# MASC: Modelling Architectural Security Concerns

Laurens Sion*, Koen Yskout*, Alexander van den Berghe*, Riccardo Scandariato†*, and Wouter Joosen*
*iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
Email: {Laurens.Sion, Koen.Yskout, Alexander.vandenBerghe, Wouter.Joosen}@cs.kuleuven.be
†Software Engineering Division, Chalmers & Göteborg University, 41756 Göteborg, Sweden
Email: Riccardo.Scandariato@chalmers.se

***Abstract*—Security decisions are an important part of software architecture design, and thus deserve to be explicitly represented in the design documentation. While UML is the best-known language for creating such documentation, it lacks security specific notations, which makes it difficult to represent the effect of the security decisions. Several security extensions for UML exist in the literature, but they represent security concerns at a lower level of abstraction, or only support a limited subset of security concerns. We propose a new notation, MASC, to model security concerns at the architectural level. It has been designed as an extension of UML, and is based on recurring security concepts that have been distilled from well-known security principles, goals, and patterns. By using our notation, a designer obtains a technique to express security concerns more explicitly in the architectural design documentation.**

***Index Terms*—security, software architecture, notation, UML, MASC**

## I. Introduction

The paper of Perry and Wolf [1] forms the starting point for the research on software architectures in 1992. Software architectures are required because of the higher form of abstraction necessary when dealing with larger and more complex software systems [2]. Because of the size and complexity of these systems, the architecting process produces and consumes a lot of knowledge [3], resulting in a shifting view to documenting not only the end result, but also the underlying rationale [3].

However, examining the different strategies for achieving security in software architectures, such as security patterns and the underlying security concepts and techniques they apply, reveals that an explicit manner to express security knowledge at this level of abstraction is missing. This makes it difficult to express and convey the security decisions and properties of a software architecture to other software architects, designers, or programmers. Nevertheless, there are several reasons why such an explicit representation for this information is necessary: (a) it allows for documentation of decisions and security information for future use; (b) it provides guidelines during the future development of the system, determining the actual implementation of the system; and (c) it can form a base for analysis, review, and compliance testing.

Our goal is to enable a software architect to represent the important security aspects of the design explicitly in the architectural documentation. Therefore, the contribution of this paper is twofold. First, we have compiled a list of important security-related concepts at the architectural level. We distilled this list from common security concepts and techniques, goals, and their usage in published security patterns. Second, we propose a new notation, Modelling Architectural Security Concerns (MASC), which is an extension of UML, dedicated to represent these concepts in a more explicit manner. Tool support for the notation has been developed as a prototype.

## II. Security Design Concepts

MASC is based on a list of security design concepts. To construct this list, we start from two security information sources: (1) well-known and often-used security concepts or techniques (e.g., compartmentalization, least privilege, etc.) from Saltzer and Schroeder [4], and Viega and McGraw [5]; and (2) security goals and objectives from Viega and McGraw [5], Pfleeger and Pfleeger [6], Firesmith [7], and Hernan et al. [8].

As illustrated in Fig. 1, these two sources are first examined with the purpose of creating an initial concept list. Using this list, we then perform an in-depth study of an elaborate set of security patterns (36 patterns from different sources [9]–[17]). For each pattern, we record the concepts that are used, and for which purpose. Usage can be explicit (i.e., mentioned in the pattern description), or implicit (i.e., the concept is used, but not explicitly mentioned in the pattern description). For example, an AuthenticationEnforcer intercepts requests, but its description does not use the term ‘interception’. Concepts that are important for the pattern but not yet in the list are added. This results in an extended list of security design concepts, together with the number of security patterns that use them.

Finally, because significant overlap among the resulting concepts exists, the concepts are grouped and merged (right-side of Fig. 1), which produces the final list as shown in Table I. The table contains the name of each identified concept, a short description, and the number of patterns that uses it. This list is used as the foundation for constructing MASC.

During the composition of the concept list, the need for a dedicated, security-oriented notation quickly became apparent.

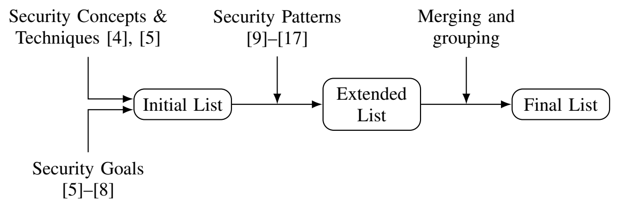


Fig. 1. Construction of the concept list.

https://doi.org/10.1109/MiSE.2015.14

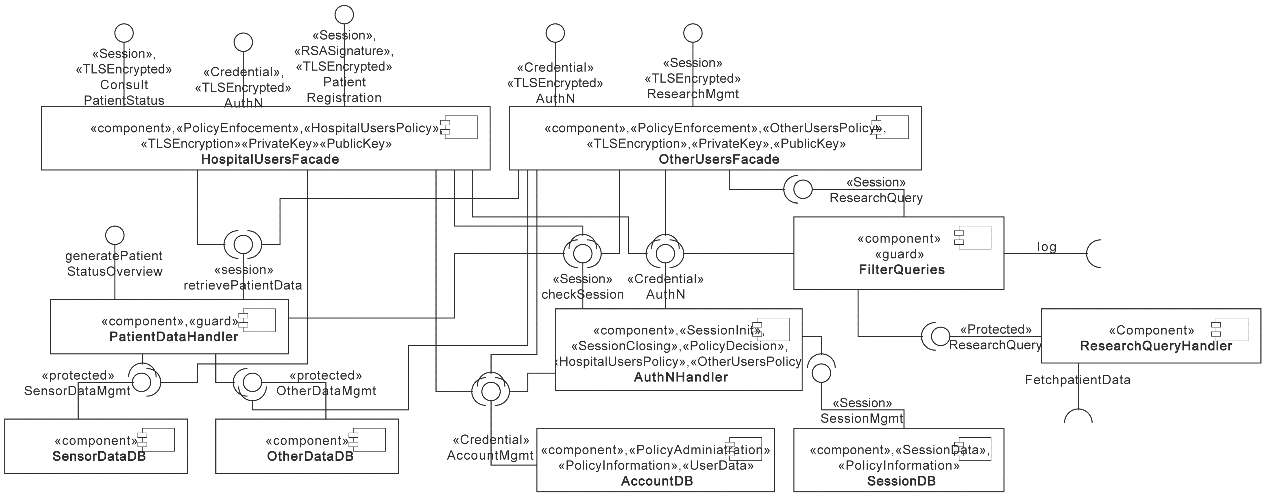

«Session», «TLSEncrypted» Consult PatientStatus
«Credential», «TLSEncrypted» AuthN
«Session», «RSASignature», «TLSEncrypted» Patient Registration
«Credential» «TLSEncrypted» AuthN
«Session» «TLSEncrypted» ResearchMgmt
«component»,«PolicyEnfocement»,«HospitalUsersPolicy» «TLSEncryption»«PrivateKey»«PublicKey» HospitalUsersFacade
«component»,«PolicyEnforcement»,«OtherUsersPolicy», «TLSEncryption»,«PrivateKey»,«PublicKey» OtherUsersFacade
«Session» ResearchQuery
«component» «guard» FilterQueries
log
generatePatient StatusOverview
«session» retrievePatientData
«Session» checkSession
«Credential» AuthN
«component»,«guard» PatientDataHandler
«component»,«SessionInit», «SessionClosing»,«PolicyDecision», «HospitalUsersPolicy»,«OtherUsersPolicy» AuthNHandler
«Protected» ResearchQuery
«Component» ResearchQueryHandler
«protected» SensorDataMgmt
«protected» OtherDataMgmt
«Session» SessionMgmt
FetchpatientData
«component» SensorDataDB
«component» OtherDataDB
«Credential» AccountMgmt
«component»,«PolicyAdministration» «PolicyInformation»,«UserData» AccountDB
«component»,«SessionData», «PolicyInformation» SessionDB

Fig. 2. A part of the initial architecture of the Patient Monitoring System. The security concerns are expressed in plain UML using custom stereotypes.

TABLE I
SECURITY CONCEPT LIST, BASED ON COMMON CONCEPTS, TECHNIQUES, GOALS, AND SECURITY PATTERNS

| | Concept | Pattern Count |
|---|---|---|
| 1 | **Interception**<br>Intercepting requests to read, modify, or block them. | 18 |
| 2 | **Policies**<br>Apply security policies, for example to make behaviour dependent on the user's role. | 18 |
| 3 | **Complete mediation**<br>Guard all access to an entity that needs protection. | 17 |
| 4 | **Credentials**<br>Require the use of credentials or session identifiers. | 14 |
| 5 | **Cryptography**<br>Usage of encryption, signatures, hash functions, and their keys. | 6 |
| 6 | **Distribution***<br>Distribute requests or calculations to redundant entities. | 6 |
| 7 | **Centralization***<br>Centralize functionality to prevent duplication. | 8 |
| 8 | **Standardization***<br>Use standardized, published, and tested algorithms. | 4 |

*: no explicit notation in MASC

Many security patterns lack a visual representation of the security concepts they contain, and only provide a textual description of the technique. If a visual representation is included, it is often cumbersome to understand or incomplete.

## III. The MASC Notation

MASC supports the top five concepts from Table I. The three concepts at the bottom of Table I are not explicitly represented in MASC for multiple reasons. They are less widely used by the patterns, and not entirely security specific. Furthermore, distribution can already be expressed in UML using multiplicities, while centralization and standardization are principles that are not easily expressed graphically.

Two concepts, namely interception and complete mediation, depend on a more invasive extension of UML, and will be discussed in-depth in subsections III-B and III-C. The other concepts are more lightweight, as they are based on icons that are added to UML diagrams. They will be discussed together, and more briefly, in III-D. Due to space constraints, we cannot discuss all aspects of MASC in depth. For more details about the notation and our prototype implementation, we refer the reader to [18].

### *A. Description of the example*

Throughout this section, we will illustrate the application of MASC with an example architecture (Fig. 2). The example is part of a patient monitoring system (PMS), designed to monitor patients remotely by collecting data from sensors they wear on their body. The system has two entry points for user interaction: one for personnel of the hospital (`Hospital-UsersFacade`), the other one for patients, general practitioners, and researchers (`OtherUsersFacade`).

Users can log in to the system via these facades. After authenticating, they can retrieve patient status information, risk assessments based on the collected sensor data, and other patient information (depending on their privileges). The processing of the requests and retrieval of the data from the databases is performed by handler components (e.g., `PatientData-Handler` and `ResearchQueryHandler`).

The architecture in Fig. 2 uses stereotypes to include security-relevant information, for example «credential» and «session» in the context of authentication.

### *B. Interception*

Interception is a common and frequently used concept in security, as illustrated by the number of patterns in Table I-1. In order to represent it in plain UML, interfaces need to be duplicated, as `FilterQueries` does for the `Research-Query` interface at the right-hand side of Fig. 2. Additionally, the diagram does not contain any information on the type of

TABLE II
OVERVIEW OF SECURITY GOAL (SUB)CATEGORIES

| **Abbrev.** | Category | Subcategory |
|---|---|---|
| AC:An | Access Control | Authentication |
| AC:Az | | Authorization |
| AHD | Attack and Harm Detection | |
| NR | Non-Repudiation | |
| I | Integrity | |
| C | Confidentiality | |
| Av | Availability | |
| Av:R | | Recovery |

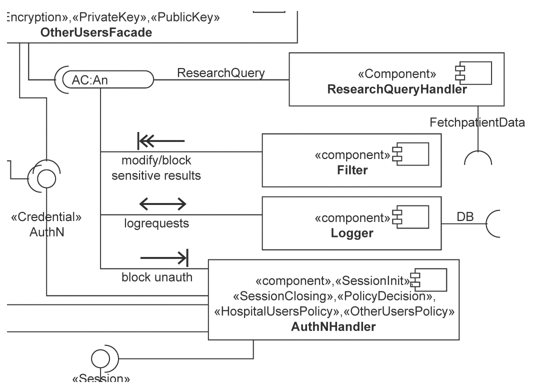


Fig. 3. The interception notation applied.

interception (e.g., reading, modifying, blocking) that can be performed by the component.

Fig. 3 illustrates the application of the interception notation in MASC for this part of the architecture. An intercepted interface is represented using a stretched interface symbol. To indicate the purpose of the interception, the symbol can include the desired security goal, for example from the list in Table II. Intercepting components are connected vertically to the intercepted interface. Furthermore, the type of each interception connection is indicated using an arrow, whose meaning is explained in Table III. For example, the `Filter` component in Fig. 3 can read, modify, or block sensitive replies from the `ResearchQueryHandler` component, while the `AuthN-Handler` can block (unauthenticated) queries.

Representing interception in this way is still ambiguous when multiple intercepting components are present, because the exact order in which they handle the requests cannot be derived. Therefore, the UML sequence diagram notation is also extended with a notation for interception, as shown in Fig. 4. In the example, requests and replies are again intercepted between the `OtherUsersFacade` (OUF) and the `ResearchQueryHandler` (RQH). Empty lifelines separate the intercepting component(s) from the rest. When a request from the client is intercepted, it goes to the intercepting component, which can process it, after which it is forwarded to the destination component. The dashed horizontal line in the center indicates the moment the request is forwarded. Note that a message from left to right below this line corresponds to the 'Post' arrow annotation. The structure for intercepting

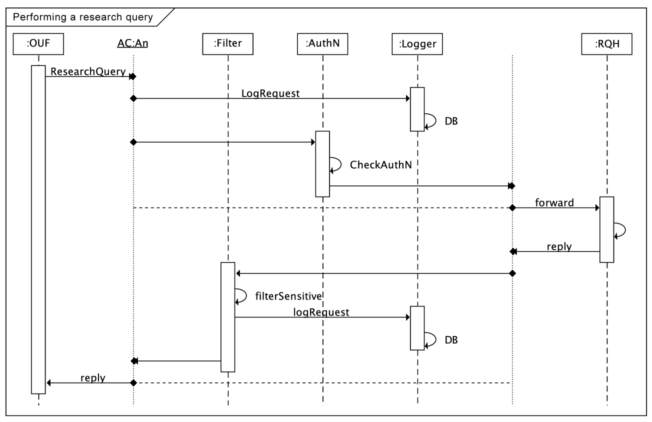


Fig. 4. Interception sequence diagram example.

TABLE III
TYPES OF INTERCEPTION ARROWS AND THEIR MEANINGS

| **Arrow** | **Meaning** |
|---|---|
| ⟶ | Reading requests |
| ↠ | Modifying requests (implies reading) |
| ⇥ | Blocking requests (implies reading) |
| ↠\| | Modifying/blocking requests (implies reading) |
| Post ⟶ | Asynchronously read a request after it has been passed on (not allowed for modification or blocking) |
| ⟵ | Reading replies. All the above apply to replies as well. |
| ← reply | The interceptor can initiate a reply itself, instead of processing an existing one. (Drawn under the connector.) |

replies is analogous to that for intercepting requests, but occurs in the other direction.

The main benefit of our notation is that it allows the visualization of interception in the architecture, including the type and goal of interception, without the need for replicating interfaces as in plain UML. A possible caveat is the reliance on the directions of the arrows to convey meaning. Changing the orientation or placement of components might lead to different interpretations of the arrows. This requires some diligence when using the notation.

## *C. Complete Mediation*

A second very common security technique is complete mediation (see Table I-3). A straightforward way to represent this concept in UML is with «guard» and «protected» stereotypes on components and interfaces respectively, as illustrated at the left-hand side of Fig. 2. This is similar to the approach taken in UMLsec [19]. While this allows the inclusion of some of the information, there is no clear indication of which protected interfaces require which guards, or whether there are multiple guards. It is also not easy to spot complete mediation on diagrams, which can lead to errors when additional interfaces

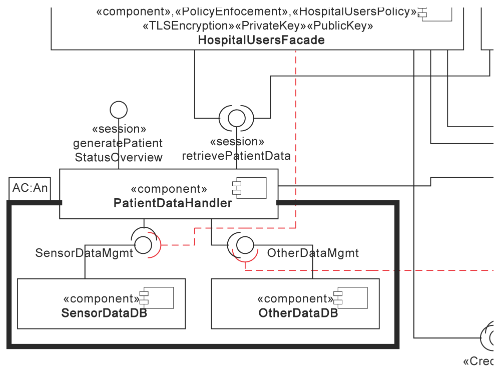


Fig. 5. The complete mediation notation applied.

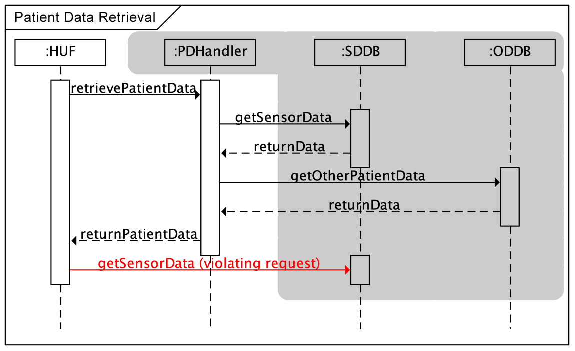
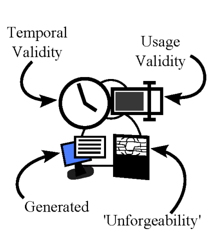


Fig. 6. Complete mediation sequence diagram.

or connections are added. For example, Fig. 2 tries to express that the interfaces of `SensorDataDB` and `OtherDataDB` should only be accessed through `PatientDataHandler`, which acts as their guard, but two violations against this rule are present.

This example can be modelled in MASC as shown in Fig. 5. Access to the guarded interfaces is limited to the components on the border. The border is annotated with the security goals, similar to the interception notation. Border components can act just like other components, and are not restricted to just passing along requests. The notation makes violations against the complete mediation principle (i.e., bypassing the border component) detectable at a glance. The two violations from before are marked with dashed lines in Fig. 5.

Because the border components can have arbitrary behaviour, MASC also supports the complete mediation notation for sequence diagrams, as illustrated in Fig. 6. `SensorData-DB` (SDDB) and `OtherDataDB` (ODDB) are shielded, indicated by their shaded backgrounds, and access to them is mediated by the border component `PatientDataHandler` (PD-Handler), with the head of the lifeline shaded. The handler is used by the `HospitalUsersFacade` (HUF) to access them. This makes the distinction between protected and unprotected components easy to see. To check if a component may access a protected component, just check its head to see if it is a border component. Hence, violations such as the one marked



**Temporal validity** expresses that the credential expires after a certain time.
**Usage validity** expresses that usage of the credential is limited (e.g., it can only be used once, such as a code generated by a token).
**Generated** expresses that the credential is randomly generated or has complexity requirements.
**Unforgeable** expresses that the credential is protected from being easily reproduced (e.g., a smartcard).

Fig. 7. The credential icon with four parameterization dimensions.

in Fig. 6 are again easily spotted. MASC also allows multiple nested applications of complete mediation, though care should be taken to limit the number of nested applications in a single diagram.

## D. Additional concepts

In contrast to interception and complete mediation, a representation for cryptography, credentials, and policies requires no invasive changes to UML. These concepts can already be expressed reasonably well with stereotypes, as illustrated in Fig. 2. Nevertheless, there is no standardized way for expressing them, which can lead to confusion or ambiguities. Furthermore, the textual representation of a stereotype quickly leads to cluttered diagrams, hindering the readability, and preventing architects from quickly parsing the information. Indeed, textual differentiation is recognized as an ineffective way of dealing with complexity [20].

MASC offers icons to represent these concepts graphically. It provides the possibility to express sufficient details, while requiring limited effort to include them in existing solutions. What follows is a brief overview of the offered representation; more details can be found in [18].

The *cryptography* notation expresses the presence of cryptographic techniques (i.e., encryption , signing , hashing , and keys ). It allows to make a distinction between performing an operation (e.g., encrypting ) and handling the resulting data (e.g., cipher text ). The algorithms that are used can also be included, contributing to the standardization concept from Table I. For example, at the top of Fig. 8, encryption using TLS is applied to the interfaces of the facades. The symbols enable the designer to check whether all requirements for cryptographic operations are fulfilled (e.g., components have access to the necessary keys).

The *credentials* notation covers the usage of credentials  (such as passwords) and session identifiers  in the architecture. They help to identify locations in the architecture where additional measures (such as encryption) are desired to protect them, or spoofing attacks may occur. The credential icon is parameterized over four dimensions, to give a rough indication of its strength (Fig. 7). The icon for sessions also allows indicating where sessions are initialized , closed , and data is stored . This notation allow the designer to verify whether credentials and sessions need to be protected

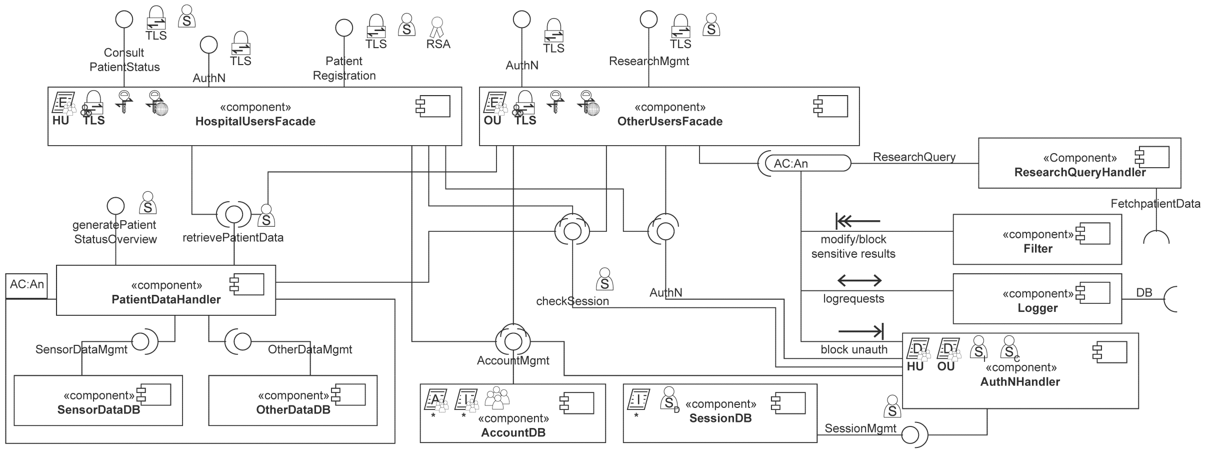


Fig. 8. The example architecture expressed in MASC.

by cryptographic techniques. In Fig. 8, it is applied to the `AuthNHandler` and its interfaces.

The *policy* notation is used to indicate the different types of policy points [21] in the architecture, such as enforcement E, decision D, administration A, and information I,as well as places where information is stored that can influence the outcome of a policy evaluation (e.g., user attributes ). This allows the designer to specify clearly where policies are stored, and which components are responsible for evaluating them. It can also be used to determine which interfaces exhibit user-role dependent behaviour due to the policies applied to them. Additionally, it allows for checking if the policy enforcers have access to the necessary data and interfaces (e.g., user data) to perform their functions. It is used at the facades in Fig. 8.

Combining these three notations with interception and complete mediation yields additional analysis opportunities. For example, it can be determined whether interception is performed at the required places, as indicated by the policies. Additionally, it can also be verified whether it is possible to perform interception at the indicated place in the architecture (i.e., can the requests be read or are they encrypted, and if so, does the interceptor have access to the keys?). The complete mediation notation offers similar possibilities in combination with the notations from this section. Currently, these checks can only be performed manually, so they depend on the expertise of the analyst, but a more precise semantics for MASC would enable automating them.

## IV. Related Work

Our work belongs to the space of model-driven security research. We do not attempt to give a thorough overview of this extensive domain here, but refer to existing surveys [22], [23]. We limit ourselves to a comparison with two of the best-known examples from this domain.

UMLsec [19] offers a UML profile for annotating a software design with security-specific information. In comparison with MASC, the concepts offered by UMLsec are at a lower level of abstraction. This follows from the different goal of both approaches: UMLsec is geared towards automated formal verification, which our approach currently does not offer. Additionally, the extensive use of stereotypes and tagged values make UMLsec less suitable as a notation for use by humans (the primary goal of MASC), as indicated earlier in section III-D. Additionally, more complex concepts such as complete mediation, and especially interception, are very difficult to clearly and unambiguously represent with UML stereotypes.

SecureUML [24], another well-known approach for model-driven security, proposes a technique to combine modelling languages (e.g., UML) and security languages (e.g., a language for RBAC) in order to formalise the access control requirements of a software system, and generate access control infrastructures. The focus of SecureUML lies primarily on access control, hence most of the concepts in Table I cannot be represented.

## V. Discussion

We identify three relevant criteria for a good notation: the quality of the *syntax* or form, the quality of the *semantics* ("What can be represented?"), and the quality of the *mapping* between the two.

*Syntax:* Moody [20] defines 9 criteria for a good visual notation. Discussing each of them in depth is beyond the scope of this paper, so we discuss the visual quality as a whole. MASC provides different, easily discernible symbols. The appearance of the symbols has been chosen to suggest their meaning, for example by using a lock icon for encryption, or a border around shielded components. MASC also uses text to complement the notation with more detailed information. The limited set of symbols keeps the notation cognitively manageable. However, there are also some less optimal aspects. As indicated earlier, the meaning of the arrows of the interception notation is derived from their orientation. Additionally, MASC does not provide explicit complexity management mechanisms. Because MASC targets a specific subset of the architecture (i.e., the security-relevant parts), there is a somewhat reduced need for such mechanisms.

*Semantics:* MASC offers a broad spectrum of frequently used security concepts, as illustrated in Table I, in order to allow architects to represent different security aspects and their interactions. The concepts are obtained from their actual usage in security principles and patterns, so they cover at least the range of security issues for which the considered patterns are applicable. Because security measures often contain a lot of important details, MASC incorporates these as parameters, so the information is still recorded and available for analysis.

*Mapping:* A final important property for a notation is the mapping between the semantic constructs and the visual syntax. Each of the 5 concepts from Table I has a separate representation, so MASC has a 1:1 mapping between syntax and semantics, which should be strived for [20].

## VI. Future Work

Extensive testing with users is necessary to verify that the right balance has been achieved between the different qualities, to collect feedback, and to further refine and improve the notation. We also plan to use MASC for modelling the security of existing systems, in order to verify whether the notation is sufficiently expressive and no gaps exist in the set of concepts.

We expect that the most important shortcomings will be identified from performing such evaluation studies. Some incremental improvements that we currently foresee are allowing user-defined parameters (balanced against the possible increase in ambiguity); extending the policy notation with a taxonomy for more precision; expanding the expressiveness by including other concepts or shifting the coverage of the concepts; and changing the balance between visible and non-visible properties.

Improvements can also originate from an in-depth comparison of MASC with other notations, to discover whether MASC is equally expressive or if there are perhaps missing concepts. In addition, further development of tool support is useful to make MASC more widely available and usable, and can help with conducting user experiments.

Furthermore, on a longer term, we intend to investigate how a MASC design can be used to perform a security analysis, and whether the design can be linked to the implementation via code generation or reverse engineering, for example.

## VII. Conclusion

Software architectures are the result of a process that depends on a lot of knowledge and many different design decisions, including decisions and knowledge pertaining to the security of the architecture. These decisions require an explicit representation in the architectural design documentation, but a language for adequately documenting them is missing.

In this paper, we present the MASC notation as an extension of UML. The design of the notation has been driven by well-known security techniques and goals from the literature, and their usage in security patterns. It allows for a concise representation of five recurring security concepts, namely interception, complete mediation, policies, credentials, and cryptography. By using MASC, we believe that software practitioners can more easily specify and become aware of the security decisions behind of a piece of software, both during the software's design phase and in later phases of its development life cycle.

## Acknowledgment

This research is partially funded by the Research Fund KU Leuven. We also want to thank the reviewers for their feedback.

## References

[1] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 4, pp. 40–52, Oct. 1992.
[2] M. Shaw, "Larger scale systems require higher-level abstractions," *SIGSOFT Softw. Eng. Notes*, vol. 14, no. 3, pp. 143–146, Apr. 1989.
[3] M. A. Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds., *Software Architecture Knowledge Management*. Springer Berlin Heidelberg, 2009.
[4] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
[5] J. Viega and G. McGraw, *Building secure software: how to avoid security problems the right way*, 1st ed. Boston, MA, USA: Addison-Wesley, September 2002.
[6] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*, 3rd ed. Prentice Hall, 2003.
[7] D. Firesmith, "Specifying reusable security requirements," *Journal of Object Technology*, vol. 3, no. 1, pp. 61–75, 2004.
[8] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Threat modeling: Uncover security design flaws using the stride approach," *MSDN Magazine*, vol. 6, November 2006.
[9] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," in *PLoP*, vol. 51, 1997, p. 61801.
[10] K. E. Sørensen, "Session patterns," in *Proceedings EuroPLoP*, 2002.
[11] T. Saridakis, "Design patterns for fault containment," in *EuroPLoP*, 2003, pp. 493–520.
[12] M. Schumacher, "Firewall patterns," in *EuroPLoP*, 2003, pp. 417–430.
[13] B. Blakley, C. Heath, and members of The Open Group Security Forum, "Security design patterns," *The Open Group*, 2004.
[14] N. Delessy-Gassant, E. B. Fernandez, S. Rajput, and M. M. Larrondo-Petrie, "Patterns for application firewalls," in *Proceedings of the Pattern Languages of Programs (PLoP) Conference*, 2004.
[15] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Pratices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall Ptr, 2005.
[16] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-hewitt, "Security patterns repository, version 1.0," 2006.
[17] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns*. Wiley, 2006.
[18] L. Sion, "Representing architectural security concerns," Master thesis, KU Leuven, Department of Computer Science, June 2014.
[19] J. Jürjens, *Secure Systems Development with UML*. Springer Berlin Heidelberg, 2005.
[20] D. Moody, "The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, Nov. 2009.
[21] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," OASIS Standard, Tech. Rep. 3.0, January 2013.
[22] J. Jensen and M. G. Jaatun, "Security in model driven development: A survey," in *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*. IEEE, 2011, pp. 704–709.
[23] P. Nguyen, J. Klein, Y. Le Traon, and M. Kramer, "A systematic review of model-driven security," in *20th Asia-Pacific Software Engineering Conference (APSEC'13)*, vol. 1, Dec 2013, pp. 432–441.
[24] D. Basin, J. Doser, and T. Lodderstedt, "Model driven Security: From UML Models to Access Control Infrastructures," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 39–91, 2006.