Towards Computer-oriented Security Patterns

ANTONIO MAÑA, UNIVERSITY OF MÁLAGA
EDUARDO B. FERNANDEZ, FLORIDA ATLANTIC UNIVERSITY
JOSE FRAN. RUIZ, FRAUNHOFER SIT
CARSTEN RUDOLPH, FRAUNHOFER SIT

Security patterns are a mature paradigm for the development of secure systems. For years, security patterns have proven their usefulness, and have demonstrated that their use, especially at the software architecture level, provides important advantages. Security patterns have also proven their value as a communication vehicle and as an educational tool. However, the ever-growing complexity and heterogeneity of software systems, which now range from tiny pieces of extremely delicate software for the control of embedded devices to internet-scale applications that are used by millions of users, are slowly but inevitably degrading the usefulness and applicability of security patterns in their current text-based form. Moreover, as new applications and technologies appear, the amount of new security solutions grows and with them the number and variety of security patterns representing them. The need to support the selection of the most appropriate pattern based on different parameters and criteria is not well-served by document-based patterns, which are more appropriate for educational purposes than for supporting current model-based and computer-assisted engineering activities. In fact, the engineering and development of software systems cannot be conceived today without a high level of automated support. Finally, security patterns in their current form provide limited support for their integration in the systems under development, resulting in different problems that may even reduce or ruin the intended security property that was meant to be provided by the pattern. The previous reasoning takes us to the core of the content of this paper: the need for a computer-oriented form of patterns, along with mechanisms for automating their use.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures—*Patterns*; **D.3.3 [Language Constructs and Features]**: *Patterns*; **K.6 [Management of Computing and Information Systems]**: Security and Protection

General Terms: Security Patterns, Security modeling and engineering, Pattern-based system engineering

Additional Key Words and Phrases: Computer-processable knowledge representation

1. INTRODUCTION

Security patterns appeared about 20 years ago but they are still not widely adopted; this is paradoxical, considering the current concern about building secure systems. After a careful analysis of the situation, we conclude that there are several reasons for this situation. One of them is that there are different definitions of the concept and realization of security patterns. Even in papers that use a similar definition, the levels of description are quite different [1]. Patterns in general are dismissed by some researchers as not being "scientific enough" and in our opinion this is not so much because of limitations in their content (mostly document-based and human oriented) or the process to define them (nothing is more scientific than experimentation and patterns provide a means to communicate results of experience) but because of the lack of a well-defined and rigorous process for developing systems based on them. Probably, one of the most significant obstacles for their use is the difficulty for system designers, which usually have a limited expertise on security, to select the right pattern to apply in their system. One of the reasons for this is the lack of a good catalog [2], not only in terms of quality of contents and size but also in terms of usability (accessibility, search capability, precision, organization, freshness, etc.); precisely, our proposal is aimed at advancing in these directions. But a catalog is not enough. An additional reason for the limited practical application of security patterns is the fact that the information contained in current pattern descriptions is often not enough to allow designers to integrate the pattern in the system under development (SUD) with enough guarantees of a sound

The work of A. Maña is supported by the SecFutur, CUMULUS, PARIS and ICES projects.
Author's addresses:
Antonio Maña, University of Malaga. Boulevar Louis Pasteur, 35, 29071 Málaga, Spain; email: amg@lcc.uma.es.
Eduardo B. Fernandez, Florida Atlantic University. 777 Glades Road, Boca Raton, FL 33431, USA; email: ed@cse.fau.edu
Jose Fran. Ruiz, Fraunhofer SIT, Rheinstraße 75, 64295 Darmstadt, Germany; email: jose.ruiz.rodriguez@sit.fraunhofer.de
Carsten Rudolph, Fraunhofer SIT, Rheinstraße 75, 64295 Darmstadt, Germany; email: carsten.rudolph@sit.fraunhofer.de

application of the pattern. Some early proposals for automated processing of patterns were rather limited and never achieved a usable level [3].

Although, as indicated, security patterns have not been used as much as one would expect, some interesting applications exist, e.g. qmail [4] and BBVA [5]. Reference [6] reports the use of security patterns to enforce security for remote healthcare in smart homes. Reference [7] describes a survey of 237 institutions in China finding out their use of security requirements for software construction. Their most common sources of security knowledge were standards and security patterns (22%); they also used information about attacks (72%). All this shows that when developers are aware of them and able to understand their advantages, they are willing to adopt them.

One of the main problems for the adoption of security patterns is the difficulty for designers to find the appropriate pattern to add at a particular development stage. To simplify the use of patterns by designers we can start by defining extended patterns that include more information about their use, as done in these works:

- Enterprise Security Patterns [5]. An enterprise security pattern (ESP) combines a wide range of items describing generic enterprise security architectures that protect a set of information assets in a specific context. They are a more comprehensive pattern that can handle more threats, in order to facilitate for the designer the selection and tailoring of security policies, patterns, mechanisms and technologies when she is building enterprise security architectures.
- Secure Semantic Analysis Patterns (SAPs). In this approach a SAP is made secure by adding security patterns after analyzing its use cases and its possible threats. A SAP is a pattern combining a set of basic use cases [8]. For example, we produced a set of secure functions for law firms [8].
- Tags [9]. Tags included the security objective, the pattern applicability, tradeoff labels (impact on other concerns) and relationships between patterns.

However, adding more information is not enough if that information does not facilitate the selection and application of the pattern. Some works in the line of facilitating the selection on the basis of extensions to the pattern description have shown the potential of this line of improvement [10]. We have already mentioned that another important limitation for the use of security patterns comes from the fact that the contents of the pattern sections are currently text based or document (graphic) based, but not automation-oriented. In fact, when present, models (mainly UML, but also any of its specialized versions like BPML, SysML or MARTE) are normally included only as figures, and are not available for direct use by system engineers. Moreover, these models only contain the basic elements to build the solution, but developers often find difficult to actually integrate the models included in the current patterns with their systems due to the lack of integration-specific details. System models can be much more complete than a set of diagrams, especially when we consider automation to use the patterns during the system engineering and design phase as we show later in the paper.

Today, systems are created using modeling and development tools and, for this reason, we consider essential that security patterns evolve from documentation artefacts to design and development artefacts, not only providing support for humans (developers) but also becoming machine-processable. This is the key idea of our proposal: to propose a new computer-processable format for patterns that facilitate the use of these. Our proposed approach is an incremental one, starting with enhancing the usability, and the search and browsing capabilities, and later addressing more ambitious goals such as supporting selection and integration as we will show in the paper. Some early versions of proof of concept tools have been developed to help demonstrate the potential of the proposed approach.

From the previous analysis we come to the conclusion that if we want security patterns to achieve a wider adoption, the time for redefining them has come. Consequently, in this paper we advocate such an evolution and present an initial proposal to be discussed with the community in order to create the new security patterns and to foster their adoption.

Throughout the paper we will use the "Credential" pattern [11] in order to illustrate the proposed approach and to present clarifying examples of the concepts introduced. The rest of the paper is structured as follows: We define in Section 2 our goals for the proposed evolution. Section 3 presents our new view of security patterns and discusses some details of their structure and the level of representational detail that we need to write useful patterns. In Section 4 we present a new view of system engineering based on the new security patterns. We end with some conclusions in Section 5.

2. GOALS FOR A NEW CONCEPT OF SECURITY PATTERNS

Analyzing the current situation with security patterns, we have concluded that a new form of security patterns is necessary to solve the problems discussed in Section 1 and to foster adoption by developers. We introduce now our vision of an evolution of security patterns towards machine-processability and automation support. We consider that in order to achieve our vision, the new security patterns should be:

- Machine-processable. Security patterns should be designed to be treated by computer-assisted means for their specification, publishing, browsing/searching and cataloguing, but also to be used and integrated in modeling and development environments. An important short-term advantage of this is that it allows the creation of an online distributed catalog of patterns with advanced searching and browsing capabilities, as well as feature-based navigation. In addition to this, the management of patterns as complex objects with advanced presentation capabilities (such as for instance condensing or expanding sections, using different views, etc.) will represent an advance that is easy to achieve but very valuable in practice.
- Engineering-oriented. In addition to current documentation-oriented or education-oriented security patterns, the new version should enable a tighter integration in engineering activities and processes, facilitating the tasks of adopting a security pattern in a SUD. Therefore, as a necessary complement to the evolved security patterns, we need to enhance existing processes and methodologies that use them and maybe find new ones. Section 4 below sketches a proposal for such a process.
- Trusted. Trust management mechanisms should be a core element in the new patterns so that developers can use security patterns from sources they trust, with guarantees of authenticity, integrity and with reliable assessment of their quality and reputation. Therefore, we propose the inclusion of mechanisms guaranteeing the authorship and integrity of the pattern. This is related to the concept of provenance in the sense that it covers origin and authorship, but the aspects of code reuse, adaption and extension, also inherent to provenance, are not really applicable here. Of course, relations between patterns must be considered as will be described in section 3.1. Additionally, we believe it would be useful to have mechanisms for associating some sort of "reputation" (i.e. community-based evaluation) to the patterns, but this should be done externally to the pattern content.
- Detailed and precise. Security patterns must allow security experts to capture a rich set of characteristics and details with enough rigor and precision to suit the needs of different users and scenarios but without sacrificing their generality, which is necessary in order to ensure their reusability. In order to achieve this, both syntax and semantics of security patterns need to be redefined to ensure that patterns produced by different sources are comparable and understandable. We believe that ontological support can be the basis to achieve this goal. In principle, ontological support can help improving selection capabilities, and interoperability of patterns from different sources. The approach we propose to follow is based on previous results on secure decentralized ontologies [12]. In essence, the idea is to allow the inclusion of references to external ontologies in different fields of the patters. As a simple example, the security properties provided by a pattern could be related using external secure ontologies. Assume the "Authentication" property provided by the "Credential" pattern has the following relations to other properties:

  "Authentication" equals "Client Authentication"
  "Mutual Authentication" implies "Client Authentication"

  In this way, a search for a pattern providing the "Authentication" property would return all patterns providing any of the "Authentication", "Client Authentication" or "Mutual Authentication" properties, as all of them could fit the original goal of the developer. Of course, pattern search mechanisms should provide flexible interfaces allowing system engineers to decide if ontology-supported equivalence is used or not in a given query. Another example is using ontologies to relate threats to actual attacks. Finally, ontologies become an essential tool in order to ensure that patterns provided by different entities can be understood and compared. As an example of this, consider the definition of the Trust element of the new patterns related to the discussion in previous point. In the definition proposed in section 3.1 (element #11) we define trust mechanisms with two components: type and value. Here, ontologies can be used to define the possible "types" (e.g. "rsa-sha1", "dsa-sha2",etc.) and their relations.
- Open and decentralized. It should be possible for security experts to publish their security patterns without the need for a central authority to approve them. Rigorous user-controlled trust mechanisms come here to complement informal trust in the repository manager that is the basis of centralized

schemes. Additionally, mechanisms for pattern identification, retrieval, tracing (versioning, composition, etc.) provenance and navigation must be provided.

3. A NEW REPRESENTATION FOR SECURITY PATTERNS

In this section we define our conceptual view of what future security patterns should be and discuss possibilities for realizing that concept. We will refer to these new security patters as COmputer-Supported Security Patterns (COSSPs). We then define the approach and level of representational detail that we need for writing useful patterns, and finally we present the new view of security patterns.

When considering the redefinition of security patterns we came into the issue of the granularity of the definition of security pattern. In fact, we found security patterns that described both, low-level mechanisms with plenty of details and high-level architectural solutions with very few details. In order to provide an adequate treatment of both types, we concluded that we should distinguish between them using an analogy with two different conceptual elements:

- Tool. Defined in Apple's British Dictionary as a device or implement used to carry out a particular function; a thing used to help perform a job; or (Comp.) a piece of software that carries out a particular function.
- Solution. Defined in Apple's British Dictionary as a means of solving a problem or dealing with a difficult problem or situation; the correct answer to a problem or a puzzle; or a product or service designed to meet a particular need.

Consequently we have distinguished two different artefacts to represent each of those conceptual elements.

- We define the term Security Building Block (SBB) to represent a piece of software and/or hardware that carries out a particular security function. SBBs are the basic tools for the security engineers. SBBs do not provide a security property by themselves: that depends on the way you use them. An example of SBB is the RSA encryption algorithm.
- We also define the term COmputer-Supported Security Pattern (COSSP) to represent a particular way to meet a given security need. COSSPs are ways to combine SBBs to provide a security property. An example of a COSSP is XML Signature, which uses the RSA encryption algorithm (an SBB) in a specific way that provides Integrity and Authenticity for XML data. Thus, COSSPs provide specific solutions for security properties, sometimes using general security artefacts (SBBs).

In terms of realization, SBBs will be materialized as a variant of COSSPs (i. e. SBBs will be represented using a variant of the XML Schema defined for COSSPs). In order to keep the paper focused and to avoid confusion derived from mixing different artefacts, in this paper we will focus on COSSPs.

In order to adequately represent the new security patterns in a way that fulfills all abovementioned goals we consider that XML is the best possible container as it allows modular representation of the security pattern elements, is easy to parse, can contain at the same time complex structuring and precise references to specific elements (including cross-referencing and references to external ontologies) and has standards supporting the envisaged trust management mechanisms. Moreover, XML is already extensively used in system engineering activities and this fact will clearly facilitate the desired integration with engineering tools such as UML modeling suites. In fact, there is a standard XML-based format to contain UML models called XMI1. This facilitates our task of developing an XML-based representation of current security patterns since it allows us to directly embed UML models in the pattern structure. Additionally, XML provides support for defining and implementing the necessary trust mechanisms by means of standards like the XML-Signature [13] and SAML [14].

One possible alternative would have been to define the new security patterns to be UML models themselves, enhancing these models with all the other security pattern elements. This second alternative has the disadvantage of making the UML models the central element of security patterns, which is not always convenient. Additionally, all cataloguing, searching and browsing capabilities over different pattern data would be more difficult. Finally, we believe that we must follow an incremental and evolutive approach, instead of a complete redefinition, because the former approach will facilitate the transition to the new concept and the translation of the very valuable collections of already defined security patterns, while the latter would have the risk of requiring a complete revision of the existing patterns that would probably be unrealistic. Therefore, in our approach we propose to use XML for the representation of security patterns and to use XMI within the pattern structure to embed UML models when needed.

[1] `http://www.omg.org/spec/XMI/Current`

Our intention is to provide digital coding for as many as possible of the current security pattern elements including the definition of appropriate semantics for these elements in order to process the patterns by automated means. Notwithstanding, we believe that we should maintain the documentation-oriented parts, following the approach taken for Software Certifications in ASSERT4SOA [15] where there was a deliberately choice to maintain the textual versions in a part of the ASSERT called ASSERT4Humans. In the same line, we propose to have human-oriented contents in COSSPs that will be similar to the current ones (but more search-friendly, structured, etc.).

3.1 Sketching out the New Security Patterns

This subsection describes the contents that we propose for COSSPs based on the most popular structures currently used to describe security patterns. It is important to clarify that, while our final goal will be to be able to represent with COSSPs any security solution that can be described with existing security patterns and at the same time to facilitate the transition from the old format to the new one, in this paper we do not intend to be exhaustive in the set of fields, values, etc. but to convey the general idea and then to encourage the security pattern community to work in a joint definition and format for COSSPs. Of course, due to the fact that COSSPs have a wider focus than traditional security patterns, some of the pattern sections will not have exactly the same semantics or goal. Moreover, some of the sections of current security patterns do not have a clear semantics and some are overlapping. Therefore, in this initial proposal we have tried to maintain all the content but with a slightly different structure. Although we propose to use XML for the actual materialization of COSSPs, in this paper we will use EBNF[2] to show the structure, as it is more convenient for readers. We do not intend to provide a full syntax, as we believe we need to do it together with the security patterns community, in order to guarantee the necessary support for the transition to the new format. A typical Computer-supported pattern description should include the following elements:

1. *Pattern reference*.

In traditional Security Patterns, patterns are often referenced using the bibliographic reference of the paper in which they were published. For a machine-processable version such means of reference are not adequate. In the proposed design of the evolution towards the new Computer-supported format for security patterns we prioritized both the searching and the automation capabilities. Therefore, the first addition to the current structure of security patterns was to add pattern unique IDs using a coding scheme that would facilitate the distributed creation of these. To achieve this goal we propose the use of a namespace-based coding. Using this scheme, the ID for the "Authenticator Pattern" described in [16] could be "Authenticator.ifs.uni-regensburg.de". An initial structure for the PatternReference element could be:

```
PatternReference = Name, ID, Version, ShortDescription
```

2. *Intent*.

In traditional Security Patterns this element summarizes the main problem that the pattern solves. In COSSPs we consider that, in order to foster interoperability and precision, we must use the concept of Security Property to describe the intent of the pattern. Consequently, in COSSPs the intent could be better represented using a composed element that contains a list of the security properties provided by this pattern along with the elements of the system that are recipients of these properties. These elements are called "Assets" in COSSPs (for instance, we may say that the property "Integrity" applies to the asset "Social Security Number"). A COSSP provides one or more security properties and each property applies to one or more assets. Assets (the elements that are protected by the pattern) refer to elements that will have new properties once the pattern is integrated. Assets are "declared" in the Solution section of the COSSPs, and are referenced here. The assets are described in that section, providing some examples of use. For each security property the specification of the pattern Intent may be completed with the specification of restrictions (for instance, to specify that a pattern providing confidentiality protects data while "in transit" but not when stored at destination); metrics (for instance, to specify that the attack coverage level of an IDS is 87%) and forces (to specify balances between aspects such as for instance level of protection vs. processing time). The structure of the Intent element could be as follows:

```
Intent = Classification, [Preconditions], [{SecurityProperty}]
SecurityProperty = Name, [Restrictions], Metrics, Forces, [AssetList]
AssetList = {AssetReference}
```

[2] ISO/IEC 14977: 1996(E). http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

3. *Example.*

In traditional Security Patterns this is a human-oriented element. In COSSPs it maintains this orientation but with additional support for automated processing. The example is provided by means of diagrams, figures, code, etc. It supports any way that can facilitate the integration of the pattern in the system. The structure of the Example element could be as follows:

```
Example = Keywords, ExampleDescription
```

4. *Context*

In traditional Security Patterns this element is used to describe the context in which the pattern can be applied. In COSSPs, we keep this field (renaming it as ContextDescription), but we complement it with information that will facilitate the integration of the pattern in the design of the SUD, including any necessary assumption about the rest of the SUD and the representation of the SUD elements that will interact with the COSSP elements. The latter are described in the next subsection.

4a. *SUD interface elements*

In COSSPs we introduce this new element to represent the SUD components that are intended to interoperate with the pattern (called "SUD Interface Elements"). SUD Interface Elements are described from the point of view of the pattern and are therefore generic elements. For example they could be "keystore storage", "client", "encryption function", etc. The mapping between SUD Interface Elements and the actual elements in a given SUD that will interoperate with the pattern elements will be done when the pattern is integrated in the design of the SUD. This structure facilitates the development of automated tools to assist engineers in the sound integration of the pattern in the SUD. SUD Interface Elements are "declared" in the Solution section of the COSSPs (see point 6 below) and are referenced here. This section describes the context where the pattern can be applied, the assumptions for applying it and then the interface elements with which it interacts. The proposed structure for the PatternContext element is as follows:

```
PatternContext = ContextDescription, [SUDInterfaceElementList], [Assumptions]
SUDInterfaceElementList = {SUDInterfaceElementReference}
```

5. *Problem*

In traditional Security Patterns this section is used to describe the details of the main problem that the pattern solves. In COSSPs we capture these details by means of restrictions and metrics of the pattern related to the Security Properties. For this reason the specification of these elements is done as part of the Intent element described in point 2 above.

6. *Solution*

In traditional Security Patterns this section is used to explain how the pattern solves the problem. In COSSPs we define this element to be a complete model[3] that includes (i) the elements of the SUD that are protected by the pattern (which we call Assets); (ii) the functional elements (which we call SUD Interface Elements) of the SUD that must be associated to some of the elements defined in the pattern; and (iii) the pattern-specific elements (i.e. the security-specific objects that must be added to the SUD in order for the pattern to perform its functions), along with their interrelations and any other necessary related models (e.g. behavior, architectural, etc.). In the model, assets and SUD interface elements must be marked using specific stereotypes. Each of these elements will have a unique identifier that will be used to refer to it in other COSSP sections (e.g. AssetReference, SUDInterfaceElementReference). Of course, as the pattern is SUD-independent, both Assets and SUD Interface Elements are generic elements that must be associated by the system designer to elements in the actual SUD. To further clarify the difference between these two concepts consider again the Credential pattern, which is defined to be used in cases "in which the users of one system may wish to access the resources of another system, based on a notion of trust shared between the systems." Aside from the internal elements that realize the pattern, it will contain a generic asset, materialized as a class called "Attribute" representing the attributes that are securely conveyed by the credential, and a SUD Interface Element, materialized as a class called "Authenticator", representing the credential verification component of the SUD. In an actual e-commerce SUD, the "Attribute" element of the pattern would be mapped to "GoldenAmazonClient" in the SUD, and the "Authenticator" element of the pattern would map to a "ShibbolethServer" in the SUD. Figure 1 below shows a class diagram for the Credential pattern, a partial model of an e-commerce SUD, and the result of integrating the pattern in the SUD, including the association of stereotypes to show the role of the

[3] We initially assume it to be a UML model, but of course, other modeling formalisms like BPML, SysML or MARTE can also be used.

elements in the models. The solution provides also the elements that will be used in the development of the system. That way the engineer will know before implementing the elements of the system that interact with the solution, the extra required elements for the solution to work correctly, etc.

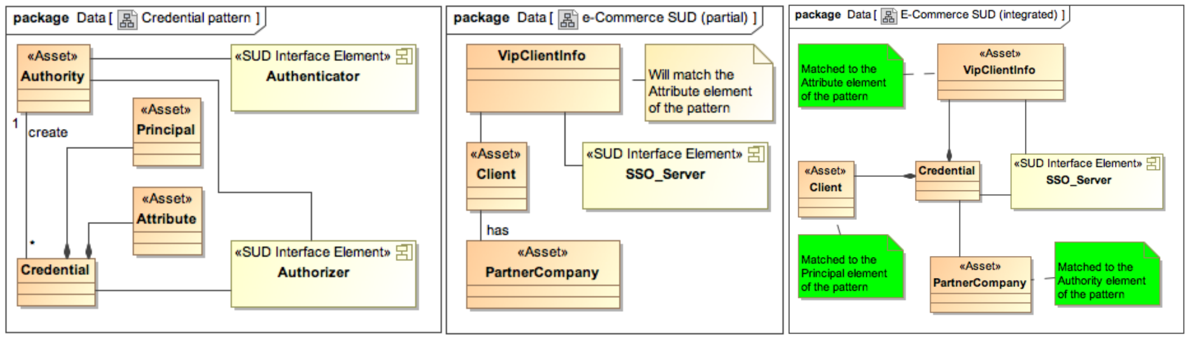
package Data [Credential pattern]
«Asset» Authority
«SUD Interface Element» Authenticator
«Asset» Principal
create
1
*
«Asset» Attribute
«Asset» Credential
«SUD Interface Element» Authorizer
package Data [e-Commerce SUD (partial)]
VipClientInfo
Will match the Attribute element of the pattern
«Asset» Client
«SUD Interface Element» SSO_Server
has
«Asset» PartnerCompany
package Data [E-Commerce SUD (integrated)]
Matched to the Attribute element of the pattern
«Asset» VipClientInfo
«Asset» Client
Credential
«SUD Interface Element» SSO_Server
Matched to the Principal element of the pattern
«Asset» PartnerCompany
Matched to the Authority element of the pattern

Fig 1. Example of "Assets" and "SUD Interface Elements"

The proposed structure for the Solution section is:

```
Solution = Description, Model, [Assets], [{ContextElements}]
Assets = {Asset}
Asset = Name, Description, ID
ContextElements = Name, Description, ID
```

7. Implementation

In traditional Security Patterns this section explains the implementation-specific considerations for the pattern. In COSSPs we maintain this important element of the traditional patterns (calling it ImplementationGuidelines), but we also propose the addition of two optional automation-oriented elements: (i) A series of rules, expressed in OCL used to verify the sound integration of the pattern in the system; and (ii) an element defining Tests (including type of testing, strategy, test suites, etc.) that can be used to verify that the pattern is adequately implemented. Additionally, it can optionally include references to existing implementations. The rules will validate and verify that the solutions work correctly in the applied system and inform of any detected problem. The structure of the Implementation element is as follows:

```
Implementations = {ImplementationGuidelines}, {Implementation}
Implementation = Name, Description, Model, (ImplementationReferences), [OCLRules], [Tests]
Tests = {Test}
Test = Name, Type, Strategy, TestSuite
```

8. Consequences

In traditional Security Patterns this element explains the advantages and liabilities that the pattern entail. In COSSPs we define this element to be a series of properties that can be assumed for the system once the pattern has been integrated, that we call "Postconditions" (to differentiate them from the Assumptions that are part of the Intention element) and a series of elements defining the limits and limitations of the pattern that we call "Scope". The structure of the Consequences element is as follows:

```
Consequences = [Postconditions], [Advantages], [Disadvantages], [Scope]
Scope = {ScopeSpecification}
```

9. Known Uses

In traditional Security Patterns this element contains references (normally in the form of bibliographic references) to known applications of the pattern. In COSSPs we have initially considered that this element is not necessary as a part of the pattern, and is even not convenient to have it inside the pattern, because in our opinion, as it changes independently of the pattern, including it would mean providing incomplete and frequently outdated information. However, this does not mean that we lose this information. On the contrary, a pattern management tool can be used to obtain up-to-date information of the pattern uses. Additionally, we are open to maintaining this element in order to provide useful engineering and development information on "selected application experiences". The actual structure of this element is not yet defined and should be decided with a wider support and collaboration of the pattern community.

10. Related Patterns

In traditional Security Patterns this element relates the pattern with other patterns appeared in the literature (normally in the form of bibliographic references). In the proposed version of COSSPs we define this element to be a series of pointers to other COSSPs using the COSSPs unique identifiers, along with version numbers and, optionally, validation mechanisms such as checksums and digital signatures. We are considering the possibility of externalizing this information, following the same approach as with the Known Uses element. However, due to the difficulty of automating the collection of this information, we have decided to include it in the pattern in this first proposal. The structure of the RelatedPatterns element is as follows:

```
RelatedPatterns = {RelatedPattern}
RelatedPattern = {Relation, [RelatedPatternReference]}
Relation = “uses” | “requires” | “includes” | “extends” | “excludes” | “overlaps”
RelatedPatternReference = {PatternID, [version], [validation]}
```

11. *Trust Mechanisms*

In traditional Security Patterns trust mechanisms are not necessary, especially because trust in the patterns is built by other non-technical means. However, for COSSPs this element is essential due to their computer-oriented format, and because we envisage the distributed and independent production of security patterns. Trust mechanisms like digital signatures ensure integrity, authenticity and origin of the patterns, and together with the unique PatternIDs support other forms of trust (e.g. reputation-based or quality-based). We are also considering trust mechanisms to cover quality evaluations, but this is still open to discussion and to finding realistic use cases in which this feature is required. The structure of the TrustMechanisms element is as follows:

```
TrustMechanisms = {TrustMechanism}
TrustMechanism= {Type, Value}
```

4. FOUNDATIONS FOR ENGINEERING SECURE SYSTEMS WITH COSSP

One of the key aspects for the adoption of any engineering paradigm or artefact is the existence of methodologies or processes aligned with the industry needs and practices that can guide its application, and the existence of tools that facilitate the adoption of the paradigm for newcomers, with enough support for ensuring its successful application and demonstrating practical advantages over other practices or over ad-hoc solutions to the problem that the new paradigm is intended to solve.

In fact, some of the problems that have hindered the adoption of traditional security patterns are the lack of systematic, predictable and reproducible methodologies for their application; the limited coverage of the engineering process (essentially limited to the selection of security solutions during the high-level design phase), the lack of integration of the few existing pattern-based methodologies with industry practices; and the lack of supporting tools for their application in the engineering process.

The proposed COSSP-based Engineering Process must cover the different phases ranging from requirements elicitation to the final implementation and testing. In this general process, the main role of COSSPs is in the design phase, as a mean to inform developers about possible solutions to fulfill their identified security requirements. A complete description of the proposed COSSP-based Process is out of the scope of this paper, but we consider it necessary to provide an overview to allow readers to understand the intended use for COSSPs.

The process is based on modeling formalisms like UML and structured as a series of activities that the engineer applies in order to transform an initial informal system specification into an executable system. The first activity deals with the requirement elicitation and specification. In this task, the system is represented as a simple UML model to which we add security requirements, specified as security properties attached to different model elements. This is the intended result of the activity, but we must highlight that the process does not make it mandatory to start with the identification of properties. Each organization can follow their current practices and start for instance with a threat analysis, but the activity must end with the identification of the security properties that are required from the system. The reason for using security properties for the specification of security requirements instead of threats or security mechanisms is that the former are much more stable and do not depend on the evolution of the security knowledge. However, threat-based engineering has been successfully used to develop innumerable systems and is an extremely valuable element in the activity. In fact, one important benefit of the threat-based engineering is that it allows more explicit tradeoffs between security and operational concerns based on the degree of threat/risk, and for this reason, threat based engineering is usually better suited for supporting design activities, while property-based approaches

are usually better for requirement specification. The proposed approach does not adopt one of these two approaches, but allow them to not only coexist, but to be related, and to switch between them when necessary. In this setting, automated support can prove very valuable for managing the requirements and for assisting engineers in the transformations between the different security concepts (threats, attacks, properties, etc.).
The main activity in the process is the system design, in which engineers design the mechanisms and functionalities necessary to fulfill the requirements identified in the previous activity. In this activity automated support can also play a central role. In the first place, automated support can help engineers identify the optimal solutions to fulfill the system requirements. In our case, these solutions are represented as COSSPs. Perhaps the most interesting and useful support that the tools can provide is in the process of integrating the COSSPs in the SUD. This is a complex process that is hard to perform correctly with current patterns, even when they provide detailed solutions because there are always many mistakes that the engineer can make. The whole process of adopting a security pattern will be facilitated by the availability of tools for automated support, and the fact that COSSPs contain different elements for guiding their adoption and for checking their correct integration. To do this, tools can be used to search different online distributed repositories of COSSPs and to present the designer with complete and reliable information about the available choices. Then, the selected COSSPs must be integrated into the System Model. Before the phase is finished, engineers must verify the coherence and correctness of the model they have obtained. This test can be automated to check whether the model fulfills the rules and restrictions of the general process, as well as the specific integration rules contained in the COSSP.
We must highlight that, we envisage a model-based approach, as opposed to a model driven one, so automated model transformations are not foreseen as the basis of model evolution. Normally, security pattern (and also COSSP) integration requires some manual work, but the way COSSPs are defined already improve the current process by providing the necessary information for implementing an automated assistant for pattern integration. Once this is done, the system designer must complete the system with specific functionalities. The form of the new COSSPs will also provide support to many other activities such as design optimization, testing definition, system evolution, etc.

5. CONCLUSIONS AND FUTURE WORK

Traditional security patterns are documents for engineers to understand a solution and apply it in their designs. Currently, high-quality patterns and catalogs are available for system engineers. Security patterns have been used in engineering activities for years, but with the move towards computer-supported system engineering, their text-based nature limits their use because their contents are not processable by computer tools. In fact, the only tools currently available for managing them are catalogs and repositories. Engineers must analyze which pattern to select for the specific system they work with and this requires them to study the whole catalog without any computer support. While the designer will still have the final say in this selection, an extended pattern description can make this selection easier and can enable the development of tools to assist engineers in their use.
COmputer-supported Security Patterns (COSSPs), are computer-oriented versions of the traditional security patterns. COSSPs are still to be used by engineers and developers, but they will not only help them understand a solution, they will also help them select it and apply it; for instance by showing different outcomes after applying the patterns. Compared with the traditional security patterns COSSPs have the potential to improve different aspects such as usability, searching and cataloguing, trust management, dynamic updating, and system integration.
We could establish an analogy between the evolution from the traditional security patterns to COSSPs and the evolution from paper plans to CAD designs for machines or buildings. System modeling and engineering tools will be able to use COSSPs directly and with additional advantages and guarantees.
An initial version of a format for COSSPs, together with a process, and requirements for support tools have been introduced in this paper, with the goal of helping designers select appropriate COSSPs to fulfill the security requirements of their systems and to integrate them in their SUD models. In particular, we envisage tools to manage COSSPs in online distributed repositories, to verify their integrity and authenticity, to browse the catalogs based on different elements of the patterns and to show the relevant documentation from within system modeling tools. In a longer term, we also envisage the introduction of interesting automation capabilities like assistants for integrating patterns in the SUD, consistency checkers, what-if analysis tools showing the effect of alternatives, guides for the developer through the stages of the software lifecycle, etc.

Our future work requires the support of the community, which we seek to gather by submitting this paper to PLOP, and focuses on further development of the new format, refining the details of the different fields, increasing the semantic precision of these, and completing the pattern description with additional fields. Additionally, we are developing precise requirements for the software tools to be used to implement the COSSPs repositories, which will be made available as open source in order to facilitate their adoption, as well as tools to support the COSSP-based process.

REFERENCES

A.V. Uzunov, E.B. Fernandez & K. Falkner (2012), "Securing distributed systems using patterns: A survey", Computers & Security, 31(5), 681 - 703.

E.B.Fernandez, "Security patterns in practice: Building secure architectures using software patterns". To appear in the Wiley Series on Software Design Patterns.

M. Schumacher, U. Roedig. "Security Engineering with Patterns". Lecture Notes in Computer Science, LNCS 2754 , 2002

Munawar Hafiz. Security architecture of mail transfer agents. Master's thesis, University of Illinois, 2005.

S. Moral-Garcia, S. Moral-Rubio, E. B. Fernandez, and E. Fernandez-Medina, "Enterprise Security Patterns: A new type of security pattern", submitted for publication.

P. El Khoury, P. Busnel, S. Giroux, and K. Li, "Enforcing security in smart homes using security patterns", Int. Journal of Smart Home, vol. 3, No 2, April 2009, 57-70.

G. Elahi, E. Yu, T. Li, and L. Liu, "Security requirements engineering in the wild: A survey of common practices", Procs. 35th IEEE Annual Comp. Software and Applics. Conf., 2011, 314-319.

E.B. Fernandez and X. Yuan, "Semantic Analysis Patterns", Procs. 19th Int. Conf. on Conceptual Modeling, ER2000 , Salt Lake City, UT, October 2000, 183-195. Lecture Notes in Computer Science, Volume: 1920 , 183-195

H. Washizaki, E. B. Fernandez, K. Maruyama, A. Kubo, N. Yoshioka, "Improving the Classification of Security Patterns," dexa, pp.165-170, 2009 20th International Workshop on Database and Expert Systems Application, 2009

K. Yskout, R. Scandariato, and W. Joosen, "Does organizing security patterns focus architectural choices?", ICSE 2012, Zurich, 617-627

E. B. Fernandez, "Designing secure architectures using security patterns", Wiley Series on Software Design Patterns. Wiley. 2013

G. Pujol, A. Maña, and Claudia Pandolfo. "A Distributed Secure Ontology for Certified Services Oriented Applications". IEEE Workshop on Ontologies for Systems Integration and Standards. Stanford (CA), USA, 2011.

D. Eastlake, J. Reagle, D. Solo. "XML-Signature Syntax and Processing". W3C. Request for Comments: 3275. March 2002

S. Cantor et al. "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0". OASIS Standard, March 2005. Document ID saml-core-2.0-os. Available online at http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

H. Koshutanski, A. Maña, R. Harjani, M. Montenegro, S. P. Kaluvuri, F. Di Cerbo, E. Damiani, C. Ardagna, A. Marco, D. Presenza, S. Gürgens, R. Menicocci, V. Bagini, F. Guida, and A. Riccardi. "ASSERT language v2.0" ASSERT4SOA Project Deliverable D1.2. Available online at http://www.assert4soa.eu/deliverable/D1.2.pdf

R. Erber, C. Schläger, and G. Pernul. "Patterns for Authentication and Authorisation Infrastructures". In Proceedings of the 18th International Conference on Database and Expert Systems Applications, DEXA 2007 Regensburg, Germany, 3 - 7 September 2007.

Appendix A:
Example COSSP (Table Format)

The example below is provided in order to sketch the proposed structure. However, given the fact that XML is not very human-friendly, we have used a table format in which XML tags appear on the left column and values appear on the right column. The indenting in the left column indicates containment (e.g. it shows that the Version field is part of the PatternReference field). Additionally, we must highlight that the Model field contains, in this format and for readers' convenience, a reference to a complete UML model in an external file, but our approach is to allow the inclusion of the model inside the pattern. This is technically simple since UML models have a standard XML serialization (XMI).

PatternReference	
Name	Credential _COSSP
ID	COSSP -Credential-PLOP13.proteus.uma.es
Version	0.9
ShortDescription	This pattern is a COSSP version of the Credential Security Pattern described in reference [2]
Intent	
Classification	General-purpose
Preconditions	
Precondition	The terms and limitations of system access of the credential are already defined
Precondition	The credentials have an expiration date
SecurityProperty	P001
Name	Authentication
Restrictions	
Restriction	The two systems exchanging data must trust each other
Restriction	There must exist other entities that produce the certifications
Restriction	The user must provide enough information to grant authorization, without being exposed to intrusive data mining
Restriction	The information must be packaged and stored in a way that survives travel between systems while allowing the data to be kept private
Restriction	The data available must be sufficient for identifying the principal to the satisfaction of the accepting system's requirements while disallowing others from accessing the system
Restriction	The data available must be sufficient for determining what actions the presenting principal is permitted to take within the accepting system while also disallowing actions the principal is not permitted to take
Restriction	The system accepting the credential must trust the system issuing the credential
Restriction	There must be entities that produce these documents such as other domains recognize them
Restriction	It should be very difficult to falsify this document
Restriction	The document should have a explicit temporal validity
Restriction	It might be necessary to use the document together with other documents
Restriction	It should be possible to conveniently revoke the document
Metrics	
Forces	Making credentials tamper-resistant takes extra time and complexity
AssetList	
AssetReference	Asset_Cred_User
AssetReference	Asset_Cred_Auth
AssetReference	Asset_Cred_Cred
Example	
Keywords	Instant messaging application; Communication; Revocation; Distributed
ExampleDescription	Suppose we are building an instant messaging service to be used by members of a university community. Students, teachers and staff of the university may communicate with each other, while outside parties are excluded. Members of the community may use computers on school grounds, or their own systems, so the client software is made available to the community and is installed on the computers of their choice. Any community member may use any computer with the client software installed. The client software communicates with servers run by the university in order to locate active participants and to exchange messages with them. In this environment, it is important to establish that the user of the client software is a member of the community, so that communications are kept private to the community. Further, when a student graduates, or an employee leaves the

	university, it must be possible to revoke their communications rights. Each member needs to be uniquely and correctly identified, and a member's identity should not be forgeable.
PatternContext	
ContextDescription	Systems in which the users of one system may wish to access the resources of another system, based on a notion of trust shared between the systems.
SUDInterfaceElementList	
SUDInterfaceElementReference	Asset_Cred_System
Assumptions	
Assumption	Credentials are created by trusted authorities
Assumption	The credentials are portable
Solution	
Description	Store authentication and authorization data in a data structure external to the systems in which the data are created and used. When presented to a system, the data (Credential) can be used to grant access and authorization rights to the requestor. In order for this to be a meaningful security arrangement, there must be an agreement between the systems which create the credential (Credential Authority) and the systems that allow their use, dictating the terms and limitations of system access
Model	https://proteus.lcc.uma.es/solutions/Auth_Cred.mdzip /** In the future, the model could be directly included in the pattern **/
Assets	
Asset	
Name	User
Description	The user or system that access the system and needs to authenticate
ID	Asset_Cred_User
Asset	
Name	Credential
Description	The credential that verifies the authentication of the user or system
ID	Asset_Cred_Cred
Asset	
Name	Authenticator
Description	The authenticator system for checking the truth of the entity
ID	Asset_Cred_Auth
Asset	
Name	System
Description	The system that is going to be accessed by the user
ID	Asset_Cred_System
ContextElements	
Name	Authority
Description	The authority that creates the credential
ID	CE_Cred_Authority
Implementations	
Implementation	
Name	Credential Encryption
Description	This method encrypts the credentials when they are issued, and to set up matching decryption on the authenticating system. This further subdivides into “shared secret” systems, where the issuing and accepting systems share the cryptographic keys necessary to encrypt and decrypt credentials and “public key” systems, where participating systems can establish means for mutual encryption/decryption without prior sharing
Model (SBB)	https://proteus.lcc.uma.es/implementations/Auth_Credentials_Encryption
OCLRules	
OCLRule	
ImplementationGuideline	The Authenticator must use the same scheme as the Authority. Kerberos tokens and X.509 certificates
Tests	
Test	
Name	Encryption Testing
Type	Active
Strategy	Check the resilience of the encryption algorithm
TestSuite	https://proteus.lcc.uma.es/testsuites/cred_encryp_test

Implementation	
Name	Credential secure channel
Description	Credentials are established and used within a closed context and encrypting the communications channels used in that context
Model (SBB)	https://proteus.lcc.uma.es/implementations/Auth_Credentials_Secure_Channel
OCLRules	
OCLRule	
Tests	
Test	
Name	Secure channel testing
Type	Active
Strategy	Check the confidentiality and integrity of the communication channel
TestSuite	https://proteus.lcc.uma.es/testsuites/cred_sec_channel_test
Consequences	
Postconditions	
Postcondition	
Advantages	
Advantage	Fine-grained authentication and authorization information can be recorded in a uniform and persistent way
Advantage	A Credential from a trusted authority can be considered proof of identity and of authorization
Advantage	It is possible to protect credentials using encryption or other means
Disadvantages	
Disadvantage	It might be difficult to find an authority that can be trusted. This can be resolved with chains (trees) of credentials, where an authority certifies another authority
Disadvantage	Making credentials tamper-resistant takes extra time and complexity
Disadvantage	Storing credentials outside of their using systems leaves system authentication and authorization mechanisms open to offline attack
Scope	
ScopeSpecification	The certificate must be generated by a trusted entity
RelatedPatterns	
RelatedPattern	
Relation	Uses
RelatedPatternReference	
PatternID	Auth_Pattern
Version	1.0
Validation	https://proteus.lcc.uma.es/repository/patterns/validator_Auth_Pattern_v10
RelatedPattern	
Relation	Includes
RelatedPatternReference	
PatternID	Metadata-based_Access_Control
Version	1.0
Validation	https://proteus.lcc.uma.es/repository/patterns/validator_Meta_AC_Pattern_v10
RelatedPattern	
Relation	Extends
RelatedPatternReference	
PatternID	Security_Session_Pattern
Version	2.0
Validation	https://proteus.lcc.uma.es/repository/patterns/validator_Sec_Session_Pattern_v20
RelatedPattern	
Relation	Overlaps
RelatedPatternReference	
PatternID	Remote_Authenticator_Pattern
Version	2.1
Validation	https://proteus.lcc.uma.es/repository/patterns/validator_Rem_Authen_Pattern_v21
RelatedPattern	
Relation	Includes
RelatedPatternReference	
PatternID	Authorizer_Pattern
Version	1.1

Validation	https://proteus.lcc.uma.es/repository/patterns/validator_Authorizer_Pattern_v11
RelatedPattern	
Relation	Includes
RelatedPatternReference	
PatternID	Delegation_Cred_Pattern
Version	1.1
Validation	https://proteus.lcc.uma.es/repository/patterns/validator_Delegation_Cred_Pattern_v11
RelatedPattern	
Relation	Extends
RelatedPatternReference	
PatternID	Circle_of_Trust_Patterns
Version	2.0
Validation	https://proteus.lcc.uma.es/repository/patterns/validator_CoT_Pattern_v20