Using Security Patterns for Modelling Security Capabilities in Grid Systems

Benjamin Aziz
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

Clive Blackwell
Department of Computing and Communication Technologies
Oxford Brookes University
Oxford, United Kingdom
cblackwell@brookes.ac.uk

***Abstract*—We extend previous work on formalising design patterns to start the development of security patterns for Grid systems. We demonstrate the feasibility of our approach with a case study involving a deployed security architecture in a Grid Operating System called XtreemOS. A number of Grid security management capabilities that aid the secure setting-up and running of a Grid are presented. We outline the functionality needed for such cases in a general form, which could be utilised when considering the development of similar large-scale systems in the future. We also specifically describe the use of authentication patterns that model the extension of trust from a secure core, and indicate how these patterns can be composed, specialised and instantiated.**

***Keywords*-Security patterns; authentication patterns; security architectures; Grid operating systems**

I. Introduction

This paper demonstrates how security design patterns, in particular those for authentication, can be used to express Grid security management capabilities within the context of a large-scale Grid operating system called XtreemOS [1], [2]. These capabilities are used by administrators, users and core services to establish and operate a Grid infrastructure.

XtreemOS provided a single abstraction layer of physical hardware and software services offered by a collection of standalone Linux operating systems to users within a Grid. These operating systems could function collaboratively to support the utilisation of computational and storage resources regardless of the geographical location of their users or machines. A major function of XtreemOS was to hide the complexity of distributed resources dynamically aggregated from large-scale cross-domain resource providers and to ensure the transparency of using such a distributed operating system. Hence, similar to a standalone operating system, once a user is registered with XtreemOS, it should be conceptually the same to utilise resources from any machine that the system is composed of, regardless of whether such resources have been recently added to the system or have been there before.

The main contribution of this paper is to propose the security capabilities in XtreemOS as a set of security patterns to aid Virtual Organisation (VO) management in future Grid and other large-scale distributed systems, such as Clouds. This aim is motivated by the sheer scale of the XtreemOS system, and the large industrial platform (more than 14 industrial applications [3]) on which its use cases were based. All of the security activities can be modelled as security patterns, but we focus on authentication patterns in this paper as a proof of concept. The main motivating factor behind our approach is to facilitate the use of such patterns in future systems based on the model provided by XtreemOS.

II. The Security and VO Management Model

In this section, we provide an overview of some of the security and VO management elements needed when designing a large-scale Grid operating system, such as XtreemOS. These elements can be described in terms of the *trust domains*, *actors* and other *core services* needed for such functionality [4].

A. Trust Domains

The definition of security and VO management services are based on three main trust domains:

- The *Resource site* domain: These are sites that offer resources to the Grid and any VOs formed within.
- The *User site* domain: This includes sites that provide users of VOs who will submit jobs to the resources included in those VOs.
- The *Core site* domain: This represents the core site in which the critical Security and VO Management services would be running.

From the trust point of view, the Core site represents the root of trust for both the Resource and User sites. In other words, a Core site must have a high level of assurance since it will be running the critical system components.

B. Actors

The main actors of the security and VO management services include:

- The *User*: this actor is the user of VOs, who is also registered in the Grid within which the VOs are created.
- The *VO Administrator*: this actor is a previous User who created a VO and became the owner and administrator of that VO. Therefore, the VO Administrator has full authority on managing the VO.

- The *Resource Administrator*: this is the actor owning the resources offered to VOs. The actor could be either a whole site administrator or the owner of one or more machines belonging to the site.
- The *Grid Administrator*: this actor is responsible for managing the security and VO management services.

C. Core Security and VO Management Services

The core services [4], shown in Figure 1, consist of the VO Management Service (VOMS), the Resource Certification Authority (RCA) and the VO Policy Service (VOPS). Additionally, monitoring and auditing services can also be deployed across the Core and resource sites to raise the level of assurance as to the general behaviour of the system and its processes.

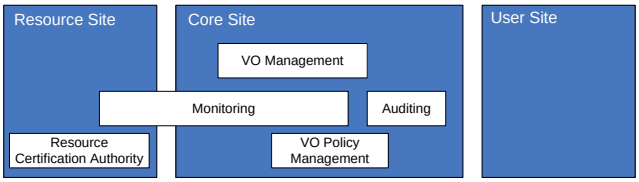

Figure 1. The security and VO management services [4].

1) The VO Management Service (VOMS): This is a VO and trust management service, which provides a logical grouping of the infrastructural services needed to manage the entities involved in a VO and ensure a consistent and coherent exploitation of the resources and capabilities inside the VO. The VO Management system consists of the following components: The *Root Certification Authority*, which is a service that creates the trust anchor, the *Root Certificate*, and uses it to certify the identity of core services within the operating system. This service can be performed offline to avoid compromise of the root private key. The certification of core services can optionally be performed by the CDA service, described next.

The *Credential Distribution Authority (CDA) Service*, is responsible for distributing the identity digital certificates to users. These certificates can be X509-based [5] and the CDA server may optionally be configured to provide service certificates, certifying the identity of the core services in the system. The *Credential Distribution Authority Client*, is a client-side program or a Web client that interfaces with the CDA service.

The *Registration Manager* is responsible for managing the initial registration of users and RCAs with the operating system. The *The VO Manager* controls the functionality associated with the different stages in the lifecycle of the VO (e.g. VO creation, operation, evolution and termination stages). There is one instance of a VO Manager component running per live VO, and its functions are controlled by the user who is the VO's Administrator. Finally, the *VO Management Database* is the main database in VOMS, in which all the information regarding the user and RCA registrations, VO membership and lifecycle is stored.

2) The Resource Certification Authority (RCA): The RCA is a certification authority at the level of administrative sites offering resources to VOs. The RCA consists of the following components. The *RCA server* is the main component responsible for issuing certificates to resources. This component is responsible for bootstrapping trust in the individual resource domains. This trust could be used by other components of the operating system, such as those for the management of job submissions. The *RCA client* is the client-side program that can interact with the RCA server. This could be optionally replaced by a Web interface. Finally, the *RCA database* stores the state of resources in each administrative domain. This state may indicate whether a resource is registered or not with the Grid and if so, whether it is currently offered to any VOs in that Grid.

3) VO Policy Service (VOPS): The VOPS is used to manage and enforce VO policies. The service consists of the typical components that a distributed policy enforcement architecture normally contain [6]; for example, a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), a Policy Information Point (PIP), a Policy Administration Point (PAP) and a Policy Store (PS).

4) The Monitoring and Auditing Service: This is responsible for receiving status data, changes and events from the resources where user jobs are running, and therefore provides feedback on user behaviour and resource performance to interested parties in the systems (e.g. the VOPS), as well as store metrics and events about this behaviour in a log database. The monitored information can span a wide spectrum of data, which can be used for assessing the security status of the operating system and analysing past behaviour of both the system and the actors. Another important aspect of the security auditing is the ability to record specific actions, providing irrefutable proof of accountability for harmful behaviour.

III. Grid Security Management Capabilities

One of the first stages in a VO lifecycle is the Grid infrastructure set-up and population, which will be the focal point of this paper. This stage is based on and requires the application of the *Grid security management* capabilities, before any VOs are created. Here, we outline the main Grid security management capabilities that we defined in XtreemOS [4]. These capabilities included the registration and removal of users and RCAs with the Grid, the registration of local resources with RCAs, the setting-up of Root Certificate Authorities (CAs) and the running of the various security and VO management core services within the system. These capabilities are depicted in Figure 2.

Generally, one can think of these capabilities as the actual setting-up of a new Grid infrastructure, which is the first step

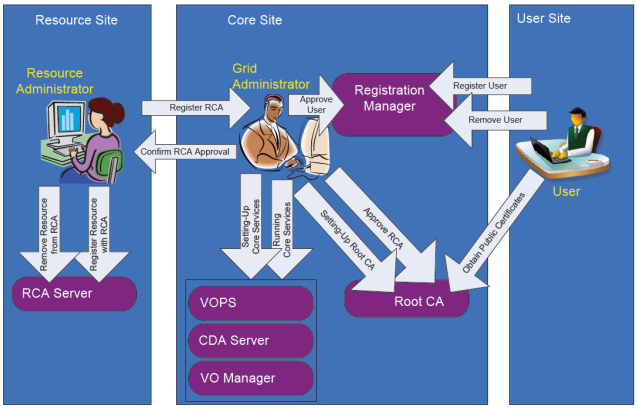

Figure 2. Grid Security Management Capabilities in XtreemOS [4].

in a VO lifecycle. We next describe some of the main Grid security management capabilities.

1) Configuring and Creating the Root CA: This capability is concerned with the creation and configuration of the root of trust in a Grid, i.e. the Root CA. The capability is performed by the Grid Administrator, who is responsible for generating the public/private key pair for the Root CA with the private key later used for signing any Certificate Signing Requests (CSRs) from other services. In the ideal case, the machine (node) running the Root CA must be of high assurance and not networked, to minimise security vulnerabilities. The public key certificate is placed on a networked core machine ready and available for public distribution when needed.

2) Creating the VOMS Database: The second capability is related to the creation of the VOMS database, which will later hold all the information related to the Grid VOs as well as the Grid membership. Again, this capability is performed by the Grid Administrator, who initialises and sets the database ready for use. This also includes setting-up a password for the database, which could be the same password assigned to the Grid Administrator (root login).

3) Setting-Up and Running Core Services: Once the Root CA and the VOMS database components have been set-up and initialised, it is now possible to set-up and run the Core Services in a Grid. The actors responsible for this capability are the Grid Administrator as well as the Resource Administrator for any organisations willing to join the Grid and provide resources. Once this capability has been executed, the Core Services of the operating system will be up and running ready for users. This also implies that their security credentials have been created. The Root CA certificate and the CDA (Credential Distribution Authority) certificate are placed on a networked node ready for distribution.

4) Obtaining Public Certificates: This capability is initiated by a User who wishes to obtain the public key certificate(s) of one or more of the Core Services from the Root CA. As a precondition, the User is expected to have the public key certificate of the trusted certification authorities installed on his/her system providing a way for checking the trust and security of registered services using the chain of keys starting with the Root CA. Once the capability is successfully executed, the User will have obtained the public key certificates of the requested Core Services in the Grid operating system.

5) Processing Certificate Requests: This capability is performed by the Grid Administrator in order to generate certificates verifying the identity of a Core Service when requested by a User as per the previous capability. A CSR (Certificate Signing Request) is converted into a public key certificate, which is sent to the originator of the request.

6) Registering Users: This capability is relevant to populating the Grid by allowing Users to register with the Grid infrastructure. This implies that a User can request a Grid account upon providing their details (e.g. user name, password, real name, organisation and email address). The capability is executed on the Registration Manager (part of the VOMS) by the User. A precondition is that the VOMS database must already have been set-up and configured ready for receiving information on Grid membership. Once the capability has been successfully executed, the User will have an account on the database and he will end-up sharing a password with the Registration Manager. The success of this capability is dependent on the completion of next capability of approving Users.

7) Approving Users: This capability is applied by the Grid Administrator, who will react upon receiving a request to join the Grid from some User as per the previous capability. The Grid Administrator will make sure that the request itself is valid (e.g. the organisation to which the User belongs is admissible to the Grid). It also implies that the Grid Administrator can contact the User by email or telephone to request further details before approving their request. If the Grid Administrator is satisfied with the request and the associated information about the User, they will then approve the request leading to a successful completion of this capability as well as the previous one.

8) Removing Users: The Grid Administrator can also remove an already-registered User from the Grid. As a result, the User will no longer be capable of logging-in to the Grid, joining VOs or submitting any computational jobs. Another form of this capability is that the User himself decides to leave the Grid by either sending a request to the Grid Administrator or by executing the relevant commands on the VO Manager.

9) Registering an RCA: This capability is related to the population of the Grid with RCAs representing resource administrative domains. The capability allows a Resource Administrator to request to join a RCA to some Grid infrastructure from the Grid Administrator. When successfully completed, the capability will allow the Resource Administrator and the Registration Manager to share a password for

managing the RCA's account in the VOMS. However, its success is dependant on the success of the next capability.

10) Approving an RCA: This capability will allow the Grid Administrator to approve a request submitted via the previous capability for joining a RCA to the Grid. Once this is achieved, the RCA will obtain an account on the VOMS for its membership in the Grid and future VOs. Similarly, the Root CA is informed of the decision in order to link the RCA to its chain of trust. The Grid Administrator will then inform the Resource Administrator of the decision of joining the RCA to the Grid. The Resource Administrator can now start applying the following two capabilities related to offering resources to the Grid.

11) Registering Resources with the RCA: Since the RCA is the main gateway for local resources (machines, nodes, services) to join the Grid, this capability is essential in that it allows the Resource Administrator to register the local resources with the RCA. As a result, the resource details will be recorded on the RCA and the resource will be issued with an identity certificate that it can use to identify itself to other Grid resources or users.

12) Removing Resources from the RCA: Finally, the last capability for Grid Management is related to the removal of any resources wishing to exit the Grid. As a precondition, it must be the case that the resource was already registered through the previous capability, and that there are no pending jobs in the Grid currently running on the resource. Once the resource has been removed from the RCA list of Grid membership, any subsequent requests for its identity or attribute certificate will fail and the resource is now outside the Grid.

IV. Grid Security Patterns

A. Design Spaces

GEBNF (Graphic Extension of BNF) notation can be used to define a metamodel for a design space [7]. BNF (Backus Normal Form or Backus-Naur Form) [8] is a well-known notation for context-free grammars, often used to describe the syntax of programming languages and document formats amongst other things. The patterns in a design space can be specified formally [7] in a language derived from the GEBNF syntax using predicate logic. They can be defined as compositions and instantiations of existing patterns by applying pattern operators [9] and then the algebraic laws proved for object-oriented design patterns [10] should also hold for patterns in these new design spaces.

We represent a design space in the following form containing the modelled elements along with different perspectives such as structural or behavioural views.

```
DESIGN SPACE <Name>;
<Element type definitions>;
<View definitions>;
END <Name>.
```

We elaborate the approach of a security design space and pattern specification [11] to authentication patterns in realistic scenarios taken from the Grid Operating System. We can model system security architecture in box diagrams [12] and then give their corresponding GEBNF formulae. A box diagram consists of a number of boxes and arrows, where each box represents a system entity or sub-system, and each arrow represents a channel of information flow or interaction. These are both element types with names and other variables for their attributes.

A view defines a set of properties for the element types and relationships between them together with some constraints that limit the valid models. Each property takes some set of defined values defined extensionally, and the relationships define associations between elements. Trust is a crucial property that is extended to new entities using digital signatures created by trusted authorities, and it is important to distinguish between strong authentication mechanisms like signatures and weak ones like passwords. We only show the structural view of the security design space by extending the previous design space [11], but the dynamic view of system behaviour modelled by sequence or activity diagrams can also be defined in GEBNF.

```
DESIGN SPACE SecurityArchitecture;
TYPE
Subsystem, Pattern:
name: STRING,
content: [Value],
description: [STRING];
InfoFlow:
name: STRING,
from, to: Subsystem,
type: [STRING];
VIEW Structure;
PROPERTY
type : Subsystem -> {data_store, computation};
trust: Subsystem -> {trustworthy, untrustworthy};
strength: Pattern -> {weak, strong}
RELATION
is-a-part-of: Subsystem * Subsystem;
inherits_from: Pattern * Pattern;
instance_of: Subsystem * Pattern;
END structure;
VIEW behaviour
...
END behaviour;
END SecurityArchitecture.
```

B. Authentication Patterns

Section III showed the main aspects of the security architecture that bootstraps off the trustworthy core systems to authenticate and authorise domains, users, machines and resources. Users are authenticated by passwords, whereas machines and their resources are authenticated by digital signatures. Trust is moved around the system using various authentication measures to enable the secure use of resources. Authentication can be modelled by authentication patterns with subpatterns for various types of authentication mechanism such as passwords and signatures. In addition, these patterns can be instantiated differently to meet the differing requirements of users, domains and resources. We

first indicate the required properties of the primitive security patterns that constitute the digital signature pattern.

```
% Fundamental property of signature and
verification key pairs
PATTERN Sign in SecurityArchitecture
EQUALS
[signkey = inv(verifykey)]
END Sign.

% Fundamental  property of hash functions
PATTERN Hash in SecurityArchitecture
EQUALS
[Hash.Data = Hash.OtherData ==> Data = OtherData]
END Hash.
```

We define a specification for digital signatures using GEBNF with the signed message created from applying a hash to the message then signing the hash using the above two primitive components. The message is concatenated with the signature and sent to the recipient who performs the reverse operations of hashing and verification. The received message is judged valid if the computed and received hashes are identical.

```
% Digital signature pattern involves elementary Hash
% and Sign patterns. Trustworthiness of the message is
% established by the receiver when the data is verified
% Trustworthiness is passed between entities by the
% Validity InfoFlow. Data-stores can easily be distinguished
% from computation. Receiver components are prefixed
% by R to distinguish them from sender components
% Some elements are missing for space reasons
PATTERN DigitalSignature in SecurityArchitecture
COMPONENT
Sender, Receiver, Message, Signature, SignedMessage,
Hash, Sign, ReceivedMessage, RSignature, RMessage,
RHash, Verify, Comparison: Subsystem;

Data, H, S, TransmittedMessage, RData, RS, VHash,
CHash, Validity, InputMessage, OutputMessage: Infoflow;

CONSTRAINT
Message, Signature is-a-part-of SignedMessage;
SignedMessage, Hash, Sign is-a-part-of Sender;
RMessage, RSignature is-a-part-of ReceivedMessage;
ReceivedMessage, Comparison is-a-part-of Receiver;
RHash, Verify is-a-part-of Comparison;
EQUALS
[Verify.RSignature = RHash.RMessage]
% ==> Message = RMessage as required
END
INFOFLOW
% InfoFlows  are omitted for space.
% Each InfoFlow has a name and a source and
% destination with from as its starting box and
% to as its terminating box. For example:
Data.from = Message; Data.to = Hash
END DigitalSignature.
```

The Grid OS uses passwords to authenticate people by something they know, as they are the easiest method of authentication, but are weak as they have little or no protection in transmission. The password is exchanged over a communication medium like the signed message, but the same password is always exchanged rather than a varying signature that depends upon the message.

There is an inheritance hierarchy for patterns with passwords and signatures inheriting from a base authentication pattern. Passwords can be sent in the clear, which is the basic authentication pattern. There are various password subpatterns for when the password is protected by a hash or key. A common password pattern is one that hashes the password, as used for user authentication in operating systems like Windows and UNIX.

The password is sent in the clear or hashed before transmission. The hash computation is performed by the sender if the password is hashed before transmission, and the receiver simply compares the received and stored password hashes. The computation of the hash is on the receiver's side if the password is transmitted in the clear, and then the computed and stored password hashes are compared. This is weak because the transmitted password or hash can be captured and reused in both cases.

We can also use signatures to sign keys in digital certificates, where a certificate is simply a special type of signature on contents containing an entity's verification key along with its name and other metadata. This is simply a specialisation of the basic signature pattern into a certificate subpattern with more structured contents. It also allows the entity being verified to sign other certificates as well as data, thereby permitting a chain of certificates starting at the Root CA.

The two differences between a signature pattern and the inherited certificate pattern are the decomposition of the message contents into its components, including the name, key and privileges of the certified entity, and additional outputs from the pattern for these data. The outputs can link to the inputs of another certificate pattern if the privileges indicate that the verified entity is now a certification authority, or to a signature pattern for verified entities to sign ordinary messages.

```
% Other Subsystems and InfoFlows are inherited
% from the basic signature  pattern
PATTERN DigitalCertificate in SecurityArchitecture
DigitalCertificate inherits_from DigitalSignature:
VerificationKey, Name, Privileges, MetaData: Subsystem
Rights, InputRights, EntityKey, InputEntityKey: Infoflow;
CONSTRAINT
VerificationKey is-a-part-of Message;
Name is-a-part-of Message;
Privileges is-a-part-of Message;
Metadata is-a-part-of Message;
INFOFLOW
Rights.from = Privileges;
EntityKey.from = VerificationKey;
Identity.from = Name;
% Output flows have inputs to another signature
END DigitalCertificate.
```

The CDA certificate may be used to authenticate services in the Grid operating system instead of the Root Authority. The root certificate authenticates the CDA certificate to sign services on its behalf for the practical reason that wider use of the root key leaves it more open to compromise, and its exposure would cause security issues for the entire system rather than just the services.

The bootstrapping of trust via a chain of certificates is modelled as a composition of patterns where trust is passed between the Root CA and CDA server and then onto the users and resources. This is a composition of the specialised

certificate pattern and the signature pattern, which is permissible as they are compatible as basic signature patterns.

The password authentication of users to the Grid then leads to the creation and distribution of their verification keys in certificates allowing recipients of their messages to verify their authenticity. However, the weak level of trust from passwords cannot be elevated by using a stronger signatures mechanism afterwards, as the password may have been compromised initially and the signing key may be in the possession of the wrong party.

The composition of signature and password patterns can show the extension of trust transitively to other parties because they are compatible as basic authentication patterns. However, the level of trustworthiness of the composed pattern cannot exceed the weakest level of authentication. This will be shown on the composed pattern that will expose the same weaknesses as the password pattern, and the rule is that any composed pattern will have the weakest trustworthy level of all its elementary patterns. Finally, the authentication of users and services are instances of the same pattern with different implementations for their specific characteristics.

V. Conclusion and Future Work

We presented in this paper a general set of security and VO management capabilities that could underlie a Grid operating system and implement the VO lifecycle that the system could offer to Grid users. We extended the previous work on formalising design patterns to initiate the development of an algebra of security patterns that can be specialised, instantiated and composed. The feasibility of the proposed approach was demonstrated with a case study involving an existing security architecture in the XtreemOS Grid Operating System. Some of the security capabilities that aid the secure set up, creation, operation, evolution and termination of the Grid operating system could be modelled in a general form as security patterns suitable for helping to develop similar large-scale systems in the future, such as Clouds. We described the use of authentication patterns to model the building of trust in the Grid from a trustworthy core as a proof of concept.

Further work will develop other security patterns, such as confidentiality and access control patterns, and show how they can be composed together to model the entire security architecture. For example, we can model how the VO Policy Service is used to manage and enforce access control policies with patterns. The service consists of the typical components that a distributed policy enforcement architecture usually contain [6]. The access control pattern may compose with an authentication pattern to model the capabilities that authenticated entities are allowed to access.

We can also model various attacks on security controls abstractly by determining if their attack patterns [13] can compose with the security patterns to exploit any weaknesses. We mentioned that the password pattern is vulnerable to certain attacks that are defeated by the signature pattern. Various attack patterns can compose with the password patterns to obtain or otherwise compromise a password transmitted in the clear or as a hash, whereas composition with the digital signature pattern will be unsuccessful as the secret is never exchanged. The composition of security and attack patterns is under active consideration.

References

[1] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld, "XtreemOS: a Vision for a Grid Operating System," XtreemOS Technical Report # 4, May 2008.

[2] C. Morin, "XtreemOS: A Grid Operating System Making your Computer Ready for Participating in Virtual Organizations," in *Proceedings of the Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007)*. IEEE Computer Society, 2007, pp. 393–402.

[3] XtreemOS Consortium, "Requirements capture and use case scenarios," in *XtreemOS public deliverables - D4.2.1*. Work Package 4.2, January 2007.

[4] XtreemOS Consortium, "Fourth Specification, Design and Architecture of the Security and VO Management Services," in *XtreemOS public deliverables - D3.5.13*. Work Package 3.5, December 2009.

[5] R. Housley, W. Polk, W. Ford, and D. Solo, "RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," April 2002.

[6] T. Moses (Ed.), "eXtensible Access Control Markup Language (XACML) Version 2.0," OASIS Standard, 2005.

[7] H. Zhu, "An institution theory of formal meta-modelling in graphically extended BNF," *Frontiers of Computer Science*, vol. 6, no. 1, pp. 40–56, 2012.

[8] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. L. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger, and P. Naur, "Revised report on the algorithmic language algol 60," *Communications of the ACM*, vol. 6(1), pp. 1–17, Jan. 1963.

[9] I. Bayley and H. Zhu, "A formal language for the expression of pattern compositions," *International Journal on Advances in Software*, vol. 4, no. 3&4, pp. 354–366, 2011.

[10] H. Zhu and I. Bayley, "An algebra of design patterns," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 3, Jul. 2013.

[11] H. Zhu, "Design space-based pattern representation," in *1st CyberPatterns: Unifying Design Patterns with Security Patterns and Attack Patterns*, C. Blackwell and H. Zhu, Eds. Springer, 2014.

[12] J. Sherwood, A. Clark, and D. Lynas, *Enterprise Security Architecture: A Business-Driven Approach*. CMP Books, 2005.

[13] Mitre Corporation, "Common Attack Pattern Enumeration and Classification (CAPEC)," 2013. http://capec.mitre.org.