remove policy assertions.
- Policies includes security assertions that can be used by trust mechanism to verify the policies.
- Each communication process should have time limit that denotes the validity of trust mechanism.
- Identity management results a high cost in terms of execution time, human resource, and administrative duties.
- A federation contains different participants having different security policies and the participants should not change their policies for accomplishing their goals.

3.1.1. Structural Aspect of SOA Security Patterns

The structural aspect of the composition of SOA security patterns is presented in Figure 2. The presented pattern is often based on WS-Policy, WS-Trust, and WS-Federation to provide security mechanism to SOA-based application. In this pattern, SecurityTokenService plays an important role act as a Web ser-

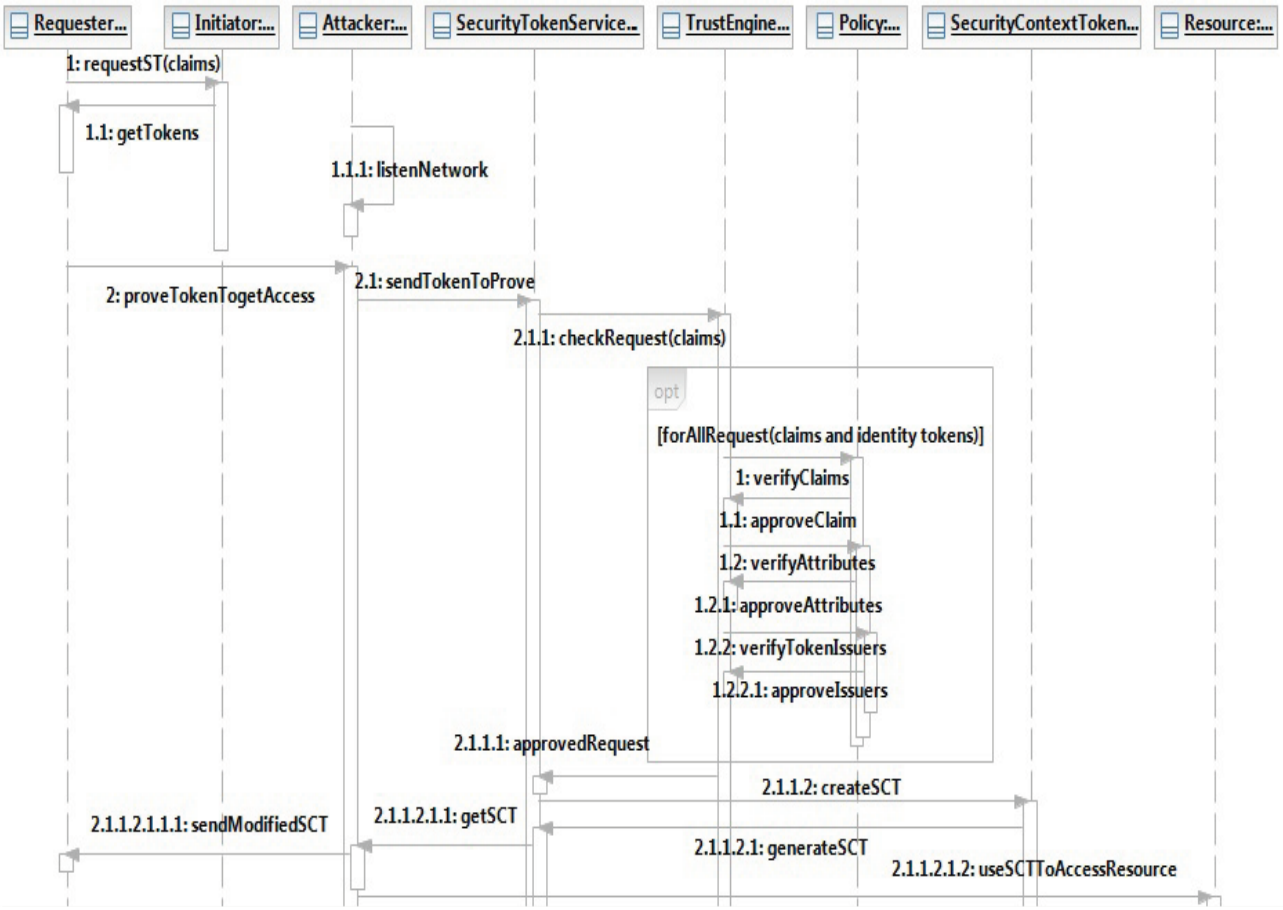

Fig. 3. Dynamic Aspect of SOA Security Patterns

vice generates SecurityContextToken (SCT). A SCT is a representation of a security context to develop an authentication state by using security tokens. Requester can consider security context tokens to encrypt a number of SOAP messages. SecurityTokenService is responsible for providing security token to insure message originality, verification of authorized use of a security token, and modifying trust in a domain of services. Each SecurityTokenService has a TrustEngine to evaluate security related issues by verifying security tokens and claims against security policies.

A SCT has a number of Security Tokens for applying signatures to tokens. Security Token is a set of claims, which is a statement about a participant, service, and resource. A Claim is available in the form of assertions in terms of authentication. A SecurityContextToken can be implemented by using Signed Security Token and Proof-of-Possession. In order to verify the claims, Trust Engine can consult with the Policy, which is a set of policy alternatives. Policy Alternative is a set of policy assertions which represent a capability and a constraint. Policy Assertion helps to identify the behavior of participants.

Identity management is an essential activity to achieve the federation. Identity Provider is a specialization of SecurityTokenService. Service provider and requester uses federation services to perform business processes using web services. A domain is a set of Resources which specifies a unit of security administration. The Identity Provider is a trusted body considered by the participants such as Service Requester and Service Provider. A Federation is a set of domains that have established business relationships. In the federation one domain can allow authorized access to its resources on the basis of identity.

Figure 2 is an extension of Web Service Security Standard patterns, where a number of security standards are mapped into the composition of security pat-

terns. The composition of patterns is specified in the presence of misuse pattern. Misuse Pattern starts from the goals of the malicious users and express the ideas of the malicious users. Misuse patterns represents the messages that malicious users transmits to various components of an application architecture to accomplish its goals. In this study a misuse pattern is considered to evaluate the security of presented patterns composition. The dynamic aspect of patterns composition is presented in the following subsection.

3.1.2. Dynamic Aspect of SOA Security Patterns

The dynamic aspect of presented patterns composition is specified by using UML sequence diagram which is shown in Figure 3. It is presented for the use case accessResource in the presence of Man-in-the-Middle attack. This dynamic aspect have three main actors, Requester, Initiator, and Attacker. According to the precondition of this specification, SecurityTokenService has Policy to verify the requester request. In this scenario requester requests for the security token to initiator in terms of claims. The initiator provides token to get Security Context Token. Requester sends token to SecurityTokenService for proving the token that can be identified by the Attacker. An attacker can send the copy of the claim to SecurityTokenService. SecurityTokenService checks the request using Trust Engine. Trust Engine verifies claim through the Policy. If policy approve the claim, it verifies the attributes against the policy. Also it verifies token issuer through the policy. After verifying all the essential elements trust engine approve the request. Subsequently, SecurityTokenService creates Security Context Token. It is generated by Security Context Token which is taken by Attacker. Attacker sends modified Security Context Token to Requester and Attacker can access Resource by using original Security Context Token. According to the post condition of this specification, an Attacker has Security Context Token to access a Resource. A number of sequence diagrams are possible for the composition of patterns presented in Figure 2 according to different use cases, such as create new policy, access resource using identity token, request a new service, etc.

3.2. Ontology-Driven Security Modeling and Refinement Framework

Ontology-Driven Security Framework (ODSF) is an extension of Model-Driven Security Framework (MDSF) by applying security pattern modeling and refinement process. ODSF is an specialization of MDE considering three layers of MDE process such as CIM, PIM, and PSM as shown in Figure 4. In CIM layer, SOA security requirements are considered to map into analysis model. In this study, a number of SOA security requirements are considered such as security requirements for the stored information, security requirements for information in transmission, security requirements for a single service, and security requirements for the composition of service. An analysis modeling can be performed by using a use-case diagram which is mapped into XMI (XML (Extensible Markup Language) Metadata Interchange) analysis file. In Model Driven Security, requirement specification is generally performed by using OCL (Object Constraint Language) expressions during the analysis phase. OCL is a formal language mainly used for the verification and validation of UML analysis and design diagrams. But in ODSF requirement validation is performed by using OWL-DL (Web Ontology Language Description Logic). ODSF offers additional operations that are not easily expressible in OCL. Table 1 shows the OCL expression for the token-based authentication and resource access. First expression denotes whether all authenticated requesters are allowed for security token. Second expression denotes the context of token request and third expression specifies about resource access where access can happen in the form of resource modification and resource access.

Table 1
Invariants for the token based authentication and resource access

context User::allAuthRequester(a:Action)
:Set(AuthenticationConstraint) **body:**
self.hasToken.allowSecurityToken().allAuthParticipant(a)

context TokenRequest
inv containsAction:
self.subordinatedactions=
self.resource.oclAsType(Token).hasattribute.action
$-> select(a \mid a.ocllsTypeOf(GetToken))$

context ResourceAccess
inv targetsAResource:
self.resource.oclIsTypeOf(Resource)
inv containsSubactions:
self.subordinatedactions = self.resource.action
$-> select(a \mid a.ocllsTypeOf(ResourceModification))$
->union(self.resource.action
$-> select(a \mid a.ocllsTypeOf(ResourceAccess)))$

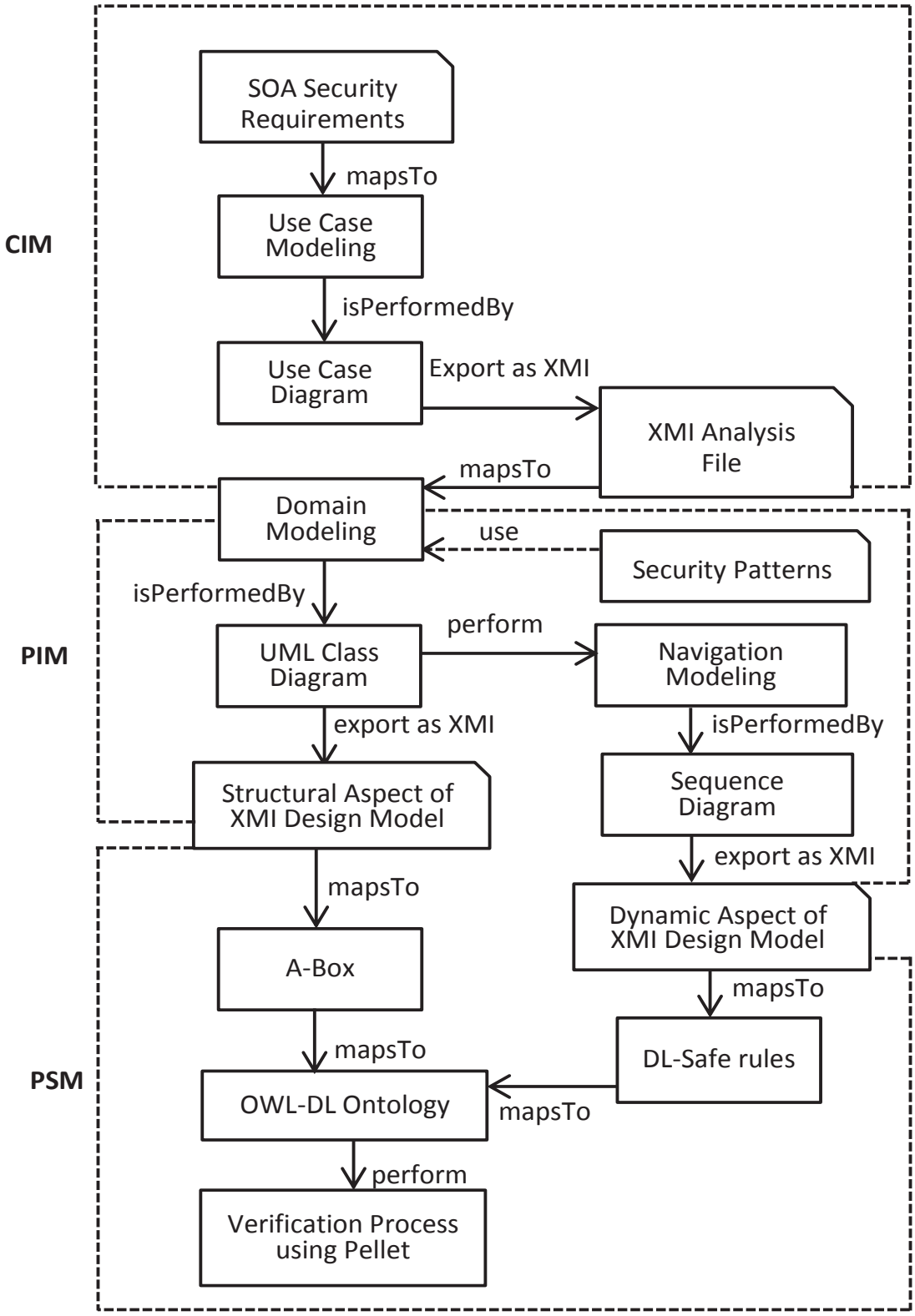

Fig. 4. Ontology-Driven Security Modeling Framework

At the PIM layer, the ODSF framework describes a metamodel which is presented in the refinement of pattern-based SOA security modeling. In PIM layer, XMI analysis file is imported by design modeling which can be performed by any UML-based tool such as RSA (Rational Software Architecture). A design modeling can be performed by using UML class diagram which denotes the static aspect of a system. Security patterns are used during the domain modeling. Security patterns provide solution to the security problems in a specific context. In this scenario authentication can be achieved by using trust mechanism in a SOA-based application. During the design modeling process, each security requirement is mapped into an abstract security pattern. From the abstract security patterns concrete security patterns can be identified. For example Security Token can be implemented by using X.25 certificate and Kerberos ticket. Security patterns are specified using class diagram that performs navigation modeling. A navigation model is a specialization of conceptual model. It can be defined as two step process; first to define navigational space model and second to define navigational structure model. In this study navigation modeling is performed by using UML sequence diagram which is a common process in a web-based system. Finally in PIM layer, UML class diagram and UML sequence diagram are exported into XMI model which helps for model-to-model transformation.

The ODSF framework mainly used to minimize the gap between PIMs and PSMs using pattern refinement process which is presented in the following subsection. PIM resolves system functional requirement in terms of problem space whereas PSM offers a solution model that resolves both functional and non-functional requirements of a system. Transformation of PIM into PSM is a challenging task that is based on a number of transformation rules. Transformation rules define as to how one or more elements in the source model can be transformed into a target model. In this framework the structural aspect of a design model is mapped into ontology A-Box which specifies the instances of concepts at instance level. The dynamic aspect of a design model is mapped into DL-Safe rules. Finally both the constructs i.e., A-Box and DL-Safe rules are mapped into OWL-DL ontology. For the verification process an automated reasoning can be performed by an ontology reasoner i.e., Pellet [55]. Pellet is an open source Java-based OWL DL reasoner. The reasoner can be used in conjunction with both Jena and OWL API library. Ontologies are the subset of models and also they accomplish the criteria for being models along with extra features.

The proposed modeling framework provides a semantic description for a SOA-based security patterns. In this study a MDD-based model2model transformation technique is presented. UML notation incorporated with security features are considered as source model that is required to mapped into a target model i.e., OWL-DL. The proposed technique is based on the concept of metamodel which supports analysis and design of rules, constraints, and models helpful for the modeling of predefined class of problems.

3.2.1. Ontology-Driven Transformation and Refinement for SOA Security Patterns

An ontology-driven pattern refinement is presented in Figure 5. In the process of pattern refinement, firstly, SOA security requirements are mapped into abstract SOA security patterns. Secondly, abstract security patterns are mapped into corresponding concrete security patterns. Thirdly, these concrete SOA security patterns are formally refined and transformed into semantic notations i.e., OWL-DL. The third step is a special form of the third layer of traditional MDS technique, which offers the platform specific models. In this step a model2model (M2M) transformation process is performed that is based on the concept of metamodel.

The proposed transformation and refinement framework is based on OMG's Ontology Definition Metamodel (ODM). During the mapping process, each model element can be represented as a resource in the RDF (Resource Description Framework) model. RDF model has a type declaration denoting to the model elements metaclas in the metamodel. The essential constructs of a model such as properties and relationships are specified by using RDF statements which denotes the properties and relationships types in the metamodel. UML diagrams incorporated with security notations (stereotypes) support for specifying information about different views of a security-critical application such as static view and dynamic view. Since it is difficult to assess all the views of a software into a single model, hence a semantic approach is considered to present OWL descriptions at the metamodeling level, which helps to disambiguate UML constructs and support to analyze logical constraints. An UML-based security design patterns associated with variation and delegation of concept in models may have a question as to how the selection of a class could be performed by using their description rather than by weaving descriptions. ODSF offers decoupling class iden-

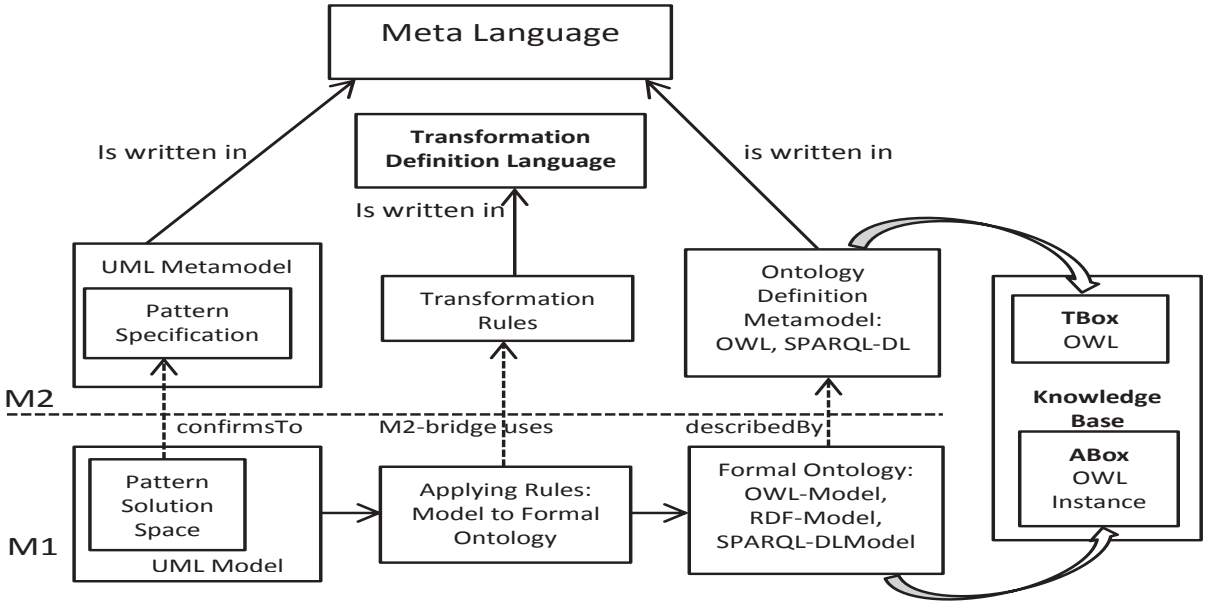

Fig. 5. Ontology-Driven UML-Based Pattern Refinement

tification from the definition of classes by considering OWL-DL.

During refinement process, UML model is used to describe pattern solution space that confirms to pattern specification represented by UML metamodel. A set of transformation rules are required in order to transform pattern solution space into formal ontology which can be described by Ontology Definition Metamodel (ODM) as shown in Figure 5. A pattern-based ODM can be represented by using T-Box (Terminology Box) where pattern-level concepts such as classes, attributes, and relationships are specified. Whereas, a pattern-based model instances such as OWL model, RDF model, SPARQL-DL model are described by using A-Box (Assertion Box). During the transformation process, pattern specification is mapped into T-Box at M2-level and pattern solution space is mapped into A-Box at M1-level for providing semantic representation to the SOA security patterns. UML metamodels and ontology definition metamodels can be written in Meta language such as OPRR (Object-Property-Relationship-Role model). Transformation rules can be written in transformation definition language such as QVT (Query/View/Transformation).

During the refinement and transformation process, description logic contains two basic elements such as concepts and relationships in order to map security concepts specified in UML notation into a semantic notation. Concept formalizes UML classes and relationship formalizes UML association. In Figure 5 knowledge base contains two boxes i.e. T-Box and A-Box. T-Box is specified by using concepts and relationships. A-Box is specified by using constraints which initiate the concepts and relationships in the T-Box. Generally, T-Box is used to specify a UML class model and an A-Box is used to specify an instance model.

3.2.2. ODSF Metamodels for Secure SOA

The ODSF metamodels are extended from TwoUse metamodels [47]. In this study, UML-based metamodel is defined to specify secure SOA-based system. Subsequently, OWL metamodel is presented to provide the semantic notation of a SOA-based system. Finally, ODSF metamodel is represented to describe ODSF approach. The UML-based metamodel defines the elements and their relationships for the secure SOA-based system, which enables a common understanding of modeling constructs. Figure 6 represents a simplified version of secure UML-based SOA metamodel that is associated with the web security requirements. These security requirements support evaluation processes to maintain a level of confidence that the security functionalities satisfy security requirements. Security requirements are enforced by security policy and satisfies security objectives. Security Policy also realizes permission for action using action assignment. An action can be performed on resource by using re-

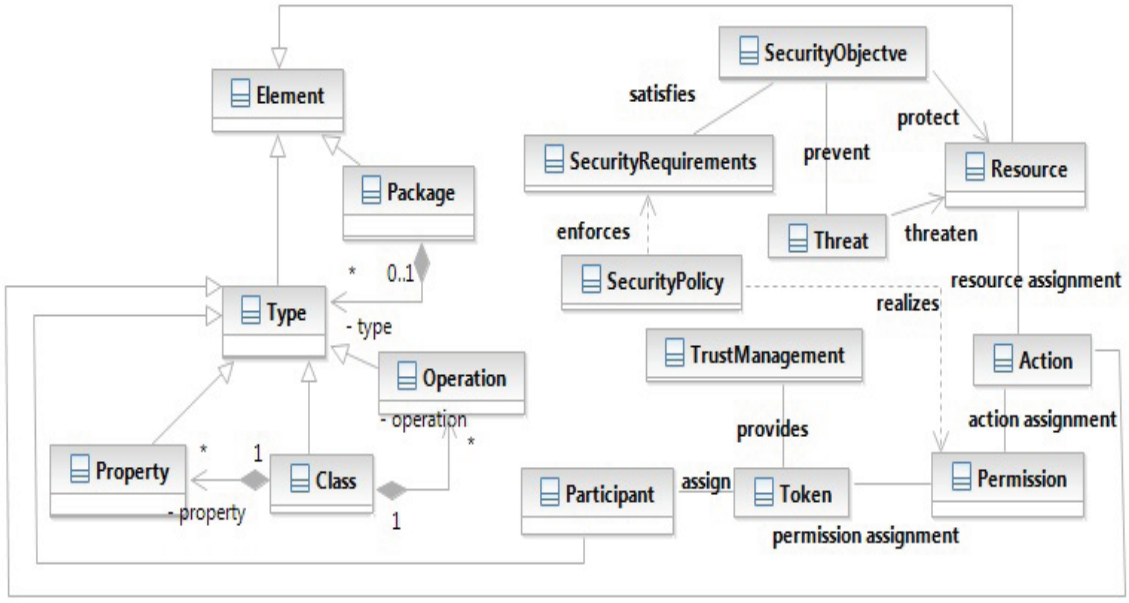

Fig. 6. UML-Based Metamodel for Secure SOA

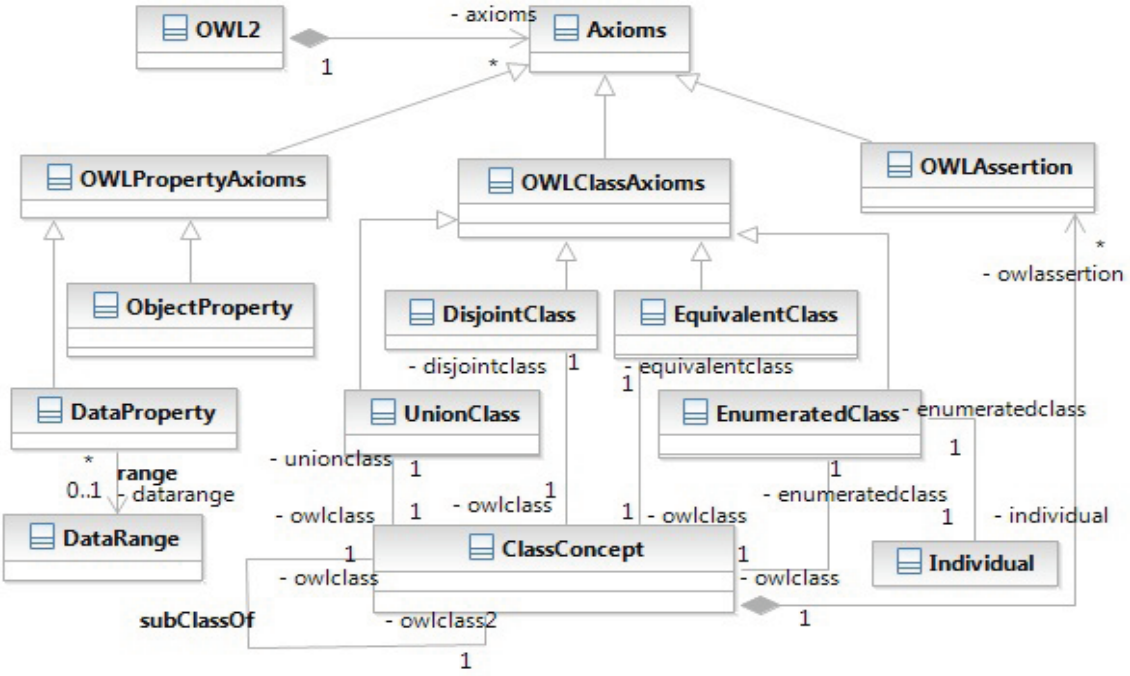

Fig. 7. OWL2 Metamodel for Secure SOA

source assignment. The presented metamodel specifies a token-based access control policy for actions on protected resources. A common criteria denotes the protection of resources from the unwanted accesses i.e, threat. The metamodel provides a subset of a UML class model that has elements which includes type,

package, participant, etc. A type has three subclasses i.e., property, class, and operation. A UML class can be a abstract class or a subclass. A UML class contains properties and operations which are represented by class Property and class Operation respectively.

The concept of OWL metamodel has been borrowed from the OMG ontology definition metamodel. OWL2 is an axiom-based language fully compatible with OWL-DL that helps to provide semantic representation of properties of a SOA-Based security critical system. OWL metamodel offers a good number of axioms, such as class axiom, property axiom, assertions, etc., which are shown in Figure 7. OWL ontology generally used to express sets of concepts which specify domain knowledge as well as specify classes by using logical notations. Axioms are used to support a number of constructs to limit classes and properties. Class constraints are available in different forms, union of classes, disjoint classes, equivalent classes, enumerated classes, etc. These class axioms are basic concepts of OWL classes. OWL property axioms are available in the form of object property and data property. OWL classes have assertions to specify the properties of a system.

The ODSF metamodel is presented in Figure 8. It offers the abstract syntax for specifying classes with semantic notations for SOA-based security critical system. The ODSF metamodel syntax allows an abstraction for the various concrete syntaxes used in ODSF modeling framework. ODSF metamodel combines both UML metamodel and OWL metamodel to represent model2model transformation process. The UML metamodel specifies structural and behavioral nature of classes whereas OWL metamodel specifies classes with OWL expressiveness. In this metamodel problem space and solution space are used by security requirement and security policy respectively. The problem space can be described by UML use case diagram and UML activity diagram. The solution space can be specified by UML class diagram that is mapped into OWL class. Here problem space and solution space are treated as artifacts which play an important role to describe software design patterns. The solution space specified by class diagram is a set of security patterns for a particular problem. The behavioral aspect of pattern-based solution can be represented by using UML sequence diagram which also needs to be semantically specified.

3.2.3. Transformation Rules

The proposed modeling framework is implemented on the basis of transformation rules, which include syntax, semantics, and pattern constraints transformation.

- In a pattern-oriented software development, a number of participants such as class, attributes, methods, instances may occur, which require variable symbol declaration. For example, Signed Security Token contains Kerberos Token as an instance. These variables need to be mapped.
- A number of predicates are used that need to be mapped. For example, each class can be declared in the ontology having a predicate *is* such as isClass and isAbstract.
- OWL properties can be defined by using a binary predicate. The name of a predicate represents the name of property. For example, *hasToken* property associates Security Context Token with Security Token.
- OWL relationship can be expressed with the help of a number of predicates, such as *isSubclassOf, types, contains*, etc. A model semantics involve a set of logical sentences by introducing all interpretations, which provide to its atomic elements, whereas syntactic mapping uses declarative and prescriptive representations.
- Pattern constraints can be transformed into OWL notation by considering pattern semantics in UML, which are specified as UML class. An UML class contains methods, attributes, and relationship. For example, method can be represented as isMethod (SecurityToken.addDigitalSignature). Similarly, relationship e.g., generalization between two classes can be represented by using *isSubclassOf* predicate, for example IdentityProvider is a sub-class of SecurityTokenService in the security pattern presented in Figure 2, which can be declared as isSubClassOf(IdentityProvider, SecurityTokenService).

According to transformation rules, a target model can be generated from the source model. In this ODSF, source model and target model confirms to corresponding source metamodel and target metamodel as shown in Figure 5.

3.2.4. Construction of TBox and ABox

In model driven software development, UML analysis and design support a stereotype set and the underlying metamodels, which represent the relationship

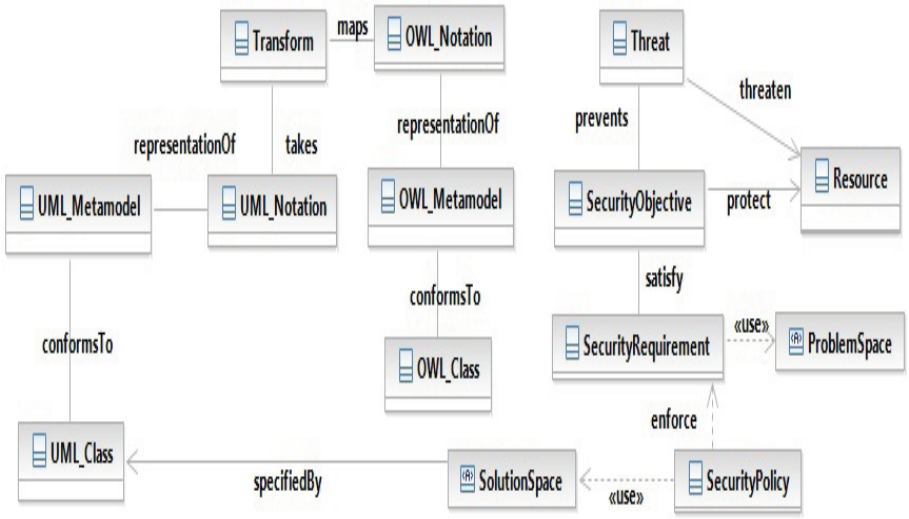

Fig. 8. Metamodel for MDSF approach

Table 2
Transformation of Meta classes into DL T-Box

Stereotype	**Meta Class**	**Meta Super Class**	**T-Box Classes**	**T-Box Generalization**
Entity	Element	NIL	T	NIL
System component	Class	Entity	Class	$class \sqsubseteq Entity$
Software component	Class	Entity	Class	$class \sqsubseteq Entity$
Database	Class	Entity	Class	$class \sqsubseteq Entity$
Role	Actor	Entity	Actor	$Actor \sqsubseteq Entity$
Role	ValueSpecification	Element	Value	$ValueSpecification \sqsubseteq Entity$
Asset	Class	Entity	Class	$class \sqsubseteq Entity$
Condition	Guard	Constraint	Guard	$Guard \sqsubseteq Constraint$
Condition	Precondition	Constraint	Precondition	$Precondition \sqsubseteq Constraint$
Condition	Postcondition	Constraint	Postcondition	$Postcondition \sqsubseteq Constraint$
Condition	StateInvariant	Constraint	Invariant	$Stateinvariant \sqsubseteq Constraint$
OperationWith FormalBody	Operation	ValueSpecification	Operation	$Operation \sqsubseteq ValueSpecification$
Authentication Exception	Message	ValueSpecification	Message	$Message \sqsubseteq ValueSpecification$
Entity	LifeLine	Element	LifeLine	$LifeLine \sqsubseteq Element$
Authentication Operation	Operation	ValueSpecification	Value	$Operation \sqsubseteq ValueSpecification$
Anonymity	Attribute	Element	Attribute	$Attribute \sqsubseteq Element$
AccessControl Model	Class	Package	Class	$Class \sqsubseteq Package$
Software Component	ProxyBasedFirewall	Proxy	Proxy	$ProxyBasedFirewall \sqsubseteq Proxy$
Software Component	UDDI	Protocol	UDDI	$UDDI \sqsubseteq Protocol$
Design Pattern	WS-Trust Pattern	SecurityPattern	Pattern	$WS-Trust \sqsubseteq SecurityPattern$
Design Pattern	WS-Policy Pattern	SecurityPattern	Pattern	$WS-Policy \sqsubseteq SecurityPattern$
Design Pattern	WS-Federation Pattern	SecurityPattern	Pattern	$WS-Federation \sqsubseteq SecurityPattern$

Table 3
Development of TBox

1. **Initialize:** $TBox = \{\}$
2. **for all** *classes*(*Class*) *and superclass* (*SupClass*) *in metamodel*
3. $TBox = T - Box \cup \{Class \sqsubseteq SupClass\}$
4. **end for**
5. **for all** *classes*(*Class*) *and superclass* (*SupClass*) *in metamodel*
6. **if** $(class \neq SupClass)\&\&(Class \sqsubseteq SupClass \notin TBox)$
   $\Rightarrow TBox = TBox \cup \{Class \cap SupClass = \emptyset\}$
7. **end if**
8. **end for**
9. **for all** *multiplicity condition between classes class*1 *and class*2
10. **if** *the value of condition is* 0 *to* 1
    $\Rightarrow TBox = TBox \cup \{class1 \sqsubseteq \leq 1.class2\}$
11. **end if**
12. **if** *the value of condition is* 1 *to n*
    $\Rightarrow TBox = TBox \cup \{class1 \sqsubseteq \geq 1.class2\}$
13. **end if**
14. **if** *the value of condition is* 1
    $\Rightarrow TBox = TBox \cup \{class1 \sqsubseteq \leq 1.class2,\ class1 \sqsubseteq \geq 1.class2\}$
15. **end if**
16. **end for**
16. **return** *TBox*

Table 4
Development of A-Box

1. **Initialize:** $ABox = \{\}$
2. **for all** $m \in M$ *and* $ob1,\ ob2 \in PO$
3. $ABox = ABox \cup \{Message(ob1,\ ob2)\}$
4. **if** $(ob1$ *send a message m to* $ob2)$
   $\Rightarrow ABox = ABox \cup \{send(ob1,\ m) \sqcap recieve(ob2,\ m)\}$
5. **end if**
6. **if** $(ob2$ *receive message* $m)$
   $\Rightarrow ABox = ABox \cup \{Operation(m) \sqcap Reply(ob1)\}$
7. **end if**
8. **end for**
9. **for all** $(ob,\ val1,\ val2,\ m1,\ m2) \in SF$
10. **if** $(val1 = val2)$
    $\Rightarrow ABox = ABox \cup \{Precede(m1,\ m2)\}$
11. **end if**
12. **end for**
13. **for all** $(l,\ c) \in AB$
14. $ABox = ABox \cup \{Instanceof(l,\ c)\}$
15. **end for**
16. **return** *ABox*

between these stereotypes, which are shown in Table 2. For description logic, an ontology can be represented as an assertion box (ABox) and a terminology-box (TBox) [56]. The ABox analyzes the instances of concepts at instance level, whereas the TBox analyzes the concept at class level. The M1 level model can be transformed into formal ontology by mapping meta-model into TBox at M2 level and mapping M1 level

model into ABox. For the development of TBox and ABox, two algorithms are specified in Table 3 and Table 4 respectively. These algorithms consider classes and metaclasses as input which are shown in Table 2.

Table 3 shows the development of a TBox that becomes helpful to map UML metamodel into a DL TBox. In this algorithm all meta classes and superclasses shown in Table 2 are mapped into a TBox. It also includes the multiplicity constraints into a TBox. In semantic analysis, behavioral model comprises both the static semantics as well as the dynamic semantics. Dynamic semantics may change during the execution time whereas the static semantics are not changeable until the model is built.

Table 4 shows the development of an ABox that becomes helpful to map UML model into a DL ABox. In this study, a sequence diagram is considered to specify using ABox. The formal semantics of a sequence diagram can be specified by using a tuple SD={ PO, M, AB, SF }, where PO is a set of participating objects, M is a set of messages, AB is a set of activation bar, and SF is a set of sequence fragments. Let us consider a sequence diagram having a message sender i.e., object ob1, and message receiver i.e., object ob2, activation life l.

4. Case Study: e-Health-Care System

In this approach a case study i.e., e-Healthcare system [57] is considered for the demonstration of ODSF modeling framework. e-HealthCare system offers digital integration of health-care related information such as patient-ID, BP monitoring report, ECG monitoring report, update patient record, patient report generation, etc., which are scattered over a myriad of traditional databases. For this case study, an analysis and design models are presented. Transformation of analysis model into design model is out of the scope of this approach.

4.1. Analyzing e-Health-Care system at the CIM and PIM level

In order to construct an analysis model, use case diagram has been taken into consideration as shown in Figure 9. A number of use cases are presented which are associated with static and dynamic authentication and authorization. In this diagram, four main actors such as Patient, Doctor, Security Designer, and Attacker are presented for the sake of simplicity. According to this model, Patient can read the record whereas Doctor can perform both the operations, read record and write record. The role of Security Designer comes into play during the deployment phase. Security Designer develops secure system by using security patterns as well as by applying security policy rules. An Attacker can assign threat to create vulnerability for accessing records.

Figure 10 represents a class diagram for securing confidential health-care data stored in databases using security patterns. These data can be accessed by using various web services such as request processing service, BP monitoring service, ECG monitoring service, etc. In this model, Authenticator and Authorizer packages are used which are instances of Authenticator pattern and Authorizer pattern respectively. Authenticator enables service controller to authenticate service consumers as well as other interacting services. Authorizer enforces access rights specified by policy rules. When a consumer wants to access a health-care related data through the Internet, e-HealthPortal forward consumer's request to trust-based security pattern. This pattern contains Security Token Service to verify the request. In this scenario, a trust-based security pattern uses Authenticator and Authorizer packages. If consumer's request verified against policy rules, it access resource through the service container package. Service container contains services used to access e-Health-Care database.

The behavioral aspect of a trust-based security patterns is represented in Figure 11. This sequence diagram is based on dynamic aspect of SOA security patterns which is shown in Figure 3. The model represents the behavioral aspect of e-Health-Care system for the use case accessBPReport. In this scenario, service consumer requests e-HealthPortal for accessing blood-pressure report. e-HealthPortal forwarded consumer's request to STS for verifying the request. STS verifies request by using Trust Engine. STS creates a Security Context Token (SCT) after verifying the request and provide it to the service consumer. SCT may have a security token in terms of claims, which are provided to consumer. These claims ensure the right to access the services. Consumer can check BP report using BP-Monitoring service after receiving the SCT. BPMonitoring service fetches the report from the database and provided to the consumer.

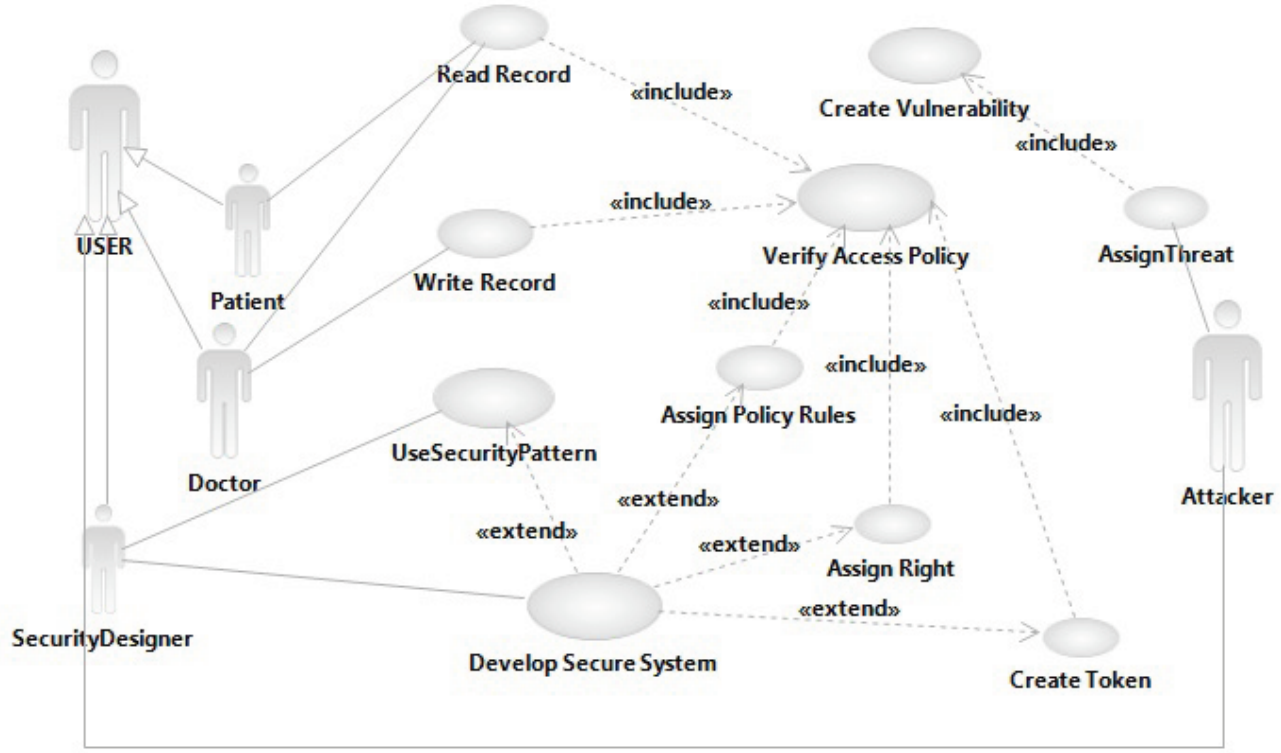

Fig. 9. Use case diagram for e-Health-Care system

4.2. *e-Health-Care system at PSM layer*

After applying security patterns in SOA-based security critical system at PIM layer, semantic web languages such as OWL, RDF, SPARQL (SPARQL Protocol and RDF Query Language), and SWRL (Semantic Web Rule Language) are applied for analyzing pattern notation. This study considers OWL associated with description logics for specifying UML-based pattern notation suffering from design inconsistency problem. Often pattern-based solution faced a problem on the decision of variation in the problem domain. In this situation, pattens fail to specify the selection of classes for a particular problem. These problems can be solved by performing OWL reasoning which infer class subassumption and object classification.

4.2.1. *Ontology for SOA Security Patterns*

The ontology-based framework provides the conceptualization for protecting web service communication process. It supports a high-level abstraction for the web service interactions. Ontology-based modeling extends the concept of URI (Uniform Resource Identifiers) for unique identification of resources. It provides the concept of namespaces for expressing consistent information spaces. A good number of ontology languages are presently available to provide a semantic description of a system. Ontology-based modeling differentiates facts about pattern templates, such as fact about problem domain (Semantic Web Service Security ontology), solution domain (Software Platform Ontology), and application domain (WS-Security Application Ontology). Figure 12 represents ontology classes and their semantic relationships. For the sake of simplicity all properties and assertions are not shown in the SOA security patterns ontology.

This ontology presents a pattern i.e., WS-Security pattern having various subpatterns, WS-Trust, WS-Policy, WS-Federation. These patterns are represented as classes. The relationships such as *isAbstract*, *isSubClassOf*, *isSubPatternOf* are used to specify pattern-based system. For example, WS-Policy pattern is a subpattern of WS-Security pattern. WebService and WS-SecurityPattern classes have *isAbstact* relationship because both are abstract classes. The relationships *isSubPatternOf* and *isSubClassOf* support transitive relation. In this ontology, SecurityTokenService

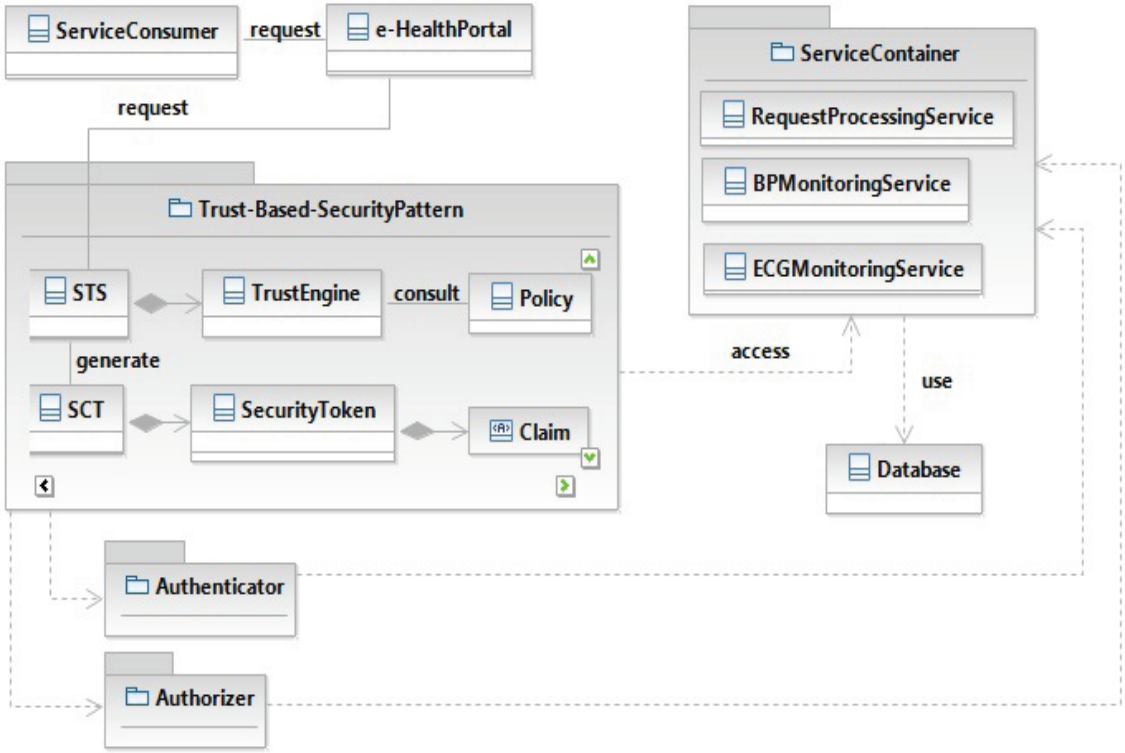

Fig. 10. Class diagram for e-Health-Care system

act as a main class related with WS-Security patterns using *hasAssurance* property for describing assurance level from SecurityTokenService. SecurityTokenService class has a number of other properties such as hasTokenVerifier, hasDefenses, and hasSCT associated with TrustEngine, SecurityThreats, and SecurityContextToken classes respectively for specifying security objectives.

OWL supports two types of properties i.e., object property and data property. Object property relates instance of one class to instance of another class. For example, The *hasDefenses* property can be specified to have a domain of the class SecurityTokenService and the range of the class SecurityThreats. Where an instance such as Secure-Logger can defense against Man-in-the-Middle attack. Data properties of OWL associate instances of a class to RDF (Resource Description Framework) literals or XML schema datatypes. For example, *hasClaims* can be a datatype property to determine whether the instance of SecurityToken contains a claim or not. Similar to isSubClassOf and isSubPatternOf relationships, properties can also be defined as sub-properties of other properties. Constraints can also be applied on the properties and sub-properties for controlling a number of instances of a property and sub-property associated with those classes.

Identity Provider class is associated with SecurityTokenService class using isSubClassOf relationship. Identity Provider has a property *hasAssociation* to specify the Domain class i.e., e-Health-Care. e-Health-Care has an object property *hasService* to relate e-health-Care services such as Request Processing service, BP Monitoring service, and ECG Monitoring service with the domain. These e-Health-Care services are subclasses of Web Service. WS-Federation pattern is associated with Identity Provider class using *isBasedOn* relationship. WS-Policy pattern is associated with Policy class using *isBasedOn* relationship. SecurityContextToken has a property *hasToken* to specify SecurityToken. The SecurityToken class includes two tokens type subclasses such as SignedSecurityToken and Proof-of-Possession token. Similarly SignedSecurityToken has two concrete tokens type subclasses such as X.509Token and KerberosToken and Proof-of-Possession include two tokens type SAMLToken and IdentityToken. The SecurityToken class has two prop-

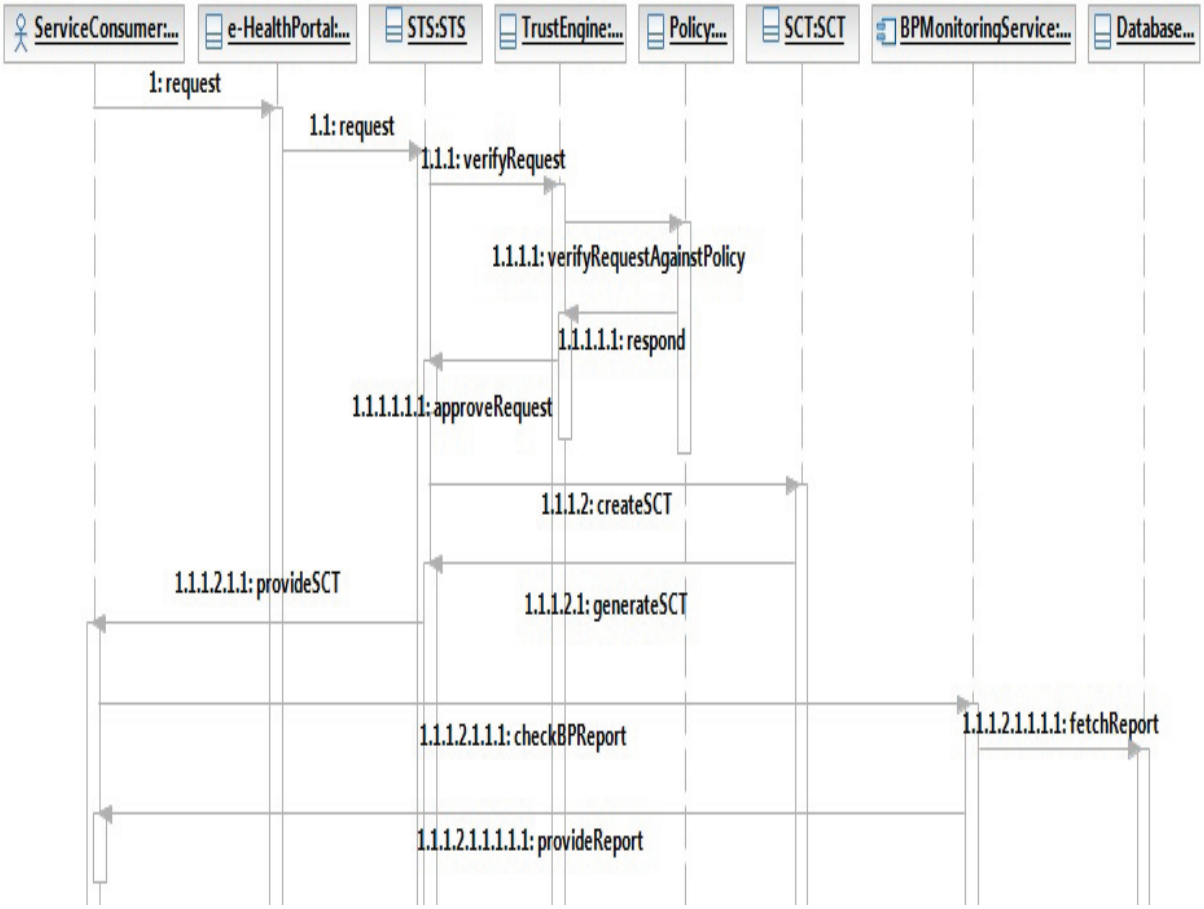

Fig. 11. Sequence diagram for e-Health-Care system for the use case access BP report

erties such as *hasClaims* and *assignPermission* to specify claim and e-Health-Care respectively.

In this ontology SecurityTokenService can request to a specific credential, such as password, keys, certificates, etc., using *reqCredential* property. It can be improved by creating classes which are essential for SOA-based security critical system. It classifies credentials into SignedSecurityToken and Proof-of-Possesion token using the statements '*what you have*' and '*what you are*' for authentication. Policy class can be classified into Policy-Based Access Control and Role-Based Access Control for providing authorization security objective using statement '*what you want*'. A number of credential requirements are also available for acquiring different types of authentication which can be represented by specifying credentials for *reqCredential* property values. For example a security requirement i.e., *LoginSystem* may has a credential i.e., X.509Certificate having value *minLength='6'*.

4.2.2. OWL notation for SOA Security Patterns profile

In the Table 5, OWL notation shows the WS-Security patterns profile presented in Figure 12. For the sake of simplicity, namespace declaration are omitted. This code represents security token assertions where X.509 certificate has a version three and serial number is one. The 'id' attribute specifies the local identification of token. Security capability represents Authorization Service capability using *XML Firewall* which provides policy-based access control. In this ontology 'wssc' namespace represents web service security capability. In a security token requirement, two requirements are specified. First token requirement represents *Login-Requirement* where login requires credential in terms of password. The role of credential is more prominent in web services for providing authentication. Credentials are available in various format such as certificates, encrypted keys, fingerprint, smartcard, etc. Second requirement contains an assertion e.g., *Se-*

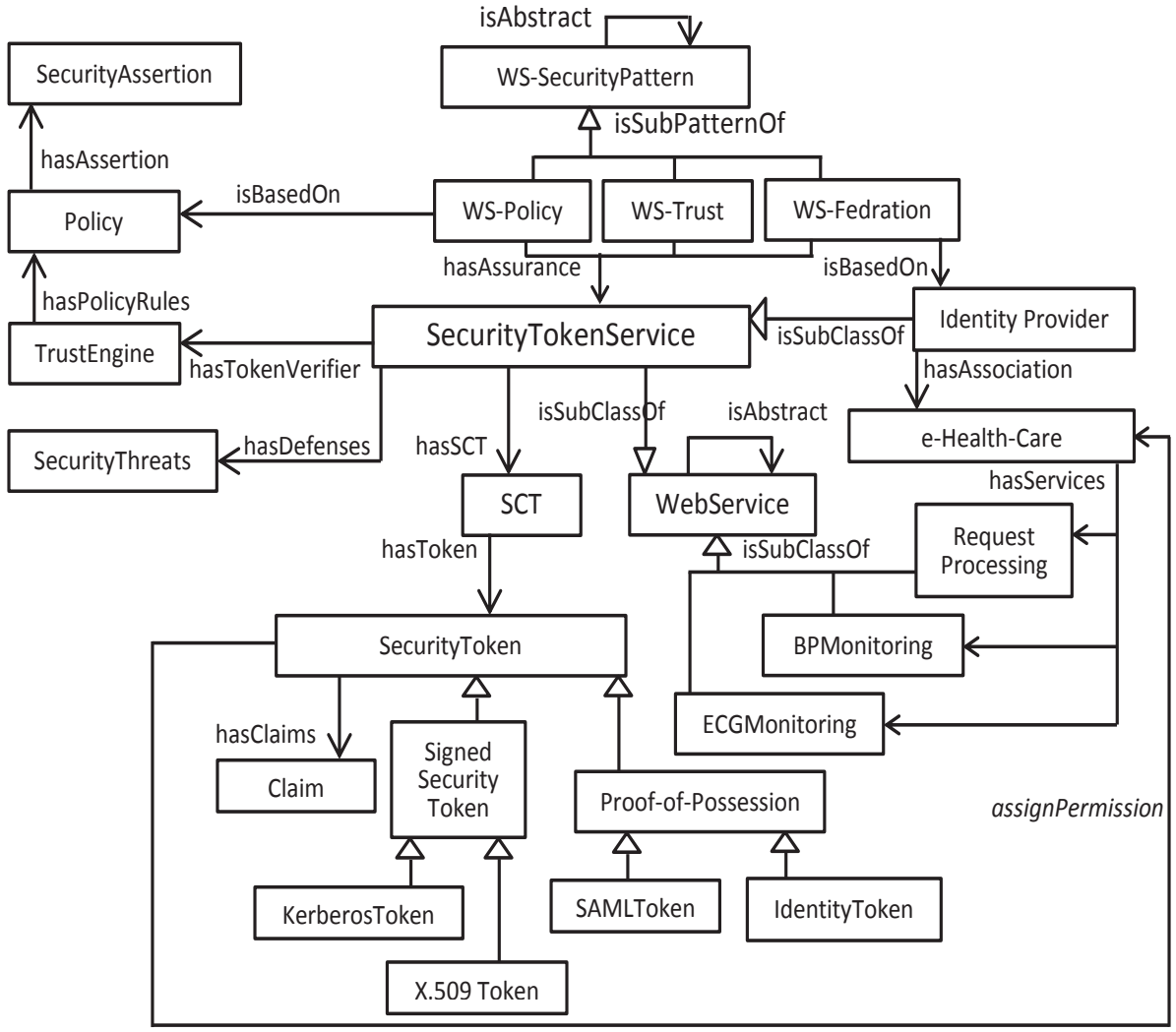

Fig. 12. Ontology for Security Pattern-Based e-Health-Care System

curityToken:TokenType, which specifies the type of a token.

A SOA-based system e.g., e-Health-Care contains number of security requirements at the consumer level as well as provider level. The service consumer can express its policy in terms of security requirements. The consumer can require authentication, access control, and confidentiality for accessing a resource through the Web. The service provider can represent their policies by specifying security requirements. The provider may require consumer authentication, change in the policy rules, a specific protocol for the consumer, etc. Table 5 also represents security requirements for e-Health-Care system such as access control requirement and authorization requirement. In this ontology 'wssr' namespace is considered, which denotes web service security requirements. In this requirement, access control provides assurance to e-Health-Care-Database by using X.509 token. In the second requirement, authorization can be achieved by using SAML (Security Assertion Markup Language).

4.2.3. DL notation for SOA Security Patterns profile

Description Logic (DL) formalizes various domains with the help of concepts and relationships, which are represented as classes and relations respectively. DL is similar to first order logic (FOL) added with some other notations. DL supports a number of mathematical operators, such as *and* ($\sqcap$), *or* ($\sqcup$), *not* ($\neg$), *subset-*

Table 5
OWL notation for SOA Security Patterns profile

$$<!--\ \textbf{SecurityToken Description}\ --!>$$
$$< SecurityToken : Certificate\ rdf : ID = \text{“}X.509Certificate'' >$$
$$< SecurityToken : version\ rdf : datatype = \text{“}\&xsd;\ int'' >$$
$$3 < /SecurityToken : version >$$
$$< SecurityToken : serialNumber\ rdf : datatype = \&\ xsd;int >$$
$$1 < /SecurityToken : serialNumber >$$
$$< /SecurityToken : Certificate >$$

$$<!--\ \textbf{Security Capability}\ --!>$$
$$< wssc : AuthorizationService\ rdf : ID = \text{“}Capability1'' >$$
$$< wssc : hasAssurance\ rdf : resource = \text{“}\&assurance;\ XMLFirewall''/ >$$
$$< /wssc : AuthorizationService >$$

$$<!--\ \textbf{SecurityToken Requirements}\ --!>$$
$$< SecurityToken : Login\ rdf : ID = \text{“}LoginRequirement'' >$$
$$< SecurityToken : hasCredential\ rdf : resource = \text{“}\&credential;\ Password''/ >$$
$$< SecurityToken : isIntitialCredential = \text{“}false''/ >$$
$$< /SecurityToken : Login >$$
$$< SecurityToken : TokenType\ rdf : ID = \text{“}Claim1'' >$$
$$< SecurityToken : hasClaims\ rdf : resource = \text{“}\&claims;\ identity''/ >$$
$$< /SecurityToken : TokenType >$$

$$<!--\ \textbf{Security Requirements for e-Health-Care}\ --!>$$
$$< wssr : AccessControl\ rdf : ID = \text{“}Req1'' >$$
$$< wssr : hasTarget\ rdf : resource = \text{“}\&resource;\ e-Health-Care-Database''/ >$$
$$< wssr : hasAssurance\ rdf : resource = \text{“}\&SecurityToken;\ X.509Token''/ >$$
$$< /wssr : AccessControl >$$
$$< wssr : Authorization\ rdf : ID = \text{“}Req2'' >$$
$$< wssr : hasTarget\ rdf : resource = \text{“}\&resource;\ e-Health-Care-Database''/ >$$
$$< wssr : hasAssurance\ rdf : resource = \text{“}\&PoP;\ SAML''/ >$$
$$< /wssr : TokenType >$$

of ($\sqsubseteq$), *equivalent* ($\equiv$), *model* ($\models$), *greater-or-equal* ($\geq$), *less-or-equal* ($\leq$) etc., for analyzing the properties of a system requirements. It also supports existential quantifier ($\exists$) and universal quantifiers ($\forall$). DL supports two other operators, such as ($\top$) and ($\bot$), which denote the meaning, having all individuals and no individual respectively. These two notations are helpful to model the constraints of UML-based patterns.

The proposed DL notations represent the formal relation and sensitive axioms for the SOA-based security critical system. The DL notations help to specify the constructs of UML diagrams. Table 6 represents various DL expressions for SOA security pattern. First expression denotes that WS-Security pattern is a subclass of Security pattern. Third expression shows that SecurityTokenService has a semantic relationship with WS-Trust, WS-Policy, and WS-Federation using *hasAssurance* property. Thirteenth expression denotes that Identity Provider is associated with e-Health-Care using *hasAssociation* property. Fifteenth expression describes about the access permission for e-Helath-Care using security token. Nineteenth expression indicates that all individual are subset of all hasPermission and all allowAccess properties. Expressions twentieth and twenty-first specify for exactly one access and exactly one permission respectively. Twenty second expression says that Authorized-Consumer and Attacker cannot be the same user. Twenty-sixth expression specifies about allowAccess operation performed by SecurityTokenService.

During the transformation process CRUD (create, Read, Update, Delete) operations may be generated for specifying pattern-based object-oriented properties. These operations are automatically generated during the SPARQL querying process. In this study, SPARQL based patterns are considered for encoding seman-

Table 6
DL notation for SOA Security Patterns profile

S.No.	DL Notation
1.	$(WS - SecurityPattern \sqsubseteq SecurityPattern)$
2.	$(WS - Policy \sqcap WS - Trust \sqcap WS - Federation \sqsubseteq WS - SecurityPattern)$
3.	$SecurityTokenService \equiv \exists\ hasAssurance.(WS - Trust \sqcap WS - Policy \sqcap WS - Federation)$
4.	$WS - Federation \equiv \exists\ isBasedOn.IdentityProvider$
5.	$WS - Policy \equiv \exists\ isBasedOn.Policy$
6.	$SecurityTokenService \equiv \exists\ hasTokenVerifier.TrustEngine$
7.	$SecurityTokenService \equiv \exists\ hasDefenses.SecurityThreats$
8.	$SecurityTokenService \equiv \exists\ hasSCT.SCT$
9.	$(SecurityTokenService \sqcap RequestProcessingService \sqcap BPMonitoringService \sqcap ECGMonitoringService \sqsubseteq WebService)$
10.	$Policy \equiv \exists\ hasAssertion.SecurityAssertion$
11.	$TrustEngine \equiv \exists\ hasPolicyRules.Policy$
12.	$(IdentityProvide \sqsubseteq SecurityTokenService)$
13.	$IdentityProvider \equiv \exists\ hasAssociation.e - Health - Care$
14.	$e - Health - Care \equiv \exists\ hasServices.(RequestProcessingService \sqcap BPMonitoringService) \sqcap ECGMonitoringService$
15.	$SecurityToken \equiv \exists\ assignPermission.e - Health - Care$
16.	$(SignedSecurityToken \sqcap Proof - of - Possession \sqsubseteq SecurityToken)$
17.	$(KerberosToken \sqcap X.509Token \sqsubseteq (SignedSecurityToken)$
18.	$(SAMLToken \sqcap IdentityToken \sqsubseteq (Proof - of - Possession)$
19.	$(\top \sqsubseteq \forall\ hasPermission.Permission \sqcap allowAccess.Policy)$
20.	$(Policy \sqsubseteq (= 1\ allowAccess.\top))$
21.	$(Permission \sqsubseteq (= 1\ hasPermission.\top))$
22.	$(AuthorizedConsumer \sqcap Attacker \equiv \bot)$
23.	$(\top \sqsubseteq (ServiceConsumer \sqcap ServiceProvider))$
24.	$(ServiceChoreography \sqcap ServiceOrchestration \sqcap ServiceComposition) \sqsubseteq ServiceOrganization$
25.	$\exists\ hasToken.\top \sqsubseteq \forall\ hasToken.SecurityToken$
26.	$\exists\ operation.allowAccess \sqsubseteq \forall\ sts.SecurityTokenService$

Table 7
OWL description using SPARQL pattern

Condition	OWL notation	SPARQL	Description
Attribute	SubPatternOf : name	*SELECT ?this WHERE{?this : name ?name FILTER regex(?this, ^PatternOf)}*	Each pattern and sub-pattern should have name
Relationship	SubPatternOf : ExtendEdge	*SELECT ?this WHERE{?this : ExtendEdge ?edge}*	A sub-pattern has an extend control edge with a corresponding pattern
Access Specifier	class : Access hasType[private, public, protected]	*SELECT ?this WHERE{?this : Access ?a FILTER(?a =′ private′ || ?a =′ public′ || ?a =′ protected′)}*	Access specifier is applicable for each class and attribute
Cardinality of relationship	class : relEdge	SELECT ?this WHERE { ?this :relEdge ?edge} GROUP By ?this HAVING(count(*?edge*) >= 2)	Each relationship has a cardinality value
One use-case includes other use-case	usecase : relEdge hasType[extend, include]	SELECT ?this WHERE { ?this :relEdge ?edge} type (ActivityNode **and** outgoing **some** (ControlFlow and Target **some** decision))	One use-case that include some other use-case

tic knowledge, because all OWL features can be expressed in SPARQL and it support semantic search in model repository. OWL2 supports a number of data properties, such as SubDataPropertyOf, EquivalentDataProperties, DisjointDataProperties, etc. as well as object properties, such as SubObjectProperty, ObjectHasValue, InverseObjectProperties, ReflexiveObjectProperty, SymetricObjectProperty, TransitiveObjectProperty, etc. For example a query *Type(?a, ObjectHasValue(securitypattern, WS-Trust))* returns all individuals that have the individual WS-Trust as value of the property securitypattern. Table 7 represents SPARQL patterns for the different types of conditions occur in UML model that can be mapped into OWL model. A resource belongs to design pattern represented using SPARQL patterns if it generates a result for the defined variable i.e., *this*. One of the advantage of SPARQL is that it is optimized for execution. It does not require a separate reasoner, the RDF storage can support querying process. Some features of SPARQL (query of access specifier in Table 7) are more prominent that are not possible using OWL. In this ODSF,

SPARQL and DL are combined to achieve the benefits of both. For the design patterns, classes are define in terms of common DL operations such as composition, union, intersection, etc.

4.2.4. Transformable elements between UML notation and OWL notation

Table 8 represents the constructs of UML notation which are mapped into OWL notation. A number of UML elements are available which have similar meaning with the OWL constructs. These elements become helpful during the transformation process. For example, OWL supports the instance concept by using OWL individual. UML association can be specified by using OWL properties. UML navigation concept is handled by OWL domain and range. OCL plays an important role to analyze system properties that can also be performed by using description logic.

5. Evaluation of ODSF

ODSF is analyzed by considering a case study i.e., e-HealthCare system. The main aim of this approach is to reduce the gap between PIM and PSM layers for providing semantic representation to UML-based security critical system. In this study, it is analyzed that how ODSF features reflect design-oriented non-functional requirements such as reusability, extendability, and security. ODSF allows developer to reuse existing ontologies, semantic annotations, and algorithm for the reusability. Extendability can be achieved by adding new assertions in the developed ontology. Developer can update the ontology if the UML model is not changed. Security analysis can be performed by using description logic.

ODSF is applied at high-level abstraction to offer a formal representation for security patterns. The evaluation of this approach is based on the fact as to how it represents the relevant concepts of a system, how it controls security threats during communication time, how precise it is, and how it can be applied to security critical system. The proposed methodology is based on sound theory and extensible for other models as well as other inconsistency problem. ODSF is also helpful for analyzing the dynamic aspect of SOA security patterns.

ODSF is compared with existing MDS approaches which is shown in Table 9. In this table, columns represent security modeling approaches, whereas rows represent attributes related to modeling approaches. The attributes taxonomy is taken from [20]. Four MDS approaches such as UMLsec [26], SecureUML [27], SECTET [28], and ModelSec [29] are considered to compare with the ODSF. Comparison between these modeling approaches are based on the parameters, such as security objective, modeling language, paradigm, problem domain, model2model transformation, verification, tool support, model as a web content. Most of the MDS techniques are developed for ensuring about security properties, CIA (confidentiality, Integrity, and Authentication). But SecureUML only concentrates on access control and others are helpful for different types of security properties, such as availability, non-repudiation, freshness, fair-exchange, auditing, etc. UMLsec, SecureUML, and SECTET use an UML language whereas ModelSec uses a non-UML-based language. ODSF uses OWL to model security-critical system. All the modeling techniques presented in Table 9 consider model-driven architecture (MDA) and domain-specific modeling paradigm except UMLsec. It uses multi-paradigm modeling platform. Most of the approaches are applicable for the web-based applications. UMLsec and SecureUML do not support model2model transformation process, whereas others support this process. SECTET and ModelSec do not offer explicit information for verifying security properties. UMLsec uses AICALL theorem prover, SecureUML uses SecureMova model checker, and ODSF uses Pellet for analyzing security properties specified in description logic. All the MDS approaches support automated tools except ODSF. The main advantage of ODSF is to model web security requirements as web contents.

6. Conclusion

Security design patterns reuse effective software design experience on solving critical security related problems. Patterns, those have good error detection and correction ability, lower data redundancy, and easy implementation, are useful for the system. In this study, an attempt has been made to systematically present a modeling approach to analyze SOA security design patterns. The presented study captures both structural and behavioral aspects of SOA-based security critical system that specifies variants using a semantic representation i.e., OWL. An ontology-based modeling and refinement framework is proposed for the web service security. During the refinement process, a DL-based approach is presented for the

Table 8
Mapping between UML-Based Pattern notation into OWL notation

S.No.	UML-Based Element	OWL-Based Element
1.	UML Package	Ontology
2.	UML Class	OWL Class
3.	UML Instances	OWL Individuals
4.	UML Attribute Values	OWL Data Values
5.	UMLAssociation.DataProperty	OWL DataProperty
6.	UMLAssociation.ObjectProperty	OWL ObjectProperty
7.	UML Datatypes	OWL Datatypes
8.	UML Generalization Relationship	OWL SubClass Relationship
9.	UML Navigation and Non-Navigation	OWL Domain and Range
10.	UML Enumeration	OWL Enumeration
11.	UML Multiplicity	OWL Cardinality
12.	OCL Expression	DL Expression

Table 9
Comparison of ODSF with Existing MDS Approaches

Modeling Approaches / Attributes	UMLsec	SecureUML	SECTET	ModelSec	**ODSF**
Security Objective	Confidentiality, Integrity, Authentication, Authorization, Non-Repudiation, Freshness, Fair-Exchange	Access Control	Integrity, Confidentiality, Non-Repudiation, Access Control	Integrity, Confidentiality, Privacy, Authentication, Availability, Non-Repudiation, Auditing, Access Control	**Authentication, Authorization, Fair-Exchange, Access Control**
Modeling Language	UML	UML	UML	SecML	**OWL**
Paradigm	MPM	MDA and DSM	MDA and DSM	MDA and DSM	**MDA and DSM**
Domain	Web Application, Embbeded System, Distributed System	Web Application	e-government, e-health, e-education	Web Application, Databases	**Web Applications**
M2M	No	No	Yes	Yes	**Yes**
Verification	Yes	yes	No	No	**Yes**
Tool Support	Yes	Yes	Yes	Yes	**No**
Model as a Web-Content	No	No	No	No	**Yes**

lightweight analysis of a trust and policy-based SOA security mechanism. A DL-based approach is considered to overcome the limitations of OCL notations. It can not be proved for its general properties using OCL expression, for example, executing OCL invariants over instance models. But it can be possible by using DL notations. Combination of OCL and DL makes an interesting analysis mechanism. In the process of model checking, analysis is a form of constraint solving. Analysis can disclose subtle flaws that software architect might not have discovered until much later. The presented guidelines are useful to check the inconsistencies and ambiguities among other security patterns.

In a current distributed and heterogeneous environment, modeling artifacts such as model, metamodel, modeling language, transformation language, etc., need to be linked, adapted, and formalized to accomplish the information requirements of various stakeholders. The proposed study helps for linking, mapping, and querying MOF based modeling languages on the web of data. Another advantage of this approach lies for simple demonstration of security re-

quirements, semantic specification of those security requirements, and evaluation of security properties along with selected software security patterns.

References

[1] Fernandez EB, Ajaj O, Buckley I, Delessy-Gassant N, Hashizume K, Larrondo-Petrie MM. A survey of patterns for web services security and reliability standards. *Future Internet* 2012; **4**(2):430–450.

[2] Kobashi T, Yoshioka N, Okubo T, Kaiya H, Washizaki H, Fukazawa Y. Validating security design patterns application using model testing. *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, IEEE, 2013; 62–71.

[3] Gamma E, Helm R, Johnson R, Vlissides J. *Design patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[4] Deepak Alur, Dan Malks, John Crupi, Grady Booch, and Martin Fowler. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Prentice Hall, 2nd edition, 2003.

[5] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, Boston, USA, 2002.

[6] Steel, C., Nagappan, R., and Lai, R. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall PTR, 2005.

[7] Schumacher M, Fernandez-Buglioni E, Hybertson D, Buschmann F, Sommerlad P. *Security Patterns: Integrating security and systems engineering*. Wiley: Hoboken, NJ, USA, 2006.

[8] Yoder J, Barcalow J. Architectural patterns for enabling application security. *In proceeding of the 4th Conference on Patterns Language of Programming (PLoP'97)*, 1997.

[9] Erl T. *SOA design patterns*. Prentice Hall: Upper Saddle River, NJ, USA, 2009.

[10] Delessy N, Fernandez EB, Larrondo-Petrie MM, Wu J. Patterns for access control in distributed systems. *Proceedings of the 14th Conference on Pattern Languages of Programs (PLoP2007), Monticello, IL, USA*, ACM, 2007; 5–8.

[11] Delessy-Gassant N, Fernandez EB, Rajput S, Larrondo-Petrie MM. Patterns for application firewalls. *Proceedings of the Pattern Languages of Programs Conference (PLoP2004)*, 2004; 8–12.

[12] Ajaj O, Fernandez EB. A pattern for the WS-Policy standard. *Proceedings of the 8th Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP 2010)*, 2010; 23–26.

[13] Ajaj O, Fernandez EB. A pattern for the WS-Trust standard for web services. *Proceedings of the 1st Asian Conference on Pattern Languages of Programs*, ACM, 2010; 1.

[14] Muñoz-Arteaga J, Fernandez EB, Caudel-García H. Misuse pattern: spoofing web services. *Proceedings of the 2nd Asian Conference on Pattern Languages of Programs*, ACM, 2011; 11.

[15] Ajaj O, Fernandez EB. A pattern for the WS-SecureConversation standard for web services. *Proceedings of the Pattern Languages of Programs*, 2012; 11.

[16] Uzunov AV, Fernandez EB, Falkner K. ASE: A comprehensive pattern-driven security methodology for distributed systems. *Computer Standards & Interfaces* 2015; **41**:112–137.

[17] Fernandez EB, Monge R, Hashizume K. Building a security reference architecture for cloud systems. *Requirements Engineering* 2015; :1–25.

[18] Ajaj O, Fernandez EB. A pattern for the WS-Federation standard for web services. *Proceedings of the Pattern Languages of Programs*, 2013; 16.

[19] Mike Dean and Guus Schreiber. Web Ontology Language. http://www.w3.org/TR/owl-ref/, 2004.

[20] Lucio L, Zhang Q, Nguyen PH, Amrani M, Klein J, Vangheluwe H, Le Traon Y. Advances in model-driven security. *Advances in Computers* 2014; **93**:103–152.

[21] Object Management Group. http://www.omg.org/ 1989.

[22] Cranefield S, Pan J. Bridging the gap between the model-driven architecture and ontology engineering. *International Journal of Human-Computer Studies* 2007; **65**(7):595–609.

[23] Staab S, Walter T, Gröner G, Parreiras FS. Model driven engineering with ontology technologies. *Reasoning Web. Semantic Technologies for Software Engineering*. Springer, 2010; 62–98.

[24] Henderson-Sellers B. Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software* 2011; **84**(2):301–313.

[25] Dermeval D, Vilela J, Bittencourt II, Castro J, Isotani S, Brito P, Silva A. Applications of ontologies in requirements engineering: a systematic review of the literature. *Requirements Engineering* 2015; :1–33.

[26] Jürjens J. UMLsec: Extending UML for secure systems development. *ń UMLż 2002ŮThe Unified Modeling Language*. Springer, 2002; 412–425.

[27] Lodderstedt T, Basin D, Doser J. Secureuml: A UML-based modeling language for model-driven security. *ń UMLż 2002ŮThe Unified Modeling Language*. Springer, 2002; 426–441.

[28] Alam M, Breu R, Hafner M. Model-driven security engineering for trust management in SECTET. *Journal of Software* 2007; **2**(1):47–59.

[29] Sánchez Ó, Molina F, García-Molina J, Toval A. ModelSec: a generative architecture for model-driven security. *Journal of Universal Computer Science* 2009; **15**(15):2957–2980.

[30] Deveci E, Caglayan MU. Model driven security framework for software design and verification. *Security and Communication Networks* 2015; .

[31] Kobashi T, Yoshizawa M, Washizaki H, Fukazawa Y, Yoshioka N, Okubo T, Kaiya H. TESEM: A tool for verifying security design pattern applications by model testing. *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, IEEE, 2015; 1–8.

[32] Katt B, Gander M, Breu R, Felderer M. Enhancing model driven security through pattern refinement techniques. *Formal Methods for Components and Objects*, Springer, 2013; 169–183.

[33] Delessy NA, Fernandez EB. A pattern-driven security process for SOA applications. *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, IEEE, 2008; 416–421.

[34] Kou S, Babar MA, Sangroya A. Modeling security for service oriented applications. *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ACM, 2010; 294–301.

[35] Alam M, Breu R, Breu M. Model driven security for web services (MDS4WS). *Multitopic Conference, 2004. Proceedings of INMIC 2004. 8th International*, IEEE, 2004; 498–505.

[36] Memon M, Menghwar GD, Depar MH, Jalbani AA, Mashwani WM. Security modeling for service-oriented systems using security pattern refinement approach. *Software & Systems Modeling* 2014; **13**(2):549–572.

[37] Basin D, Clavel M, Doser J, Egea M. Automated analysis of security-design models. *Information and Software Technology* 2009; **51**(5):815–831.

[38] Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne T, *et al.*. OWL-S: Semantic markup for web services. *W3C member submission* 2004; **22**:2007–04.

[39] Kim A, Luo J, Kang M. Security ontology to facilitate web service description and discovery. *Journal on data semantics IX*. Springer, 2007; 167–195.

[40] Garcia DZG, De Toledo MBF. Ontology-based security policies for supporting the management of web service business processes. *Semantic Computing, 2008 IEEE International Conference on*, IEEE, 2008; 331–338.

[41] D'Agostini S, Di Giacomo V, Pandolfo C, Presenza D. An ontology for run-time verification of security certificates for SOA. *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, IEEE, 2012; 525–533.

[42] Yu B, Yang L, Wang Y, Zhang B, Cao Y, Ma L, Luo X. Rule-based security capabilities matching for web services. *Wireless personal communications* 2013; **73**(4):1349–1367.

[43] Ben Brahim M, Chaari T, Ben Jemaa M, Jmaiel M. Semantic matching of web services security policies. *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, IEEE, 2012; 1–8.

[44] Dietrich J, Elgar C. Towards a web of patterns. *Web Semantics: Science, Services and Agents on the World Wide Web* 2007; **5**(2):108–116.

[45] Boaro L, Glorio E, Pagliarecci F, Spalazzi L. Semantic model checking security requirements for web services. *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, IEEE, 2010; 283–290.

[46] Di Modica G, Tomarchio O. Matchmaking semantic security policies in heterogeneous clouds. *Future Generation Computer Systems* 2015; .

[47] Parreiras FS, Staab S. Using ontologies with UML class-based modeling: The TwoUse approach. *Data & Knowledge Engineering* 2010; **69**(11):1194–1207.

[48] Katasonov A. Ontology-driven software engineering: Beyond model checking and transformations. *International Journal of Semantic Computing* 2012; **6**(02):205–242.

[49] Hästbacka D, Kuikka S. Semantics enhanced engineering and model reasoning for control application development. *Multimedia tools and applications* 2013; **65**(1):47–62.

[50] Elaasar M, Briand L, Labiche Y. Domain-specific model verification with QVT. *Modelling Foundations and Applications*. Springer, 2011; 282–298.

[51] Tounsi I, Hadj Kacem M, Hadj Kacem A, Drira K. A refinement-based approach for building valid SOA design patterns. *International Journal of Cloud Computing 2* 2015; **4**(1):78–104.

[52] Kim SK, Carrington D. A formalism to describe design patterns based on role concepts. *Formal aspects of computing* 2009; **21**(5):397–420.

[53] Brown KP, Capretz MA. ODEP-DPS: Ontology-driven engineering process for the collaborative development of semantic data providing services. *Information and Software Technology* 2013; **55**(9):1563–1579.

[54] OASIS. Reference ontology for semantic service oriented architectures version 1.0. http://docs.oasis-open.org/semantic-ex/ro-soa/v1.0/pr01/see-rosoa-v1.0-pr01.html November 2008.

[55] Clark & Parsia L. Pellet. http://semanticweb.org/wiki/Pellet August 2011.

[56] Baader F. *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[57] Hafner M, Breu R. *Security engineering for service-oriented architectures*. Springer Science & Business Media, 2008.