Softw Syst Model
DOI 10.1007/s10270-015-0486-9

REGULAR PAPER

Design notations for secure software: a systematic literature review

Alexander van den Berghe[1] · **Riccardo Scandariato**[2] · **Koen Yskout**[1] · **Wouter Joosen**[1]

Received: 25 July 2014 / Revised: 2 July 2015 / Accepted: 10 July 2015

Abstract In the past 10 years, the research community has produced a significant number of design notations to represent security properties and concepts in a design artifact. These notations are aimed at documenting and analyzing security in a software design model. The fragmentation of the research space, however, has resulted in a complex tangle of different techniques. Hence, practitioners are confronted with the challenging task of scouting the right approach from a multitude of proposals. Similarly, it is hard for researchers to keep track of the synergies among the existing notations, in order to identify the existing opportunities for original contributions. This paper presents a systematic literature review that inventorizes the existing notations and provides an in-depth, comparative analysis for each.

Keywords Security · Notation · Software design · Empirical study

Communicated by Prof. Alexander Pretschner.

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund KU Leuven.

✉ Alexander van den Berghe
alexander.vandenberghe@cs.kuleuven.be

Riccardo Scandariato
riccardo.scandariato@chalmers.se

Koen Yskout
koen.yskout@cs.kuleuven.be

Wouter Joosen
wouter.joosen@cs.kuleuven.be

[1] iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium

[2] Software Engineering Division, Chalmers and Göteborg University, 41756 Göteborg, Sweden

1 Introduction

Dealing with security right from the start in a software development project has the potential of avoiding higher costs related to fixing security flaws later on. Ideally, security concerns should be tackled as early as the software design phase. This is the central tenet of security by design, which is advocated by both industry and academia.

To ensure that the security considerations of a design do not remain in the head of the designer, thereby evaporating over time, and also that analysis and verification tasks can be performed at the design level, the security aspects need to be explicitly represented within the design of the software system. This implies that a designer needs to have access to a concrete technique that supports the recording of security-relevant information in relation to the design. This paper focuses on approaches that enable this, by offering support for explicitly representing and/or analyzing security concepts and properties in a software design model, which we will henceforth refer to as *notations* for security. For example, an approach that includes a graphical convention for the designer with the purpose of indicating which parts of the software system are subject to access control is considered as a notation. Clearly, notations are only one aspect of security by design. We acknowledge that, besides notations, design methodologies are equally important, as well as design support tools, such as security patterns and the like. However, these are outside the scope of our work.

To elaborate on this point a bit further, this paper considers secure design notations separate from any methodology that they possibly belong to. A secure design methodology supports the process of converting security requirements into a design that satisfies these requirements, and can range from a set of guidelines to a predefined sequence of steps. In contrast, a notation for security (as defined above) offers a way

Published online: 06 August 2015

to explicitly represent or analyze the security aspects of a design. As mentioned before, a designer eventually needs such a concrete way of representing security at the design level, with the purpose of documentation, analysis, and verification, irrespective of the methodology that is followed (if any).

A methodology can either rely on one or more notations to achieve its goals, or it can be defined without specific security notations in mind. This allows the designers to apply the methodology using any notation that they see fit. The question remains whether a notation, when proposed as part of a methodology, can be considered separately from this methodology. We argue that we can separate both for the purpose of this paper, because we only focus on the security-specific aspects that currently can (or cannot) be represented in a design, rather than the process of incorporating them into it. Of course, some notations will be more adequate for use with one methodology than with another. Also, note well that we by no means intend to suggest that using a suitable notation will lead to a design secure by itself. Indeed, a notation does not take away from the careful reasoning that the designer must perform whenever designing a secure system. Nevertheless, a good notation can support the designer in expressing the outcome of this process in a natural manner (using what could be called a domain-specific language for security) and serve as input for an analysis process.

Within this context of design notations for security, the research community has produced a sizeable number of approaches in the past ten years that are aimed at documenting security concepts and analyzing security properties in a software design model. The fragmentation of the research space, however, has resulted in a complex tangle of different techniques, which might have hindered the emergence of a leading approach that could be taken up and promoted by the industry. Hence, practitioners are confronted with the challenging task of scouting the right approach from a multitude of proposals. Similarly, it is hard for researchers to keep track of the synergies among the existing approaches, in order to identify the existing opportunities for original contributions.

We define three research objectives to mitigate these problems. First, we want to create an inventory of the existing notations and characterize each notation from the perspective of its applicability. For instance, we want to assess whether tool support is provided and whether the notation is specific to a given design style (e.g., service-oriented architectures, rather than data-centric systems). Second, as there are multiple facets to security, such as confidentiality, integrity, auditability, and so on, we want to create a map of which notation is suitable for each of the aforementioned security concerns. Third, we want to assess to what degree the notations (and associated techniques) are validated in realistic conditions, which is a measure of the reliability of an approach.

This paper achieves our objectives via a systematic literature review (particularly, a mapping study) that we have performed to disentangle the domain of notations for secure software design. We followed the methodology of Kitchenham et al. [41]. This means that the selection of the papers being included in this study has been dictated by a precise, repeatable process. As a result, we ended up analyzing 42 papers, corresponding to a total of 28 notations. This paper is accompanied by a web site containing all the "raw" data used to compile the information presented here, together with additional tables and graphs that might be of value for the interested reader [71].

The results of this empirical study show that most notations are very specialized with respect to the security concerns they address, which is possibly a reason for the fragmentation of the research space mentioned above. The study also shows that several security concerns (such as auditability and availability) are poorly covered. This is even more the case when it comes to analyzing a design with respect to those concerns. Further, most design notations lack a credible validation. These results are of interest to both researchers and practitioners. Practitioners can use this overview as a shopping guide to find the most suitable technique. Researchers can leverage this study to determine where the interesting opportunities for original contribution lie.

The remainder of this paper is structured as follows. Section 2 summarizes the research questions addressed by this study, and Sect. 3 describes the research methodology. Sections 4–6 present our findings, which are discussed in Sect. 7. Section 8 outlines the limitations of this study. Finally, Sect. 9 discusses the related work, and Sect. 10 presents the concluding remarks.

2 Research questions

The goal of this study is to inventorize and characterize the existing notations and associated techniques in the field of secure software design. This is achieved by means of a systematic analysis of the relevant literature. As mentioned before, the study addresses three research questions.

RQ1: Applicability. What is the practical applicability of existing notations for secure software design? The paper starts by building a general overview of the state of the art. In this study, software design refers to both architectural and detailed design. The notations are then characterized in terms of their scope (e.g., do they specifically apply to service-oriented architectures?) and their tool support. Clearly, these

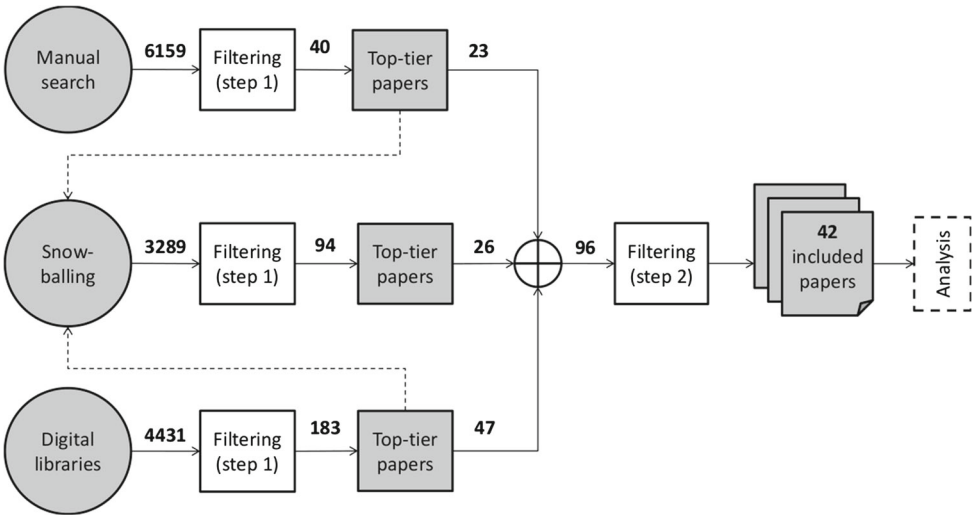


Fig. 1 This study employs three search methods to collect the relevant papers

aspects are valuable to practitioners that are scouting for a notation to use in their projects.

RQ2: Coverage. What security concerns are addressed by each notation? The purpose of the second research question is to link the notations and associated techniques to specific security concerns. Such concerns are, for example, confidentiality, integrity, and availability. The full set of concerns used in this study is discussed later on. Practitioners can select useful notations based on the security concerns that matter the most in their application domain or design project at hand. Researchers can determine gaps in the state of the art by assessing which security concerns are not (or rarely) considered.

Furthermore, notations might have been devised for different purposes. For instance, a notation can focus more on documenting security decisions in a software design for the sake of tracking such decisions (e.g., to facilitate the maintenance of the software system). Alternatively, the focus could lie on analyzing software designs in order to verify whether they satisfy the security requirements (e.g., that unauthorized users are not able to gain access to confidential data). Practitioners could use this information to further refine the set of potential notations. For example, practitioners who require to be able to verify the design (e.g., because of compliance requirements) would discard notations that do not support formal analysis.

RQ3: Validation. What evidence is provided that the notations works? The third and final research question provides an assessment of how each notation is validated and evaluated. The answer to this question is primarily relevant to practitioners, so that they can better judge the maturity of a notation and its applicability to real-world cases. However, the answer to this question also matters to researchers, as they can discover opportunities in the area of empirical research.

3 Methodology

In order to answer the above questions, the authors have analyzed a large corpus of research papers. We limited the time frame to a decade, namely from January 2002 until December 2012. We choose to begin from the year 2002 as this was the point when top-tier publications concerning secure design notations started to emerge. Notations described in papers published before this period have either been developed further (thus being included through later papers) or have become outdated for current technology.

3.1 Collection of the papers

The papers have been collected according to three complementary search methods in order to achieve the maximum coverage of the domain. The search strategy is summarized in Fig. 1. First, we performed a manual search of the proceedings of several conferences which are particularly relevant to the domain of software security (i.e., dealing with security in the software artifacts) and software engineering. In particular, we have looked into the past ten editions of the conferences mentioned in Table 1. We also manually searched the issues of relevant journals that have been published over the past ten years, which are also mentioned in Table 1. These sources were selected based on the experience of the authors and are not meant to be exhaustive, yet still representative. In case a source is not considered by the manual search, we count on the other two search methods to retrieve relevant papers from it.

Second, we searched through the digital libraries mentioned in Table 1 using the search string of Listing 1. Note that Google Scholar was not used because it imposes a maximum length of 256 characters on the search string, which is not sufficient in our case. However, Google Scholar is still

Table 1 Sources for the collection of research papers

Source	Relevant papers
Conferences	
European Conference on Object-Oriented Programming (ECOOP)	0
European Conference on Software Architecture (ECSA)	1
Symposium on Engineering Secure Software and Systems (ESSoS)	4
International Conference on Software Engineering (ICSE)	7
International Symposium on Architecting Critical Systems (ISARCS)	1
Model Driven Engineering Languages and Systems (MODELS)	11
Working IEEE/IFIP Conference on Software Architecture (WICSA)	3
Total	**27**
Journals	
Journal of Systems and Software (JSS)	3
Software and Systems Modeling (SoSyM)	6
ACM Transactions on Software Engineering and Methodology (TOSEM)	1
IEEE Transactions on Software Engineering (TSE)	3
Total	**13**
Digital libraries	
ACM Digital Library (https://dl.acm.org)	119
CiteSeerX (https://citeseerx.ist.psu.edu)	32
Compendex (https://www.engineeringvillage.com)	28
IEEE Xplore (http://ieeexplore.ieee.org)	82
ISI Web of Science (https://apps.webofknowledge.com)	122
Springer Link (http://link.springer.com)	12
Total (without duplicate results)	**183**

instrumental to the search strategy, as described later (e.g., to find citations). The search string has been initially constructed using terms selected from a set of relevant papers. It has been fine-tuned in order to reduce the number of irrelevant papers using the top 100 results returned by the ACM Digital Library and CiteSeerX. The search string contains four parts. The first part (line 1) defines two mandatory terms, while the second (line 2) delimits the domain of the results to software design. The third part (line 3) lists all considered security concerns. The fourth part (line 4) delimits the paradigm in which the results could be situated.

Listing 1: Search string used to retrieve papers from digital libraries.

```
1 (secur* AND model*) AND
2 ((software AND (develop* OR design* OR engine* OR architect* OR
      specification)) OR (system* AND (develop* OR design* OR
      engine* OR architect* OR specification)) OR (architect* AND
      (develop* OR design* OR engine* OR specification)) OR (
      application* AND (develop* OR design* OR engine* OR
      architect* OR specification))) AND
3 (authentication OR privacy OR (access AND control) OR
      confidentiality OR secrecy OR integrity OR avail* OR account
      * OR audit* OR log* OR authorization OR (threat AND model*)
      OR (attack AND model*) OR (intrusion AND detect*) OR (
      intrusion AND model*) OR (information AND flow) OR
      encryption) AND
4 (model-driven OR MDD OR MDS OR MDA OR aspect-oriented OR AOD OR
      AOSD OR AOM OR (unified AND modeling AND language) OR UML*
      OR (modeling AND language))
```

Third, we performed the so-called snowballing on included papers. For each paper that is part of the study, snowballing consist of retrieving the papers that are cited by the considered paper (forward snowballing) and the papers that cite the considered paper as a reference (backward snowballing). For the forward snowballing, we used the bibliography sections of the papers that are part of this study. Google Scholar was used to perform the backward snowballing.

The papers collected through the different search methods overlap with each other. A majority of 54 papers (out of 96) was found by more than one search method. Of the 42 papers that were only found by a single method, 4 papers were obtained from manual search, 12 came from digital libraries, and 26 papers were obtained by snowballing. When only considering the final set of 42 included papers, a large majority (30 papers) was found through multiple methods. The digital libraries and snowballing each account for 6 of the remaining papers. These numbers show that, while significant overlap exists between the results from each of the search methods, it is advisable to use a combination of methods in order to ensure a broad coverage of the available literature.

Filtering irrelevant papers. The considered conference proceedings and journal volumes, as well as the results obtained via the digital libraries and the snowballing, contain many papers that are not relevant to this study (false positives) and need to be filtered out.

As shown in Fig. 1, relevance is assessed in two steps. In the first step (left-hand side of figure), the papers are browsed and only the title, abstract, and keywords are considered. In case of doubt, the conclusion section of the papers is also consulted. Based on this information, a decision is made on whether to conditionally include the paper in the study. The papers that pass this initial selection step are then read fully in order to reassess their relevance in a second filtering step (right-hand side of figure), and a final decision is made about the inclusion of each paper in this study.

When assessing the relevance to this study of a given paper (in both steps), crisp criteria are needed in order to make the

process objective and repeatable. We have defined both positive and negative criteria. Negative criteria, when met, yield to the exclusion of the considered paper. These exclusion criteria are:

- The paper only mentions security as a general introductory term or
- is available only as extended abstract, poster, or presentation (no full version)
- is a duplicate of another relevant paper or
- is superseded by another relevant paper.

If no exclusion criteria hold, the paper is assessed vis-a-vis a list of positive criteria for inclusion. The inclusion criteria are:

- The paper represents security concerns in software design or
- analyzes security concerns in software design or
- models attacks or threats in software design or
- validates a notation described in another relevant paper.

If no inclusion criteria are met, the paper is definitively discarded. Note that in case of duplicate papers only the most recent or most extensive version is included.

As this study is mainly performed by one researcher (with the support of two more for quality control), due to time and resource constraints, the authors had to reduce the number of papers that needed to be reassessed. Therefore, to make the study feasible, the authors introduced an additional filtering step (see the gray boxes in the center of Fig. 1). That is, we excluded from the study those papers that are not published at top-tier venues. In particular, we considered conferences that are rated as "A" or higher and journals that are rated "B" or higher by the Excellence in Research for Australia (ERA) initiative [1]. We assume that most interesting and promising notations wind up being published in top-tier venues eventually.

As mentioned in Fig. 1, the collection phase of this study resulted in the final inclusion of 42 papers.

3.2 Analysis of the papers

For each paper, we start by gathering the type of publication (e.g., conference vs journal) and the venue of publication. The paper is then read carefully and classified according to a multidimensional taxonomy, as described in the remainder of this section.

Applicability. First, we characterize the applicability of each notation (see top of Fig. 2). In particular, we categorize the scope of each notation as being either generic or specific to a technology (like service-oriented architectures), design concern (like data storage), and so on. Furthermore, we assess whether tool support is described in the paper and whether this tool support is publicly available. Clearly, the presence of a tool is of utter importance for the industrial application of the notation. However, a publicly available open-source tool could be of interest for researchers as well. Finally, we collect the citation count (using Google Scholar) as an initial approximation to measure each notation's popularity within the research community.

Support for security. For each included paper, we list the security concerns that are addressed by the research work. As shown in the middle-left part of Fig. 2, the security concerns are divided in declarative properties and operational mechanisms. Declarative properties are the well-known CIAA properties (confidentiality, integrity, availability, and auditability), whereas operational mechanisms are the mechanisms used to implement the declarative properties.

Purpose. For a given security concern, a paper can just propose a way to represent such security concern in a design model, or also include a technique to verify the model vis-a-vis the concern (see the middle-right part of Fig. 2). In this study, a representation is every explicit modeling notation, either graphical or textual. The representation category is further divided into documentation and construction. Documentation means that the representation is only used as a mean to record certain security-related design decisions, without serving any other explicit purpose during the development. Construction means that the representation is actively used. For example, an annotation about confidentiality could be used to generate glue code to incorporate a cryptographic library in the implementation. This is often the case in model-driven engineering.

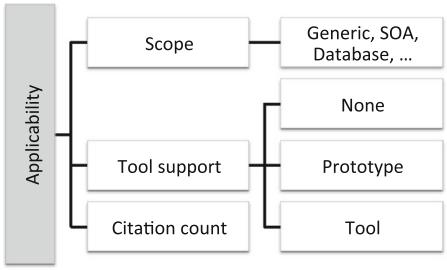
Analysis is the verification activity that leverages a representation in order to check whether a design satisfies the security requirements. The presence of some support for analysis could be important for practitioners that have compliance requirements. Analysis can be algorithmic, by which we mean that the analysis procedure is described algorithmically. It can be performed automatically, or by a human. Conversely, we define an expertise-based analysis as one that relies on implicit knowledge, requiring human reasoning. The first type could be less expensive from a management perspective, provided that good tool support exists. Neither of the defined analysis types considers the scope, in terms of addressed security concerns, of a design notation.

Usage. For each paper, practical usage information is also collected (see the middle-right part of Fig. 2). This comprises the types of models that are involved in the notation, as well as the notation mechanism that is used. When selecting a notation, practitioners often have constraints related to their own

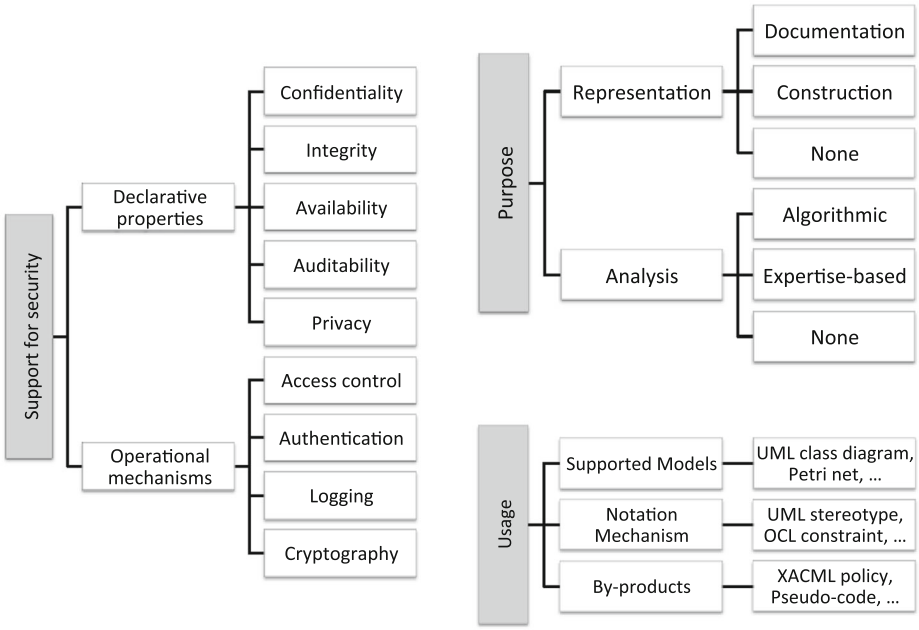
[1] http://www.arc.gov.au/era/era_2010/archive/era_journal_list.htm.

Fig. 2 The papers are categorized according to the depicted taxonomy

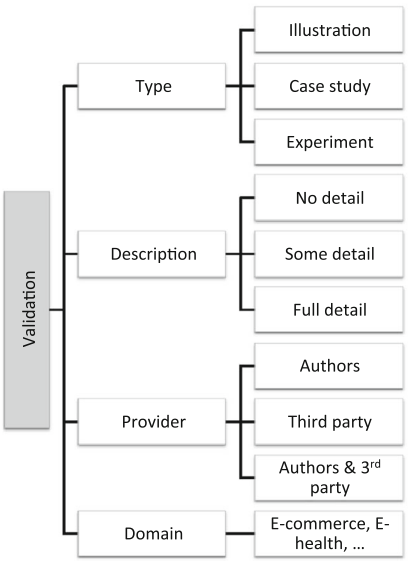
RQ1 (Section 4)



RQ2 (Section 5)



RQ3 (Section 6)



development environment. For instance, someone working on a project that already uses UML as the main modeling language might more likely be interested in a notation that deals with security concerns in a homogeneous way, i.e., using the same language. We also record whether any by-product artifacts are generated, such as security polices expressed in a domain-specific language. These artifacts could be of use later on in the software development process.

Validation. As shown at the bottom of Fig. 2, the notation presented in a paper can have been evaluated in different ways and with different degrees of rigor. In an illustration, a notation is applied to a small example as a demonstration of how the notation works. In a case study, a notation is applied to an industrial-size project. In an experiment, a notation is used under controlled conditions and its performance is evaluated empirically. Such experiments allow to assess different aspects of a design notation. For example, an experiment can assess whether applying the proposed notation improves the quality of the design, i.e., leads to a more secure design, compared to other notations. Experiments also provide the opportunity to asses the usability of a design notation in a controlled environment. In any case, the paper can provide different degrees of details with respect to the evaluation. The full detail category means that a thorough description of the evaluation procedure is provided, and an analysis of the results that states the observed benefits and limitations of the notation, the lessons learned, and so on. This type of description is obviously more valuable. An evaluation without any details only mentions/claims the existence of the evaluation without further description. The cases in between are categorized as "some detail."

The provider category refers to who performed the evaluation. This can be the authors of the notation themselves, a third party or a combination of both. Clearly, a third party evaluation adds credibility to the notation. Finally, we track the type of application domain (e.g., e-health) in which the evaluation is situated. This information might be instrumental to practitioners that are inclined to adopt the notation and are developing a system in the same application domain.

3.3 Quality control

The data analysis in this study is mostly performed by a single researcher. To ensure correctness and objectivity of the results, two quality control activities have been put into place.

Calibration via a pilot study. Prior to the study, the above-mentioned researcher performed the analysis of an initial set of 12 papers. Another senior researcher has repeated the analysis independently. The inconsistencies have been discussed extensively in order to clear any ambiguities and correct issues in the protocol of the study (for instance, the list of security concerns has been updated). This guaranteed a smoother execution of the rest of study, i.e., when the full set of papers has been considered.

Random quality checks. During the execution of the study, we have selected a sample containing about 15 % of the papers (out of the 96 mentioned in Fig. 1). This sample consisted of papers for which the decision to include or exclude it was difficult, supplemented with randomly selected papers for which inclusion was reasonably straightforward. These papers have been reassessed independently by another researcher. The second researcher has evaluated the relevance of the papers using the inclusion/exclusion criteria and also performed the analysis itself.

Concerning the relevance of papers, the researchers disagreed on 3 papers (which belong to the difficult cases added to the sample), caused by subtle differences in opinion about the fulfillment of the inclusion criteria. These discrepancies were resolved through an in-depth discussion on why the paper is relevant or not, and (when necessary) a refinement of the inclusion/exclusion criteria for the rest of the evaluation.

A comparison of the analysis results of the relevant papers showed very few discrepancies (i.e., <5 % of the collected data), which were not systematic (i.e., they were spread over the various categories of the taxonomy). The discrepancies mostly originated from a different interpretation of a sentence in the paper and were also resolved through a discussion between the two researchers. At all times, a third researcher would join to allow a majority vote in case the two researchers could not come to an agreement, but this was never necessary.

4 Applicability of the notations (RQ1)

Figure 3 plots the number of publications included in this study for each year of the 10-year window we selected as our time frame. It seems like the research interest for the topic of design notations for security has been rather constant over

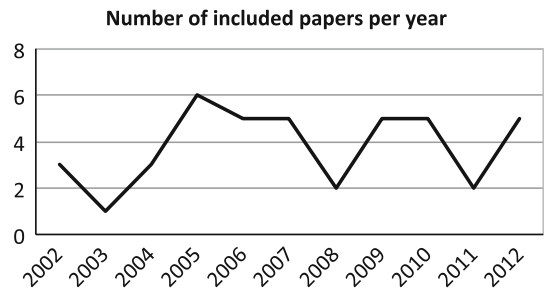


Fig. 3 Over time, there has been a constant stream of top-tier publications

time, with an average of about 4 top-tier publications per year. The 42 included papers describe a total of *28 design notations*, which are listed in Table 2. Concerning the naming convention for the notations, when the original authors do not explicitly mention a name, we defined one by combining the name of the main author with a suffix. In this paper, we provide an analysis of the design notations by aggregating related papers. The interested reader can find the analysis of each individual paper online [71]. The companion website also contains additional tables and graphs that have not been included in this manuscript.

The large majority of the notations (17) are generic and do not specialize in a specific application domain (see Table 2). Among the domain-specific notations, service-oriented architecture (SOA) is quite popular (6 notations). Possibly, this architectural style calls for a different approach to security than conventional distributed software development. Existing standards such as WS-Trust [65] and WS-SecurityPolicy [64] are likely perceived as to low level from a designer perspective. This could justify the need for a domain-specific, higher-level notation.

The 3 notations focussing on data security discuss the general importance of information security with respect to databases. However, no further reasons are provided concerning the specific security requirements that need to be addressed in a specialized way at design level.

Finally, 2 notations are very specialized. AMF focusses on authorization systems, while ADM-RBAC extends a method for modeling hypermedia and web systems (the method is defined by the same authors in previous work).

Tool support. UMLsec and SecureUML provide a more mature tool support which is publicly available for download. Only 2 other design notations (Hoisl-SOA and Memon-SECTET) offer the same level of tool support. A striking total of 12 notations offer no support at all. The remaining 12 notations mention the existence of some kind of prototype, which, however, is not released. Due to this unbalanced situation, we have not investigated further on the matter of tool support (e.g., licenses, documentation, and so on). We remark that the general lack of tool support is a testimonial of a deficiency with respect to maturity in

Table 2 This study analyzes 28 design notations for secure software

				Scope					Tool support	
Notation	**Papers**	**Cit.**	**Cross-ref.**	Generic	SOA	Database	AuthZ systems	Web systems	Tool	Prototype
ADM-RBAC	[16]	4	0					•		•
Ahn-AC	[3]	9	0	•						
Alam-SECTET	[4]	26	1		•					•
AMF	[26]	3	0				•			•
Buyens-LP	[9]	1	0	•						•
FDAF	[10,12]	22	0	•						•
Georg-AO	[20,19]	140	4	•						
Giordano-AC	[21]	2	0	•						•
Gomaa-UML	[22]	12	1	•						
Hafner-SOA	[23]	31	1		•					
Hoisl-SOA	[25]	4	0		•				•	
Kim-AC	[40]	3	0	•						•
Kong-Threat	[44]	6	0	•						
Mariscal-AC	[57]	14	2	•						•
Medina-DB	[17,18,67]	109	5			•				•
Memon-SECTET	[48]	0	0		•				•	
Nakamura-SOA	[51,60]	84	3		•					
PbSD	[1,2]	0	0			•				•
Ray-AC	[58]	80	3	•						
SecureSOA	[49,50]	42	1		•					•
SecureUML	[7,6]	399	15	•					•	
Sohr-AC	[62]	36	0	•						•
UML AC	[42,43]	143	3	•						
UMLS	[24]	18	1	•						
UMLsec	[30,31,33] [8,35,34]	853	24	•					•	
Vela-DB-XML	[69]	3	0			•				
Xu-Petri	[72]	83	0	•						
Yu-AC	[73]	13	0	•						
			Total	17	6	3	1	1	4	12

the field of secure software design. In any case, there is a potential business (or applied research) opportunity in this area.

Popularity. Citation count is a possible metric to provide an indication concerning how popular the existing design notations are within the research community. We count the citations for each notation (third column in Table 2) as the sum of the citation counts of each paper included for this notation. The shown citation counts thus constitute an upper bound. UMLsec and SecureUML are in the command position, while Georg-AO, UML AC, and Medina-DB fall somewhat behind.

Furthermore, limiting the citation count to cross-references among the 42 analyzed papers (fourth column in Table 2), we observe UMLsec and SecureUML taking a clear lead. These two notations seem to serve as baseline comparisons with other notations, whereas Georg-AO, UML AC, and Medina-DB barely distinguish themselves anymore among the other notations.

Clearly, the citation count is influenced by age. The 2 most referential notations mentioned before were among the first to introduce security notations into software design. UMLsec was introduced in 2001, and SecureUML was first published in 2002. However, being early alone seems insufficient to become popular. In fact, Georg-AO, UML AC, and Medina-DB were also published in 2001–2002, but their citation count is not as high.

Interestingly enough, the two most referenced notations have a different level of specialization. UMLsec is broadly applicable and covers a variety of security concerns. SecureUML is more specific and geared toward role-based access control only. It seems like 'popularity' favors these two extremes rather than notations in the middle ground.

Furthermore, an obvious commonality between the most referenced notations is that they are all based on UML. Using a widespread modeling language such as UML is likely to increase usability and hence adoption.

A final important quality concerning the popularity of notations is their continued development, i.e., publications with new additions. All the top notations have been extended throughout the 10-year window considered by this study, either by including support for additional security concerns or by providing further validation.

5 Coverage of security concerns (RQ2)

Table 3 maps the identified notations to the security concerns that they support both declarative properties and operational mechanisms (see Sect. 3.2). The reader can appreciate that the number of design notations addressing each concern varies strongly. With 20 design notations, access control is by far the most supported security concern. A possible explanation for its popularity is the fact that a number of relevant standards, for example, role-based access control (RBAC, [59]), are widespread and well-described in the literature. These standards provide a solid guideline for incorporating access control into design. Incidentally, most of the notations addressing access control do not cover any other security concern, i.e., access control is mostly considered in isolation. Integrity and cryptography are distant second in terms of coverage (7 notations each). Notably, availability and privacy are not considered at all in the investigated literature.

Table 3 Summary of the security concerns addressed by each design notation

	Declarative					Operational				
Notation	Confidentiality	Integrity	Availability	Auditability	Privacy	Access Control	Authentication	Logging	Cryptography	Total
ADM-RBAC						•				1
Ahn-AC						•				1
Alam-SECTET						•				1
AMF						•				1
Buyens-LP						•				1
FDAF						•			•	2
Georg-AO							•			1
Giordano-AC						•				1
Gomaa-UML						•	•		•	3
Hafner-SOA	•	•		•						3
Hoisl-SOA	•	•							•	3
Kim-AC						•				1
Kong-Threat		•								1
Mariscal-AC						•				1
Medina-DB						•		•		2
Memon-SECTET				•					•	2
Nakamura-SOA		•					•		•	3
PbSD						•				1
Ray-AC						•				1
SecureSOA	•	•					•		•	4
SecureUML						•				1
Sohr-AC						•				1
UML AC						•				1
UMLS						•				1
UMLsec	•	•				•	•		•	5
Vela-DB-XML						•		•		2
Xu-Petri		•								1
Yu-AC						•				1
Total	4	7	0	2	0	20	5	2	7	

'•' means the notation offers some kind of support for the concern

Table 3 also shows that a majority of 18 notations consider only a single security concern, with 15 of these focusing exclusively on access control. In general, most notations specialize in a small number of security concerns. Only one notation (UMLsec) is more complete in terms of coverage, having five (out of nine) security concerns covered. This specialization makes it difficult to single out one notation to be used in larger and more complex design projects, which typically entail several security requirements of diverse nature.

Among the notations considering multiple concerns, only 3 are domain independent (UMLsec, Gomaa-UML and FDAF). Developing a design notation that is both generic

in scope and broad in coverage seems to be a challenge that only few are willing to tackle.

Looking at the declarative properties and operational mechanisms separately, the first observation is that 20 notations only consider operational mechanisms. It is worth noting that only 3 notations (Hafner-SOA, Kong-Threat, and Xu-Petri) work solely with declarative properties. Thus, considering declarative properties alone is rare. This may be due to declarative properties being rather closely related to requirements, whereas operational mechanisms are more solution-oriented, and easier to integrate in a software design. However, declarative properties are important because they allow to trace the rationale behind operational mechanisms adopted in a design. This rationale is often instrumental in preserving the integrity of the design and to avoid decay over time. Finally, it is remarkable that of the 5 notations that support both declarative properties and operational mechanisms (Hoisl-SOA, Memon-SECTET, Nakamura-SOA, SecureSOA, and UMLsec), 4 are oriented toward service-oriented architectures. It could be that there is a larger need to declaratively specify security concerns in service-oriented designs, or perhaps that it is easier to do so because of the rather high abstraction level of such designs.

5.1 Purpose of the design notations

We have further investigated the purpose of each design notation (i.e., the type of support provided) with respect to the security concerns it covers. A notation can provide just a way to represent the security concern in a design or can also provide the means to perform some form of analysis. These two aspects are further detailed in Sects. 5.2 and 5.3, respectively.

First, we start with some general observations, based on Tables 4 (representation) and 5 (analysis), which provide a more detailed view on the information from Table 3. Figure 4 graphically summarizes these two tables in order to make the gaps in the coverage more visible.

Except for availability and privacy, each of the security concerns in the taxonomy of Fig. 2 (middle-left part) can be represented by at least two notations. However, only five of the considered concerns can be analyzed, so in general, analysis is not as well supported as representation. A total of 17 notations provide support for representation only, 8 notations provide support for both representation and analysis, and 3 notations provide support for analysis only. These last three cases (Buyens-LP, Kong-Threat, and Xu-Petri) perform their analysis on top of existing representations and design models, without extending them for the sake of security. That is, these approaches use an implicit representation for the security concerns. Buyens-LP extracts the intended access policy from the documentation and architecture, whereas Kong-Threat and Xu-Petri require the designer to model the intended behavior using sequence diagrams and petri nets, respectively.

Table 4 Representation support of the different design notation for each security concern

Representation	Declarative					Operational				
Notation	Confidentiality	Integrity	Availability	Auditability	Privacy	Access Control	Authentication	Logging	Cryptography	Total
ADM-RBAC						C				1
Ahn-AC						D				1
Alam-SECTET						C				1
AMF						C				1
Buyens-LP										0
FDAF						D			D	2
Georg-AO							D			1
Giordano-AC						C				1
Gomaa-UML						D	D		D	3
Hafner-SOA	D	D		D						3
Hoisl-SOA	C	C							C	3
Kim-AC						D				1
Kong-Threat										0
Mariscal-AC						C				1
Medina-DB						C		C		2
Memon-SECTET				C					C	2
Nakamura-SOA		C					C		C	3
PbSD						C				1
Ray-AC						D				1
SecureSOA	D	D					D		D	4
SecureUML						C				1
Sohr-AC						D				1
UML AC						D				1
UMLS						C				1
UMLsec	C	C				C	C		C	5
Vela-DB-XML						C		C		2
Xu-Petri										0
Yu-AC						D				1
Total	4	5	0	2	0	19	5	2	7	

Possible values are Documentation (D) or Construction (C). An empty cell means no support

We also note that analysis is found almost exclusively among notations specializing in a single security concern, with UMLsec being the only exception. Finally, among the 11 notations providing support for analysis, only 4 notations (Georg-AO, Kong-Threat, UMLsec, and Xu-Petri) consider security concerns beyond access control.

5.2 Representation support

Table 6 summarizes the representation put forward by each notation and indicates the design models that are involved. Among the modeling languages used to represent the security concerns, UML [53,54] is by far the most common language (23 notations). Furthermore, Hoisl-SOA also uses SoaML [56] and UML4SOA [47], both of which are UML extensions and thus increase UML's dominance even further. A likely reason for UML's popularity is that it offers an extensive modeling language for software development and provides several extension mechanisms such as profiles and metamodel extension. Using such an existing, extensible language allows the authors of a notation to focus on the new

Table 5 Analysis support of the different design notations for each security concern

Analysis	Declarative					Operational				
Notation	Confidentiality	Integrity	Availability	Auditability	Privacy	Access Control	Authentication	Logging	Cryptography	Total
ADM-RBAC										0
Ahn-AC										0
Alam-SECTET										0
AMF						A				1
Buyens-LP						A				1
FDAF										0
Georg-AO							E			1
Giordano-AC										0
Gomaa-UML										0
Hafner-SOA										0
Hoisl-SOA										0
Kim-AC										0
Kong-Threat		E								1
Mariscal-AC										0
Medina-DB										0
Memon-SECTET										0
Nakamura-SOA										0
PbSD						A				1
Ray-AC										0
SecureSOA										0
SecureUML						A				1
Sohr-AC						A				1
UML AC						E				1
UMLS										0
UMLsec	A	A					A		A	4
Vela-DB-XML										0
Xu-Petri		E								1
Yu-AC						E				1
Total	1	3	0	0	0	7	2	0	1	

Possible values are Algorithmic (A) or Expertise-based (E). An empty cell means no support

Table 6 The representations provided by the notations are largely based on UML

	UML-based						
Notation	Class diagram	Sequence diagram	Activity diagram	Collaboration diagram	Other UML diagram(s)	Non-UML diagrams	**Mechanism**
ADM-RBAC						•	AC policy modeled separately
Ahn-AC	•			•	•		AC policy modeled separately + OCL constraints
Alam-SECTET	•						Separate model + SECTET-PL rules
AMF	•						AC policy modeled separately + OCL constraints
Buyens-LP	(no explicit representation)						
FDAF	•						Aspect represents security concern
Georg-AO	•	•		•			Aspect (static and dynamic view) models
Giordano-AC						•	AC policy defined using visual models
Gomaa-UML	•			•			Security service classes
Hafner-SOA	•		•				OCL-like expression added to elements
Hoisl-SOA			•			•	Secure Pins, Stereotypes and Notes
Kim-AC	•	•					RBAC artifacts modeled using aspect model
Kong-Threat	(no explicit representation)						
Mariscal-AC	•						Policy defined in separate models (Role-slice, User, delegation diagrams)
Medina-DB	•						AC policy and logging defined using tagged values and OCL constraints
Memon-SECTET	•		•				Stereotypes + SECTET-PL rules
Nakamura-SOA	•				•		Stereotype + attributes
PbSD	•						Association as a class + stereotypes + OCL constraints
Ray-AC	•			•			RBAC artifacts modeled using aspect model
SecureSOA	•					•	UML classes with stereotypes
SecureUML	•				•		Association as a class + Stereotypes + OCL constraints
Sohr-AC	•				•		Separate model + OCL constraints
UML AC	•				•		UML Class and Object diagrams model policy and rules
UMLS	•	•	•	•			Labels
UMLsec	•	•	•		•		Stereotypes + Tagged values
Vela-DB-XML	•						Stereotypes + Tagged values + Rules (modeled as UML classes)
Xu-Petri	(no explicit representation)						
Yu-AC	•	•					Policy in separate model + OCL constraints
Total	22	5	5	5	6	4	

elements they introduce for their representation, instead of having to define the syntax and semantics from the ground up.

Taking a closer look at the diagrams used, we observe that UML class diagrams are by far the most used by notations. In fact, only a single UML-based notation, Hoisl-SOA, does not support class diagrams. UML sequence, activity, and collab-

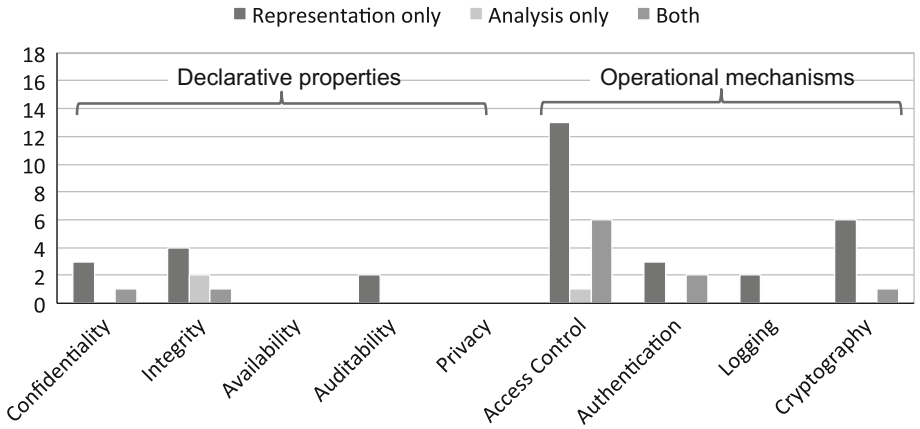


Fig. 4 Several gaps can be observed in the overall coverage of security concerns

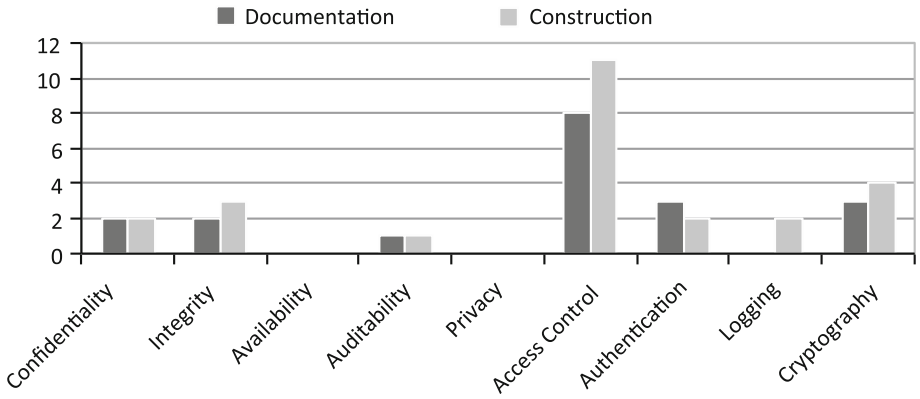


Fig. 5 The dominance of construction design notations is largely due to access control

oration diagrams follow at considerable distance, each being used by only 5 notations. Other UML diagrams, e.g., deployment and object diagrams, are rarely used. This difference in popularity indicates two trends. First, the notations focus on the structural aspects of security concerns. Second, the notations are more oriented toward the detailed design phase than the architectural design phase.

Documentation versus construction. Before delving deeper into the actual representations used by the notations, we first briefly look into the type of representation. Recall from Sect. 3.2 that the representation can be used just to document the security concerns, or as a starting point to construct other artifacts. A first observation from Table 4 is that 14 notations use their representation constructively, while the representation of 11 notations serves as documentation only. Recall that 3 design notations do not propose any custom representation at all. In a model-driven setting, support for construction is often expected, which may explain why more notations focus on this aspect. Nevertheless, the numbers for the two types do not lie far from each other.

A closer look at how the types of representation are distributed over the security concerns (Fig. 5) indicates that the observed difference is largely due to access control. A representation for access control properties indeed allows for a straightforward use in further development phases. One possibility is to generate a run-time access control policy from it. For example, Giordano-AC generates XACML policies, and both Hoisl-SOA and Nakamura-SOA generate WS-SecurityPolicy specifications for web services. Furthermore, this can be complemented with enforcement logic (e.g., SecureUML can be used to generate skeleton implementations for the EJB and .NET platforms). Another option is the creation of test cases that verify the implementation of the policy, which is done by AMF and UMLsec, for example.

Finally, note that all design notations that use the representation for constructive purposes do so for one or more operational mechanisms. This is not surprising, given that the generation of an artifact requires sufficiently concrete information. The operational mechanisms capture such information better than the declarative properties, which specify requirements rather than solutions. Nevertheless, 4 notations (Hoisl-SOA, Memon-SECTET, Nakamura-SOA, and UMLsec) consider also declarative properties for artifact generation.

UML-based notations. To discuss the large number of design notations based on UML, we divide them into four broad types, namely notations that (1) use standard UML elements; (2) use UML extension mechanisms; (3) combine UML with other languages; or (4) use UML in an aspect-oriented way. The following paragraphs discuss each of these types in more detail.

Four notations (Ahn-AC, AMF, Sohr-AC, and Yu-AC) use standard UML class diagrams to model (role-based) access control policies, augmented with OCL to specify constraints such as separation of duty. Ahn-AC also employs collaboration diagrams to model dynamic aspects of access control such as checking permissions. It is worth noting that standard UML is only used by notations focusing on access control.

The most common way in which UML is used, with a total of 12 notations, is extending it with security-specific elements. The manner in which these notations extend UML varies strongly, though. On one hand, there are lightweight extensions that define new stereotypes and tagged values, for example, using a UML profile. On the other hand, there are heavyweight extensions in which the UML metamodel itself is extended with new elements, or new diagram types are defined.

An example of a lightweight extension is Medina-DB, which adds tagged values to classes in order to indicate the security levels and roles of the annotated classes. Supplemented with OCL constraints, these tagged values define an access control policy. Similarly, UMLsec defines an extensive UML profile containing various stereotypes and tagged values. For example, a class that contains confidential information can be stereotyped as "critical," and the "secrecy" tag is used to specify the confidential data itself. Gomaa-UML,

Nakamura-SOA, and PbSD make similar use of stereotypes to model their security concerns.

UML AC goes a bit further than the above notations, by not only adding stereotypes but also giving a new interpretation to object diagrams. The authors use object diagrams to graphically model both policy rules and constraints. A policy rule for assigning a role to a subject is modeled as an object diagram containing two objects, which represent the subject and the role. The objects are connected by a stereotyped link.

UMLS proposes a heavyweight extension, by adding so-called labels to, for example, standard UML classes and attributes, in order to model access control information such as the owner of an attribute, and who is allowed to read it. Another example is Vela-DB-XML, which adds, among others, new types of classes to model the rules that are used for modeling access control and logging policies.

A number of notations define new diagram types, based on existing UML elements. For example, Mariscal-AC defines four new diagrams, mostly using class diagram elements, for modeling access control. The secure subsystem diagram models the public interface that is subject to access control. The role-slice diagram models the role hierarchy and specifies, for each role, the allowed and disallowed operations. The user diagram models the assignment of users to roles, and the delegation diagram models how users may delegate their roles.

Rather than just extending UML itself, some design notations combine UML with one or more other languages. Hoisl-SOA, for example, combines UML, SoaML, and UML4SOA, in order to represent security concerns for SOA applications at multiple levels during the development. First, the metamodel for UML activity diagrams is extended to represent security requirements at the business level, by adding new elements (e.g., *SecurePin* and *SecureNode*) and new semantics (using OCL constraints). Furthermore, SoaML and UML4SOA are extended to model security at the service level. Hence, Hoisl-SOA extends the different modeling languages in a complementary manner in order to represent the security concerns.

SecureUML, in contrast, merges its security design language with the design language that is used to design the system into a so-called dialect language. UML is used as the concrete language for the security concerns, and the authors only demonstrate this approach using design languages which are subsets of UML (in particular, ComponentUML and ControllerUML). This leaves the feasibility of merging with design languages that are not based on UML as an open question.

SecureSOA uses the SecureUML dialect mechanism to merge Fundamental Modeling Concepts (FMC) [37], more specifically FMC block diagrams, as a design language with their security design language. The FMC metamodel is extended with SecureSOA-specific concepts, where UML classes and stereotypes are used as concrete syntax for the security concerns of these concepts.

Finally, four notations use UML in an aspect-oriented way. FDAF represents a security concern as an aspect, whose different pieces must be woven into the design at hand. The aspect encapsulates both the attributes and the operations that are required for the security concern. For example, the role-based access control aspect [12] defines, among others, a `CreateRoleSession` and a `CheckRolesForActions` operation. The former must be woven in after each login operation, while the latter must be attached to the execution of every user request.

The three other aspectual notations use one or more models to define their security aspects. Georg-AO defines authentication protocols using UML class and sequence diagrams. Kim-AC uses UML class and sequence diagrams to represent role-based access control aspects. Similarly, Ray-AC uses UML class and collaboration diagrams for this purpose. To use these notations, these aspect models must eventually be woven into the design model(s) of the system itself.

Non-UML-based notations. We have identified only two notations that do not use UML for their representation, namely ADM-RBAC and Giordano-AC. Both define a visual language for role-based access control. They introduce models for defining roles, with their mutual relations, and assigning permissions to roles. Furthermore, access control policies can be generated using both notations. ADM-RBAC supports the generation of access tables, whereas Giordano-AC generates XACML policies.

Despite their similarities, both notations have different reasons for defining their own modeling language. ADM-RBAC is an extension of the Ariadne Development Method (ADM), a web system development methodology by the same authors [15]. In contrast, Giordano-AC explicitly aims to be usable by all different kinds of users, from developers to top-level enterprise managers. To achieve this aim, the authors decided to construct a specially crafted visual language.

Interaction with design models. How the security-specific representations provided by a notation interact with the (often already existing) model of the software system is another interesting point of view besides the used modeling language. By abstracting the notation mechanisms given in Table 6, we identified four different interaction types that are used by the notations (see Table 7).

First, annotations can be used to add security information to the existing system model elements. For example, UMLsec defines stereotypes and tagged values to annotate several UML elements with security requirements. In total, 11 notations take this route.

Table 7 A large majority of design notations either annotate system design models or provide separate models

Notation	Annotation	Separate element	Aspect	Separate model
ADM-RBAC				•
Ahn-AC				•
Alam-SECTET				•
AMF				•
Buyens-LP				
FDAF			•	
Georg-AO			•	
Giordano-AC				•
Gomaa-UML		•		
Hafner-SOA	•			
Hoisl-SOA	•			
Kim-AC			•	
Kong-Threat				
Mariscal-AC				•
Medina-DB	•			
Memon-SECTET	•			
Nakamura-SOA	•			
PbSD	•			
Ray-AC			•	
SecureSOA	•			
SecureUML	•			
Sohr-AC				•
UML AC				•
UMLS	•			
UMLsec	•			
Vela-DB-XML	•			
Xu-Petri				
Yu-AC				•
Total	11	1	4	9

Second, separate elements that represent the security concerns can be added to the existing models. These elements differ from annotations in that they also have a meaning on their own. Gomaa-UML introduces separate service classes encapsulating security concerns. For example, an `AccessControlAgent` in [22] offers operations to verify whether a user is authorized to use certain services. The authors explicitly choose to clearly separate security from the rest of the application in order to reduce the complexity of the design and to allow the reuse of services across different applications. It is noteworthy that no other notation takes this route.

Third, aspects are used to model security concerns independently from the specific application context. The aspects are then instantiated by weaving them into a given design. For example, Georg-AO models authentication using a UML class diagram and a sequence diagram. In order to provide authentication in a software system, these two models are woven into their corresponding system design models. The weaving is a mostly manual process in which the designer defines the mapping of the elements in the aspect and design models. Next to Georg-AO, three other notations (FDAF, Kim-AC, and Ray-AC) follow a similar route.

Fourth, separate models can represent security concerns orthogonal from the main design. The difference with the above aspect-based notations is that separate models are not incorporated into the software design models. For example, Giordano-AC models access policies in separate models orthogonal to the software being designed. Thus, the policy and design models are developed and maintained independently from each other. Interestingly, all 9 notations using separate models consider only access control, indicating that for other concerns it may not be feasible to model them orthogonally to the rest of the design.

Table 8 A summary of the analysis techniques used by each notation

Notation	Analysis technique
AMF	UML + OCL and Alloy
Buyens-LP	Set theory
Georg-AO	OCL + Alloy
Kong-Threat	Graph grammar
PbSD	UML + OCL
SecureUML	OCL
Sohr-AC	UML + OCL
UML AC	Graph
UMLsec	Model checking + Theorem proving
Xu-Petri	Petri net
Yu-AC	UML

5.3 Security analysis support

As mentioned before, only 11 design notations support the analysis of security concerns. As shown in Table 8, most

notations combine different types of techniques for analysis. A recurring technique is the use of OCL constraints (Object Constraint Language, [55]) which are applied to UML diagrams and verified via constraint solvers.

Algorithmic analysis. From Table 5, it follows that 6 notations offer an algorithmic analysis (according to the definition of algorithmic given in Sect. 3.2). It is remarkable that, except for UMLsec, these notations are all limited to access control. This is likely due to the fact that requirements concerning access policies are rather clear and unambiguous, thereby allowing easier (algorithmic) analysis.

UMLsec bases its analysis on the notion of an adversary with certain knowledge and behavior. It utilizes several formal methods, such as theorem provers and model checkers, to perform the actual analysis. For example, a UMLsec design, together with the adversary model, can be translated to Promela, allowing the Spin model checker to automatically verify whether the adversary can successfully violate a security requirement [35].

Among the five notations supporting an algorithmic analysis for access control, Buyens-LP is the only one not relying on OCL. Buyens-LP allows to analyze a software architecture for least privilege and separation of duty violations by deriving a so-called Task Execution Model (TEM) from the architecture and its documentation. The TEM identifies the relations between principals and the tasks the system at hand can perform (i.e., it represents the policy defined in the architecture). The analysis consists of verifying whether the TEM is consistent with the intended policy as defined by the requirements. Note that the analysis is agnostic to how the intended policy is described, i.e., the actual representation of the intended policy is not part of the notation.

The remaining four notations supporting an algorithmic analysis (AMF, Sohr-AC, PbSD, and SecureUML) use OCL constraints to analyze access control policies, although they differ in how they use OCL. AMF and Sohr-AC both generate possible system states and verify these against OCL constraints that specify, for example, separation of duty constraints. It is worth noting that AMF also uses Alloy to verify whether the designed policy conforms to the formal specification by generating test cases.

PbSD, on the other hand, works with templates of common security patterns (e.g., RBAC), which are modeled in UML and are augmented with template OCL constraints. The patterns are instantiated into application models, and PbSD verifies whether the pattern application conforms to the multiplicities, cardinalities, and language specified by the pattern template. Furthermore, the OCL constraints are verified within the application in which the security pattern is instantiated.

SecureUML uses OCL to formalize parts of the models to be analyzed, as well as the access control properties to be verified. The required semantics of the used dialect (the combination of SecureUML and a design language) are expressed using OCL. The properties to be verified are consequently expressed as OCL queries which are verified against snapshots (i.e., run-time instances) of the dialect metamodel or policy model. The provided machinery allows an automatic analysis of the modeled RBAC policies.

Expertise-based analysis. Some of the notations supporting an expertise-based analysis (Georg-AO, Kong-Threat, and Xu-Petri) require the designer to manually define the sequence of steps in an attack and to relate these steps to the elements of the design. In particular, Georg-AO represents possible attacks as aspects using UML class and sequence diagrams. Each generic attack must be instantiated in the context of the software under development. Therefore, the designer must have in-depth knowledge of the workings of the attack in question, and know how it relates to the software at hand. Kong-Threat and Xu-Petri require the modeling of threats (which must be gathered through risk analysis) using UML sequence diagrams and petri nets, respectively. Again, extensive knowledge on how a threat can exploit the software being designed is required.

An alternative analysis approach is followed for Yu-AC, where scenarios for verifying the defined access control policy are generated from so-called operation invocation patterns. These patterns are manually defined by a designer and constrain the initial state and allowed sequence of operation invocations. It is up to the designer to select and define the most interesting patterns (i.e., those that are most likely to show faults in the policy).

Finally, UML AC analyzes an access control policy by verifying whether all reachable states satisfy the access control constraints. These constraints are derived from the access control model (e.g., a role must have at least one associated permission for RBAC) and the application domain (e.g., only treating physicians may access a patient file in a medical application). The actual analysis consists of a designer verifying whether the policy rules, which control the reachable states, can violate any of the constraints. If a violating policy rule is found, this rule must be refined in order to mitigate the violation.

6 Validation of the notations (RQ3)

Table 9 summarizes the type of validation that is performed for each design notation. In Fig. 6, the data from Table 9 are combined with the information about the level of detail provided in the papers describing the validation.

Almost all notations are illustrated by means of one or more examples. However, these illustrations are generally lacking in detail as far as their documentation is concerned.

Table 9 The notations are mostly validated through (poorly documented) illustrative examples. Table shows for each notation the number of validations per validation technique

Notation	Illustrations	Case studies	Experiments	Total
ADM-RBAC	1		1	2
Ahn-AC				0
Alam-SECTET	1			1
AMF	1			1
Buyens-LP	5			5
FDAF	2			2
Georg-AO	2			2
Giordano-AC		1	1	2
Gomaa-UML	1			1
Hafner-SOA	1			1
Hoisl-SOA	2			2
Kim-AC	2			2
Kong-Threat	1			1
Mariscal-AC	1			1
Medina-DB	2	1		3
Memon-SECTET	1			1
Nakamura-SOA	2			2
PbSD	2		1	3
Ray-AC	1			1
SecureSOA	2			2
SecureUML	2			2
Sohr-AC				0
UML AC	2			2
UMLS	1			1
UMLsec	6	7		13
Vela-DB-XML	1			1
Xu-Petri	1			1
Yu-AC	1			1
Total	**44**	**9**	**3**	**56**

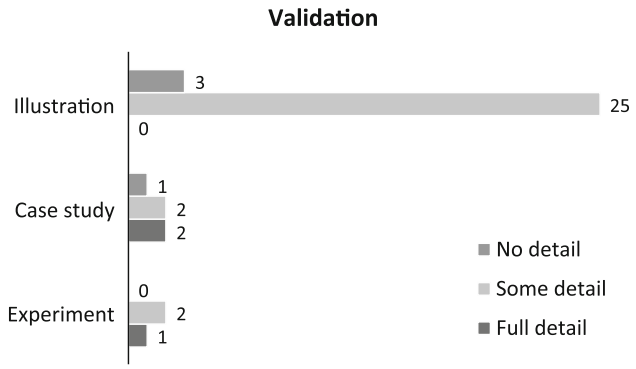


Fig. 6 Most notations are lacking a credible validation

This could pose some issues when it comes to learning a given notation. However, the most striking observation is that, in most cases, the validation does not go beyond the illustrative examples that are used to demonstrate the notation. Such illustrations have their use to initially show the applicability of a notation or highlight particular strengths of a technique, but they are typically small and, to some extent, crafted to showcase the strengths of the notation at hand. Very few notations are validated by means of a controlled experiment or a case study (in both cases, 3 notations). Concerning the case studies, only two are documented in full detail, meaning that they provide an in-depth analysis of the application of the design notation. Case studies are an interesting form of evaluation as they put the notations to the test in realistic industrial conditions. However, we acknowledge that case studies are costly in terms of time and effort. Furthermore, it is not always possible to carry out a case study because it can be challenging to find an interested industrial partner for a novel technique for instance.

Next to the limited quality of the validation, we also observed limitations as far as quantity is concerned. The majority of the notations only provide one or two evaluations (12 and 10 notations, respectively). In some cases, no evaluation is provided at all (see Ahn-AC and Sohr-AC in Table 9). Given that we only consider papers published in top-tier venues, this was a surprising discovery.

A notable exception is UMLsec, which provides a total of 13 evaluations (see Table 9). Two fully detailed case studies describe the analysis of a single-sign-on-mechanism of a search engine at a car manufacturer [8] and a mobile system architecture at a telecommunication company [34]. These case studies are performed by industry partners in collaboration with the authors. Incidentally, these are the only cases we saw of third party validation.

Also Medina-DB reports an in-depth case study performed by the authors, concerning the redesign of an accounting database for a government agency [17].

PbSD provides a well-documented experiment [2]. The authors compare their notation to specify access control policies to directly specifying the policies using SQL and Oracle's VPD. This experiment was conducted with 148 students as participants divided into two groups. The authors evaluated, among others, the quality of the resulting access control policies and the time required to construct them.

ADM-RBAC [16] and Giordano-AC [21] report similar types of experiments where the authors compare their notation to an existing one, respectively, NIST RBAC and XGrid. These experiments lack an in-depth analysis and are performed on a smaller scale.

7 Discussion

To summarize the findings of our study, we identified four major gaps in the state of the art concerning design notations for secure software. First, not all security concerns are equally supported by the existing notations. Access control is considered by a large number of notations, whereas the other concerns are more sparingly addressed. In particular, availability and privacy are completely neglected. Looking at the possibilities for verifying a design against its security requirements, we observed that just five out of nine security concerns can currently be analyzed. This could represent a problem when it comes to more critical systems, which require a higher level of assurance. Second, a large majority of the notations are highly specialized and address only a single security concern. Third, most notations are evaluated using only illustrations. A limited amount of case studies are documented in the literature, but they mostly belong to one specific notation. Fourth, the available tool support is generally limited. We found that only four notations have a tool that is publicly available for download.

The landscape depicted by this systematic study outlines that the "secure by design" goal could be very difficult to attain in practice, given the technological support available today. If you were the head software designer of a medium/large development project, you would be faced with an impassable obstacle when selecting a suitable secure design technique for your team to work with. As an example, take two security requirements such as the authorization of users and the auditability of user actions (which are only sketched here, for the sake of staying general). These requirements are very common in most of the realistic systems, as observed during the authors' own experience of collaboration with industrial partners. When designing a system that encompasses these security concerns, a designer would not find any notation in our inventory that could be used to document the design choices and analyze the resulting design model vis-a-vis the security requirements. Beyond the specific example, this would be the case for many other groups of security requirements. In summary, unless you were concerned with a restricted set of specific security problems, finding an overarching secure design notation would not be possible.

The alternative would be to adopt a collection of synergetic techniques. As many notations in our inventory are based on UML, this task appears to be feasible. However, as clearly visible in Table 6, this would require the designers to deal with a mix of OCL specification, stereotypes, aspects, and domain-specific declarative languages. The learning curve would be rather steep as considerable expertise would be required. Further, the coherence of the resulting artifacts would be lost, which might negatively affect the integrity and soundness of the design over time. In essence, the combination of the existing specialized notations is far from straightforward.

A charter for future research. Despite a track record of over ten years of efforts, the research community has yet quite some way to go in the field of design notations for secure software. The results of this study can be used to sketch a guideline for future research. First, there is a need for a comprehensive, yet elegant notation that is able to cover the many concerns of security. Such notation should play the role of a "one-stop shop" for secure design. From the practitioners' perspective, it is more effective to invest in learning an approach that winds up being used routinely for all the secure design tasks faced in the field. Still, it could be useful to have a modular notation that allows to incrementally extend the amount of notational concepts that are used, depending on the complexity of the design task at hand (e.g., as soon as more security concerns need to be considered). This way, the endeavor of mastering the notation could be tackled incrementally. Similarly, it is advisable to keep the notation close to the current practice of software design, which is largely dominated by the use of UML. Further, the notation should be applicable to the mostly used design diagrams (including class and sequence diagrams), but also stretch to more abstract design artifacts such as the software architecture and business processes.

The notation should take care to support the possibility to analyze its security concerns. This aspect is undervalued by the current notations. Especially a tool-supported algorithmic analysis can provide significant gains concerning the security of resulting systems. Such an analysis can provide strong guarantees concerning the satisfaction of the security requirements by a system. The tool support should shield developers from requiring any specific expert knowledge, e.g., formal methods, minimizing the learning curve

and allowing a straightforward incorporation into the used development process.

To achieve this aim of a thorough notation, two inroads can be taken. First, one could attempt to merge and extend the best of the existing work. The challenge, in this case, is to assemble a coherent and elegant notation, which is far from trivial. Alternatively, one could start from scratch, although this would be a major undertaking. In both cases, having a real impact on the practice of software design should be the main goal. Therefore, it is advisable to include practitioners in the loop and join forces in a collaborative effort of the research community (eventually, via a standardization consortium). A suggested way to achieve impact is to investigate the real design needs of practitioners. This has been neglected so far and could be achieved, for instance, by setting up surveys with practitioners.

Finally, the community should take the responsibility of thoroughly validating future proposals via large case studies and multiple controlled experiments.

8 Limitations of the study

As any empirical study, this work also suffers from some threats to validity. First, the restriction to top-tier journals and conferences resulted in discarding a considerable number of papers. This restriction could potentially lead to biased conclusions. However, we are confident that the final set of 42 papers constitutes a representative subset of all relevant literature concerning design notations for secure software. Further, since we selected the top-tier papers, the set of analyzed articles contains the most influential research work in the field.

Second, since most of the work in this study is performed by a single researcher, there is a risk of subjectivity in the selection of relevant papers as well as in their analysis. To mitigate this risk, we put in place an elaborate strategy for quality control (see Sect. 3.3), which, however, does not rule out entirely the possibility of human errors.

Third, the list of security concerns used in this study (see Fig. 2) is somewhat coarse-grained. For example, integrity can be divided further in data and transmission integrity, and so on. Similarly, privacy is also a complex concern that can be further decomposed [14]. However, most referential security taxonomies are similar to ours [5,63].

Finally, there are many aspects that we did not investigate due to the lack of time and resources. For instance, we did not look into the usability of the tool support. Further, we did not assess the learnability of the design notations, e.g., by considering whether the notations provide sufficient documentation. These and other aspects are interesting areas of future work.

9 Related work

Over the years, various studies comparable to ours have been performed. We have divided them into three categories: systematic reviews, surveys, and comparative studies. We will discuss the work in each of these categories and compare it to the study presented here.

Systematic reviews The systematic review category only includes studies that follow the guidelines by Kitchenham and Charters [41]. Therefore, these studies are closest to ours. We have identified two other systematic reviews in the area of secure software design.

Firstly, Nguyen et al. [52] study model-driven security with a focus on the model-driven aspects such as the usage of model-to-model and model-to-text transformations by the approaches. The study analyzes a total of 80 papers covering UMLsec, SecureUML, and Medina-DB (named Secure data warehouses). The authors conclude that most approaches support authorization, especially access control. Furthermore, the authors observe a lack of thorough semantic foundations and in-depth evaluation of approaches.

Secondly, Jensen and Jaatun [29] also study security in model-driven development. Due to their focus on code generation, the scope of their study is narrower than ours. Also, their objective is to provide only a rough overview. The review includes a total of 30 papers, overlapping with four notations from our study, namely SecureUML, Hafner-SOA, Alam-SECTET, and Medina-DB. The authors limit themselves to a qualitative reflection on the state of the art, without an explicit comparison of the included approaches. Their main conclusion is that there is a need for more empirical studies and standardization.

Surveys The survey category also contains studies that provide an overview of the domain, possibly including comparisons of the different notations. The major difference with the systematic review category is that a survey is not performed in a systematic manner, which makes them more difficult to repeat or update.

Uzunov et al. [68] survey a total of 31 security methodologies for distributed systems, including UMLsec (named MBSE/UMLsec), SecureUML (named Model-Driven Security), Alam-SECTET and Memon-SECTET (combined as SECTET), Georg-AO and Ray-AC (combined as AORDD), SecureSOA and Gomaa-UML. The authors define security methodologies *"as systematic approaches combining security and modern software engineering"*, resulting in a broader scope than our survey. First, the approaches are classified along four classes: model-, architecture-, pattern-, or agent-driven methodologies. Furthermore, each approach is described along different dimensions such as specificity, security concerns, and supported software development life

cycle (SDLC) stages. These descriptions are supplemented with a qualitative discussion of the strengths and weaknesses of the approaches. Second, the authors evaluate the more mature methodologies, 18 in total, for industry adoption according to 12 criteria. Based on these two parts of the survey, the authors conclude that no ideal security methodology tailored to industry needs exists. Although each approach has its merits, more effort is required to advance the state of the art as state of the practice. The authors list a number of future directions such as extending the range of security solutions and encapsulating both security solutions and related knowledge into catalogs.

Dai and Cooper [11] survey 11 approaches, including SecureUML, Gomaa-UML (named Separating Modelling of Application and Security Concerns), UMLsec (named UML/Theorem Prover Approach), and the authors' own FDAF. The authors first evaluate the strengths and limitations of each approach individually. Second, the authors shortly compare the approaches based on the supported security concerns, the modeling language used, the supported analysis, and the available evaluation. They conclude that the wide variety of modeling notations indicates the need for more thoroughly investigating which are suitable for modeling and analyzing security concerns. Furthermore, they remark that additional evaluation of the approaches is required.

Lúcio et al. [45] provide a detailed survey of five model-driven security approaches, including UMLsec, SecureUML, and SECTET, using a taxonomy based on that used in [36, 39,52]. The authors focus on model-driven aspects such as model transformations. A main conclusion is that a better level of maturity should be reached. The authors state that this requires both building tool support and provide more systematic industrial experimentation.

Talhi et al. [66] survey 17 UML-based approaches (including UMLsec, Mariscal-AC, SecureUML, and Ray-AC) focussing on the usability of UML as security specification language. The authors group the approaches in five different groups based on how they use and/or extend UML. The groups are stereotypes and tagged values, OCL, behavior diagrams, extending the UML metalanguage and defining a new metalanguage. These groups are evaluated for the expressiveness, tool support, verifiability, and complexity. The authors conclude that stereotypes are most usable for security specification, whereas OCL is not suited. Furthermore, the authors conclude that creating a new metalanguage is preferable to extending the UML metalanguage due to the fact that the latter is too constraining.

Dehlinger and Subramanian [13] survey aspect-oriented approaches for designing and implementing secure software. The authors review only three approaches of which one, Georg-AO, is also included in our study. The approaches are evaluated using the standards of Shah and Hill [61]. The authors conclude that neither approach measures up to the standard, and thus further development and evaluation of the approaches are required before they are usable in an industrial setting. Furthermore, the authors discuss other aspect-oriented approaches such as secure coding outside the scope of our review.

Jayaram and Mathur [28] aim to provide a representative overview of approaches throughout the entire software development life cycle, without the intention of being complete. Focussing mainly on approaches for requirements engineering and specification, however, their scope is complementary to ours. The authors consider, amongst others, SecureUML and UML AC for modeling access control during requirements engineering and specification. Furthermore, they discuss using UMLsec for the analysis and design of secure software. They conclude that more research is necessary concerning the use of formal methods in software security, which ideally leads to the removal of all manual intervention in every phase of the development life cycle. Furthermore, the authors conclude more research is required to better integrate security requirements with system requirements and code generation.

Villarroel et al. [70] survey and compare 11 manually selected methodologies including UMLsec, Medina-DB, and Georg-AO. To compare the methodologies, the authors use the comparison framework of Khwaja and Urban [39]. Because this framework lacks security-specific criteria, the comparison handles security in a rudimentary way. The authors conclude that there is a need for a standardized methodology which considers security aspects from the earliest development stages. Furthermore this methodology should be an extension of already existing development methodologies and standards, in order to increase its chance of adoption.

Kasal et al. [36] survey the position of security in model-driven development. The authors aim to find which approaches are applicable to which development problems and what specific features characterize each technique. The authors compare six approaches, including UMLsec, using a framework inspired by that of Khwaja and Urban [39]. The framework compares, among others, the formality, availability of tool support, and security mechanisms considered by each approach. The authors conclude that no approaches for analyzing implementations of modeled systems exist and that each aspect-oriented approach considers only a single security mechanism. On the contrary, our study includes approaches that were not included by Kasal et al. [36], but that do support implementation analysis (e.g., UMLS) and aspects for multiple security mechanisms (e.g., FDAF). This illustrates the advantage of using a broader overview such as ours as the foundation for more detailed comparisons.

Khan and Zulkernine [38] survey secure software development mostly from the requirements engineering perspective.

Within this scope, the authors discuss 11 security specification languages, including SecureUML and UMLsec, and five security requirements engineering processes. The specification languages are compared based on six desired qualities, partially inspired by the mandatory requirements for security specification languages outlined by Jürjens [30, chap.4]. Furthermore, the authors note that most languages for specifying security requirements can also be used for specifying the design, because the low-level requirements modeled by the requirements specification languages are very close to design solutions. We agree with this point, and remark that we have considered both SecureUML and UMLsec to be design notations in our study. The authors conclude that there is a need to further develop security requirements and design specification languages, although little guidance is given concerning the goals that these specification languages should achieve.

Jürjens [32] shortly reports on a number of model-based development approaches in the field of security and dependability engineering. The approaches include Alam-SECTET, Medina-DB, Ray-AC, SecureUML, UML AC, UMLS, and UMLsec. The author concludes with four open problems. First, security requirements should be traceable throughout the different phases of the system life cycle. Second, the verification of legacy implementations against high-level security concerns is barely tackled. Third, little is known concerning the preservation of security concerns when applying techniques such as modular composition or decomposition of system parts. Fourth, the interaction between security and other nonfunctional requirements requires more attention.

Finally, Hussain et al. [27] provide a rough overview of the state of the art, divided into four categories: model-driven methodologies, methodologies having automated tool support, methodologies having no tool support, and methodologies based on formal methods. These categories contain a total of 24 methodologies, including SecureUML and UMLsec. No more details concerning the coverage in terms of security concerns are given, however. The development of a methodology based on a lightweight application of formal methods is presented as a direction for further research.

Comparative studies To the best of our knowledge, the only detailed comparison of two specific notations is performed in the study by Matulevičius and Dumas [46]. They compare SecureUML and UMLsec with respect to their applicability to role-based access control. For the comparison, the authors use a meeting scheduler system example. The intention of this comparison is to help practitioners choose the most suitable approach given their business constraints. The conclusion of the comparison is that both SecureUML and UMLsec are applicable for modeling RBAC policies and that they complement each other by providing different viewpoints.

In contrast to such a comparative study, we aim to provide a broader overview of the existing notations, capturing the most important characteristics of the various notations, without performing a detailed comparison (on a single case study, for example). As such, our study is complementary to comparative studies, as it can provide a starting point to perform more in-depth comparisons of the identified notations.

Summary While multiple surveys concerning notations for secure software design exist, our study is only the third systematic literature review in this research domain, each of which has a different focus. In contrast to the existing systematic reviews (which focus on generating code from design artifacts), we focus on the actual representation and analysis of security concerns in software design. This difference in focus yields limited overlap in surveyed papers by us and other studies.

Nevertheless, the major findings of all studies (including ours) are aligned. There exists a wide variety of design notations, but these notations are scattered, and as such, there is an opportunity for integration. Also, most of the existing notations lack a thorough evaluation. Finally, better guidance is needed on which notations to use in which concrete situation, which can be obtained from in-depth comparative studies.

10 Conclusion

This paper has reported the results of a literature review we conducted in the field of design notations for secure software. We have systematically scanned the published, peer-reviewed literature and studied an extensive set of 28 notations. This study plays the role of a field map for a complex landscape, which is populated by numerous design techniques. From a practical perspective, software architects and design teams could use this work as a "buying guide" when investigating which technique (or techniques) they could use. However, the authors have the longer-term ambition to stimulate the research community to reorientate its research efforts and join forces for the creation of a more structured and thorough design notation for secure software systems.

Acknowledgments This research is partially funded by the Research Fund KU Leuven and by the EU FP7 project NESSoS, with the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE).

References

1. Abramov, J., Anson, O., Dahan, M., Shoval, P., Sturm, A.: A methodology for integrating access control policies within database development. Comput. Secur. **31**(3), 299–314 (2012)
2. Abramov, J., Sturm, A., Shoval, P.: Evaluation of the pattern-based method for secure development (PbSD): a controlled experiment. Inf. Softw. Technol. **54**(9), 1029–1043 (2012)

3. Ahn, G.-J., Hong, S.-P., Shin, M.E.: Reconstructing a formal security model. Inf. Softw. Technol. **44**(11), 649–657 (2002)
4. Alam, M., Breu, R., Hafner, M.: Model-driven security engineering for trust management in SECTET. J. Softw. **2**(1), 47–59 (2007)
5. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. **1**(1), 11–33 (2004)
6. Basin, D., Clavel, M., Doser, J., Egea, M.: Automated analysis of security-design models. Inf. Softw. Technol. **51**(5), 815–831 (2009)
7. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: from UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. **15**(1), 39–91 (2006)
8. Best, B., Jürjens, J., Nuseibeh, B.: Model-Based Security Engineering of Distributed Information Systems Using UMLsec. In: Proceedings of the 29th International Conference on Software Engineering, ICSE '07, pp. 581–590. Washington, DC, USA (2007). IEEE Computer Society
9. Buyens, K., Scandariato, R., Joosen, W.: Least privilege analysis in software architectures. Softw. Syst. Model. **12**(2), 1–18 (2011)
10. Dai, L., Cooper, K.: Modeling and performance analysis for security aspects. Sci. Comput. Program. **61**(1), 58–71 (2006)
11. Dai, L., Cooper, K.: A survey of modeling and analysis approaches for architecting secure software systems. Int. J. Netw. Secur. **5**(2), 187–198 (2007)
12. Dai, L., Cooper, K.: Using FDAF to bridge the gap between enterprise and software architectures for security. Sci. Comput. Program. **66**(1), 87–102 (2007)
13. Dehlinger, J., Subramanian, N.: Architecting Secure Software Systems Using an Aspect-Oriented Approach: A Survey of Current Research. In: Technical Report, Iowa State University (2006)
14. Deng, M., Wuyts, K., Scandariato, R., Preneel, B., Joosen, W.: A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. Requir. Eng. **16**(1), 187–198 (2012)
15. Díaz, P., Aedo, I., Montero, S.: Ariadne, a development method for hypermedia. In: Mayr, H.C., Lazansky, J., Quirchmayr, G., Vogel, P. (eds.) Database and Expert Systems Applications. Lecture Notes in Computer Science, vol. 2113, pp. 764–774. Springer, Berlin (2001)
16. Díaz, P., Aedo, I., Sanz, D., Malizia, A.: A Model-Driven Approach for the Visual Specification of Role-Based Access Control Policies in Web Systems. In: IEEE Symposium on Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. pp. 203–210 (2008)
17. Fernández-Medina, E., Piattini, M.: Designing secure databases. Inf. Softw. Technol. **47**(7), 463–477 (2005)
18. Fernández-Medina, E., Trujillo, J., Villarroel, R., Piattini, M.: Developing secure data warehouses with a UML extension. Inf. Syst. **32**(6), 826–856 (2007)
19. Georg, G., Ray, I., Anastasakis, K., Bordbar, B., Toahchoodee, M., Houmb, S.H.: An aspect-oriented methodology for designing secure applications. Inf. Softw. Technol. **51**(5), 846–864 (2009)
20. Georg, G., Ray, I., France, R.: Using Aspects to Design a Secure System. In: Proceedings of the Eighth International Conference on Engineering of Complex Computer Systems, ICECCS '02, p. 117. IEEE Computer Society, Washington (2002)
21. Giordano, M., Polese, G., Scanniello, G., Tortora, G.: A system for visual role-based policy modelling. J. Vis. Lang. Comput. **21**(1), 41–64 (2010)
22. Gomaa, H., Eonsuk Shin, M.: Modelling Complex Systems by Separating Application and Security Concerns. In: Proceedings of Ninth IEEE International Conference on Engineering Complex Computer Systems, pp. 19–28 (2004)
23. Hafner, M., Breu, M., Breu, R., Nowak, A.: Modelling Inter-organizational Workflow Security in a Peer-to-Peer Environment. In: Proceedings of 2005 IEEE International Conference on Web Services, 2005. ICWS 2005. p. 540 (2005)
24. Heldal, R., Hultin, F.: Bridging Model-Based and Language-Based Security. In: Snekkenes E., Gollmann D. (eds) Computer Security ESORICS 2003, volume 2808 of Lecture Notes in Computer Science, pp. 235–252. Springer, Berlin (2003). doi:10.1007/978-3-540-39650-5_14
25. Hoisl, B., Sobernig, S., Strembeck, M.: Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach. Softw. Syst. Model. **13**(2), 513–548 (2014). doi:10.1007/s10270-012-0263-y
26. Hu, H., Ahn, G.-J.: Constructing authorization systems using assurance management framework. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **40**(4), 396–405 (2010)
27. Hussain, S., Rasool, G., Atef, M., Shahid, A.K.: A review of approaches to model security into software systems. J. Basic Appl. Sci. Res. **3**(4), 642–647 (2013)
28. Jayaram, K.R., Mathur, A.P.: Software Engineering for Secure Software—State of the Art: A Survey. In: Technical Report CERIAS 2005-67, Purdue University (2005)
29. Jensen, J., Jaatun, M.G.: Security in Model Driven Development: A Survey. In: 2011 Sixth International Conference on Availability, Reliability and Security (ARES), pp. 704–709 (2011)
30. Jürjens, J.: Secure Systems Development with UML. Springer, Berlin (2004)
31. Jürjens, J.: Sound Methods and Effective Tools for Model-Based Security Engineering with UML. In: Proceedings of the 27th International Conference on Software Engineering, ICSE '05, pp. 322–331. ACM, New York (2005)
32. Jürjens, J.: Security and dependability engineering. In: Kokolakis, S., Gómez, A.M., Spanoudakis, G. (eds.) Security and Dependability for Ambient Intelligence, Volume 45 of Advances in Information Security, pp. 21–36. Springer, Berlin (2009)
33. Jürjens, J., Lehrhuber, M., Wimmel, G.: Model-Based Design and Analysis of Permission-Based Security. In: Proceedings of 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005. pp. 224–233 (2005)
34. Jürjens, J., Schreck, J., Bartmann, P.: Model-Based Security Analysis for Mobile Communications. In: Proceedings of the 30th International Conference on Software Engineering, ICSE '08, pp. 683–692. ACM, New York (2008)
35. Jürjens, J., Shabalin, P.: Tools for secure systems development with UML. Int. J. Softw. Tools Technol. Transf. **9**, 527–544 (2007)
36. Kasal, K., Heurix, J., Neubauer, T.: Model-Driven Development Meets Security: An Evaluation of Current Approaches. In: 2011 44th Hawaii International Conference on System Sciences (HICSS), pp. 1–9 (2011)
37. Keller, F., Wendt, S.: FMC: An approach towards architecture-centric system development. In: Proceedings of 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003, pp. 173–182. IEEE (2003)
38. Khan, M.U.A., Zulkernine, M.: A Survey on Requirements and Design Methods for Secure Software Development. In: Technical Report 2009-562, School of Computing, Queen's University, Kingston, Ontario, Canada (2009)
39. Khwaja, A.A., Urban, J.E.: A synthesis of evaluation criteria for software specifications and specification techniques. Int. J. Softw. Eng. Knowl. Eng. **12**(5), 581–599 (2002)
40. Kim, S., Kim, D.-K., Lu, L., Kim, S., Park, S.: A feature-based approach for modeling role-based access control systems. J. Syst. Softw. **84**(12), 2035–2052 (2011)
41. Kitchenham, B., Charters, S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering. In: Technical Report EBSE 2007-001, Keele University and Durham University Joint Report (2007)
42. Koch, M., Mancini, L.V., Parisi Presicce, F.: A graph-based formalism for RBAC. ACM Trans. Inf. Syst. Secur. **5**(3), 332–365 (2002)

43. Koch, M., Parisi-Presicce, F.: UML specification of access control policies and their formal verification. Softw. Syst. Model. **5**(4), 429–447 (2006)
44. Kong, J., Xu, D., Zeng, X.: UML-based modeling and analysis of security threats. Int. J. Softw. Eng. Knowl. Eng. **20**(6), 875–897 (2010)
45. Lúcio, L., Zhang, Q., Nguyen, P.-H., Amrani, M., Klein, J., Vangheluwe, H., Le Traon, Y.: Advances in Model-Driven Security. Adv. Comput. **93**, 103–152 (2013)
46. Matulevičius, R., Dumas, M.: A Comparison of SecureUML and UMLsec for Role-Based Access Control. In: Databases and Information Systems, pp. 171–185 (2010)
47. Mayer, P., Koch, N., Schroeder, A., Knapp, A.: The UML4SOA Profile. In: Technical report, LMU Muenchen (2010)
48. Memon, M., Menghwar, G., Depar, M., Jalbani, A., Mashwani, W.: Security modeling for service-oriented systems using security pattern refinement approach. Softw. Syst. Model. **13**(2), 549–572 (2014). doi:10.1007/s10270-012-0268-6
49. Menzel, M., Meinel, C.: A Security Meta-Model for Service-Oriented Architectures. In: IEEE International Conference on Services Computing, 2009. SCC '09. , pp. 251–259 (2009)
50. Menzel, M., Meinel, C.: SecureSOA Modelling Security Requirements for Service-Oriented Architectures. In: 2010 IEEE International Conference on Services Computing (SCC), pp. 146–153 (2010)
51. Nakamura, Y., Tatsubori, M., Imamura, T., Ono, K.: Model-Driven Security Based on a Web Services Security Architecture. In: 2005 IEEE International Conference on Services Computing, vol. 1, pp. 7–15 (2005)
52. Nguyen, P.-H., Klein, J., Le Traon, Y., Kramer, M.E.: A Systematic Review of Model-Driven Security. In: Software Engineering Conference (APSEC, 2013 20th Asia-Pacific), vol. 1, pp. 432–441 (2013)
53. OMG. OMG Unified Modeling Language (OMG UML), Infrastructure (2011). OMG. http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF
54. OMG. OMG Unified Modeling Language (OMG UML), Superstructure (2011). OMG. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF
55. OMG. OMG Object Constraint Language (OCL) (2012). OMG. http://www.omg.org/spec/OCL/2.3.1/PDF
56. OMG. Service Oriented architecture Modeling Language (SoaML) Specification (2012). OMG. http://www.omg.org/spec/SoaML/1.0.1/PDF
57. Pavlich-Mariscal, J.A., Demurjian, S.A., Michel, L.D.: A framework of composable access control features: preserving separation of access control concerns from models to code. Comput. Secur. **29**(3), 350–379 (2010)
58. Ray, I., France, R., Li, N., Georg, G.: An aspect-based approach to modeling access control concerns. Inf. Softw. Technol. **46**(9), 575–587 (2004)
59. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
60. Satoh, F., Nakamura, Y., Ono, K.: Adding Authentication to Model Driven Security. In: Proceedings of the IEEE International Conference on Web Services, ICWS '06, pp. 585–594. IEEE Computer Society, Washington (2006)
61. Shah, V., Hill, F.: An Aspect-Oriented Security Framework: Lessons Learned. In: AOSD Technology for Application-level Security (AOSDSEC) (2004)
62. Sohr, K., Ahn, G.-J., Gogolla, M., Migge, L.: Specification and Validation of Authorisation Constraints Using UML and OCL. In: de Capitani, S., di Vimercati, P., Syverson, D. Gollmann, (eds.) Computer Security ESORICS 2005. Lecture Notes in Computer Science, vol. 3679, pp. 64–79. Springer, Berlin Heidelberg (2005)
63. Standard. The Common Criteria: Security functional components. https://www.commoncriteriaportal.org (2012)
64. Standard. WS-SecurityPolicy v1.3. OASIS Standard incorporating Approved Errata. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/errata01/os/ws-securitypolicy-1.3-errata01-os-complete.html (2012)
65. Standard. WS-Trust 1.4. OASIS Standard Incorporating Approved Errata. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/os/ws-trust-1.4-errata01-os-complete.html (2012)
66. Talhi, C., Mouheb, D., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: Usability of security specification approaches for UML design: a survey. J. Object Technol. **8**(6), 103–122 (2009)
67. Trujillo, J., Soler, E., Fernández-Medina, E., Piattini, M.: An engineering process for developing secure data warehouses. Inf. Softw. Technol. **51**(6), 1033–1051 (2009)
68. Uzunov, A.V., Fernandez, E.B., Falkner, K.: Engineering security into distributed systems a survey of methodologies. J. Univ. Comput. Sci. **18**(20), 2920–3006 (2012)
69. Vela, B., Blanco, C., Fernández-Medina, E., Marcos, E.: A practical application of our MDD approach for modeling secure XML data warehouses. Decis. Support Syst. **52**(4), 899–925 (2012)
70. Villarroel, R., Fernández-Medina, E., Piattini, M.: Secure information systems development—a survey and comparison. Comput. Secur. **24**(4), 308–321 (2005)
71. Website. https://people.cs.kuleuven.be/alexander.vandenberghe/review/overview.html
72. Xu, D., Nygard, K.E.: Threat-driven modeling and verification of secure software using aspect-oriented petri nets. IEEE Trans. Softw. Eng. **32**(4), 265–278 (2006)
73. Yu, L., France, R., Ray, Indrakshi, Ghosh, S.: A Rigorous Approach to Uncovering Security Policy Violations in UML Designs. In: 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, pp. 126–135 (2009)

Alexander van den Berghe is a PhD student in the DistriNet research group of the department of Computer Science at KU Leuven, Belgium. His main research interests are in the area of secure software engineering, focussing on representing and analyzing security concerns in the early design phases.

Riccardo Scandariato received his PhD in Computer Science in 2004 from Politecnico di Torino. Since 2014, he is with the department of Computer Science and Engineering, which is shared between the Chalmers University of Technology and the University of Gothenburg. His main research interests are in the area of secure software engineering, with a particular focus on (i) empirical methods for security and (ii) security&privacy in software design. He has published over 50 papers in the area of security and software engineering. He is an Associate Editor of the International Journal of Secure Software Engineering (IJSSE) and a member of the Review Editorial Board of Frontiers in ICT. He regularly participates to the Program Committees of several top-rated conferences in the area of security and software engineering.

Koen Yskout is a postdoctoral researcher at the iMinds-DistriNet group of the Department of Computer Science at the KU Leuven in Belgium. He obtained a master degree in engineering (computer science) in 2005 and thereafter joined the iMinds-DistriNet group as a researcher. He obtained his PhD on the use of patterns for connecting software security requirements and architectural design. Since then, he continued with research focusing on empirical software engineering, secure software architecture and design, patterns, security requirements, and model-driven development. He has also been involved in several national and international research projects.

Wouter Joosen is full professor at the Department of Computer Science of the Katholieke Universiteit Leuven in Belgium, where he teaches courses on software architecture and component-based software engineering, distributed systems, and the engineering of secure service platforms. His research interests are in aspect-oriented software development, focusing on software architecture and middleware, and in security aspects of software, including security in component frameworks and security architectures.