Model Federation based on Role Modeling

Bastien Drouot and Joël Champeau
Lab STICC UMR6285, ENSTA Bretagne, Brest, France

Keywords: Model Federation, Role Modeling, DSML Interoperability.

Abstract: Modeling approaches could be a powerful solution for specification, design and analysis. At a system level, models must take into account many system concerns. Thus, several system modeling approaches are based on several viewpoints expressed in Domain Specific Modeling Languages. Cyber threat analysis takes place within this modeling context with the need for several DSMLs to address several viewpoints of the system. So, the analysis of this domain is supported by DSML interoperability to perform simulation or other algorithms. Therefore, in this paper, we present an approach to face DSML interoperability based on role modeling. The Role4All framework is based on a metamodel including the Role concept. The Role4All language provides the capacity to define shared semantics between the DSMLs. Role4All and role modeling avoid model transformations and promote a federation approach between several DSMLs. The federation mechanisms of Role4All are illustrated in the cyber threat modeling framework to emphasize information gathering and the updates of the role model.

1 INTRODUCTION

In several domains, modeling approaches are accepted as solutions for the support of specification, design and, in some cases, implementation. These approaches are considered as powerful if the models are used for a precise and dedicated purpose such as systems engineering, system knowledge management, system analysis (structural, behavioral), performance evaluation (worst case analysis, scheduling) and also model or code generation. To take into account the diversity of system aspects, modeling approaches are mostly based on several Domain Specific Modeling Languages (DSMLs) or a standard Language with specializations (SysML and its profiles). In any case, these approaches aim to focus each DSML on a precise domain to provide adequate abstractions to fit with the domain concepts, or a subset of the concepts.

Currently, one of the active research domains is the security domain for systems integrating digital subsystems. These systems must be analyzed on several levels of abstraction (from binary to system levels) and from several viewpoints (network infrastructure or operating system of course). For this kind of problematics, models are relevant to outline system features, to provide relationships between features or concepts and to support different kinds of analysis from human expertise to automatic algorithms such as artificial intelligence.

In this context, we demonstrate that cyber threat analysis must be supported by a set of DSMLs, dedicated to providing abstractions for system representation, vulnerability definition, attacker competencies, etc. Cyber threat analysis is an activity using current knowledge and hypothetical attacker ability in order to anticipate cyber attacks. So one of the goals of this activity is to explore attacker behavior in the current system to understand and anticipate attack scenarios. Relative to this situation, models are a powerful means to formulate hypotheses and obtain attack scenarios to prevent real cyber attacks or to produce defense or monitoring artifacts.

To improve the results of this kind of analysis, we need to correlate different models to obtain the most relevant analysis. In the resulting modeling context, we face the interoperability problem of several DSMLs which model a system from several viewpoints. In this paper, we propose a model federation approach to solve the interoperability problem between several DSMLs. This federation approach is based on role modeling, the main property of which is to provide dynamic interface definitions. Our approach is illustrated by a relevant cyber threat modeling context.

Our paper is organized as follows: we present the cyber threat analysis context to clarify our requirements in Section 2. Section 3 focuses on the

Drouot, B. and Champeau, J.
Model Federation based on Role Modeling.
DOI: 10.5220/0007363500720083
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 72-83
ISBN: 978-989-758-358-2

DSML interoperability and the role modeling approach. We introduce our model federation language and approach in Section 4. Section 5 illustrates our Role4All framework in the cyber threat modeling context. Finally we conclude the paper in Section 6.

2 CYBER THREAT CONTEXT

2.1 Cyber Threat Analysis

Cyber threat analysis aims to look for system vulnerabilities, in relation to current vulnerability knowledge and tries to anticipate an attacker's behavior, while making assumptions as to his system knowledge and his attack goals.

The Threat analysis approach is at the crossroads between penetration testing methodology (Holik et al., 2014), cyber threat intelligence (Conti et al., 2018) and attack analysis (Elahi et al., 2010). Given this crossroads, several methodological steps have been identified:

• *Modeling*: Several system elements are conceptualized to produce an abstraction of the considered system. For example, attack trees are part of the modeling artifacts used to represent the goal of a potential attack and all the possible actions to achieve this goal. The main objective of this modeling step is to gather information and to create a representation of the current knowledge of the system.

• *System Discovery*: The knowledge of a system depends on the current view of the real system. For example, extending knowledge of the system can lead to the identification of accessible vulnerabilities regarding the architecture and configuration of the system. System discovery consists in the description of the actions and commands that can be performed on the system to extend the knowledge of the latter.

• *Vulnerability Exploitation*: A vulnerability is an error or weakness in design, implementation, or operation (Schneider, 1999). Based on the previous discovery step, vulnerabilities can be exploited to reach the attack goal or an intermediate step in the overall attack scenario. The selection of suitable vulnerabilities depends on the system configuration and is motivated by the expected progress in the current scenario. The purpose of the vulnerability exploitation step is to perform the attack scenario based on attacker ability, system access and system configuration.

Thus, the threat Analysis approach can be split into three steps: Modeling, System discovery and Vulnerability exploitation. Each of these steps can be approached using models. Indeed, the vulnerability exploitation is based on the modeling step and the system discovery can be simulated through models. Therefore, in the following parts we focus on the model approach to performing threat analysis.

2.2 Cyber Threat Modeling

Threat modeling is about using models to find security problems. The use of models means an abstraction of the system, the threats, the attackers and some other details (Shostack, 2014). There is no consensus about the threat modeling approach, for example in (Myagmar et al., 2005) Myagmar et al. propose a threat modeling approach consisting in the following three high-level parts: characterizing the system, identifying assets and access points, and identifying threats. While Pauli and Xu in (Pauli and Xu, 2005) center the threat modeling around three ideas: describing the decision-making process of an attacker, modeling the security threats and designing the system. Although the threat modeling approaches of Myagmar et al. and Pauli and Xu are different, it is possible to review the subgroups they create using a common theme.

Regarding the literature, we separated the threat modeling into three parts:

• *System Modeling*: System modeling consists in characterizing the system by describing the system behavior, its features, its boundaries, etc. System modeling could be split into sub-parts such as the system topology and the system configuration. System modeling represents the characterization of the system for Myagmar et al. and the design of the system for Pauli and Xu.

• *Attacker Modeling*: Attacker modeling consists in characterizing the attacker and his behavior, that includes the attacker's goal, the attacker's competencies and attack scenarios. Attacker modeling represents the identification of assets and access points for Myagmar et al. and the description of the decision-making process of an attacker for Pauli and Xu.

• *Threat Description*: Threat description consists in describing the possible vulnerabilities of the system. Threat description represents the identification of threats for Myagmar et al. and the modeling of security threats for Pauli and Xu.

Our threat modeling process includes the same steps as the processes of Myagmar et al. and Pauli and Xu but with a different grouping. Our grouping highlights the modeling part of the threat modeling and allows a clear distinction between the system, the threats and the attackers. One of the possible implementations of our representation of the threat modeling, based on various DSMLs, is:

• *System Topology*: A subset of the system modeling describing the boundaries of the system and the relations between elements in the system. The DSML chosen to describe the system topology was PimCa (Hemery, 2015). PimCa was defined by a cybersecurity entity of the French Ministry of Armed Forces, dedicated to representing and analyzing a system at several levels of abstraction, from an electronic to a systems level. The system topology is modeled by several entity types (processing, storing, customs, passport, etc.) and specialized relationships between the entities (controls, uses, owns, etc.).

• *System Configuration*: The configuration of each element of the system. We decided to use a configuration file based on structured data as a DSML. This DSML allowed the element configuration to be described including version numbers but did not allow the interaction between configurations to be modeled.

• *Vulnerabilities*: A vulnerability library. All systems have vulnerabilities, most of them referenced in a library. We chose to limit our vulnerabilities to a single library: NVD the US National Vulnerability Database (Zhang et al., 2011).

• *Attacker*: Attacker decision-making during an attack. This decision depends on the attacker's competencies and knowledge of the system. The decision-making of the attacker is symbolized by an attack tree (Mauw and Oostdijk, 2005). The attacker's competencies are modeled by structured data based on the description of an attack in CVSS (Mell et al., 2006). His knowledge of the system is represented by a specific viewpoint of the system changing step-by-step over the attack.

Each of these DSMLs is one of the possible DSMLs, for example, an attack tree can by replaced by Misuse cases (Alexander, 2003), or a fault tree (Lee et al., 1985), or a data-flow diagram (Li and Chen, 2009). In this article, the set of DSMLs chosen is: PimCa, configuration files, NVD, attack tree and attacker competencies structured data. Moreover, to generate and animate the attacker viewpoint step-by-step, we decided to operate a simulation. Our simulation allows the action of the attacker to be simulated independently of his attack scenario. The attack scenario only depends on actions taken, according to the attacker's viewpoint and abilities.

To summarize, we chose to group the threat modeling into these three groups and we implemented our representation of the threat modeling with several replaceable DSMLs that may evolve. This is why interoperability between DSMLs must be managed, taking into account that they can be modified or replaced, and the way that DSMLs interact can also change.

3 BACKGROUND AND PROBLEMATICS

3.1 DSML Interoperability

Interoperability between several formalisms is a recurrent problem in Model Driven Engineering (MDE) approaches in the scope of a modeling and simulating process or a development process including models at several steps. Traditionally, interoperability approaches are classified through integration, unification or federation of the formalisms (Rio, 2012; Guychard et al., 2013; Niemoller et al., 2013). Integration is based on the creation of a modeling language including the union of all the concepts of the different formalisms. This approach is applied if the concept number is limited and in particular if the number of formalisms remains fixed because any new DSML integration requires the redefinition of the entire integrated language which is the major drawback of this approach.

The unification mechanism is based on the identification of common concepts between the formalisms and the definition of the correspondence between these core concepts and the concepts of the modeling languages. This approach is usually named as the pivot language approach. This strategy is one of the most used to conceptualize and implement interoperability between several formalisms. In the case of a limited number of concepts, pivot language is a powerful solution if any concept of the languages finds its correspondence in the pivot definition. After the correspondence definition, transformations can be applied between all the formalisms and the pivot language, and also in the reverse direction. However, the definition accuracy of the pivot language remains a difficult task. If the definition is too abstract, this leads to lost concepts (or properties), if the concepts (or properties) have no correspondence in the pivot language. On the other hand, defining a rich pivot language, to preserve any property of the languages, produces an overly broad pivot language. In this case, the unification is equivalent to the integration approach with these drawbacks, and the management of the language transformations.

The federation approach shifts the point of view and focuses on modeling the semantics of the links between the concepts (Guychard et al., 2013; Niemoller et al., 2013). These links or relationships define the correspondence between the concepts without a concrete pivot language definition. The main goal of the federation is to specify the interoperability without modifying the language concepts, by concentrating on the semantic definitions. Based on this interoperability specification, the transformations are applied only on demand. Several federation approaches exist but for all, the main advantage is the clear separation between the source element with its original type (the DSML's concept) and the semantic modeling relative to the federation context (the federation's concept). A source concept could have several semantics within the same federation and also several source concepts could share the same semantics.

Relative to the cyber threat modeling context, we have chosen to apply the federation approach to ensure the DSML interoperability while taking into account the immature modeling context and thus the probable evolution of the DSML definitions. This evolution feature is mainly the major drawback for DSML interoperability. So our federation requires a flexible definition and a support to agree with the DSML definition evolutions. In this context, the ability to define a dynamic interoperability is ensured by role modeling, which is one of the main strengths we will expand upon in the next section.

3.2 Role Modeling

Role modeling has its roots in the work on data modeling during the seventies and as natural extensions the roles were used in object-oriented design and implementation, and modeling and metamodeling. In (Steimann, 2000) Steimann presents a short survey on role modeling which emphasizes the ontological definition of roles. This definition clearly highlights the difference between the natural type of individuals and the roles where the individuals could enter or leave without losing their identity.

To summarize, roles are used to provide dynamicity on classic type approaches (Kühn et al., 2014) and to define dynamic interfaces which can be adapted over time (Gottlob et al., 1996). Furthermore, these usages can be on several levels, metamodeling, modeling, and implementation.

Relative to our context, the natural type is provided by a metaclass of the DSML definition, and roles support the semantic interpretation of the elements according to the current attached role. Steimann provided a reference set of 15 features, with their semantics, that must be fulfilled by a role-based modeling framework (Steimann, 2000). Based on this reference list of features, Kühn et al. upgraded the framework by adding the ability to define a context for a group of roles (Kühn et al., 2014).

Roles are successfully used to interconnect heterogeneous design tools in order to create tool chains dedicated to system design with many necessary tools. So, building a tool chain requires an interoperability support that can be based on roles. Seifert et al. connected various models produced by different modeling tools thanks to roles (Seifert et al., 2010). Moreover, Champeau et al. used roles to guarantee the preservation of the semantics of a model entity during the model exchange process between modeling tools (Champeau et al., 2013). Roles enabled the modification of tools underlying metamodels to be avoided.

To define the necessary interoperability support between our DSMLs, we chose to improve the Role4All framework (Schneider et al., 2015) to add a federation support from the previous design and apply the framework to our cyber security context.

4 Role4All: ROLES AND FEDERATION

In this section, we introduce Role4All: a role-based framework including a role language. This language takes advantage of the role concept to define dedicated viewpoints and to map these viewpoints on several DSMLs. The language Role4All is based on the relevant subset of role features to ensure the federation of several modeling languages.

The purpose of the federation is to clearly separate the syntax of the model elements and the semantics defined by the federation model. In our case, the syntax is supported by the DSMLs and the semantics by the role model. Thanks to this context, we define and present the Role4All language and the weak coupling between the model elements, the syntax, the roles, and the semantics.

4.1 Role Features

In the literature, a significant effort has been made to provide an exhaustive formalized feature list to characterize and define the role concept in the context of role modeling (Steimann, 2000; Kühn et al., 2014). T.Kuhn et al. listed 26 necessary role features, reported in Table 1, to cover all the potential aspects of a role language definition. Following on from defining a role-based language is first a selection of the relevant features in the identified set, regarding the targeted context and domain. This feature selection creates various role language definitions. The associated tooling on these languages can introduce some usage variations but the role semantics remains preserved if the selected features are a strict subset of the identified features (Kühn et al., 2015).

Our main driver defines dynamic viewpoints between several DSMLs and therefore between the sup-

Table 1: Classification of role features.

1. Roles have properties and behaviors	15. An object and its roles have different identities
2. Roles depend on relationships	16. Relationships between roles can be constrained
3. Objects may play different roles simultaneously	17. There may be constraints between relationships
4. Objects may play the same role several times	18. Roles can be grouped and constrained together
5. Objects may acquire and abandon roles dynamically	19. Roles depend on compartments
6. The sequence of role acquisition and removal may be restricted	20. Compartments have properties and behaviors
7. Unrelated objects can play the same role	21. A role can be part of several compartments
8. Roles can play roles	22. Compartments may play roles like objects
9. Roles can be transferred between objects	23. Compartments may play roles which are part of themselves
10. The state of an object can be role-specific	24. Compartments can contain other compartments
11. Features of an object can be role-specific	25. Different compartments may share structure and behavior
12. Roles restrict access	26. Compartments have their own identity
13. Different roles may share structure and behavior	
14. An object and its roles share identity	

ported tool languages. In this context, our objectives are to take into account:

• A common semantic interpretation of several concepts of different languages. For example in a design process, at the implementation step, a system function of the application domain and a *Thread* in a concurrent design are federated to a *Class* concept of an object language, to provide a common implementation interpretation.

• The semantic evolution of a language concept during a modeling process. For example along a design process, a system function relative to the application domain is interpreted as a *Thread* in a concurrent design and as a *Class* of an implementation language.

• The provision of a dynamic semantics to the federated concepts. In our system design process, a system function can be implemented as *class* or *Threads*. Moreover, in a simulation framework the two can be simulated as an agent concept to analyze and evaluate the system design.

In Role4All, each role, called *Role*, gathers a subset of these 26 features. In this paper, instead of a list of role features, we classify them according to their underlying objectives in Role4All. The three most important objectives are: defining viewpoints, federating behaviors between elements and ensuring dynamic definition of these viewpoints.

• Defining viewpoints: As in role modeling approaches, in Role4All a *Role* defines a viewpoint of one or several model elements. As previously explained, we need to create a single viewpoint to define a common semantics from several DSMLs. This objective is covered by Features 1, 3, 4, 7 and 15. Each feature has a dedicated purpose:

- *Feature 1* enhances elements thanks to properties and behaviors.
- *Features 3 and 4* deal with several interpretations and thus several role models applied at the same time.
- *Feature 7* defines a viewpoint related to different elements of different languages.
- *Feature 15* requires a separation between the model elements, which keep their native identity, and the roles of these elements, to model viewpoints and relationships between model elements.

• Federating behaviors between elements: Roles are used to federate model elements (Wende et al., 2009). Role4All uses *Role* instances (named *roles*) to federate model instances from different languages. The federation is supported mainly by two features:

- *Feature 18* groups *roles* together for the federation.
- *Feature 13* enables behavior between federated *roles* to be shared.

• Defining dynamic viewpoints: One of the most important features in role modeling is to support dynamic evolution of the viewpoint definitions. The roles are attached or detached to the viewpoint to adapt the semantics relative to viewpoint evolution. For example simulation results can generate new roles relative to some existing model elements. This objective is supported by the following features:

- *Feature 5* provides the capacity to attach new roles, including the integration of simulation results.
- *Features 8 and 9* deal with the adaptation of viewpoint definition by adding new roles on a role or by migrating a role through another object.

Finally, a role in Role4All is totally defined by the Features: 1, 3, 4, 5, 7, 8, 9, 13, 15 and 18. In the next section, we describe how these features are combined and linked in the Role4All metamodel.

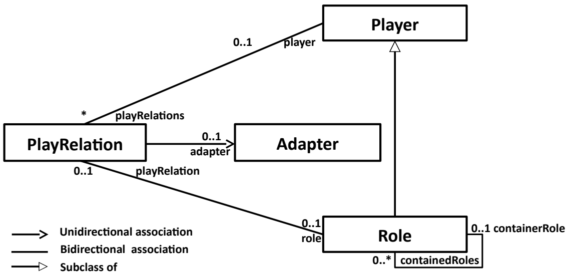

Figure 1: Role4All metamodel.

4.2 Role4All Design: Viewpoint

Based on the previous design of Role4All (Schneider et al., 2015), in this paper we present an extension of Role4All to ensure model federation. The Role4All definition is based on a metamodel, presented in Figure 1. This metamodel results in the selected Features 1, 3, 4, 5, 7, 8, 9, 13, 15 and 18. In this paragraph and the following, we present how this metamodel gathers the features together.

As a reminder, we use *role* to talk about a role instance and *Role* to talk about a role class and idem for the other metaclasses (*player* and *Player*, *playRelation* and *PlayRelation*, *adapter* and *Adapter*). Regarding our DSML interoperability context, a *Player* represents an element of one of the DSML's metamodels and a *Role* is a viewpoint of one or several *Players*. The key structure in our metamodel is: A *player* plays a *Role* means that a *player* is linked with a *role*. Moreover, relative to the metamodel, the link is reified by a *playRelation*. This relation ensures a clear separation between the *Player* and the *Role*. This relation can support its own properties, in particular, the *Adapter* definition. The *adapter* is in charge of transforming the *player* properties into *role* properties.

Each role feature is supported by the Role4All metamodel substructure. The definition of a viewpoint is supported by:

- *Feature 1: Roles have properties and behaviors*: A *Role* is defined by the metaclass Role. Thus, the generic semantics of a role defines this metaclass. Furthermore, a *Role* is defined by a class with behaviors and properties. So the operational semantics of the role is defined by the methods to encapsulate behaviors and property access.
- *Feature 3: Objects may play different roles simultaneously*: A *player* may have many *playRelations* according to the cardinality "*" of the relation between *Player* and *PlayRelation*. As we stated before, the *PlayRelation* emphasizes the separation between a DSML element and several Roles.
- *Feature 4: Objects may play the same role several times*: In addition to the cardinality of the relation between *Player* and *PlayRelation*, a *player* may be linked to the same *Role* several times but with a different *adapter*. So, a *player* may play the same *Role* several times with a different *Adapter*, or not.
- *Feature 7: Unrelated objects can play the same role*: The dissociation between *Role*, *Player* and *Adapter*, allows various *players* to play the same *Role*, with a specific *adapter*.
- *Feature 15: An object and its roles have different identities*: A *role* and the *player* linked to it are instances of different classes in Role4All.

So, Role4All metamodel substructures support Features 1, 3, 4, 7 and 15 allowing viewpoints to be defined. In the next section, let us focused on the features mainly dedicated to the concept federation.

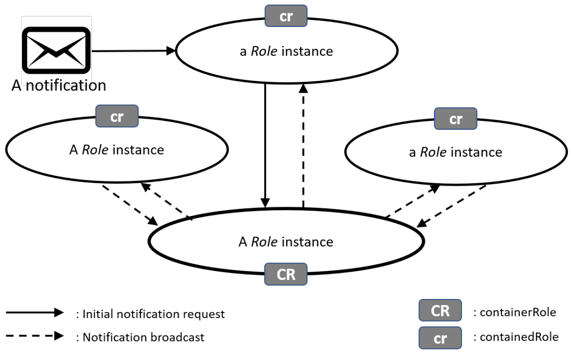

Figure 2: Notifications in the federated models.

4.3 Role4All Design: Federation and Dynamicity

The main goal of the federation approach is to face the issue of the definition of relationships between several formalisms. Each formalism defines a viewpoint of a system, and the federation specifies cross cutting concerns on the formalisms. The resulting federation model has two main properties: first, taking into account the dynamics of the relationships between the formalisms and second, ensuring consistency between the model elements of the different viewpoints. So the main features to support the federation are:

- *Feature 5: Objects may acquire and abandon roles dynamically*: The *Player* class has a behavior method, "*play*", to allow at run time, the creation of a *playRelation* to connect a *role*, an *adapter* and itself. Moreover, the removal of a *playRelation* entitles a *player* to release a *Role* dynamically.
- *Feature 8: Roles can play roles*: The class *Role* is a subclass of *Player*, so a *role* has the inherited be-

havior (especially the method "*play*") of a *player*. Therefore, a *role* can play a *Role*.

- *Feature 9: Roles can be transferred between objects*: A role can be transferred to another object via the creation of a new *playRelation* or a mutation of the older relation towards the other object.
- *Feature 13: Different roles may share structure and behavior*: A *Role* is defined via a class, so subclasses create *Roles* to share behavior and structure with the parent *Role*.
- *Feature 18: Roles can be grouped and constrained together*: The class *Role* and the association between *containerRole* and *containedRoles* allow a *role* to contain other *roles* or to be contained by another *role*. This association allows the creation of role sets. The behavior of the container includes constraints applicable to all the roles of this set. As we presented below, this feature is one of the main supports of the role federation.

Feature 5, 8 and 9 are inherently role concept properties to ensure dynamicity and interoperability between the formalisms from the role model. Indeed, if new concepts (or properties) are required in the federation definition, we add *roles* to adapt this definition. Furthermore, if concepts (or properties) are no longer necessary in the federation definition, we can detach the corresponding *roles* and remove the associated *playRelation* and *adapter*. These capacities ensure a dynamic definition of the federation approach.

Feature 13 and 18 ensure consistency between roles and model elements. In the metamodel (Figure 1), this feature is supported by the reflexive reference on the Role class, with one *containerRole* and several *containedRoles* if each *containedRoles* is connected with a model element. This *containerRole* provides the capacity to gather a logical assembly of *roles*, in order to define a set of federated *roles*. This container also maintains the overall behavior of this set based on local *role* behaviors and supports broadcasting notifications which come from local behavior.

To illustrate this definition, when a model element is updated, it notifies its associated *role*. This *role* transmits the notification to its *containerRole* and then, the *containerRole* broadcasts the notification to all the *containedRoles*. This algorithm, illustrated in Figure 2, ensures the goal to notify each *containedRole* federated in the container. In the case of notification with no returned object, each *containedRole* processes the notification, and thus typically updates a property of every model element associated with each *role*. In this case, the container does not ensure a particular consistency rule and the updating of all the model elements, via the *containedRoles*, is performed. In the case of notification with a returned object, a sequential order semantics is applied and the first non null object is considered as the result of the broadcast. This answer selection algorithm could be extended to select an answer relative to the evaluation of an assertion or a property of the container. These two examples are detailed in the next section.

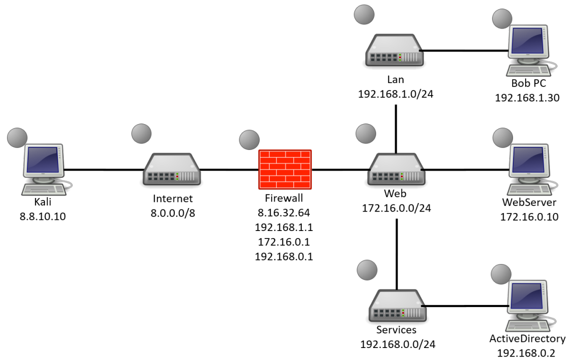

Figure 3: Network architecture of the use case.

Regarding the integration of the behavior in our metamodel, we precise that Role4All is implemented as an embedded DSL in Pharo/Smalltalk[1]. This kind of implementation provides facilities to have a clear DSL's definition and also a powerful implementation based on Smalltalk code for the behavior definition of the meta-entities and entities.

5 THREAT MODELING USE CASE

5.1 Case Study

In the previous parts, we presented our federation approach based on role modeling supported by the Role4All framework. In this part, we exemplify our approach through the modeling space of the cyber threat analysis use case. As defined previously, cyber threat analysis requires several viewpoints to take into account the system modeling, vulnerability description and the attacker modeling.

Each viewpoint is defined by a dedicated DSML and applied to a precise domain. In this context, a model federation approach is required to create an attacker viewpoint, which represents the attacker's current view of the real system throughout the scenario.

This attacker viewpoint emphasizes the main purpose of the role modeling by creating a semantic viewpoint of several source DSMLs. Based on this

[1] Pharo Distribution (http://pharo.org)

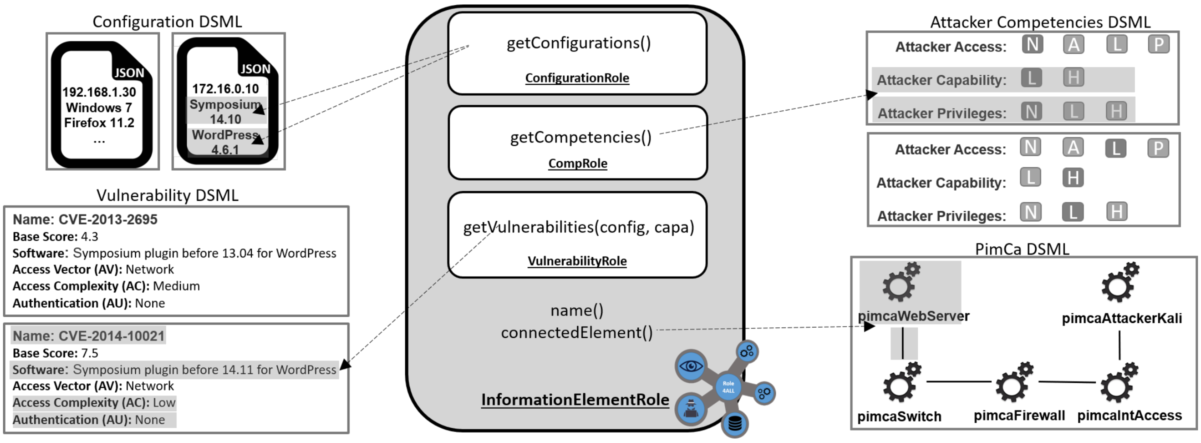

Figure 4: Conceptual view of the federation framework.

modeling infrastructure, we illustrate three main features of our federation approach:

• How to gather information from several DSMLs to create a common and shared semantics ?

• How this shared semantics is used to notify and update the DSML syntax elements ?

• How to dynamically update this shared semantics ? This is one of the foundation features of role modeling and federation approaches.

To obtain representative models of the cyber threat analysis, several sources exist, from bibliographic descriptions (Holik et al., 2014) to penetration testing techniques on the real system. Another way is to use an isolated virtual platform[2] and experiment sophisticated cyber attack scenarios in this sandbox environment. This flexible way provides the capacity to experiment several strategies and create quick iterations between modeling and experiments, in order to obtain the most representative models.

In this context, models and simulation focus on a system level and on entity behaviors, to overlook low level constraints and activities such as penetration testing. However, to be representative, a sophisticated cyber attack scenario is held as the reference on which the system is modeled , with all the associated features (configuration and vulnerabilities), and also the attacker skills required to perform the scenario.

This scenario is performed on a networked system (Figure 3) including three parts: the attacker's computer, the Internet and the target local network which embodies most company architectures with a web server and a local network.

In the network figure, the Kali machine symbolizes the attacker's computer, the Internet is abstracted by a switch, connecting the Kali machine to the local network. The target local network consists of three subnets, all protected by the same firewall. The cyber threat analysis includes a risk analysis emphasizing that the active directory is a critical resource due to the contained access accounts (login and password).

In the virtual environment, the attack scenario includes three main steps:

• Identifying vulnerabilities of the web server by using first the *nmap* command to localize the web server and *wpscan* to identify vulnerabilities in the WordPress framework deployed on the web server.

• Using a vulnerability to take control of the machine by privilege elevation with the *metasploit* program to execute the payload *cowroot* malware.

• Inside the local network the active directory administrator is targeted and his access account is obtained by a Trojan horse.

This experimental approach to perform this scenario on the virtualized system is close to reality. However, we need to test enough scenarios to be relevant and this task is time consuming and requires competencies, on several levels of abstraction: network, operating system, applications.

Our approach is to provide a modeling framework on this use case to simulate the scenario based on the federated models. The expected benefit is to experiment and analyze several system configurations and several scenarios according to different attacker competencies. In the next section, we present the modeling space and the federated approach, both illustrated relative to this use case.

5.2 Federation through Role4All Role Models

As presented previously, we federate several DSMLs through a common semantics. This common semantics defines the attacker viewpoint and enables DSML

[2] HNS Platform (https://www.hns-platform.com)

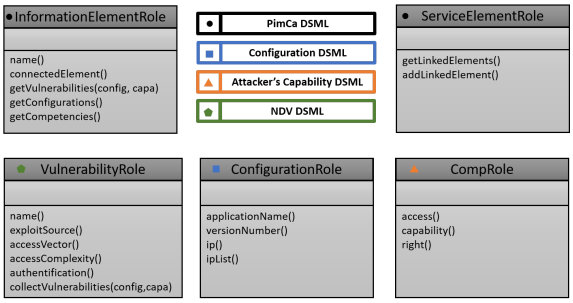

Figure 5: Federation Role Model.

model access without any model copy or transformation. The common semantics is supported by the behavioral properties of the roles as illustrated in Figure 4. This conceptual view shows the common semantics in the center (i.e the roles of the attacker viewpoint), and around it, accessed models via the behavioral properties of the roles.

Figure 4 represents the relationships between *InformationElementRole*, a *Role* of the attacker viewpoint (detailed in Section 5.2.1), and the different DSMLs. The different DSMLs are PimCa for the system model, a vulnerability DSML based on a subset of the NVD data structure, a DSML for the system configuration and an attacker competency DSML based on CVSS.

The *InformationElementRole* references three types of *Role*, *VulnerabilityRole* (to select all the vulnerabilities for a specific configuration exploitable by a relevant attacker), the *ConfigurationRole* (to define the configuration of the system entities) and the *CompRole* (to define the attacker competencies)

So this role model illustrates a semantic definition applied to a set of DSML concepts to provide a dedicated behavioral meaning.

In the rest of this section, we focus on the federation mechanisms used in Role4All to:
- Create a common and shared semantics.
- Use and dynamically update this semantics.

5.2.1 Definition of the Common and Shared Semantics

Regarding the Role4All metamodel, the relationship *containerRoles-containedRoles* on the Role class specifies that any *role* can reference a collection of *roles*, see Figure 1. This relationship between *roles* is not explicitly reified by a metaclass in the metamodel. So the role model does not contain relationship definition between *Roles* as we illustrate in Figure 5. This feature provides extreme flexibility because the role model states only the *Role* definitions individually and the role instantiation schema (Figure 6) provides relationships between *roles*. Two main advantages are underlined: first these relationships can be changed throughout the federation runtime to evolve and thus conform to the evolution of the attacker viewpoint of the system, in our example. Secondly, any *role* can be moved from a *containerRole* to another without any constraints except those of the domain.

The attacker role model contains two main *Roles* allowing the system elements to be interpreted: first, the elements that contain relevant information regarding the attacker goals (the attacker must control or corrupt these elements) and second, the elements that produce a service in the system (the attacker must use or bypass these elements). These *Roles* are *InformationElementRole* and *ServiceElementRole*. Three other *Roles* are defined to access subsets of DSMLs, presented in Figure 5. These five *Roles* are:
- *InformationElementRole*: This *Role* specifies a semantics for a subset of the PimCa DSML and contains three *roles* to access the other DSMLs for the configuration elements, the attacker competencies and vulnerabilities.
- *ServiceElementRole*: This *Role* provides another viewpoint of the PimCa DSML to view the topology elements which provide a service which can be used or bypassed by the attacker. These elements are necessary but they are not part of final the goal. Thus, the PimCa elements could be viewed as an *InformationElementRole* or *ServiceElementRole* according to the knowledge and perception of the attacker.
- *ConfigurationRole*: This *Role* mainly provides the application name and version on the topology elements of role type *InformationElementRole* or *ServiceElementRole*.
- *CompRole*: This *Role* specifies the attacker's competencies, such as to take into account the use of existing malware (via metasploit, for example) or the development a new malware.
- *VulnerabilityRole*: This *Role* specifies the possible vulnerability viewpoint relative to the current element configuration and the attacker's competencies. This *Role* selects the vulnerabilities by applying a filter (composed of configuration and attacker competencies) because typically the NVD data base is too huge and detailed. The viewpoint selects only the relevant vulnerability features to reach the targeted scenario.

With our role model, the instantiation process can create one instance of *InformationElementRole* as a *containerRole* and one instance of each *ConfigurationRole*, *CompRole* and *VulnerabilityRole* which are as *containedRoles*, see Figure 6.

Conforming to the Role4All metamodel, each *role* has behavioral methods to federate model data from the DSMLs and each model element is a *player*. For example, the method *getVulnerabilities* (shown in

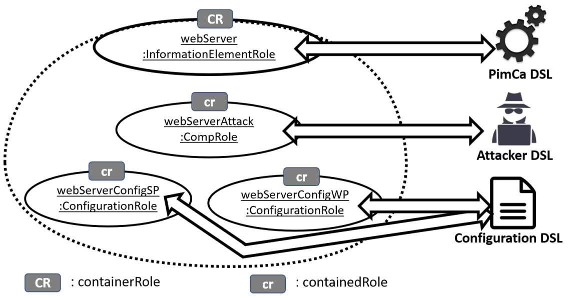

Figure 6: Instantiation schema of a container and its contained roles.

Figure 4) provides the behavior to select the vulnerabilities regarding the current configuration and the attacker competencies.

In the following parts we exemplify the usage of the behavioral methods of shared semantics.

5.2.2 Usage and Dynamic Update of the Shared Semantics

To focus on the use of the shared semantics, we exemplify the gathering of vulnerabilities on an exploitable element targeted by the attacker. Thanks to the behavioral method *getVulnerabilities* of *InformationElementRole*, we gather model data from the configuration, the attacker competencies and the vulnerability DSMLs. Finally, this method selects and returns all the exploitable vulnerabilities relative to the *InformationElementRole* instance.

In our example, Figure 7, we take an instance of *InformationElementRole* called *webServer*. *WebServer* corresponds to the federation of a PimCa machinery instance called "*pimcaWebServer*", a dedicated configuration defining WordPress version 4.6.1 and Symposium version 14.11, and attacker competencies with a network access and without authentication privilege like in Figure 4.

The instantiation schema, illustrated in Figure 6, provides a *webServer* as a *containerRole* and three instances: *webServerConfigWP*, *webServerConfigSP* and *webServerAttack* as *containedRoles*.

To accomplish a simulation step, the simulator must request the message *getVulnerabilities*. This simulator DSL is based on a dedicated set of roles and on an interactive approach. Without an in-depth presentation of this simulator, we consider here that this environment requests the federation and waits for a response from the role model with or without information from the DSMLs.

So the request of *getVulnerabilities* by the simulator, triggers the behavior of the federation in several steps, illustrated in Figure 7 and presented below:

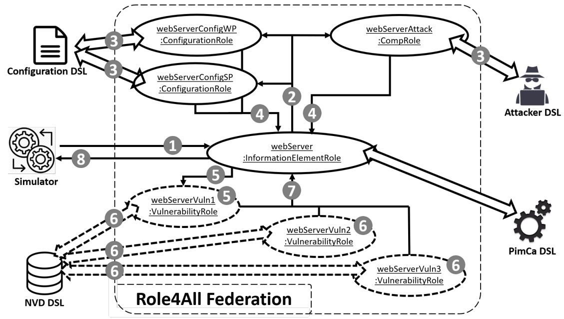

Figure 7: Federation mechanism details for information gathering and updating role model.

1. The simulator triggers the behavior method *getVulnerabilities* on the *containerRole*, *webServer*. This method requires arguments: a configuration and attacker's competencies. So *webServer* must request this information from its *containedRoles*.

2. The *containerRole* requests all the *containedRoles*, and especially *webServerConfigWP* and *webServerConfigSP* about the configuration of *webServer* and *webServerAttack* about the attacker's competencies.

3. *WebServerConfigWP*, *webServerConfigSP* and *webServerAttack* are linked to their player, i.e the model elements of the DSMLs. Conforming to the metamodel, the *PlayRelation* and *Adapter* instances collect the data from the *players*.

4. *WebServerConfigWP*, *webServerConfigSP* and *webServerAttack* send the requested information to *webServer*. Then the *containerRole* calls the method *getVulnerabilities* with arguments resulting of the collected information.

5. The method *getVulnerabilities* generates an instance of *VulnerabilityRole* contained by *webServer*, called *webServerVuln1*.

6. *WebServerVuln1* triggers the method *collectVulnerabilities*. This method requires the same arguments as *getVulnerabilities* and generates instances of *VulnerabilityRole*. This method selects all the existing vulnerabilities for the *webServer* configuration which are exploitable by the attacker. For each selected vulnerability, an instance of *VulnerabilityRole* is created with a sub-set of the features from NVD vulnerability database. In our example *webServer* has three selected vulnerabilities, so we generate three instances of *VulnerabilityRole* to update the role model with the vulnerabilities.

7. Each instance of *VulnerabilityRole* sends the vul-

nerability content to *webServer*.

8 Then the *containerRole* returns all the federated *VulnerabilityRole* instances to the simulator.

In our use case, the returned list of vulnerabilities contains the vulnerability CVE-2014-10021. This vulnerability allows remote attackers to execute an arbitrary code by uploading an executable file. Thanks to this vulnerability, the attacker obtains full access to the Web server. Thus the attacker has access to the target local network, and he can obtain the administrator account of the Active Directory using Trojan horse.

With this example we illustrate the use of the shared semantics and highlight the relation between *containerRole* and *containedRole* to ensure the information gathering and the update of the role model. The role model is composed of role classes and behavioral methods. These methods act on the DSML elements, the *players*, to get and set the element properties without any copy of these elements. The creation of *playRelation* at run time (Step 6 of Figure 7) allows the federation system to be adapted dynamically to the evolution of the *players*. As a future work we would like to try a web based implementation similar to WebDPF (Rabbi et al., 2016).

6 CONCLUSION

DSML interoperability remains tedious to obtain and it is traditionally handled by transformations with potential extensions to have bi-directional transformations. Our claim is that the model federation approach facilitates the DSML interoperability issue. In this paper, we demonstrate that role modeling provides the capacity to define a shared semantics between the considered DSMLs. The goal of our role modeling approaches is to act as a semantics viewpoint on the model elements.

In this approach, the model elements remain independent of the federated model and no transformations are applied. The role model is based on behavioral functions to obtain and set model elements without the creation of intermediate model elements. Our Role4All metamodel is based on a formalization of the role concept which provides a clear context of our work. The framework must be extended to take into account, for example, dedicated connectors to facilitate the interaction with classic data formats such as JSON.

The case study used to illustrate our approach is really relevant in the sense that cyber threat analysis requires several tools to improve this kind of critical analyses. The analysis needs to take into account many data and metadata on a system, correlate these data and process the resulting federated data. This example must be extended to increase the data sources but it remains a very interesting field of study for the federation approach.

ACKNOWLEDGEMENTS

This work is accomplished in the context of a PhD grant from the French Ministry of Armed Forces and the Brittany Regional Council.

REFERENCES

Alexander, I. (2003). Misuse cases: Use cases with hostile intent. *IEEE software*, 20(1):58–66.

Champeau, J., Leilde, V., and Diallo, P. I. (2013). Model federation in toolchains. In *MODELS Companion Proceedings*.

Conti, M., Dargahi, T., and Dehghantanha, A. (2018). *Cyber Threat Intelligence: Challenges and Opportunities*, pages 1–6. Springer International Publishing, Cham.

Elahi, G., Yu, E., and Zannone, N. (2010). A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements engineering*, 15(1):41–62.

Gottlob, G., Schrefl, M., and Röck, B. (1996). Extending object-oriented systems with roles. *ACM Transactions on Information Systems (TOIS)*, 14(3):268–296.

Guychard, C., Guerin, S., Koudri, A., Beugnard, A., and Dagnat, F. (2013). Conceptual interoperability through models federation. In *Semantic Information Federation Community Workshop*.

Hemery, D. (2015). PimCa: Définition du langage. Technical report, DGA Maitrise de l'Information.

Holik, F., Horalek, J., Marik, O., Neradova, S., and Zitta, S. (2014). Effective penetration testing with Metasploit framework and methodologies. In *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 237–242. IEEE.

Kühn, T., Böhme, S., Götz, S., and Aßmann, U. (2015). A combined formal model for relational context-dependent roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 113–124. ACM.

Kühn, T., Leuthäuser, M., Götz, S., Seidl, C., and Aßmann, U. (2014). A metamodel family for role-based modeling and programming languages. In *International Conference on Software Language Engineering*, pages 141–160. Springer.

Lee, W.-S., Grosh, D. L., Tillman, F. A., and Lie, C. H. (1985). Fault tree analysis, methods, and applications - a review. *IEEE transactions on reliability*, 34(3):194–203.

Li, Q. and Chen, Y.-L. (2009). Data flow diagram. In *Modeling and Analysis of Enterprise and Information Systems*, pages 85–97. Springer.

Mauw, S. and Oostdijk, M. (2005). Foundations of attack trees. In *Proceedings of the 8th international conference on Information Security and Cryptology*, pages 186–198. Springer-Verlag.

Mell, P., Scarfone, K., and Romanosky, S. (2006). Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6).

Myagmar, S., Lee, A. J., and Yurcik, W. (2005). Threat modeling as a basis for security requirements. In *Symposium on requirements engineering for information security (SREIS)*, volume 2005, pages 1–8. Citeseer.

Niemoller, J., Mokrushin, L., Vandikas, K., Avesand, S., and Angelin, L. (2013). Model federation and probabilistic analysis for advanced OSS and BSS. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2013 Seventh International Conference on*, pages 122–129. IEEE.

Pauli, J. and Xu, D. (2005). Threat-driven architectural design of secure information systems. In *Proceeding of First International Workshop on Protection by Adaptation PBA 2005, Miami*.

Rabbi, F., Lamo, Y., Yu, I. C., and Kristensen, L. M. (2016). Webdpf: A web-based metamodelling and model transformation environment. In *Model-Driven Engineering and Software Development (MODELSWARD), 2016 4th International Conference on*, pages 87–98. IEEE.

Rio, M. (2012). *A l'interface de l'ingénierie et de l'analyse environnementale, fédération pour une éco-conception proactive*. PhD thesis, Université de Technologie de Troyes.

Schneider, F. B. (1999). *Trust in cyberspace*. National Academy Press Washington, DC.

Schneider, J.-P., Champeau, J., Teodorov, C., Senn, E., and Lagadec, L. (2015). A role language to interpret multi-formalism system of systems models. In *9th Annual IEEE International Systems Conference (SysCon)*, pages 200–205. IEEE.

Seifert, M., Wende, C., and Aßmann, U. (2010). Anticipating unanticipated tool interoperability using role models. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, pages 52–60. ACM.

Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.

Steimann, F. (2000). On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106.

Wende, C., Thieme, N., and Zschaler, S. (2009). A role-based approach towards modular language engineering. In *International Conference on Software Language Engineering*, pages 254–273. Springer.

Zhang, S., Caragea, D., and Ou, X. (2011). An empirical study on using the national vulnerability database to predict software vulnerabilities. In *International Conference on Database and Expert Systems Applications*, pages 217–231. Springer.