Journal of Universal Computer Science, vol. 18, no. 20 (2012), 2920-3006
submitted: 24/2/12, accepted: 28/11/12, appeared: 1/12/12 © J.UCS

Engineering Security into Distributed Systems: A Survey of Methodologies

Anton V. Uzunov
(University of Adelaide, SA, Australia
anton.uzunov@adelaide.edu.au)

Eduardo B. Fernandez
(Florida Atlantic University, Boca Raton, FL, USA
ed@cse.fau.edu)

Katrina Falkner
(University of Adelaide, SA, Australia
katrina.falkner@adelaide.edu.au)

Abstract: Rapid technological advances in recent years have precipitated a general shift towards software distribution as a central computing paradigm. This has been accompanied by a corresponding increase in the dangers of security breaches, often causing security attributes to become an inhibiting factor for use and adoption. Despite the acknowledged importance of security, especially in the context of open and collaborative environments, there is a growing gap in the survey literature relating to systematic approaches (methodologies) for engineering secure distributed systems. In this paper, we attempt to fill the aforementioned gap by surveying and critically analyzing the state-of-the-art in security methodologies based on some form of abstract modeling (i.e. model-based methodologies) for, or applicable to, distributed systems. Our detailed reviews can be seen as a step towards increasing awareness and appreciation of a range of methodologies, allowing researchers and industry stakeholders to gain a comprehensive view of the field and make informed decisions. Following the comprehensive survey we propose a number of criteria reflecting the characteristics security methodologies should possess to be adopted in real-life industry scenarios, and evaluate each methodology accordingly. Our results highlight a number of areas for improvement, help to qualify adoption risks, and indicate future research directions.

Keywords: Computer Security, Security Engineering, Secure Software Engineering, Distributed Systems, Security Methodologies, Model Driven Security, Secure Software Architectures, Security Patterns, Model-based Development, Survey
Categories: C.2.0, C.2.4, D.2.0, D.2.1, D.2.2, D.4.6, K.6.5, L.4

1 Introduction

Ever since the advent of the Internet and its popularization, the face of computing has increasingly been defined by a physically distributed, collaborative model in which software components interact in parallel over a network – potentially spanning vast geographical distances – to perform complex computational tasks. Advances in hardware technologies for commodity parallelism and software paradigms for large-scale distribution [Erl 2009, Foster and Kesselman 2003, Milojicic et al. 2002,

Vaquero-Gonzalez et al. 2009], as well as a range of new system varieties tailored for file sharing, resource aggregation, e-commerce and others, have ensured that businesses as well as social groups and individuals are increasingly exposed to a distributed computing environment.

Along with the fast-paced progress in technologies and paradigms, the corresponding dangers of intentional and unintentional security breaches have increased exponentially, so much so that among the more recent trends in distributed computing, security has become an inhibiting factor for adoption (see, for example, [Rosado et al. 2010a, Zhang et al. 2010]). While the importance and significance of security has never really been questioned, especially for open and collaborative distributed environments, its introduction into software by the industry has not been proportionally enthusiastic. Of course, there is no shortage of research in the area – a prodigious body of work addressing a myriad of access control models, intrusion detection techniques, cryptographic exchange protocols and many other concerns not only exists since the 1970's, but is being enlarged all the time. It appears, however, that much of this has remained in the realm of the theoretical, while software continues to suffer from vulnerabilities whose constant exploitation gives rise to ever new and interesting headlines in the popular news.

In a recent study of the state-of-the-practice in the industry, [Whyte and Harrison 2011] reveal a number of important factors influencing this slow uptake:

1. The business case for employing security best practices is missing;
2. Developers lack security expertise, which is currently required to employ security best practices;
3. The risk of committing to a particular security approach is too high.

Besides these points, and perhaps of greater importance, is the purport and overall conclusions of the study, namely: "a large majority of experts involved agreed that a very significant, if not the most significant, positive impact on secure software development would be concentration on measures that improve the overall quality of the 'state of the practice' rather than the 'state of the art'". In this respect, "research will only have significant impact if it is compatible with the commercial environment of developers and their existent skills" [Whyte and Harrison 2011].

This indicates that for development teams to take security seriously it must be integrated into their everyday activities, i.e. security must be concomitant with software engineering practices, in a manner that is compatible with their current skillsets and (non-security) knowledge-base. In itself, this implication is not new: it has already been argued for in the research literature for over a decade [Fernandez 1999, Devanbu and Stubblebine 2000, Tryfonas et al. 2001, Ghosh et al. 2002, McGraw 2004, McGraw 2006, Jürjens 2005a, Mouratidis 2007, Anderson 2008, Jaatun and Tøndel 2008, Haley et al. 2008, etc.], with some researchers, e.g. [Mouratidis and Giorgini 2006] even envisaging an altogether new field of computing – secure software engineering – as its realization.

Such a full integration of security and software engineering is especially important in the context of distributed systems, where heterogeneity, collaborative behaviour and emergent properties all lead to increased complexity demanding systematic development approaches. A "systematic way of doing things in a particular

discipline", following [Gonzalez-Perez and Henderson-Sellers 2008], is termed a methodology; therefore, the systematic approaches referred to above can justifiably be termed secure software engineering methodologies, or more simply security methodologies, applicable to distributed systems.

A security methodology provides tools, techniques and processes throughout the software development life-cycle (SDLC) to guide the introduction of security and is thus, by its nature, focused on improving the state-of-the-practice in developing secure software. Such methodologies do indeed exist – in fact there are almost two dozen applicable to distributed systems alone – and, from our definition and previous remarks, their application certainly seems like a prudent best-practice that should be popular in the industry. Nevertheless, in light of Whyte and Harrison's findings, it appears that security methodologies are not being applied in any significant measure, or, if they are, they are not successful in their aims. Are these methodologies not widely known? If so, why are they not known? And if they exist and are well-known, why are they not receiving more attention from the industry?

Currently, there is no single survey that gives a broad and comprehensive overview of available security methodologies, not only applicable to distributed systems, but to general software systems, that can support answers to these questions. The more recent overviews of [Villarroel et al. 2005, Jayaram and Mathur 2005, Khan and Zulkernine 2009, Jürjens 2009, Fernández-Medina et al. 2009, Talhi et al. 2009 and Kasal et al. 2011] have all made important contributions in this regard, but, even taken together, do not provide essential details and comprehensive analyses that would allow a fair assessment of the range of methodologies available for practical use. Furthermore, very few (if any) security methodologies have ever been (independently) evaluated with respect to their potential for real-life adoption. This leaves a gap in the literature that, based on current practices and future directions, would inevitably grow in significance.

In this paper, we aim to fill the aforementioned gap by surveying and critically analyzing a wide range of security methodologies for, or applicable to, distributed software systems, both generally and for specific types of such systems, and evaluating their potential for real-life, industry use. Our selection of methodologies is based on their comprehensiveness, applicability and uniqueness, and is aimed at giving a comprehensive and detailed picture of the state-of-the-art. We have attempted to cover all relevant mature methodologies, as well as most emerging methodologies. In the instances where known omissions have been made, we either provide references or short summaries. It is hoped that our detailed reviews and subsequent evaluation provide a first step in increasing awareness about a range of security methodologies, allowing industry stakeholders to better qualify risks and make informed decisions in adopting a systematic security approach.

1.1 Scope and Organization

According to [Baskerville 1993], security design methodologies can be classified into several generations, akin to programming languages: first generation, ad-hoc approaches (approx. 1970s onwards), in which developers introduce security by consulting checklists of all possible solutions; second generation, requirements-driven approaches (approx. 1980s onwards), in which a set of activities are followed alongside (but apart from) the software development process to introduce security;

and third-generation approaches (approx. late 1980's/early 1990s onwards), which are based on abstract modeling approaches concomitant with the development process. The focus of this survey stems from our definitions of security methodologies as systematic approaches combining security and modern software engineering, i.e. we consider third-generation security approaches based on some form of abstract modeling. As a side-effect to surveying these approaches in detail, which 20 years ago were only in their germinal stages, we extend Baskerville's work and bring the security methodology survey literature up to date, albeit from a distributed systems viewpoint.

To accomplish the goals set out above, we first present a taxonomy of security methodologies in [Section 2], discussing their main ingredients and most pertinent characteristics. In [Section 3], we review our selection of security methodologies according to the taxonomy. Our reviews attempt to provide a reasonable amount of detail (more rather than less), so as to promote a fuller understanding and appreciation of each approach, and to allow a more complete assessment of the (above all practical) value of any given methodology. This section therefore aims to address the question we posed previously – why are security methodologies not better known? In [Section 4] we present a set of criteria reflecting industry requirements for adoption based predominantly on the study of Whyte and Harrison, and evaluate the different methodologies accordingly. The merit of the analysis results presented in this section lies in revealing the current deficiencies in existing approaches and establishing trends. This section is an attempt to address the other question we posed previously – why are security methodologies not receiving more attention from the industry? Finally, in [Section 5] we conclude and discuss future directions.

2 Taxonomy of Security Methodologies

2.1 Background: Key Ingredients of a Security Methodology

In this section we present some background material on security methodologies as a prelude to the taxonomy and the rest of the survey.

Previously, we defined a methodology as a systematic way of doing something in a given discipline. Another, more specific point of view on what constitutes a methodology is provided by [Booch 1994], according to whom "a methodology is a collection of methods applied across the software development life-cycle and unified by some general, philosophical approach". The collection of methods referred to by Booch imply a set of activities performed by a methodology, which can provide further insight into the nature of security methodologies. To describe these activities, we first consider a methodology's alignment with a generic software development life-cycle [Ramsin and Paige 2008, Rodriguez et al. 2009] as illustrated in [Figure 1] below.

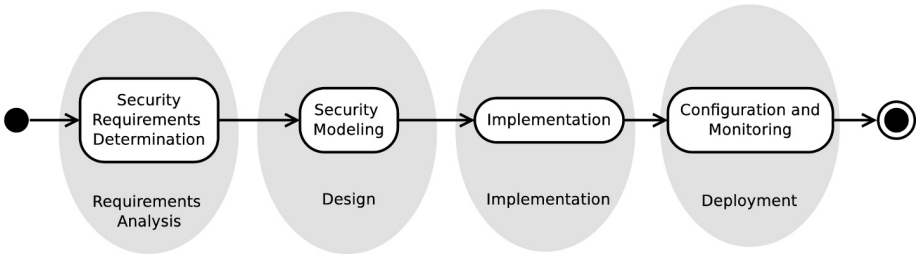

Figure 1: Alignment of a security methodology with a generic SDLC

In general, a software system passes from a set of requirements, to a set of models, to a set of implementation artefacts, until it reaches deployment. Analogously, a security methodology's activities can be aligned into (corresponding) requirements analysis, design, implementation and deployment stages. This alignment of activities may not perfectly fit the (generic) development process above, e.g. some or all of the "design" activities of a methodology may actually be performed during the "analysis" development stage; but in any given case the overall progression of activities will be the same, regardless of the overall development paradigms and process models used. In light of this alignment, we can briefly describe a (generic) security methodology's activities as follows:

— **Security requirements determination** *(Requirements analysis)*: during this stage, activities are performed to determine a set of security requirements (see, for example, [Haley et al. 2008]). Some of these requirements are prescribed by the organization in the form of high-level policies, or are directed by regulations (government, state etc.). A methodology may further undertake some form of security assessment on conceptual or design models to determine and possibly enumerate threats and associated risks. This is often collectively termed "threat modeling" (see, for example, [Swiderski and Snyder 2004]). The outcome of these assessment activities are a set of resultant security requirements, which either augment or refine the initial prescribed set.

— **Security modeling** *(Design)*: during this stage, security properties are modeled, usually as a fulfillment of the security requirements from previous stages, alongside the standard (architectural) modeling activities. Security modeling activities aim to embody the security properties in some form of architectural models, just as functionality is embodied in models during system design.

— **(Security) Implementation** *(Implementation)*: during this stage, the security properties (as models, for example) from the previous stages are implemented, either manually, or automatically. This can include the generation of security configurations or parameters for certain software elements or the target platform, as well as manual coding activities and the selection of COTS components. Other activities may include testing, code-level security verification etc.

— **Configuration and monitoring** *(Deployment)*: during this stage, a methodology may prescribe certain configuration activities, and may provide, through integral

use of software support, monitoring facilities during system run-time. While not strictly part of a methodology's activities, such configurations and monitoring provisions may be an integral part of the overall approach (e.g. if a methodology is tied to a particular software framework).

We will designate a methodology providing activities for all the stages outlined above a *comprehensive methodology*, whereas a methodology providing activities for only one or two of the stages will be designated a *partial methodology*.

2.2 Classification Dimensions

Security methodologies, like the systems they secure, can vary greatly from one another, depending on a set of fundamental characteristics. In order to encompass their wide variety, in this section we introduce a number of classification dimensions as part of a taxonomy of security methodologies. Our classification dimensions do not represent a complete set of features of a methodology, but attempt to capture the main characteristics. The dimensions are presented in order of those that are inherent to a methodology ([Section 2.2.1] – [Section 2.2.3]) and unlikely to change over time; those that are inherent but mutable ([Section 2.2.4] – [Section 2.2.7]); to those reflecting current work ([Section 2.2.8]).

The taxonomy complements the descriptions (in the previous section) of the general activities performed by a security methodology.

2.2.1 Methodology Paradigm

In [Section 2.1] we presented Booch's extended characterization of a methodology as collection of methods applied across the SDLC, unified by a general, philosophical approach [Booch 1994]. The philosophical approach referred to by Booch can be termed the methodology's paradigm, and is one of the most distinguishing features. A methodology's paradigm may influence its organization, prescribing certain activities to be performed during a certain life-cycle stage, and often determines corresponding development paradigms, integral use of standards and a concentration on certain software engineering artefacts or approaches. All the above allow us to divide security methodologies into a hierarchical set of distinct (though not always disjoint) classes.

2.2.1.1 Code-Based methodologies

Code-based methodologies attempt to introduce security into a system by enforcing certain security-related activities throughout a software process without explicit regard for a system's design or architecture. The main distinguishing feature of code-based methodologies is their focus of security-related efforts primarily or exclusively on requirements and implementation-level artefacts (hence the term “code”-based). Activities performed throughout the whole SDLC may include security requirements gathering, threat modeling (at either the requirements or design stage), penetration testing, developer education and others. Following [Baskerville 1993], code-based methodologies can be classified as second-generation security approaches.

The most widely recognized and used code-based methodologies are Microsoft's SDL [Howard and Lipner 2006] and OWASP's CLASP [OWASP 2011] [De Win et al. 2009, Khan and Zulkernine 2009]; other less known approaches include the early

work of Breu and colleagues [Breu et al. 2004], who consider introducing security within the object-oriented development life-cycle, and Secure UP ([Steel at al. 2005], Chapter 8), a security-enhanced version of the Rational Unified Process (UP). Such code-based approaches have a number of benefits. The activities performed provide managed guidelines for software projects to reduce security flaws most broadly, and determine well-defined checkpoints at which to enforce security. Moreover, other, more specific methodologies, can be incorporated into a code-based methodology to enforce one or more security practices (e.g. secure coding).

Taken by themselves, however, code-based methodologies do not give concrete guidance on how to introduce security properties into a software system, or which such properties are required – they only provide a framework in which to use separate security improvement techniques (cf. [Schumacher 2003], Chapter 5). Thus, they only offer high-level solutions, without any "'practical mechanisms' […] that would permit [them] to implement the approach in a short space of time and with minimal effort" [Rosado et al. 2010a]. Their lack of a modeling stage in particular implies that security must be achieved via rigorous code inspections, testing and manual techniques.

By nature, code-based methodologies are generic, and tailoring them to a specific type of system (e.g. a distributed system) may require a significant amount of work, making them inappropriate for all system types (cf. [Rosado et al. 2010a]). All code-based approaches require significant security expertise in some form, such as a security expert (Microsoft SDL) who oversees the security-related activities or a dedicated team member (CLASP) who performs specific development tasks [Belapurkar et al. 2009].

[Gregoire et al. 2007] present a detailed comparison of Microsoft's SDL and CLASP, which is extended in [De Win et al. 2009] to include McGraw's Touchpoints [McGraw 2006]. Both papers discuss at length the activities done at each stage for each individual approach and in relation to each other. [Jayaram and Mathur 2005, Khan and Zulkernine 2009] collectively survey a number of code-based approaches, including Microsoft's SDL, CLASP and AEGIS [Flechais et al. 2003, Flechais et al. 2007], with concentration on the requirements and design stages.

2.2.1.2 Model-Based methodologies

In contrast to code-based methodologies, **Model-based methodologies** (the focus of this survey) are based on some form of abstract modeling and hence take a system's design or architecture into account explicitly. In this sense, they are the logical complement of code-based approaches.

Model-based methodologies form a very broad class of approaches that can be further divided into four hierarchical classes. A methodology can have characteristics from more than one of these classes, in which case the class into which most of its characteristics fall becomes its primary class, and the others secondary or even tertiary classes. In most cases, one characteristic will prevail over the others, allowing the methodology to be categorized in a single class. In what follows we briefly describe the (model-based) methodology classes. They are further described in detail throughout the survey ([Section 3]), in sections preceding the reviews of the methodologies belonging to a particular paradigm class.

— **Model-driven methodologies** ([Section 3.2]) focus on system models as first class entities within the MDE (Model-Driven Engineering) paradigm [Schmidt 2006], and add security properties by enriching the system's functional models with security artefacts. These methodologies can be subdivided further into Model-Driven Architecture/Security (MDA/MDS) approaches ([Section 3.2.1]), which use transformations on security-enriched models to generate implementation-level (security) artefacts; and Aspect-Oriented Software Development (AOSD) approaches ([Section 3.2.2]) – emphasizing separation of concerns – which model security as aspects and weave the aspects into system functional models.

— **Architecture-driven methodologies** ([Section 3.3]) are distinguished from model-driven and aspect-driven methodologies by the (broader) emphasis on software architectures as opposed to first-class models or aspects, and a focus on creating a "secure software architecture". Architecture-driven approaches either typically use specialized Architecture Description Languages (ADLs) – [Taylor et al. 2010] to incorporate security at the architectural level – often employing formal methods to reason and analyze the resulting architecture – or base themselves on certain design principles.

— **Pattern-driven methodologies** ([Section 3.4]) emphasize the use of patterns as the central technique to achieve the introduction of security properties into a system, and can be subdivided further into methodologies utilizing components and/or architectures as patterns ([Section 3.4.1]) and methodologies utilizing on security patterns ([Section 3.4.2]).

— **Agent-driven methodologies** ([Section 3.5]) are based on the Agent-Oriented Software Engineering (AOSE) paradigm [Jennings 2001], and are distinguished by a goal-oriented approach and an explicit focus on software agents.

2.2.2 Specificity

A methodology can be applied to a wide range of systems or it can be tailored to only a small subset or a particular system type. Generic security methodologies can be applied to all software systems by providing universal guidelines, models, tools and techniques for the incorporation of various security properties. As a trade-off to their wide applicability, these methodologies do not address the challenges raised by particular system types. In such cases, a generic methodology's guidelines and techniques must be adapted for the specifics of the system, relying on the security expertise of the architects and developers. Specific security methodologies, on the other hand, are tailored and applicable to one or several types of systems, but may not be appropriate for securing more general systems. If one or more parts of a system employ, for example, different technologies or design approaches, specific methodologies can become inadequate. Between these two extremities is a whole spectrum in which some or all aspects of a methodology can make provisions for or be suited to specific system types. The latter can broadly be referred to as hybrid specific, or hybrid generic methodologies, depending on their accent.

2.2.3 Modeling Language and Notation

The modeling language and corresponding notation used by a methodology is often an important defining factor. Different modeling languages include a variety of ADLs (such as Darwin, xADL and AADL), UML, SysML, Petri nets and others. They differ in their syntax and semantics, ability to model the various aspects of a system, level of formality etc. [Taylor et al. 2010] provide a detailed taxonomy of ADLs that is applicable to modeling languages more generally, and can be referred to for more detail.

2.2.4 Range of Security Properties

Securing a whole system requires the use and application of a broad range of security properties. These are often divided among the categories of confidentiality, integrity, availability and accountability [Gollmann 2011], and include properties such as entity identification, non-repudiation, data origin authenticity and others. A methodology can support the introduction of many of these security properties or only one or several such properties. Some properties (e.g. access control) are arguably more important than others, particularly in distributed settings where systems are inherently more exposed to external threats (e.g. via the Internet). In the face of leaving vulnerabilities left open to exploitation, however, all security properties can become equally important. Ensuring that an attacker cannot gain access to a company database (access control), for example, is essential, however, it may be equally important to prevent an attacker eavesdropping on a communication channel carrying database data (communication secrecy). Ultimately, the aim of applying a security methodology is (or should be) to produce a secure system, and in this sense the scope of supported security properties to a large degree determines the utility of a methodology.

Adding support for the introduction of new security properties to a methodology is also an important characteristic in this context. In some cases it may take significant effort to interpolate or even extrapolate a methodology's support for introducing new properties, while in other cases this addition may be trivial.

2.2.5 Use of Formal Methods and Verification

Using formal methods during the application of a methodology as an integral or optional element can give assurances that the security solutions introduced are correct, consistent and compatible, and, in some cases, verified or even certified to be such. The use of formal methods, however, also has disadvantages: it involves significant human labour, leading to increased project costs [Devanbu and Stubblebine 2000]; and it also impacts adversely on the ease of use of a methodology, requiring developer training in the relevant formal techniques.

Formal methods are usually used in conjunction with some form of security verification. Verification approaches may be manual, such as the creation of test cases; automatic, using specialized tools; or semi-automatic, and may be performed at any stage of development. In any of the above cases, the application of formal methods is almost invariably localized to critical aspects of a system: no single formal technique is capable of formally verifying a complete complex system.

2.2.6 SDLC Stages Supported

A methodology can encompass the whole SDLC, or only certain stages, i.e. one or more of the early stages, which we define to be the stages spanning across (early) requirements to (late) design. These stages are dependent on the paradigm used for development (e.g. object-oriented, functional, architecture-centric), and may include an analysis stage, an architecture-development stage and others.

2.2.7 Ease of Use

The ease of use or usability of a methodology is a qualitative measure of how easy it is, in relative terms, for an average software development team to introduce security into a system by applying the methodology. This is an extremely important factor in deciding the uptake of a methodology, since high complexity and high levels of expertise imply steep learning curves and, therefore, prolonged project schedules – making for an unattractive choice. Features such as tool-support, detailed guidance, usage of popular notations, support for re-usability and others all contribute to a methodology that is not only more usable but also more practical. Such methodologies would be classified as easy to use. Conversely, methodologies requiring detailed knowledge and application of formal methods, such as higher-order predicate logics and Petri nets, or domain knowledge, such as cryptography and complex security models, would be classified as difficult to use.

In assessing the ease of use of a methodology there is inevitably a factor of subjectivity, both from those performing the assessment, and the basis on which it is performed – namely, factors such as whether the most popular notation or modeling language is the easiest to use, the trade-off between a steep learning curve at the beginning and simple application of a technique thereafter etc. Case studies are more significant in this respect for showing how easy it is to use a methodology in actual experience, although here, too, it is not always clear how an approach can scale for larger and/or more complex systems. Despite the presence of subjectivity, the assessment of the ease of use of a methodology provides an important indicator when considering its real life application.

2.2.8 Tool Support

Tool-support for a methodology can be realized in many ways, for example, as a set of scripts to help transform models, a GUI-based application for specifying and cataloging threats, or a whole suite of CASE tools for modeling security architectures. Tools help to make a methodology more acceptable to a broader range of developers and, therefore, increase its practical value. If tool-support is not available, the methodology must rely on its ease of use in addition to using techniques realizable without significant developer or team effort; conversely, with tool-support a methodology can allow more sophisticated tasks to be accomplished automatically or semi-automatically.

3 Survey of Model-Based Security Methodologies

In what follows we review our selected range of security methodologies according to the taxonomy presented in the previous section. In order to provide more detailed reviews for more detailed approaches and retain relative similarity between review lengths, we have striven to review comprehensive methodologies in more detail, and partial methodologies in less detail proportionally, based on their scope and number of activities performed.

The methodologies are grouped based on their primary paradigms. Each review includes a very brief outline, capturing the essence or distinguishing elements of the methodology, a detailed description and a discussion, the latter concentrating on limitations and features of the methodology. The detailed description is segregated into the SDLC stages for which a methodology is applicable, though not always in the strict order (i.e. requirements analysis, then design, then implementation) of application. We preface each methodology paradigm with a very brief overview of the corresponding development paradigm and/or relevant concepts, insofar as these have relevance to the security methodologies under review. The exception to this is the (general) model-based paradigm class, which was previously discussed in [Section 2.2.1.2].

3.1 (General) Model-Based Methodologies

3.1.1 Jürjens (MBSE / UMLsec)

OUTLINE:

Jürjens [see Jürjens 2002a, Jürjens 2005a, Jürjens 2006a] was amongst the first to propose a modern model-based approach to securing distributed software. Jürjens' Model-Based Security Engineering (MBSE) follows the standard life-cycle and corresponding engineering practices [see Section 2.1], and proceeds by first eliciting security requirements along with the system's functional requirements; embodying the resulting security requirements in models during design; and finally implementing the models into code either manually or automatically. UMLsec – an extension of UML for the specification of security requirements – is the modeling language and core of Jürjens' MBSE. Below we describe UMLsec and its associated tool-suite for design and implementation and then briefly mention approaches for security requirements determination.

DETAILED DESCRIPTION:

Design: UMLsec is a light-weight extension of the Unified Modeling Language, initially as a UML 1.4/1.5 profile in [Jürjens 2002a, Jürjens 2005a] and recently updated to conform to UML 2.3 via the UMLsec4UML2 profile in [Schmidt and Jürjens 2011b], defining stereotypes, tags and constraints for expressing a number of security requirements, such as data secrecy and integrity, message secrecy and integrity, information flow restrictions, role-based access control and others. Each stereotype is applicable to different UML diagrams, including deployment (accounting for communications between nodes), class (accounting for static data and relationships between elements), activity (accounting for intra-object properties) and

sequence diagrams (accounting for interactions and protocols). By annotating UML models with the UMLsec security extensions, developers embody a system's security requirements at the design level.

While useful in itself, simply annotating models with security requirements does not guarantee that the models will conform to their corresponding requirements. MBSE/UMLsec's real value lies in the use of advanced tool-support for precisely this kind of security verification. The UMLsec tool-suite [Jürjens 2005b, Jürjens 2006b, Jürjens et al. 2008] consists of a number of analysis tools and plugins for verifying static and dynamic models as well as checking access control permissions. To support formal verification, the subset of UML employed by UMLsec is given a formal (behavioural) semantics via UML machines (see [Jürjens 2005a] for details), allowing for the execution semantics of UMLsec models to be converted to first-order logic and checked via automated theorem provers (e-SETHEO in earlier work, and more recently SPASS) against the relevant security requirements. If conformance fails, an attack sequence violating the security requirements is automatically generated and provided to developers for analysis.

For the purposes of verification, UMLsec allows the capabilities of adversaries to be specified via an adversary type, which formally models the threats related to the deployment view of a system, namely, those related to communication (delete, read, insert) and individual nodes (access). Two standard adversary types are defined in UMLsec that can be annotated on UML package diagrams: a default attacker, representing "an outsider adversary with modest capability [Jürjens 2005a], and an insider, representing an attacker with internal access to a system.

Since the verification process can be applied to dynamic models captured as sequence diagrams, it is ideal for checking cryptographic security protocols (cf. [Jürjens 2006b]). In comparison to purely formal security protocol verification approaches [see Gritzalis et al. 1999] not employing the more intuitive UML notation, specifying and verifying protocols in UMLsec can be seen as one of its strongest points, albeit requiring expertise accordingly.

MBSE/UMLsec is tailored for object-oriented and component-based systems, making its applicability broad. UMLsec can also be used to specify security aspects [Jürjens 2006a] (see the beginning of [Section 3.2.2] for definitions) as well as security components and/or partial architectures treated as patterns ([Schmidt and Jürjens 2011a], see the beginning of [Section 3.4] for definitions), but, contrary to [Jürjens et al. 2002, Jürjens 2005a], is not appropriate for security patterns such as those found in [Schumacher et al. 2006], since the latter cannot be represented by a fixed, formally specifiable architecture.

Implementation: One of the problems with using MBSE/UMLsec is that heavy reliance on security verification at the design stage "has no implication on the real system, as long as the application code is not generated directly from the model[s] and there is no assurance that the generation process itself is correct" [Best et al. 2007]. According to [Jürjens 2006a], such assured generation is currently not feasible, leaving as an (only) alternative the necessity to perform verification procedures on the code resulting from UMLsec models. This is again achieved via the UMLsec tool-suite, augmented for the implementation stage, which can (1) help to ensure that the (manually developed) code resulting from UMLsec design models adheres to the

specified security requirements by inserting run-time assertions; (2) automatically generate a series of test-cases to help guarantee code compliance; and (3) analyze (cryptography-related) source code by generating call-graphs, translating them into first-order logic – analogously to the design-level verification procedures – and passing them to an automated theorem prover for checking [Jürjens 2006b]. Legacy code can also be a target of such analysis.

MBSE/UMLsec is unique among other methodologies in its direct support for implementation-level security.

Requirements analysis: To complete the MBSE approach, a security requirements stage is necessary in which the requirements that are later specified using UMLsec and verified to hold using the tool-support can be determined. There are several approaches that cover this stage. [Jürjens 2002b], for example, utilizes use cases and security goal trees to guide the introduction of requirements into UMLsec models; [Popp et al. 2003] employ UMLsec within an object-oriented methodology determining security requirements via security-extended use cases; [Hatebur et al. 2011] employ problem frames as patterns to capture requirements (cf. [Section 3.4.1.2]) and use OCL to generate UMLsec models; and [Mouratidis and Jürjens 2010] use Secure Tropos for security requirements modeling and architecture development, mapping the resulting architectural artefacts to UMLsec models during detailed design via a set of heuristics.

Despite the existence of several competing approaches for determining security requirements, none of them are really perfect matches for UMLsec. Hatebur et al.'s approach in particular is quite advanced and intricate, however, generating design artefacts straight from requirements ignores the activity of architectural design, making it appear a more theoretical approach applicable in practice to simple systems. The approaches of Mouratidis and Jürjens (which we discuss in more detail later from the perspective of the agent-driven Secure Tropos methodology in [Section 3.5.1]) and Popp and colleagues are attempts to integrate UMLsec into other methodologies, not the converse. Finally, Jürjens' early work with goal trees does not include a comprehensive threat enumeration strategy. This leaves the requirements aspect less well-defined than the rest of MBSE.

DISCUSSION:

Overall, MBSE/UMLsec is a rigorous secure software engineering approach particularly well-suited for use during detailed design, with a powerful tool-suite for security verification. UMLsec's deployment view and network-related stereotypes («LAN», «Internet» etc.) make it quite applicable to distributed settings, (cf. [Jürjens 2002a, Best et al. 2007]), for example, demonstrate the use of UMLsec in the industrial development of a distributed (client/server) information system. UMLsec has also been used on a number of other real-life industry projects [see Jürjens 2005a, Jürjens 2006b]. Nevertheless, MBSE/UMLsec promulgates a formal approach that is at times complex [Best et al. 2007] and requires a reasonable amount of security expertise, though less so than purely formal approaches (cf. [Gritzalis et al. 1999]) due to the use of UML and the intuitive, often self-explanatory nature of the stereotypes. Where expertise begins to exponentially increase is in the modeling of security protocols and the development of custom stereotypes with custom tool-

support (using UML machines). The latter, in particular, also requires strong mathematical skills and sound knowledge of formal methods, which not all developers posses.

Since no guidance is provided for actually designing secure systems, choosing security solutions etc., the MBSE/UMLsec approach can be seen as a "power user" modeling and analysis suite for guaranteeing security properties in systems for which security is a critical factor.

As slight digression, we should point out that more recently UMLsec has been described as a model-driven security (MDS) approach (see for example, [Fernández-Medina et al. 2009, Kasal et al. 2011]) as per [Section 3.2.1]. While MBSE shares certain similarities with MDS approaches, no model transformations are used (although the possibility for this does exist – see [Jürjens 2005a]), making it model-based, but not model-driven with respect to our taxonomy.

3.2 Model-Driven Methodologies

A model is an abstract representation, capturing some but not all aspects, of an actual system or entity. Model-driven security methodologies are based on the Model-Driven Engineering (MDE) paradigm [France and Rumpe 2007, Schmidt 2006], in which models are treated as first-class entities in the overall development process. A fundamental characteristic of MDE is the use of model transformations, either automatic or manual, between models (model-to-model transformations) across abstraction layers, or as a means for generating implementation artefacts (model-to-code transformations). While model-to-code transformations, otherwise known simply as code generation, is one of the attractive features of MDE and perhaps one of the most advertised, in practice the paradigm is also used as a means of obtaining re-usable models and as a disciplined and systematic development approach [Hutchinson et al. 2011]. All these benefits can be transferred to security methodologies within the MDE paradigm.

Model-driven methodologies can be further divided into two sub-classes described individually in [Section 3.2.1] and [Section 3.2.2].

3.2.1 Model Driven Architecture / Security (MDA/MDS) Methodologies

Model-Driven Architecture is OMG's realization of MDE [Frankel 2003], in which models are created in three levels of abstraction measured with respect to a particular platform and transformed gradually from one level to another. The levels correspond approximately to conceptual models (CIMs – computationally independent models), design models (PIMs – platform-independent models) and detailed, implementation-dependent design models (PSMs – platform specific models). In MDA models can be transformed between and across levels to ultimately generate implementation-level artefacts.

Model-Driven Security (MDS) – a term first used by [Lodderstedt et al. 2002, Lang and Schreiner 2003] and, more definitively, by [Basin et al. 2006] – is an attempt to specialize the MDA paradigm to security by using models and transformations to enforce security properties from high-level designs to implementation artefacts [Fernández-Medina et al. 2009]. In MDS methodologies, a system's platform-independent functional models are annotated or composed with a

well-defined set of security constructs, which can be verified and collectively transformed together with the functional models either directly or through intermediate stages to security configurations in some underlying platform or to actual software components (i.e. to code). [Fernández-Medina et al. 2009] survey the area of model-driven security both concisely and comprehensively, and we refrain from repeating their work here.

Although UML is the standard modeling language used in most MDS approaches, the use of other languages is also possible [see Dai and Cooper 2007]. Dai and Cooper's treatment of SecureUML as a modeling approach in that work is supplemented by our treatment of SecureUML in the context of Loddertedt, Basin and Doeser's full Model-Driven Security methodology.

3.2.1.1 Lodderstedt / Basin / Doeser (Model-Driven Security)

OUTLINE:

The work of [Lodderstedt et al. 2002] arguably marked the beginning of the model-driven security paradigm with SecureUML – a security modeling language realized as a heavy-weight UML extension for allowing RBAC access control policies to be modeled concurrently with functionality at the design stage – within a code-generative approach. Lodderstedt's later work in [Lodderstedt 2003], and following it [Basin et al. 2006], generalizes and completes the approach begun with SecureUML for arbitrary security modeling languages while still retaining the focus on RBAC access control. The resulting (partial) methodology, termed simply Model-Driven Security, emphasizes the specification of security properties in design models (somewhat akin to MBSE/UMLsec), their verification and their automatic transformation to security configurations. The latter transformation can target arbitrary software platforms, however, in practice they are most useful in generating policy specifications for distributed middleware.

DETAILED DESCRIPTION:

Design and Implementation: The early work on Model-Driven Security begins with Lodderstedt et al.'s proposal of the SecureUML modeling language [Lodderstedt et al. 2002] for expressing extended RBAC policies in functional models during design that are subsequently transformed via a model-to-code transformation into security configuration parameters for the J2EE distributed platform. The latter ideas of modeling security at the design stage and transforming it to configuration parameters (or code artefacts, as required) are extended by [Lodderstedt 2003], appearing later in [Basin et al. 2003, Basin et al. 2006], as part of the fully developed Model-Driven Security methodology.

Model-Driven Security is based on a linguistic approach to modeling security, which proposes the creation of custom language schemas combining security modeling languages with functional (system or domain) modeling languages, enabling developers to model the security and functional aspects of a system simultaneously. To bridge the gap between the system and security modeling languages in each schema, a third, intermediate language – a "dialect" – is required, which tailors the security language in reference to a particular (system) language with more concrete or refined semantics and helps to merge the corresponding language metamodels (abstract syntax) and vocabulary. A dialect for a security language

allowing the specification of access control policies such as SecureUML, for example, can specify which model elements of a system language can be valid protected objects and what actions can be performed on them. This general composition of languages results in families of secure design modeling languages that provide the security properties specifiable by each security language as system modeling constructs in a coherent fashion, with the concrete syntax (notation) used for functional models.

In order to realize this abstract vision of different language schemas and thereby produce a practical security approach, Lodderstedt, Basin and Doeser propose two example UML-based functional modeling languages in [Lodderstedt 2003] and [Basin et al. 2003, Basin et al. 2006] – ControllerUML for state-based systems; and ComponentUML for distributed component systems – along with SecureUML as a security modeling language. The proposed functional modeling languages are merged with SecureUML to produce two usable schemas: ComponentUML+SecureUML and ControllerUML+SecureUML, demonstrating the viability of the abstract approach. As in [Lodderstedt et al. 2002], the resulting languages can be used to create security-enhanced design models that can subsequently be transformed to security configurations in the supported middleware platforms; for example, ComponentUML+SecureUML models can be transformed to security policies for EJB as well as Microsoft's .NET platform.

The verification of security-enhanced design models, first proposed as a goal for future work in [Lodderstedt 2003], is addressed in the work of [Basin et al. 2009], both abstractly and with relation to ComponentUML+SecureUML. The proposed verification strategy is based, like security modeling, on a linguistic approach, and consists of formulating queries in OCL about whether certain security conditions can hold and evaluating the queries with respect to the metamodel corresponding to a target design model. The approach requires the formalization of the modeling languages used (something already accomplished for SecureUML and ComponentUML in the previously cited publications), as well as the mapping of design models and possible run-time scenarios (expressed in object models, for example) to their associated metamodels for evaluation. Although supported by a modeling and OCL query tool (SecureMOVA), verification clearly requires an expert.

Requirements Analysis: The determination of security requirements is considered only briefly in [Lodderstedt 2003], who proposes a rights assignment scheme very similar to that of [Fernandez and Hawkins 1997] (see [Section 3.4.2.1] for more details on the latter). Nevertheless, security requirements determination is not an essential aspect of Model-Driven Security.

DISCUSSION:

Most model-driven MDA/MDS methodologies have been influenced by the general strategy taken in Model-Driven Security of annotating higher-level models with security properties and transforming these to implementation-level artefacts. However, Model-Driven Security's abstract basis of creating general language schemas has not been as influental.

First of all, creating new security modeling languages is a demonstrably difficult task requiring expertise, as evidenced by the formal descriptions presented by [Lodderstedt 2003] and [Basin et al. 2006] of the abstract syntax and semantics of

SecureUML (which, it must be kept in mind, can only specify RBAC policies). Formal specifications are necessary, however, for Basin et al.'s security verification strategy. Perhaps more problematic than complexity and high expertise requirements is that, in practice, the set of properties specifiable by a security modeling language is restricted by the capabilities of the target software platforms. Hence, according to [Basin et al. 2006]: "the modeling constructs in the security modeling language and their semantics should be designed with an eye open to the class of architectures and security mechanisms that will later be part of the target platforms. This requires care during the language design phase". Thus designing new languages is not only complex, but can also be precarious. Even if a platform were to support a very wide (fixed) range of security mechanisms, there remains an issue of scalability: as the security semantics become richer, so does the complexity of the security modeling language.

As a practical methodology – consisting of SecureUML merged with ControllerUML and ComponentUML together with tool-support – aiming to generate security configurations for distributed platforms, Model-Driven Security is a sound security approach, albeit one focusing exclusively on RBAC policies.

The methodology has been trialled in the industry on at least one real project. [Clavel et al. 2008] report a successful use of Model-Driven Security with ComponentUML+SecureUML in the context of a test-report configuration utility, adding, however, that (good) tool-support is required for the wide-spread adoption of the methodology (in fact, at the time the developers did not actually implement a transformation function to generate the security configuration – this was done by hand). It is worth pointing out that Clavel et al.'s test-report utility had an almost identical design to the distributed calendar/scheduler application used as an example in [Lodderstedt et al. 2002]. Nevertheless, even the small-scale example in [Lodderstedt et al. 2002] showed an impressive improvement over creating manual security configurations in terms of effort and precision. [Lodderstedt 2003] also reports on using a prototypical e-commerce application for J2EE as a pilot study with similarly positive outcomes in generating security configurations.

As long as a new security modeling language is not required (i.e. if the existing SecureUML language schemas are sufficient for the task at hand), the ease of use of Model-Driven Security is relatively high, especially given the popularity of UML. However, since no guidance is provided as to where and why security specifications are introduced into design models, the methodology requires a security expert. In this sense, Model-Driven Security is akin to MBSE/UMLsec in helping experts to perform their tasks better.

Model-Driven Security is supported by ArcStyler plugins, and more recently by a custom Eclipse-based development environment for modeling and analyzing security design models (see [Basin et al. 2011] for details).

3.2.1.2 Hafner / Breu / Memon (SECTET / SECTISSIMO)

OUTLINE:

SECTET [Alam et al. 2007, Hafner et al. 2006, Memon et al. 2008, Hafner and Breu 2009] is a comprehensive model-driven (MDS) (primary class) and pattern-based (secondary class) methodology consisting of a framework, run-time infrastructure and analysis process for securing SOA-based systems, concentrating on

peer-to-peer, inter-organizational workflow scenarios. Like other model-driven approaches aspiring to produce security configurations from high-level models, SECTET's main goal is to support the annotation of functional business-process models with security properties, and to transform the resulting security-enhanced models to implementation-level artefacts in supporting web-service platforms.

DETAILED DESCRIPTION:

Analysis, Design and Implementation: The main analysis, design and implementation level aspects of the methodology are embodied in the SECTET framework. In early work on SECTET, e.g. [Hafner et al. 2006], the framework's architecture closely adhered to the principles of MDA, and in particular Basin et al.'s MDS specialization [Basin et al. 2006], by transforming security-enhanced functional models through two layers of abstraction corresponding approximately to PIM (security-enhanced functional models) and PSM to security configurations. The SECTISSIMO project [Memon 2011, Memon et al. 2008], which, judging by later publications, e.g. [Hafner et al. 2009] and presentations [Katt et al. 2010] seems to have been subsumed into SECTET, extends the latter architecture by adding an intermediate transformation layer, introducing an additional platform-independent, model-to-model transformation.

Using the SECTET framework (in its extended, current form), models are first created at the Security-enhanced Functional Models Layer using SECTET-DSL, a set of domain specific languages for business process modeling realized as UML 2.0 profiles [Hafner and Breu 2009]. Since the aim of SECTET is to support inter-organization workflows, the SECTET-DSL languages provide appropriate constructs to model documents, interfaces as well as global and local views for business processes. Together with modeling functionality using SECTET-DSL, models can be annotated with security requirements using SECTET-PL, a high-level policy language [Alam et al. 2007] to produce security-enhanced models. During development, these security-enhanced models are transformed via a model-to-model transformation to abstract security patterns and abstract policy models in the Abstract Security Models Layer, where they are subsequently refined manually by two degrees: the first by generating several platform-independent patterns and policy models at a lower level of abstraction; and the second by determining their platform-specific details (technologies, standards etc.). Pattern and policy model refinement thus gradually transforms security artefacts from PIMs to PSMs. In the last transformation step, the refined, platform-specific patterns and policy models are mapped and transformed, respectively, to security services and configurations in the Runtime Platform Layer (i.e. a compatible target platform), and subsequently deployed.

Implementation and Deployment: For the Runtime Platform Layer, SECTET utilizes a custom security as a service (SeAAS) infrastructure [Hafner et al. 2009] – inspired by the [ISO/IEC 7498-2 1989] security architecture for open distributed systems [Memon 2011] – in which security functionality is provided by a collection of flexible components or services. SeAAS is used during the SECTET framework's last (model-to-code) transformation step described above, in which the platform specific patterns are mapped to security workflows, i.e. they are transformed to

compositions of SeAAS security services in BPEL, and policy models are transformed to security configurations of those services.

SeAAS is split into two layers of services: basic services like encryption and digital signatures, and advanced services like authentication, authorization, non-repudiation and others, which make use of the functionality provided by the basic layer. Instances of SeAAS are deployed in each participating organizational/administrative domain and connected by an Enterprise Service Bus (ESB), which can intercept all messages – thus playing the role of a reference monitor – and pass them to the SeAAS infrastructure. The decision of which security services to use, alone or in a composed sequence, at any given time is taken by the SeAAS Security Engine, thereby decoupling security functionality from application code. This approach promulgates a declarative security model, where most security-related configuration is placed in the underlying infrastructure of the application (i.e. SeAAS connected to the ESB).

Requirements analysis: The SECTET framework is complemented by a security analysis process, ProSecO [Breu et al. 2008, Hafner and Breu 2009], which consists of a series of steps to determine security requirements from general business-driven security goals. The latter is accompanied by a set of metamodels, which provide a (UML-based) conceptual framework in which to consider stakeholders, business concerns and system components (as services) at a large granularity level, from both global and local views. ProSecO is applied to a system's (local) model, which is decomposed into three layers according to the (local) metamodel: business layer, application layer and physical layer. The process proceeds by setting certain security goals, which in turn give rise to security requirements; considering the threats and their associated risks with respect to these requirements; and choosing a suitable (set of) countermeasure(s). This is done by further partitioning the system model into sub-models, determining a "dependency graph" of model elements that are related (top-down, starting at the business layer) and performing the analysis iteratively on each sub-model. The process lacks support for choosing the actual countermeasures (security controls), and this may be future work. ProSecO is supported by a tailored development environment to help with the task of annotating model elements with security artefacts (threats, risks etc.).

Although setting security goals and effectively refining them into security requirements is a useful endeavour in securing a system, associating threats and corresponding risks to model elements only with respect to security requirements fails to consider any attacks on the system, meaning that a valid threat arising from a possible attack will never be considered unless it falls within the scope of a security goal. The drawback of this scheme is that a system can be considered secure without actually being protected. In this respect, the remark by the authors that OCTAVE [Alberts and Dorofee 2002] can be used as a complementary process (i.e. a process to enumerate security requirements starting from threats, not the other way around) should be taken in practical terms, too, if complete security is to be achieved. Besides its goal-oriented nature, the main merits of the ProSecO security analysis process are in the use of tool-support, formalization via UML metamodels and the relating of security artefacts (threats and risks) to system models, as opposed to considering them in the abstract. ProSecO complements the SECTET framework (models to security

configurations) with a corresponding set of security activities to determine the necessary security solutions (albeit with the limitation noted above).

DISCUSSION:

SECTET offers a detailed and strongly model-driven security approach. In comparison to Lodderstedt, Basin and Doeser's Model-Driven Security (see [Section 3.2.1.1]), SECTET can handle a much wider variety of security mechanisms, including authentication and non-repudiation. This is not altogether surprising, since SECTET takes its start from the latter approach, which is extended further and applied comprehensively. With ProSecO, SECTET also motivates the need for the security mechanisms as responses to threats within the context of set security goals. Nevertheless, SECTET is only applicable to inter-organizational workflow scenarios, and not to more general distributed settings.

In [Memon 2011], it is envisaged that the various activities performed when using SECTET are delegated to development team members with certain roles and experience. In particular, functional models are to be specified by a domain expert, who also determines the security requirements; a so-called framework engineer creates a catalog of security patterns, as well as their refinements, together with policy models for use in the abstract models layer; a security expert selects the apposite patterns and follows the process until code generation; and an application administrator takes care of deployment, configuration and management of the implemented security services. This distribution of roles fits the SECTET approach well, however, it implies that security expertise is essential to the success of the methodology. Security expertise is required in many forms: e.g. the application administrator should be versed in security concepts to configure and manage whole security infrastructures; the domain experts must possess sound security knowledge, otherwise it would be impossible for them to specify correct security requirements; and, finally, the framework engineer is really a security expert in disguise. These observations imply that SECTET is tailored for a very security-aware development team, making the methodology more restricted in scope. Re-use of the pattern refinements, models etc. across projects certainly helps to mitigate this situation, but not for the first project, nor with respect to differing security scenarios.

Regarding the use of patterns, strictly speaking, the security patterns in SECTET are not treated in the manner of the security patterns appearing in, for example, [Schumacher et al. 2006] (cf. [Section 3.4.2]), but are more properly intermediate abstractions of security mechanisms represented as models suitable for further transformation. Therefore, the advantages patterns have in guiding developers are not exploited. The modeling and refinement of security policies to configure the mechanisms represented by the patterns is a unique and important feature of the SECTET approach, and complements the use of models and patterns (as models) very well.

The methodology has been trialled in real-life projects from the e-government and e-health domains [see Hafner et al. 2006] and [Hafner and Breu 2009], attesting to the applicability of the approach.

SECTET provides an extensive tool-suite, both for the security requirements process (ProSecO) and the model-to-code transformations. The SECTET-DSL

profiles are compatible with a number of development environments [Hafner and Breu 2009].

3.2.1.3 Lang / Schreiner (OpenPMF)

OUTLINE:

[Schreiner and Lang 2008] present a partial model-driven (MDS) methodology building on a security enforcement system (PMF/OpenPMF) for various distributed middleware platforms. The approach is strongly influenced by the work of [Lodderstedt et al. 2002], aiming to specify high-level security policies in a domain-specific language and transform them to lower-level policies that can be enforced by PMF/OpenPMF. Below we describe Lang and Schreiner's the early work leading up to the PMF/OpenPMF system, and review the current form of the methodology afterwards.

DETAILED DESCRIPTION:

Design: Lang and Schreiner's early work [Lang and Schreiner 2003] presents an approach to securing distributed applications based on decoupling security policies from the software, security technologies and underlying distributed platform (middleware). The approach loosely follows the MDA philosophy, aligning the use of security policies and the relevant transformations to the MDA modeling levels (views) by analogy. Security policies are specified in PDL, a custom policy language that can capture complicated authorization and authentication semantics, allowing for the specification of a number of security models. Conceptually, security policies are mapped via transformations to the equivalent security functionality present in the underlying platform. With respect to implementation, this mapping is realized by extracting platform or technology specific attributes (transformation aspect) and matching policies against them at run-time (policy enforcement aspect). Access is mediated on a per-object invocation basis by the platform or middleware and intercepted by a custom Adapter software component, which passes on the mediated invocation to a dedicated Policy Enforcement component. Policy enforcement occurs by decomposing the set of policies into a tree-like hierarchy (which, most importantly, can take into account authority delegation), using a set of Transformer components individual to each platform to extract situational attributes and matching policy rules accordingly. The realization of this approach is embodied in the Policy Management Framework (PMF) and later OpenPMF [Lang and Schreiner 2004].

This approach is extended to address the limitation of manually specifying security policies in [Schreiner and Lang 2008]. There the authors state candidly their experience in a larger, real-life project with using OpenPMF alone and specifying policies by hand: “[the human administrator] had to use a primitive trial and error approach: Start the system, wait for a policy violation, correct the policy, and go to start again! After about one week, the application was running without new policy violations, with a policy of about 1000 rules. The policy was far from optimal, and provided little assurance.”

In its extended (and now current) form the methodology strictly follows MDS principles by concurrently specifying security policies in SPL – a domain-specific language – together with functional system models as per MDA standards, transforming the security models to PDL policies and relying on PMF/OpenPMF for

enforcement. This accomplishes Schreiner and Lang's aim of "correct enforcement of correct policies" [Schreiner and Lang 2008]. The policies themselves reflect the available solutions in the underlying platforms supported by OpenPMF. As [Schreiner and Lang 2008] point out, the correctness of transformations, platform security capabilities, additional security functionality etc. all have to be considered, making the approach useful for security experts but not for the average developer. The emphasis of the methodology is thus on the design-level specification and run-time enforcement of security.

DISCUSSION:

Since Lang and Schreiner's work has close commercial ties including "patent-pending methods and algorithms" [Schreiner and Lang 2008] (one assumes for the high-level SPL policy transformations) the relevant publications are sparse in detail, making it very difficult to fully assess the merits and deficiencies of the approach.

As with Lodderstedt/Basin's MDS ([Section 3.2.1.1]), Lang and Schreiner's approach has been trialled on real projects. [Schreiner and Lang 2008] reports on their use of SPL and OpenPMF in an industrial air traffic simulation system, and note the positive outcome of producing a large number of valid PDL policies from a small set of high-level SPL policies.

The methodology is supported by an Eclipse-based tool-suite for specifying SPL policies, editing resulting PDL policies and performing automatic transformations. OpenPMF, currently in version 2.0, supports a wide variety of middleware platforms.

3.2.1.4 Gunawan / Hermann / Kraemer (security-enhanced SPACE)

OUTLINE:

[Gunawan et al. 2009, Gunawan et al. 2011] present a partial model-driven methodology applicable to distributed systems as an extension of the SPACE development approach [Kraemer 2008, Kraemer et al. 2009].

SPACE is an MDE approach in which system functionality is specified using collaborative building blocks (or simply collaborations), which are higher level service models capturing the distributed functionality of a number of collaborating components, notated using UML 2.0 collaboration (static) and activity (dynamic) diagrams. SPACE collaborations can be composed hierarchically and combined in a "LEGO block" fashion to produce more complex functionality. The resulting architectural models can be checked due to their formal, temporal logic based semantics and transformed automatically to sets of components carrying the collaboration/service functionality, which are further transformed to Java code artefacts.

DETAILED DESCRIPTION:

Design: Guanawan et al.'s methodology extends SPACE by adding an additional security analysis stage and proposing a library of security building blocks that can automatically be integrated into a system's functional models using tool-support. In the early work of [Gunawan et al. 2009], the process for introducing security consisted in: (1) creating a functional system design; (2) analyzing the design for valuable assets, assessing their value; (3) identifying threats to those assets; (4) assessing associated risks using a simple but effective matrix relating asset scores to

threat scores; and (5) designing and integrating security countermeasures in the form of security collaborations/building blocks. The process is iterated between domain and security experts for each introduced solution.

In recent work on the methodology [Gunawan et al. 2011], emphasis is placed on automation via tool-support, thereby removing the need for domain/security expert iterations, but introducing a set of requirements on the original architecture for security integration that makes for a significantly more rigid approach. The current process replaces step (5) above with the following four steps: (5') identifying collaborations requiring the introduction of security countermeasures; (6) automatically checking that a system's architecture fulfills certain pre-defined collaboration constraints for a given countermeasure; (7) automatically composing countermeasure security building blocks with identified collaborations via graph-based, information-flow algorithms; and (8) semi-automatically integrating the resulting secured collaborations into the functional models.

The use of security building blocks requires run-time support, and the approach currently considers only communications security between two participants (client/server), making its scope rather limited.

DISCUSSION:

The generative nature of SPACE and the high degree of automation in Gunawan et al.'s methodology is one of its most attractive factors. Since SPACE collaborations/building blocks are essentially template-like patterns or partial architectures containing specific functionality, they can be re-used across different systems. Nevertheless, no guidance is provided in actually choosing particular security building blocks. In both the older and most recent variants of the methodology security is integrated after, not during, functional modeling, which implies that security introduction is effectively, despite the aim of early security integration, an afterthought to the design stage. With respect to implementation, it is not valid to assume as [Gunawan et al. 2009] do that the SPACE model transformations and subsequent code-generation preserve all security properties. It remains to be seen in future work whether the methodology can encompass security solutions other than TLS-based secure communications in a useful and relatively constraint-free manner.

3.2.1.5 Sánchez and colleagues (ModelSec)

OUTLINE:

Sánchez et al.'s ModelSec [Sánchez et al. 2009] is a partial model-driven (MDS) methodology applicable to general distributed systems, whose focus is on capturing security information and applying fine-grained model transformations across the development life-cycle to produce implementation-level security artefacts.

DETAILED DESCRIPTION:

Requirements analysis: In ModelSec, security introduction begins in the early development stages. During analysis, a domain-specific language (SecML) defined according to a general requirements metamodel is used for capturing security requirements. SecML allows common requirements and concepts to be represented graphically, helping to analyze security requirements, threats and goals in a single

model. The resulting requirement model elements are synchronized with a corresponding conceptual model (which similarly conforms to the general requirements metamodel) by mapping assets in the SecML model to concepts in the conceptual model.

Design: During design, SecML requirements models are transformed via a model-to-model (PIM-to-PSM) transformation to generate "skeleton" security design models that conform to another metamodel capturing security design decisions. More specifically, security design models constitute platform-dependent encapsulations of technology-related decisions for each mechanism corresponding to a security requirement in the requirements metamodel. The latter are elaborated and transformed once again via a PSM-to-PSM transformation to a set of platform-dependent security implementation models, which further specify the design models with respect to a given platform. A number of such implementation models can be created, according to different target implementation artefacts.

Implementation: For implementation, the security requirements, design decision and implementation models, together with a set of platform metamodels capturing details about particular platforms for each implementation model, undergo another model-to-model transformation to produce a set of generation models from which code artefacts, such as policies or configurations, can be generated directly.

Discussion: ModelSec is a fine-grained model-driven approach aiming to encompass the whole SDLC and support developers with security expertise to generate security configurations more easily. The models used in ModelSec, however, do not model security solutions, but security-related information – requirements, technology decisions, platform details – in the form of schemas, meaning that the alignment to MDS is rather loose. ModelSec security models are not strongly related to a system's functional models, and in particular, security design models are not related in any way to the corresponding functional design models. Capturing design decisions as models is an interesting approach, but embedding technical decisions in the related metamodel implies that it is liable to change as new technologies appear. Regarding the transformation process, we should point out that the SecML models and corresponding conceptual models are actually computation-independent models (CIMs), not, as treated in ModelSec, platform-independent models. The implications of this are that the approach does not encompass PIMs, but transforms CIMs directly to PSMs, which carry platform-independent characteristics. Since design models do not capture solutions, but decisions, and are not related to functional models, verification is not an option.

The methodology is accompanied by an Eclipse-based tool-suite for SecML modeling and model transformations.

3.2.2 Aspect-Oriented Software Development (AOSD) Methodologies

An aspect, used as a technical term, is a particular part or feature of a program or design that cuts across or is present in numerous separate modules or components of a system. Historically, the use of aspects as a central concept takes its roots in Aspect-Oriented Programming (AOP) [Kiczales et al. 1997] (see also [García et al. 2012] for

a security-oriented discussion), where parts of a program's functionality are localized in aspects and woven (composed or integrated) at particular join points specified either manually or automatically. The ideas from AOP were subsequently applied to higher levels of abstraction in the work of [Clarke and Baniassad 2005], [France et al. 2004a] and others to form the Aspect-Oriented Software Development (AOSD) paradigm, focusing "on the identification and representation of crosscutting concerns, and their modularization in separate units [aspects], as well as their automated composition into a complete system" [Jakob et al. 2009]. In such aspect-oriented approaches, each concern embodied by an aspect is woven into a base or primary model describing some part (or the whole) of a system's software architecture. A join point model defines where in that architecture aspects can be woven, while pointcuts determine a selection of possible join points. Depending on the approach, aspects can consist of pointcuts and/or valid join points and advice – the actual action or content of the aspect embodying the particular concern.

With respect to security, AOSD approaches allow developers to consider security concerns in isolation and weave them either into base design models to obtain integrated security-enhanced models or into implementation-level artefacts. AOSD methodologies are model-driven inasmuch as aspects are first-class models and aspect weaving is essentially a model-to-model transformation.

[Dehlinger and Subramanian 2006] survey several recent aspect-oriented approaches for securing software, among them [Georg et al. 2002a] and [Yu et al. 2005]. The former is treated without any detail, however, thereby not allowing a fair appraisal of Dehlinger and Subramanian's pointed criticisms. This is redressed by our review of the same approach in [Section 3.2.2.1] below.

3.2.2.1 Georg / Ray / France (AORDD)

OUTLINE:

[Georg et al. 2002a, Georg et al. 2002b, Georg et al. 2006, Georg et al. 2009, Georg et al. 2010], [Ray et al. 2004] and [France et al. 2004b] present a model-driven (AOSD) (primary class) and pattern-driven (secondary class) security methodology based on Aspect-Oriented Modeling (AOM) for securing general software systems. The approach is also applicable to distributed systems inasmuch as Georg et al.'s aspects can, at least theoretically, be constructed and/or tailored to account for more specialized security scenarios.

In Georg et al.'s approach, security solutions are expressed as aspects, which are treated as patterns based on the previous work of [France et al. 2002, France et al. 2004a]. The latter work is an attempt to make rigorous specifications of design patterns in UML by the use of role models (approximately meta-model level templates) and accompanying OCL constraint templates, which restrict the scope of possible pattern realizations in terms of model elements. Such rigorous specifications are useful because they determine the degree to which the design patterns (i.e. their models) can be varied. We should note that Georg et al.'s aspects [Georg et al. 2002a, Georg et al. 2009] are not to be confused with security patterns (cf. [Schumacher et al. 2006]), to which they are nevertheless related (see [Section 3.4] for more details). France et al.'s specifications are unable to capture the complete semantics of corresponding solutions, and, unlike security patterns, do not aspire to provide guidance or models of tried-and-tested solutions. In this sense, the aspects used by

[Georg et al. 2002a, Georg et al. 2009] resemble the patterns used in the SECTET approach, which are likewise treated as first-class models, or in other pattern-based methodologies using template-like patterns ([Section 3.4.1]).

DETAILED DESCRIPTION:

In the research group's early work [Georg et al. 2002a, Georg et al. 2002b], the approach can be described by a limited sequence of security activities: identifying necessary security aspects via a simple table mapping possible assets, attacks and corresponding countermeasures, and incorporating the countermeasures (aspects) by weaving them into a system's primary model. The (representative) countermeasures table is a good first attempt to identify security solutions, but is not a complete strategy.

With Georg et al.'s work on AORDD (aspect-oriented risk-driven development) [Georg et al. 2006, Georg et al. 2010] and the group's work on formal verification, the methodology is complemented by an attack modeling strategy, Alloy-based verification and trade-off analysis. In this latest version of the methodology, presented in [Georg et al. 2009, Georg et al. 2010], security introduction becomes an iterative six-part process, consisting of six consecutive steps: (1) determination of threats and risks using CORAS [Lund et al. 2011] alongside the elaboration of the primary model; (2) modeling of high-risk attacks related to threats as aspects; (3) weaving of the attack aspects into the primary model and formally analyzing the resulting misuse model to evaluate the attack's damage potential and hence to determine (resultant) security requirements, with the latter being mapped immediately to countermeasures; (4) introduction of relevant countermeasures as aspects into the primary model, producing a security-treated model; (5) re-weaving of the attack aspects into the security-treated model, followed by automated formal analysis, to evaluate the success of the security solution in mitigating the attacks; (6) calculating a fitness score for the solution using a BBN (Baysean Belief Network). The process is repeated for each attack, to decide and trade-off between different solutions, and to discover newly introduced discrepancies such as solution conflicts. The approach is only applicable to the design stage. Below we describe the steps of this process in more detail.

Design: The first step in Georg et al.'s methodology is to use a security assessment strategy to evaluate the risks to an existing primary (system) model. CORAS is proposed for this purpose in [Georg et al. 2009], whose use results in a set of resultant security requirements embodied in the form of "treatments". The subsequent creation of misuse models and their analysis (discussed further below) effectively refines the risk (threat) model from CORAS in view of the primary model, allowing a more detailed set of security requirements to be specified. The overall approach is valid even if CORAS is replaced by another threat modeling/risk analysis method, making the methodology more flexible in this regard.

Following the determination of resultant security requirements, high-risk attacks are modeled as aspects, woven (as per the earlier work on the methodology in [Georg et al. 2002a, Georg et al. 2002b, France et al. 2004b, Ray et al. 2004]) individually into the primary model (one attack per iteration of the methodology) and verified, as explained below for security aspects, to produce a misuse model. If the verification results show that the attack is feasible, a relevant countermeasure is selected based on expert advice.

Currently, the methodology does not provide guidance or a structured approach for the selection activity. The range of countermeasures are those expressible as generic security aspects, which, judging by their availability, must be developed by security experts impromptu. Aspects appearing in the related literature include those for TLS secure communication [Georg et al. 2006, Georg et al. 2009], simple authentication and auditing [Georg et al. 2002b] and access control [Ray et al. 2004]. We should point out in this respect that [Georg et al. 2010] indicate their intent to create a repository of aspects associated with particular security properties, as well as to formalize dependencies between attacks and security solutions, which suggests that future improvements of the methodology may include features to support a more structured solution identification process using pre-defined solutions.

Once chosen, the countermeasures are subsequently woven into the primary model (from the previous methodology iteration, if applicable) to produce (a further) security-enhanced primary model. Aspect weaving is a two-phase process (following [Ray et al. 2004]), in which, firstly, the generic aspect (a role model-based pattern) is realized by manually mapping or binding the abstract roles (structural and behavioural) to existing primary model elements, if possible, or to new model elements, in both cases following the semantics and syntax (naming, concepts etc.) of the primary model, to create a context-specific aspect; and secondly, semi-automatically merging the context-specific aspect into the primary model, resolving conflicts where possible. Merging (UML) model elements can be accomplished via algorithmic strategies based on names [Straw et al. 2004] and signatures [Reddy et al. 2006], which can be tailored via composition directives, e.g. specifying which elements override others, rules for removal of elements etc. – see [Straw et al. 2004]. From a high-level perspective, the weaving process corresponds well to Fleurey et al.'s matching and merging model composition stages for AOM [Fleurey et al. 2008]. The aspect weaving process can also be thought of as a very rigorous but somewhat rigid form of pattern instantiation.

Once the aspect has been woven and all conflicts resolved, the attack aspect developed earlier can be re-woven into the security-enhanced primary model to determine whether the most recently woven countermeasure is effective in stopping the corresponding attack. This determination is accomplished with the aid of formal verification, which was also used to analyze the misuse models in the earlier stages of the methodology. The security-treated model (in UML) is translated into the Alloy formal language by simplifying the static UML models, writing OCL specifications of their methods (dynamic behaviour), and subsequently translating the simplified models together with accompanying OCL constraints using the UML2Alloy tool, see [Georg et al. 2009] for references and details. An Alloy Analyzer is used on the resulting model to automatically check for instances in which a particular condition – specified in OCL – fails.

The whole analysis process requires detailed specifications of the models such as would usually be obtained at the detailed design stage (complete method signatures, attributes etc.), which is slightly at variance with the aim of the methodology as stated in [Georg et al. 2009]: “our method can expose bugs and security issues of a system early in the development process before the model is refined enough to be implemented”. If verification is performed prior to detailed design, however, further refinement of the models may invalidate the results.

One of the most salient features of Georg et al.'s AORDD methodology is the trade-off analysis performed between different security solutions, following the Alloy-based verification process. A BBN is used to calculate a fitness score for a particular security countermeasure, which is "a measure of the degree that a particular security solution meets the security, development, project and financial constraints of the project" [Georg et al. 2010]. Each of these constraints, as well as the result from the verification (whether a solution mitigates an attack or not), is specified in the form of a number of parameters used as input to the BBN. The fitness scores can be used to guide the selection of which solution for a given attack should be chosen for implementation. This has the important ramification that a security solution failing to mitigate an attack can nevertheless be chosen for implementation , e.g. the case study in [Georg et al. 2010] based on its lower cost of implementation, project schedule constraints etc.

Trade-off analysis is a best-effort, semi-automated approach, but one in which subjectivity cannot be removed entirely, since the assignment of values in the BBN as well as its setup is a subjective process based on experience and expertise. Therefore, even more than the Alloy verification results, which are similarly best-effort, the fitness scores are not absolute indicators. Nevertheless, the approach offers developers support in making sound decisions in the face of numerous other project factors, and, given a correctly set up BBN, can be used by non-experts. Considering the impact resulting from the introduction of security aspects to other non-functional properties is a goal for future work in [Georg et al. 2010], which may be a valuable extension. Preliminary work addressing performance is presented in [Houmb et al. 2011].

DISCUSSION:

Overall, the methodology presented in [Georg et al. 2002a, Georg et al. 2009, Georg et al. 2010] is a design-focused approach distinctive for its use of rigorous security modeling and semi-automated trade-off analysis. While the trade-off analysis in particular is a unique and valuable feature, the methodology as a whole has several major drawbacks. Besides those already noted throughout the review, Georg et al.'s approach does not appear to be scalable to larger systems, and as the authors themselves state, "repeated generic aspect instantiation, composition [weaving] and analysis can be tedious and error-prone" [Georg et al. 2009]. In many ways the introduction of security properties is also experimental, or ad-hoc: "it is important to continue integrating security mechanisms and analyzing the resulting security-treated system […] since some mechanisms may interfere with each other. When such conflicts arise, the designer can integrate alternative solutions until a usable combination is identified through achieving acceptable analysis results" [Georg et al. 2009]. The efforts required in aspect weaving, verification and trade-off analysis make the approach unrealistic in cases where more than just a (very) small number security solutions must be introduced.

With respect to guidance, in all the case studies the authors know a priori how to best construct and weave a security aspect into a system's primary model, but it never becomes clear how this is known, nor how the strategy can be generalized. Finally, from a more fundamental perspective, the application of the AOM paradigm as done by [France et al. 2004a], in which "fixed-to-measure" models of security solutions are

rigorously specified, results in a need to consider fixed, automated weaving strategies via algorithms or the like carefully matching each UML model classifier, association etc., making for a rigid approach overall, in which the emphasis is on automation instead of guidance. While automation is a helpful feature, it is not possible to fully automate the securing of a system. The current lack of guidance and high expertise and effort requirements make Georg et al.'s methodology seem somewhat difficult to use.

The methodology is supported by a suite of tools: *ArgoUML* and *UML2Alloy* for the verification process [Georg et al. 2009], *Hugin* for the decision trade-off analysis [Georg et al. 2010] and custom tools for aspect weaving [France et al. 2004a, Mekerke et al. 2002]. It may also be possible to support the aspect weaving strategies using the KerMeta-based tool developed as part of the work in [Reddy et al. 2006]. More than other methodologies, the numerous descriptions of algorithms and processes of the AORDD approach indicate that using a software tool is essential for its practical use.

3.2.2.2 Mouheb and colleagues

OUTLINE:

[Mouheb et al. 2009, Mouheb et al. 2010] present a partial model-driven security methodology, similar in many ways to Georg et al.'s work described above. Mouheb et al. closely follow the AOSD paradigm by encapsulating security solutions in aspects using UML-based domain-specific languages, automatically weaving the aspects in base or primary models and subsequently (as a planned stage) generating code artefacts. Like Georg et al.'s AORDD, the approach is applicable to distributed systems on account of its support for constructing relevant, specific aspects.

DETAILED DESCRIPTION:

Design: One of the aims of Mouheb et al.'s approach is to lower the expertise required of developers in implementing security by localizing that expertise to security aspects. For this purpose, the approach provides a domain-specific language (DSL) in the form of a UML metamodel, together with a pointcut language, allowing both the structural and behavioural properties of aspects to be specified in the form of changes to the primary model (adaptations) together with the places in that model where the aspect can be woven (pointcuts). The resulting aspect specifications define parameterized model templates similar to the template patterns of [France et al. 2004b], in which the embodied security solutions can be re-used in different contexts. The simultaneous specification of valid join points in the primary model within aspects is a significant advantage over France et al.'s aspects [France et al. 2004b], since this information helps to better define the scope and applicability of a security solution. It should be pointed out that Mouheb et al.'s aspect and pointcut languages are not specific to security, but define general languages for specifying aspects in UML.

During design, developers select appropriate security aspects and specialize them using tool-support by choosing which elements (parameters) of the aspects map to which primary model elements according to the aspect-specified pointcuts. This can be likened to mapping pattern roles to valid model elements, and is very much analogous to Georg et al.'s context-specific aspect instantiation (see [Section 3.2.2.1] above). The mapping to model elements chosen by the developers defines a set of join

points in the primary model around which the aspects are woven. The weaving process is defined in terms of model transformations written in QVT and is fully automated. Woven aspect behaviours appear in sequence diagrams as composed fragments that hide the underlying complexity, something especially useful in long security protocols. Unlike the similar process in Georg et al.'s approach of weaving using strategies, Mouheb et al.'s aspect weaving does not appear to address issues of conflict resolution. This may be problematic in instances where an aspect adaptation requires the addition of a class already contained in the primary model.

Requirements analysis: The specification of security requirements is mentioned as being part of the overall approach in [Mouheb et al. 2009], where [Jürjens 2005a] is cited for further details. While UMLsec stereotypes do indeed appear in some of Mouheb et al.'s examples [Mouheb et al. 2009], the precise role of UMLsec, or requirements specification in general, remains unclear, which suggests that a security requirements stage is not an essential part of the methodology.

DISCUSSION:

Mouheb et al.'s work shares many similarities to Georg et al.'s AORDD approach with respect to aspect representation and weaving, arguably making improvements in both areas. One important element omitted from Mouheb et al.'s methodology, or at least its presentation in the literature, is a structured selection stage for the security aspects. Currently this selection requires security expertise, thus weakening the initial aim of the methodology to be developer-friendly. Missing support for conflict resolution in aspect weaving and a lack of security verification also leave open questions regarding the validity of the final security-enhanced models.

The methodology is supported by an IBM Rational Software Architect plug-in, used for aspect specialization and weaving.

3.2.2.3 Jakob / Loriant / Consel (DiaAspect)

OUTLINE:

[Jakob et al. 2009] present a partial model-driven (AOSD) methodology to securing distributed (pervasive) systems based on the DiaSpec/DiaGen generative software development framework [Cassou et al. 2009, Jouve et al. 2008].

DETAILED DESCRIPTION:

Design and Implementation: In the DiaSpec/DiaGen approach, a system's software architecture is first modeled using DiaSpec – a lightweight ADL tailored for distributed (pervasive) systems. The architecture description is subsequently used to generate a typed programming framework with accompanying run-time support in the form of a custom middleware layer (DiaEnv) abstracting away underlying platform details.

To introduce security concerns, [Jakob et al. 2009] closely follow the AOSD paradigm and superimpose an aspect language called DiaAspect on top of the DiaSpec ADL that, together with its attendant join point model, associates pointcuts (effectively join point filters) with advice (security solution code) to allow the specification of a range of security concerns. DiaAspect's join point model is defined relative to DiaSpec's connector constructs encompassing synchronous RPC, publish-

subscribe and session-based semantics, according to inter-component communication behaviour (e.g. a component publishing events).

In its particulars, DiaAspect is constructed to closely resemble AspectJ. DiaAspect's pointcut syntax and semantics allow for the specification of which join points should be selected for aspect weaving, while aspect advice (i.e. security solutions) is specified using Java. Aspect advice can encompass an arbitrary range of crosscutting security concerns. Since DiaAspect is built on top of DiaSpec, the advice in the aspects can refer to the architectural constructs of the ADL; however, since the latter are compiled by DiaGen to an implementation framework, the advice can also contain code referring to the associated (to-be-generated) framework via a specific API. Thus DiaAspect allows advice to refer to both architectural and implementation artefacts, making it quite powerful, albeit complex to use.

The process of introducing security properties consists in the (ad-hoc) modeling of additional security components and connectors, and the specification of security aspects (advice together with appropriate pointcuts) in DiaAspect at the architecture-development stage. Instead of weaving the DiaAspect aspects into the primary DiaSpec architectural models, however, they are translated to AspectJ aspects and, transitioning to the implementation stage, woven into the actual code-level artefacts, namely, the generated programming framework and the DiaEnv middleware layer. Since aspect weaving is performed on a codebase with known structure, precision can be ensured.

DISCUSSION:

Overall, Jakob et al.'s methodology is unique for its use of an aspect language on top of an ADL within a generative framework. The close adherence of Jakob et al.'s approach to the AOSD paradigm, coupled with strong similarity in syntax between DiaAspect and AspectJ helps to place the methodology in a more familiar context and thus increase its usability. Nevertheless, the absence of any guidance implies that security and domain expertise are required both in developing the security aspects and in determining via pointcut specifications where they should be woven. A real problem in this respect is that while the security solutions contained in the aspects can be verified independently, an overall correct security implementation relies on the correct specification of pointcuts, which is a potentially error-prone task even for experts. Some form of (security) verification would thus be required to make the approach more robust. Another, related, problem stems from the intermixing of architectural and implementation decisions in aspects. In particular, this may lead – at least on occasions – to a bottom-up (implementation-driven) as opposed to a top-down (design-driven) approach to security, in which developers effectively code security solutions at the design stage.

The DiaGen/DiaSpec development framework comes with an Eclipse plugin, but it is not clear whether this is used in Jakob et al.'s methodology, which otherwise does not appear to be supported by any tools.

3.2.3 Other Model-Driven Methodologies

There are several other model-driven methodologies for securing general and distributed software systems that deserve mention.

[Menzel et al. 2009, Menzel et al. 2010] and [Menzel and Meinel 2009, Menzel and Meinel 2010], for example, specialize Model-Driven Security ([see Section 3.2.1.1]) to SOA-based systems using SecureSOA, a security modeling language inspired by SecureUML. Security requirements can be captured at the business process modeling level in platform-independent models and transformed to web-services policy specifications via a set of security configuration patterns (WS-specific security patterns formalized using a custom DSL). We do not cover Menzel and Meiner's work, since we have already reviewed a comprehensive model-driven approach for SOA-based systems (SECTET), and, furthermore, SecureSOA is a direct application of the principles outlined in [Basin et al. 2006].

[Yu et al. 2005] propose a partial model-driven AOSD (primary class) and architecture-driven (secondary class) methodology using the SAM ADL [see Dai and Cooper 2007]. A high-level SAM architectural model is separated into a base architectural model and a security aspect model containing the various security solutions in aspects. The latter are not discussed in detail, however, and there seems to be no structured approach to creating or selecting aspects. Only brief indications are given regarding aspect weaving, which, together with the extensive use of formal methods and lack of any tool support, implies a technically sound but unrealistic approach.

[Reznik et al. 2007] take a very similar approach to [Schreiner and Lang 2008], using Qedo (a CCM implementation) and OpenPMF combined with MDS principles for specifying high-level security policies. The emphasis of Reznik et al.'s work is on the creation of appropriate tool-support.

A more architecture-driven approach can be found in the work of [Oladimeji et al. 2007], who present a policy-based, software architectural perspective on secure model-driven development via a UML 2.0 extension. The emphasis, however, is on general software systems.

3.3 Architecture-Driven Methodologies

In contrast to model-driven engineering approaches, architecture-centric development approaches [Taylor et al. 2010] concentrate on a system's software architecture as the main artefact of importance. An architecture consists of more than a collection of models: it embodies a set of principal design decisions and determines the overall structure and behaviour of a piece of software in relation to various views [see Bass et al. 2003]. Accordingly, the aim of architecture-driven security methodologies is to create a secure software architecture: an architecture with embedded security properties.

As with their model-driven counterparts, architecture-driven security approaches generally begin with an architectural model of a system and annotate it with security properties to produce a security-enriched model (or security design model in the terms of [Basin et al. 2006]). The modeling languages used to achieve this security enrichment, the emphasis on architectures as opposed to first-class models and the absence of model transformation distinguish architecture-driven from model-driven methodologies. While model-driven approaches make use of extensions of UML, architecture-driven approaches typically use a precise architectural description of a system produced by ADLs and/or formal modeling methods. In architecture-driven methodologies, the verification of security properties often assumes a role of greater

importance. Due to their nature, such methodologies generally concentrate on the design stage of the SDLC, without much regard for the remaining development life-cycle.

[Ren 2006] has surveyed a number of architectural-level approaches to introducing security, placing emphasis on modeling languages and older methodologies using formal methods such as [Moriconi et al. 1997] (architecture refinement through mappings) and [Deng et al. 2003] (Petri nets and temporal logic) aiming to implement security at the architectural level. We omit these, and review several more recent approaches instead.

3.3.1 Ren / Taylor

OUTLINE:

A representative (partial) architecture-driven security methodology is described in the work of [Ren and Taylor 2005], [Ren et al. 2005] and [Ren 2006]. The essence of the approach consists in extending an existing ADL to support security constructs and allow reasoning about security – specifically access control – at the architectural level via architecture analysis techniques [Taylor et al. 2010].

DETAILED DESCRIPTION:

Design: The substrate language used in Ren and Taylor's approach is xADL [Dashofy et al. 2005], an extensible XML-based ADL, whose semantics are enriched via a set of general security concepts to provide direct support for modeling subjects (users on whose behalf software components execute), principals (essentially the identity/ies of a subject), resources (protected entities), privileges (approximately subject permissions), safeguards (approximately object permissions), security policies (in XACML) and contexts.

In many ways, the resulting modeling language – Secure xADL – is a software architectural version of Lodderstedt et al.'s SecureUML. Unlike [Lodderstedt et al. 2002] and later [Basin et al. 2006], Ren and Taylor do not seek to generate code from their architectural models, but, as noted above, to analyze and reason about security properties prior to the implementation phase.

Secure xADL can model access control concerns at the architectural level by specifying "contracts" between software components via their associated access permissions and privileges. The security semantics are applied and enforced exclusively through connectors (for which see [Shaw 1996]), which are treated as first class citizens in all xADL architectural models. In this context connectors act as both policy enforcement and policy decision points [Gollmann 2011], and, above all, as multiple (or single) reference monitors. Higher-order connectors can be composed from multiple other connectors; connectors can be replaced (potentially dynamically); and connectors can be created from component specifications, to perform a (potentially) wide variety of security functions. Since xADL supports component compositions, the approach can account for multiple sub-architectures and extend access control to their constituent components and connectors. Since the approach works at the architectural level, it can also determine "architectural security" concerns, such as component creation and destruction, execution etc.

Analyzing the resulting secured architecture is achieved by using an algorithm that parses the Secure xADL model (which is represented as XML text) and

validating the semantics. In doing so, the context for each component and connector must be taken into account and permissions and privileges aggregated to form an overall perspective on the access control (on a per component/connector basis). The algorithm provides assurances that the modeled access control is correct and consistent.

DISCUSSION:

Ren and Taylor's approach offers a comprehensive methodology for incorporating access control semantics at the architectural level. The approach has relevance to distributed systems inasmuch as it places preeminent emphasis on software connectors. However, the approach has several limitations.

Like [Lodderstedt et al. 2002] and [Basin et al. 2006], only access control semantics are demonstrated, although Ren and Taylor proceed further in allowing the semantics of multiple access control models to be captured – including those based on the traditional Matrix model (ACLs and capabilities), roles (RBAC) and trust-management (TMBAC). Nevertheless, delegation is not considered. In practice, access control policies are specified as part of the connector in XACML, limiting the range of access control models to those supported by the standard. Attacks and threats are also not considered, placing reliance on the security expertise of the architects. The largest drawback of the approach, however, is in its level of abstraction. Using connectors as security safeguards does not allow for the specification of detail, e.g. whether the connector is a firewall, a filter or some form of security proxy. Moreover, stipulating that security functionality should be placed in connectors is equivalent to stating that security should be placed in "special components", which provides little practical help. A case study using a simple distributed (military) coalition application demonstrates the connector-centric approach by replacing a group of components and connectors with a single secure connector, which, while useful from a conceptual point of view, does not appear to address the issue of secure design.

Overall, the approach can be conceivably extended to encompass authentication and secure communications, but it is not entirely clear how easy it would be to encompass other security properties, and how many or what extensions (if any) would be required to Secure xADL and/or necessary verification algorithms.

Regarding ease of use, Ren and Taylor's methodology is quite easy to use, but requires a security expert due to the lack of support in modeling security properties. With respect to access control, the only currently supported property, the associated verification algorithm improves usability inasmuch errors can be detected and corrected early in the development process.

The methodology is supported by the ArchStudio development environment [see Dashofy et al. 2005]).

3.3.2 Ali / El-Kassas / Mahmoud

OUTLINE:

[Ali et al. 2009] present a partial architecture-driven methodology based on threat modeling, formal architectural modeling and formal verification approaches. The methodology builds on the work of a number of other architecture-driven methodologies such as [Deng et al. 2003] (see also [Ren 2006]), [He et al. 2002] and

the threat modeling approach of [Myagmar et al. 2005]. Thus, it is representative of a range of formal architecture-driven approaches.

In Ali et al.'s approach, the ultimate aim is to construct a software architecture in which a number of security properties are embedded and verified to be correct. This is achieved by constructing a high-level architectural model, performing threat modeling to produce countermeasures, translating the countermeasures to formal models which are fitted back into the architecture and verifying the architecture using model checking. The approach applies only to the early architecture development stage, and only to SOA-based systems.

DETAILED DESCRIPTION:

Design: The methodology proceeds by first constructing a high-level architectural model of a system using Service-Oriented SAM (Software Architecture Model – [Dai and Cooper 2007]) – an ADL utilizing Petri nets and temporal logic as the underlying formalisms, tailored for web-services [Fu et al. 2006]. Petri nets in particular can be seen as modeling the information flow through the system. Threat modeling is then undertaken based on the architectural model, where the threats to vulnerable assets and access points are categorized according to STRIDE [Swiderski and Snyder 2004] with the additional extension that "architectural" mitigations (countermeasures), logical constraints and property specifications can be captured as well (using LTL – linear temporal logic – and informal descriptions). Security requirements resulting from the threat model are thus immediately mapped to particular countermeasures, which are translated to equivalent Petri net behavioural models (cf. [Deng et al. 2003]), and added to the architecture model as actual components to elaborate it further. Constraints link the new components into the architecture. Unless the actual mitigations are COTS components (modeled as "black-boxes" with just interfaces/ports), the threat modeling is re-iterated to refine their internal structure. Once the architecture with its components, connectors and constraints is completed, the architectural model is translated into an SMV (Symbolic Model Verifier) ready form. The security properties captured by the threat model (in LTL) are similarly translated to ensure that the properties hold in the presence of the corresponding threats (translated as a set of parameters influencing the model-checker). The counterexamples generated by the SMV model-checker are used to refine the architecture and the threat model iteratively.

[Ali et al. 2009] do not clarify whether the applicability of the approach to SOA-based systems extends beyond the ADL used (Service-Oriented SAM).

DISCUSSION:

Although Ali et al.'s approach is valid, offers justification for the introduced security properties using threat modeling and offers assurance via model-checking, the trade-off in complexity is not to the methodology's advantage. Extensive and almost exclusive use of formal techniques renders the methodology inaccessible to all but the most well-trained development teams. Besides expertise in formal modeling methods, a great deal of security expertise is demanded to specify the necessary security properties in the first place, without any guidance. While LTL is general and expressive, it only allows for low-level, technical policies to be captured. It does not appear realistic to apply Ali et al's methodology, and indeed any methodology with

such a heavy use of formal methods, to large-scale and complex distributed systems; however, in small, critical applications, such approaches can help to provide the necessary assurance that certain security properties hold, at least theoretically, prior to implementation.

In the context of formal verification, the applicability of the approach only to architecture development seems to be a limitation: the guarantees that certain security properties hold may be broken during detailed design, for example.

Besides the SMV model-checker, there does not appear to be any tool-support for the methodology.

Related approaches relevant to distributed systems that concentrate on building and verifying an architecture with embedded security properties include [Deng et al. 2003], [Yu et al. 2003] and [Yu et al. 2004], all of who similarly employ SAM as a basis, but without suggesting any form of adversary modeling.

3.3.3 Other Architecture-Driven Approaches

Other, smaller-scale approaches that can be classified as architectural include [Schneider 1999], who considers security architectures abstractly in the context of information flow domains. Schneider's approach is surveyed in [Sachitano et al. 2004], who points out the unrealistic assumptions entailed by information domains when constructing actual systems.

[Jensen 1998] suggests securing distributed application architectures by considering component and connector security. Although the approach seeks to “integrate security specifications into the software architecture”, the (potential) solutions are described in the most general terms. Contrary to the stipulation that the approach applies to distributed applications, there is very little (if anything) specific to that setting, excepting several brief suggestions regarding access control (e.g. using domains and pseudo-users).

[Whitmore 2001] adheres to the Common Criteria (CC) and proposes a methodology in which security services are expressed as distinct subsystems, one for each class of security issues (e.g. authentication, authorization etc.). In this respect Whitmore's proposal resembles the [ISO/IEC 7498-2 1989] security architecture, transposed to meet the CC guidelines, as well as other methodologies discussed earlier which employ similar (generic) security reference architectures. It is not, however, altogether clear how Whitmore's separate security subsystems should be integrated with a system's functionality.

[Shin and Gomaa 2007] present a methodology in which the system is modeled using components and connectors, and afterwards made secure in some sense by evolving non-secure connectors to secure components and connectors "when the application requires security services". Only application level security is treated: the network and underlying operating system must be secured in some other way. Also, only a small range of well-known, standard security mechanisms are adduced when describing the process of evolution from non-secure to secure architectural constructs.

Finally, [Sachitano et al. 2004] argue that a software system can be made secure simply by good design or finding a correct architectural style that will ensure greater system security. The authors adduce qmail as an example of a secure application (cf. [Hafiz et al. 2004]), proposing that qmail has an architecture which is secure, since there have been no vulnerabilities found in the program. The design decisions taken

by the qmail developer, however, such as “don't parse, do as little as possible” etc., are far too generic to be of use in particular situations or to dictate the structure or behaviour of a software system; principles such as keeping everything simple are not sufficient in guiding the system architect to create a more secure system, and do not constitute a universal approach.

3.4 Pattern-Driven Methodologies

The concept of a pattern in design and architecture began with the work of the architect Christopher Alexander in “The Timeless Way of Building” [Alexander 1979] and related works. Alexander conceived patterns as solutions that can be applied again and again in different contexts in a generative fashion. The idea was taken into the world of computer science by [Beck and Cunningham 1987], but reached its mass popularization in the book by [Gamma et al. 1995], in the form of design patterns. Software patterns have proved to be extremely useful in software engineering because they help organize expert knowledge into a standardized format; they directly support re-use; and they provide a domain-specific vocabulary, helping to improve communication in project teams [Buschmann et al. 1996, Erl 2009, Schmidt 1995, Taylor et al. 2010].

Pattern-driven security methodologies are distinguished by their aim to use patterns in a significant way for the introduction of security properties throughout the various stages of the SDLC. Over time, diverse interpretations of the pattern concept have emerged, leading to several distinct classes of patterns and, accordingly, of security methodologies using those patterns. For our purposes, we broadly distinguish between only two classes of patterns (and hence methodologies): (1) template-like patterns, appearing as software components and/or architectures (treated as patterns); and (2) security patterns.

Security patterns [Yoder and Barcalow 1997, Schumacher et al. 2006, Fernandez 2009, Uzunov et al. 2012] are full inheritors of the software pattern concept. Like software patterns in general, security patterns capture successful secure designs in a generic form that can be applied or instantiated to produce solutions with well-defined properties. They build on the success of design patterns and software patterns more generally, encapsulating the experience and expertise of security professionals [Schumacher 2003, Steel et al. 2005] in a form accessible to the security non-expert [Fernandez and Larrondo-Petrie 2006], including guidance for application, trade-offs, implementation hints etc. Security pattern instantiation implies highly flexible customization of a solution strategy, often (though not always) accompanied by representative architectural models. We review methodologies based on security patterns in [Section 3.4.2].

In contrast to security patterns, template-like patterns, in the form of parameterized model templates or interface specifications for security components or whole security architectures, are re-usable software elements that can be instantiated by replacing parameters in a fixed fashion akin to languages type-instantiation. Template-like patterns can be thought of as specific instances of the representative architectural models appearing as part of some security patterns. We review methodologies based on components and/or architectures used as patterns in [Section 3.4.1].

3.4.1 Methodologies Using Security Components and/or Architectures as Patterns

3.4.1.1 Rosado / Fernández-Medina / Lopez (PSecGCM / SecMobGrid)

OUTLINE:

PSecGCM ([Rosado et al. 2008] – also referred to as SecMobGrid in [Rosado et al. 2011b]), is a comprehensive methodology for Mobile Grid systems attempting to support all the stages of the SDLC, including system inception and requirements, analysis [Rosado et al. 2009a, Rosado et al. 2009b, Rosado et al. 2010a, Rosado et al. 2010b], design [Rosado et al. 2011a, Rosado et al. 2011b] and implementation. Every stage defines certain activities tailored to grid systems, taking into account mobility requirements, to output tangible software engineering artifacts. The latter include all the standard documents such as project scope and technology specifications for the planning stage, requirements specifications, architecture descriptions etc. Two of the main features of the PSecGCM methodology are requirements traceability and software re-use, realized in the form of a repository of re-usable artefacts including use cases and models. The latter are introduced to reduce development time and effort as well as to ensure that tried-and-tested solutions (in the spirit of software patterns) are being utilized. Individual tasks or activities in each phase are iterated with continual improvement and verification before proceeding to the next phase of the SDLC. Overall, the methodology aims to construct an "engineering process that defines the steps to follow so that starting from the necessities [...] we can construct a secure grid system". The three cornerstones of the approach are a specialized UML profile allowing security fine-grained use-case modeling at the analysis stage, an abstract security reference architecture used in the design stage, and a large repository of re-usable artefacts. Below we outline how each of these are used.

DETAILED DESCRIPTION:

Requirements Analysis: The introduction of security in PSecGCM begins in the analysis stage [Rosado et al. 2010a], where the analysis model is created using a custom UML profile called GSecUC-profile discussed in detail in [Rosado et al. 2009a, Rosado et al. 2009b, Rosado et al. 2010b]. One of the most important features of the profile is that it allows the modeling of security by providing semantics for graphically representing misactors, attacks and security countermeasures. Modeling elements can be tagged for greater detail, for example, by specifying kinds of attacks, assets involved etc. Use cases and models from previous iterations of the activity or the whole process can be taken from and recycled into the re-usable artefacts inventory to promote re-use. In terms of the analysis activity, a number of tasks are specified which, with respect to security, consist of functional modeling with UML using a standard software process (RUP, OPEN etc.), identifying assets, considering threats and attacks to the assets and performing risk analysis. A number of security use cases, divided into general and grid-related, are the result of the latter task. The final analysis model consisting of an extended collection of UML diagrams with use cases, misuse cases and security use cases is carried over into the design phase.

Design: In [Rosado et al. 2011a] the authors present a "reference security architecture" for mobile grid systems with the idea of creating a service-oriented security infrastructure akin to SeAAS ([Hafner et al. 2009] - discussed previously in [Section 3.2.1.2]) to be used in the design phase [Rosado et al. 2011b]. Unlike SeAAS, the security reference architecture referred to here has an auxiliary methodological function, aiding the designer to relate security requirements to particular services. In essence, the reference architecture is a large architectural (template-like) pattern; its actual realization is dependent on the system architects. Rosado et al.'s reference architecture is thus somewhat similar in conception to the [ISO/IEC 7498-2 1989] Security Architecture or Cole's ECMA/TG9 security model [Cole 1990]. In terms of structure, the security reference architecture is divided into five layers with (approximately) each layer using the services of the layer below. Basic security services form the lowest layer (confidentiality, authorization etc.) while delegation, anonymity, trust and identity management and auditing services reside in the higher layers. The relationships between each of the services are considered in detail by Rosado and colleagues, as well as the interfaces of each individual service, determining its abstract operations. A very similar idea is proposed (but not realized) by [Naqvi and Riguidel 2004], who suggest a virtualized, pluggable security architecture for grid systems while also discussing mobility aspects. Although Naqvi and Riguidel's security architecture is more concrete, it is also less comprehensive in scope.

Rosado et al.'s reference security architecture is used in a central way during the design stage of the PSecGCM methodology, described fully in [Rosado et al. 2011b]. The design stage is separated into two parts: the construction of a software architecture and the construction of a security architecture, both of which must be integrated in some way and then documented according to IEEE 1471-2000. To create the security architecture, Rosado and colleagues mine and aggregate general security requirements for mobile grid systems from the open literature to form a tableau of requirements against which to consider countermeasures. Particular security use cases from the analysis phase are mapped to the set of requirements, which are then mapped, according to a predefined rule-set, to the security services of the security reference architecture in [Rosado et al. 2011a]. Only a subset of the reference architecture needs to be used for a given (general) security requirement, determined by the interrelations of the security services. For example, an "identify user" security use case can be mapped to the general requirement Authentication, which is mapped to the Authentication Service in the security reference architecture; however, the Authentication Service is related to services for identity management, confidentiality etc., thus combining a whole set of services to form the security architecture. The latter mapping process greatly helps with requirements traceability. Furthermore, mapping use cases to requirements to security services provides clear justification for the necessity of security mechanisms, and, if the reference architecture is considered as an abstract schema, the approach presents a basis for refinement to more concrete security solutions. The final security architecture is the largest interrelated subset of the security reference architecture obtained from the mapping of all possible security use cases via (general) security requirements to security services. As in previous activities of PSecGCM, the activity can be iterated for quality improvement.

DISCUSSION:

Unlike some of the other approaches surveyed so far, the PSecGCM methodology advocates good software engineering practices as a means to achieving higher quality software with less design flaws, which in turn implies improved software security. This is very much in accordance with the principles of secure software engineering [Mouratidis and Giorgini 2006]. Moreover, PSecGCM sees security required throughout the whole SDLC, not just a particular stage (as in, for example, Basin et al.'s Model-Driven Security), with use cases, generic security requirements and risk analysis helping to determine why and when a particular security solution is needed. Like other approaches though, the PSecGCM methodology has certain drawbacks and limitations.

One drawback of using abstract security architectures like Rosado et al.'s reference architecture lies in the fact that such architectures capture only general properties applicable to a broad class of systems. Rosado et al.'s reference architecture details interfaces and provides high-level requirements of the functionality of each security service as well as service interrelations, but does not consider actual service realization or internal design. According to [Rosado 2011b], "once we have defined the security architecture from the reference security architecture [by instantiating the relevant sub-architecture] […] it is necessary to design the final security architecture with UML diagrams, relationships between elements, protocols [etc.]", which is where PSecGCM stops short of offering support. Another, related drawback of using the security reference architecture pertains to integration: it is not altogether clear in what way "[the resulting security] architecture must be integrated with the software architecture to obtain a security software architecture with all the security aspects required by the users and the system" [Rosado et al. 2011a] in a systematic fashion. Understandably, security separation can be useful insofar that domain experts can concentrate on domain functionality rather than security aspects; however, the converse of this is also true: security designs dissociated from functional models or, as in PSecGCM, whole security architectures requiring some form integration with system functionality may, if not handled carefully, lead to design clashes and/or the traditional post-hoc introduction of security. It should be noted that such integration problems are symptomatic of using abstract security architectures generally, not just in PSecGCM.

In spite of these drawbacks, the reference security architecture is certainly a valuable tool in helping to build a global security architecture more efficiently, and the authors affirm its usefulness in their design case study (part of the GREDIA EU project – see [Rosado et al. 2011b]).

In [Rosado et al. 2011b] the authors suggest using MDS principles akin to [Basin et al. 2006] (see [Section 3.2.1.1]) to automate some aspects of the process, in particular transformation between different models, which may progress into a step towards automated integration.

PSecGCM is supported by SMGridTool, a custom CASE tool for building use case diagrams and managing the re-usable artefact repository during the analysis stage. Future work hinted in [Rosado et al. 2011b] may evolve the tool to encompass aspects of the design stage also (such as the security requirement mappings). The use of tool-support (essential due to the high number of development artefacts), clear traceability, re-usability features and standard development process features

(including the use of UML as a modeling language) makes the methodology easy to apply.

3.4.1.2 Schmidt / Hatebur / Heisel (SEPP)

OUTLINE:

Security Engineering Process with Patterns (SEPP) [Hatebur et al. 2007a, Hatebur et al. 2007b, Hatebur et al. 2008, Schmidt 2010a, Schmidt 2010b, Schmidt et al. 2011] is a comprehensive, pattern-driven methodology applicable to the early stages of the SDLC, with a strong security requirements engineering focus. The approach is based on a partitioning and detailed analysis of the problem domain and the environment of a software system using abstract and concrete security problem frames and the elaboration of a corresponding security architecture via the composition of pre-defined sets of components, where both the security problem frames and corresponding components are treated as patterns.

DETAILED DESCRIPTION:

Requirements analysis: In SEPP, the process of introducing security begins after a set of initial security requirements are defined [Hatebur et al. 2008] both prescriptively and arising from some form of adversary modeling. These requirements are matched with a set of Security Problem Frames (SPFs), which are based on Jackson's problem frames [see Côté et al. 2008]) and treated as patterns. Problem frames are a technique to capture and analyze a system or software unit's problem domain in detail. The software system or system unit itself (called the "machine") is placed in its environment, consisting of various physical and logical entities (domains) including users (biddable domains), data objects (lexical domains) and real-life entities (causal domains). SPFs in particular help to analyze a system's malicious and benign environments and capture the associated security requirements apart from any solution details. Concretized Security Problem Frames (CSPFs) on the other hand, make concrete, as the name implies, a particular SPF by embodying a particular security solution strategy, but without implementation details. By instantiating SPFs and CSPFs, a system's problem domain is effectively partitioned into a number of sub-problems and associated solutions. SPF and CSPF instantiation implies placing the software system or unit in a concrete context, whereby the problem-frame domains are mapped to context-specific domains. With respect to the development life-cycle, SPFs and CSPFs can be seen, respectively, as requirement and analysis stage artefacts.

SPFs and corresponding CSPFs in SEPP are organized as a pattern system by [Hatebur et al. 2007a, Hatebur et al. 2007b], in which relations such as "conflicts", "complements" and others are defined. The pattern system can be represented as a 2-dimensional matrix or table [Hatebur et al. 2008] to aid the selection of relevant SPFs and CSPFs. A detailed solution selection stage via instantiating CSPFs is one of the salient features of the approach and also one of its strong points. After the initial security requirements have been analyzed and the relevant SPFs have been instantiated, corresponding CSPFs are chosen according to the columns of Hatebur et al.'s matrix in [Hatebur et al. 2008]. Related, required or alternative CSPFs are also selected and instantiated according to the chosen SPFs (matrix columns) and corresponding CSPFs (matrix rows) to cover new and existing requirements in an

iterative fashion. After each CSPF has been considered, a threat and risk assessment is undertaken [Schmidt 2010b] to ensure that the CSPF's security solution cannot be bypassed or broken easily, and does not introduce any new vulnerabilities. Risk levels are considered according to assumptions about a system's environment, and, in the event of a weak solution for which the risk level is not tolerable, additional SPFs and CSPFs are instantiated. CSPFs and SPFs are thus iteratively and incrementally instantiated and assessed until all security requirements, both initial and those pertaining to the introduction of new solutions, are covered, thereby ensuring a good level of system protection.

The range of security solutions embodied in CSPFs focuses on confidentiality and integrity concerns and is relatively broad, covering communication and data security, as well as key distribution, security management and others. Extending this range, while requiring expertise, does not appear to be a difficult task, and would consist in fitting new CSPFs into the problem-frame pattern system (cf. [Hatebur et al. 2007b, Hatebur et al. 2008]) and evaluating conflicts, related patterns etc. for each existing SPF and CSPF.

As a next step, security requirements (and most notably confidentiality requirements) can be specified formally using CSP (Communicating Sequential Processes) ([see Schmidt 2010a] and [Schmidt et al. 2011] for details). In particular, the CSP specifications can be used to show that the instantiated CSPFs are step-wise refinements of the corresponding SPFs. Although formal specification is an important feature of SEPP, we also consider it to be an optional feature, and do not describe it here.

Design: Once all CSPFs have been selected, indicating the end of the analysis stage, a system's security architecture can be elaborated. Security architectures in SEPP are comprised of a number of Generic Security Components (GSCs) and associated helper components termed Generic Non-security Components (GNCs), both of which are (simple) abstract components treated as patterns. GSCs include components for encryption, password input, hashing, access control and random number generation; GNCs are general purpose and include user interface, storage and communication manager components. The GSCs and GNCs can be composed in various combinations – somewhat like "LEGO blocks" – to form Generic Security Architectures (GSAs), which are treated as larger-grained architectural (template-like) patterns, to satisfy the solution requirements embodied in particular CSPFs, and, from the point of view of problem frames, structure the CSPFs' machine domains. Each GSA also contains a manager (application) component, which acts as a facade to the interfaces of all the constituent GSCs and GNCs. [Schmidt 2010a] has defined a set of standard GSAs, each one corresponding to exactly one CSPF [see Schmidt 2010a], Ch. 14 for details). GSCs, GNCs and GSAs are all modeled using UML 2.0 composite structure and sequence diagrams.

During architecture development, individual GSAs as well as additional GSCs and GNCs are instantiated and composed to form a global security architecture. The approach is analogous to Hatebur and Heisel's problem-driven approach to constructing software architectures [Hatebur and Heisel 2009], in which a problem domain is recursively partitioned into problem frames and the solution to the individual sub-problems are composed and mapped to particular (pre-defined)

architectural structures. In SEPP, dependencies between GSAs are analyzed according to the corresponding CSPFs and identical components across different GSAs are merged where possible (e.g. manager components). Wrapper components are used to resolve interface conflicts. The resulting global security architecture represents a complete solution set to the solution requirements in all relevant CSPFs.

In many ways SEPP's global security architectures are flexible and customized versions of Rosado et al.'s security reference architecture ([Rosado et al. 2011a], see [Section 3.4.1.1] above), sharing the similar drawback in requiring some sort of integration with a system's functional architecture. In contrast, SEPP requires careful composition of individual GSAs, which may not always be easy to achieve, and may give rise to emergent properties – something likely to be addressed in future work (cf. [Schmidt and Jürjens 2011a, Schmidt and Jürjens 2011b]).

Detailed design: In recent work, [Schmidt and Jürjens 2011b] extend SEPP to employ UMLsec via the UMLsec4UML2 profile (see [Section 3.1.1]) in the specification of GSAs. Individual UML model elements of the GSAs are annotated using UMLsec's stereotypes, tags and constraints such that they embody the security requirements contained in the CSPFs. This allows UMLsec's advanced tool-support (static checks, automated theorem provers etc.) to be utilized for verification, and is a logical extension of SEPP to the detailed design stage.

The view advanced by [Schmidt 2010a, Schmidt 2010b] as well as [Hatebur et al. 2008] that standard security patterns [Schumacher et al. 2006] are mainly used during detailed design (and, as a consequence, may be used after the stages covered by SEPP) is erroneous, since security patterns can be applied during both the analysis and architecture-development stages of the SDLC (cf. [Section 3.4.2.1]). Rather, security patterns are alternatives to CSPFs and GSAs, or, from another point of view, CSPFs and GSAs can be thought of as direct analogs of abstract and concrete security patterns respectively (cf. [Fernandez et al. 2008]).

DISCUSSION:

Overall, SEPP is a thorough security methodology with a strong focus on the problem domain and a detailed approach to selecting appropriate security solutions. The use of problem-frames and components as patterns promotes a flexible approach to introducing security, and helps to consider security properties within a system's unique context.

Although there is a good deal of formality associated with SEPP's SPFs and CSPFs, they are not difficult to apply. If the formal specification stage of SEPP is skipped, the approach becomes quite easy to use due to the clear SPF/CSPF selection strategy and the straightforward “LEGO block” approach to constructing security architectures via GSAs. Nevertheless, some security expertise is required since GSCs, GSAs etc. are abstract components and architectures without packaged guidance or trade-offs. Furthermore, familiarity and experience with problem frames is necessary to fully understand and appreciate SPF and CSPF descriptions. If the CSP formal specification stage is included, then the level of required expertise, as well as the complexity of application, is increased considerably and the methodology becomes cumbersome to use.

With respect to the development life-cycle, SEPP is quite a problem-domain heavy approach, i.e. the problem domain determines everything in the solution domain in a "surjective" fashion. Analysis models (contained in SPFs and CSPFs) are not used to guide the development process to a solution, but to enforce a set of possible solutions. Thus, solution-space considerations, such as constructing an optimal software architecture, the interplay between the software and security architecture of an application, are not considered.

Regarding the nature of the approach, problem frames are not as popular as, for example, use cases, which has a bearing on the adoption of SEPP in real-life projects. [Hatebur and Heisel 2010] have created a UML profile allowing for SPFs/CSPFs to be modeled with UML, which offers an improvement over using problem-frame specific diagrams, and may be a beneficial future improvement to SEPP.

The methodology is supported by CSPFTool, an Eclipse plugin for Linux that allows the creation and verification of SPFs and CSPFs. An Eclipse plugin is also available for the UMLsec extension of SEPP.

3.4.1.3 SERENITY

OUTLINE:

The SERENITY project [Maña et al. 2006, Spanoudakis et al 2009] is an effort to address the security and dependability (henceforth S&D) issues peculiar to Ambient Intelligence (AmI) environments. AmI environments can be characterized by possessing a high degree of dynamism, heterogeneity and, above all, distribution. Somewhat like SECTET ([Hafner and Breu 2009], [Section 3.2.1.2]), SERENITY is a methodology consisting of a software development and run-time framework with an accompanying development process.

DETAILED DESCRIPTION:

One of the major goals of SERENITY is to provide adaptable security via application run-time support and variable componentized security solutions. To realize this goal, the approach employs a formalized extension of security patterns embodied in S&D solutions [Gallego-Nicasio et al. 2009, Maña et al. 2006, Maña and Pujol 2008, Serrano et al. 2008, Serrano et al. 2009a, Serrano et al. 2009b], which are the vehicle for security mechanisms. S&D solutions lie at the center of the SERENITY approach. The SERENITY framework supports a hierarchy of S&D solutions at several layers of abstraction:

1. S&D Classes, at the highest level, which guarantee particular semantics and a common interface, and abstract the S&D properties provided by a collection of S&D patterns.
2. S&D Patterns are concrete, individual specifications of solutions with precise formal semantics having multiple possible implementations. These implementations are represented as:
3. S&D Implementations, which are specifications of run-time solutions, realized as:
4. Executable Components, in the SERENITY run-time environment (described below).

The solution hierarchy is thus composed top-down, i.e. from abstract solution classes to concrete solution specifications to implementation-dependent specifications and

finally to actual components. SERENITY's S&D patterns are fundamentally implementation-oriented. According to [Sánchez-Cid et al. 2009], "S&D Solutions are well-defined mechanisms […] that provide one or more S&D properties", while "S&D patterns are semantic descriptions of S&D solutions" containing the necessary information for the "selection, instantiation and adaptation, and dynamic application" of S&D solutions during run-time.

S&D Classes and Patterns must be rigorously specified by a security expert using an XML template. Patterns and associated implementations in particular can be formally certified, and are envisaged as "off-the-shelf" re-usable building blocks. Pattern specification include at least: (1) a well defined interface, along with Adapters bridging the interface to that of any associated S&D classes; (2) pre-conditions for applying the solution; (3) S&D properties provided; (4) some form of certification; and (5) a set of monitoring rules. The resulting artefacts are stored in a library for use during development and also at run-time. Some examples of S&D Patterns and Classes can be seen in [Dolinar et al. 2008] and [Sánchez-Cid et al. 2009].

The SERENITY framework itself is split into two parts: a development framework (SDF) and a run-time framework (SRF). The development framework provisions selection of S&D solutions by developers and also provides tools to develop new S&D solutions. The run-time framework deals with the monitoring of security solutions, their instantiation as executable components and their adaptation as the need arises. Monitoring and execution support help to realize the goal of providing adaptable security at run-time, allowing different security solutions (as executable components) to be swapped dynamically.

The SERENITY development process can be reconstructed from [Sánchez-Cid and Maña 2008], [Maña and Pujol 2008] and [Serrano et al. 2008, Serrano et al. 2009a, Serrano et al. 2009b] as follows.

At the beginning of the process S&D experts must determine any new solutions to be used and codify them as S&D Classes or Patterns; the latter are then coded by S&D developers as Executable Components, and fitted into the framework by writing corresponding S&D Implementation specifications.

Requirements: During the early development stages, application developers must identify the necessary S&D properties from the target application's security goals. A custom language [Serrano et al. 2009a] providing simple constructs for expressing, amongst other features, (canonical) S&D properties and contexts is used to specify the S&D requirements, which are later used as semantic queries in the SDF to determine a set of conforming S&D solutions that can be selected.

Analysis and Design: A UML profile is used for notating required S&D properties on classes during analysis and representing relevant S&D solutions (as single classes) during design. The requirements-as-queries from the earlier stages are used in conjunction with the SDF to determine the set of all available S&D solutions for selection, thereby establishing a conceptual mapping from S&D requirements expressed as required S&D properties to S&D solutions providing those properties. This mapping is realized by the selection of an appropriate solution at some level of abstraction (i.e. Class, Pattern or Implementation). The higher the level, the more

choice is left for the SRF algorithms to make the decision as to what security solution is best during run-time. If an appropriate solution is not found in the S&D solution catalog, a new S&D Class, Pattern or Implementation must be made by the S&D experts and developers.

Implementation: For the approach to work, all applications must be made SERENITY-aware, i.e. they must use the underlying interfaces and functions defined by the S&D classes and patterns, making security largely programmatic. A Java API exists for this purpose, which provides handlers to applications to make calls on the Executable Components instantiated in the SRF instead of communicating with them directly.

Deployment: One of the benefits of using SERENITY is that the SRF provides automated monitoring capabilities for the security solutions during run-time. When the conditions or environment of the system change, the monitoring rules contained in the S&D solutions can potentially be triggered, leading to a replacement of the solutions with a different one (e.g. a stronger cryptographic algorithm). While not a methodological feature as such, monitoring has implications on how S&D patterns are specified and is connected with the overall aims of SERENITY, making it an essential feature of the approach.

DISCUSSION:

Like SECTET and, to a lesser extent, PMF/OpenPMF [Lang and Schreiner 2003], SERENITY is only concerned with application-level software. Essentially, applications run on top of the SRF, which acts as a middleware layer providing adaptable security. The idea of having executable components encapsulating security services is similar to Rosado et al.'s reference security architecture [Rosado et al. 2011a] and SeAAS [Hafner et al. 2009] and hence in some degree to the ISO/IEC 7498-2 Security Architecture model of security services in open distributed systems. In a sense, this idea is taken to its logical conclusion in SERENITY, where all security solutions ultimately reduce to pre-coded security components. However, the scope and scenario coverage of solutions is variable and hence the number of (equivalents of) security services in SERENITY are potentially unlimited.

Comparing S&D Patterns in SERENITY to security patterns in SECTET reveals a number of similarities; for example, both are used as intermediate representations of implemented security mechanisms in an underlying platform or framework and therefore cannot capture software design decisions. An S&D pattern does not "describe the internal functioning of [a security] solution, but its semantics (i.e. properties provided, limitations etc.)" [Sánchez-Cid et al. 2009] in a formal fashion. The same could be said of S&D Implementations, which likewise capture additional platform-dependent information about the corresponding Executable Component.

Although S&D solutions can capture an extremely wide variety of security mechanisms, the SERENITY approach suffers from the drawback of relying exclusively on its own security framework. Interoperability between the SRF and other middleware is not fully addressed, reflecting the specialization of SERENITY to AmI systems.

If SEPP ([Section 3.4.1.2] above) was portrayed as being requirements-heavy, then it could said of SERENITY that it is, in the same measure, implementation-heavy. SERENITY patterns are simply encapsulations of descriptions of run-time security components. While there is support for modeling security architectures via the provided UML profiles, the activity of creating secure designs is a secondary concern. All analysis and design activities ultimately tend towards the selection of appropriate solutions, which developers are required to manage purely programmatically during implementation. The S&D artefacts themselves (classes, patterns etc.) are re-usable only within the scope of SERENITY, and in particular its run-time constituent (the SRF).

Nevertheless, the SERENITY approach advances much farther than most other major security methodologies outlined previously in catering for the needs of security non-experts, predominantly by allowing the prescription of security solutions at the design stage at varying levels of abstraction, according to expertise; the SRF is then left to decide (at run-time) the best implementation for a particular solution. This makes the approach easy to use.

SERENITY is supported by a tool-suite for the development and selection of S&D solutions, including a pattern management tool to aid security experts in creating new S&D solutions and an Eclipse-plugin for solution searching during development.

3.4.2 Methodologies Using Security Patterns

3.4.2.1 Eduardo B. Fernandez and colleagues

OUTLINE:

[Fernandez 2004] and later [Fernandez et al. 2005, Fernandez et al. 2006a, Fernandez et al. 2006b, Fernandez et al. 2011] present a comprehensive methodology using security patterns throughout the early stages of the SDLC within the object-oriented paradigm. The introduction of security properties proceeds in two complementary dimensions: along the system's development life-cycle, closely following the stages of object-oriented analysis and design [Fernandez et al. 2006a, Fernandez and Mujica 2010]; and across a set of functional and non-functional hierarchical layers of abstraction, representing an extended software stack, superimposed on a system's software architecture [Fernandez and France 1995, Fernandez and Nair 1998, Fernandez 1999, Fernandez 2003]. Security constraints are defined in accord with this approach both at the earliest stages of the SDLC and at the highest layers of abstraction – a dedicated security layer in [Fernandez and France 1995] or a meta-layer in [Fernandez 2003] where the conceptual model resides, as well as the application layer [Fernandez 2004] – and propagated horizontally and vertically respectively. Security patterns can be fitted into the various (lower) layers and across stages in increasingly more specialized variants [Fernandez et al. 2008] to enforce the high-level constraints (requirements).

DETAILED DESCRIPTION:

Requirements: In Fernandez et al.'s approach, consideration of security starts from the requirements stage with an analysis of use cases, individually or in sequence (workflows). The purpose of this analysis is twofold: firstly, to determine a set of

possible threats and thereby derive a set of security requirements for later stages of the methodology [Braz et al. 2008, Fernandez et al. 2006b]; and secondly, to determine a set of minimal authorization rights for each subject (or role) [Fernandez and Hawkins 1997].

For the determination of subject (or role) rights, use cases are extended via stereotypes to allow the explicit specification of security authorizations for each action by a given actor. "The union of the set of authorization rights across all use cases for every actor defines the complete set of authorization rights for the system" [Fernandez and Hawkins 1997]. Since only valid actions are authorized based on all possible (external) system uses, the resulting set of rights is minimally sufficient in accordance with the need-to-know principle. This approach, therefore, determines rights based on what subjects are allowed to do, not on what they are not allowed to do. The authorization rights determine the first set of security constraints for an application or system.

For determining threats and their corresponding security requirements, [Fernandez et al. 2006b] and [Braz et al. 2008] provide a systematic approach to threat modeling. First of all, threats are enumerated by analyzing each action within a single use case or a sequence of such use cases (per existing actor) and determining how that activity can be subverted, as well as adding external actors such as intruders or hackers and analyzing their specific actions. Each threat is related to one of the affected standard security requirement classes, namely, confidentiality, integrity, availability or accountability, and their related sub-classes [see Braz et al. 2008] for more details), as well as whether the actor initiating the given action is an authorized insider, unauthorized insider or outsider. The resulting threat model is expressed in the form of extended, composite UML activity diagrams. The actual determination of threats is based on experience and domain knowledge, as is the usual case, although threat patterns, which re-use this experience, are also proposed in [Fernandez et al. 2006b]. Performing risk analysis is also part of the process, although specific analysis techniques are not considered. The use cases, threat model, security requirements and initial security constraints are the main results of the requirements stage of the methodology.

Analysis: During the analysis stage, analysis patterns [Blaimer et al. 2010] and in particular Semantic Analysis Patterns (SAP – [Fernandez and Yuan 2000, Fernandez and Yuan 2007]) are used in addition to standard object-oriented modeling practices to create a conceptual (domain or business concept) model efficiently. The use case analysis results from before are carried over to determine a set of abstract security patterns [Fernandez and Mujica 2010, Fernandez et al. 2008] that collectively embody a subset of the security requirements from before and also constitute a set of application-level constraints. From the point of view of [Braz et al. 2008], the abstract patterns can be seen as the mitigating policies corresponding to some or all of the security requirements. These include at least an overall access control model, which is realized by repeated instantiations of the desired (abstract) access control pattern defined by [Fernandez and Pan 2001], thereby realizing the authorization rights per subject (or role) determined from before. The final conceptual model represents a security-enhanced model of the system's main requirements, and is used to guide the rest of the development.

Design: During the design stage, the conceptual model from analysis is used to construct a corresponding software design (i.e. software architecture). This initiates a re-newed consideration of security attacks, since the threat model from before must now be considered with respect to the elaborated architecture, and, moreover, there is scope for new attacks at each layer of the (extended) software stack – i.e. the overall or superimposed hierarchical architecture. This is performed with the aid of misuse patterns [Fernandez et al. 2009], which describe in detail how an attack is performed (including what software components are required) from the attacker's perspective, which patterns can mitigate the attack, and how the attack can be traced once it occurs. Misuse patterns aggregate a sequence of attack patterns [Hoglund and McGraw 2004] that represent the steps or certain aspects of the whole attack. Misuse patterns can be cataloged according to their context, and, once instantiated, represent a specific attack with respect to a particular system's architecture at some level of abstraction. Using misuse patterns has a number of advantages in contrast to ad-hoc approaches, including guidance for the non-expert in considering possible attacks and easier identification of corresponding design-level countermeasures. This approach is combined with the previous security requirements and, following [Braz et al. 2008], the total set of unsatisfied or partially satisfied security requirements are mapped to appropriate groups of security patterns to secure each software layer. This ensures traceability from requirements to design, and aids in restricting the selection of patterns according to attacks and the previous security constraints.

At this stage, the abstract patterns in the conceptual model are also specialized into more concrete security patterns and, along with all other security patterns introduced during design, are propagated down the different software layers, thus giving rise to a set of concrete countermeasures that are required to collectively participate in enforcing the initial security constraints [Fernandez 2003]. To achieve this, the concrete or specialized security patterns – representing security mechanisms at varying layers of abstraction – must be mapped through [Fernandez et al. 2006a] or coordinated between [Fernandez 1999] their respective layers. In [Fernandez 1999] this coordination consists in creating object models of each mechanism at each layer and formally defining the relationships (i.e. mappings of model elements), with Z and OCL being proposed as suitable languages for the purpose. Given that in the latest approach patterns represent the solutions at each layer, it does not seem strictly necessary to create object models, but simply to relate the security patterns to each other, which in turn implies at least the mapping of different security policies or rules across different mechanisms in each layer. It is beyond doubt that this is a difficult task, especially in the context of solutions crossing administrative boundaries, and a fully structured approach appears to be an open problem (cf. [Fernandez et al. 2005]). Since the patterns are instantiated in the system's design and elaborated along with the rest of the system, integration problems are avoided.

As a central example of satisfying access control constraints, and one that can be considered as an essential part of the approach, [Fernandez and Hawkins 1997] and later in a more finalized form [Fernandez et al. 2006a] suggest the (possibly) multiple instantiation of the Model-View Controller pattern [Buschmann et al. 1996] across the architecture, allowing for the division of objects into Model and View classes and the subsequent restriction of all external interactions (such as user access) to View objects (i.e. user interfaces). This can also be seen as a simultaneous instantiation of the

Single Access Point (SAP) pattern [Yoder and Barcalow 1997], which likewise restricts external system accesses to a single point. In this scheme, the authorizations specified in the conceptual model are enforced at the View objects playing the role of policy enforcement points [Gollmann 2011] and mapped down the software stack, being reflected conceptually in database table privileges, filesystem rights etc. The practical application of this appears to require manual administration.

Distribution is considered as a special layer of the hierarchical architecture and is thus secured, like other layers, using patterns. Distributed concerns are therefore addressed during the design stage, whenever the conceptual model is mapped onto a pattern-based distributed architecture. [Fernandez and Larrondo-Petrie 2007] discuss this approach in the context of component middleware and propose secure variants of a number of standard distribution patterns (such as the Broker pattern in [Buschmann et al. 1996] – see [Fernandez and Uzunov 2012] and [Uzunov et al. 2012]). In [Fernandez et al. 2007b] an initial attempt is made to extend and tailor the methodology to SOA-based systems.

Implementation: During implementation, the necessary countermeasures as patterns are realized as software units or COTS components. Security can be tested at every stage by checking for the absence or presence of definite security patterns, alignment with security requirements, whether certain attacks are mitigated, etc. The most significant security verification activity employs test cases generated (manually) during or from the requirements stage, to test, during the design and implementation stages, whether the designed and implemented countermeasures can withstand the subverted uses and misuses of the system.

DISCUSSION:

In comparison to other approaches, Fernandez et al.'s methodology is the only one to consider security in a holistic manner – i.e. the need to secure a system or application and all its related functionality in the underlying layers of the software stack. This is, however, a difficult and expensive task, and would require automation of certain aspects, such as determining lower-level policy rules, to make the approach practical for large-scale systems, especially those using black-box legacy sub-systems. On the other hand, coordinating security solutions throughout the whole stack as proposed by Fernandez and colleagues is the only way to ensure global system protection.

Close alignment to the object-oriented development paradigm has advantages – in that object-orientated analysis and design and UML are widely used and readily comprehensible – and disadvantages – in that the methodology may not be applicable in conjunction with other paradigms. The advantages in particular greatly aid to increase the acceptance of the methodology, while using patterns and guidance throughout helps to make the methodology easier to use. Future work in [Fernandez and Mujica 2010] and [Fernandez et al. 2011] indicates that the methodology is moving away from strict adherence to the object-orientated paradigm and heading towards an MDE paradigm.

The methodology relies on coherent catalogs of security patterns, which can cover a range of security concerns. Such catalogs exist in the form of books [Fernandez 2013, Schumacher et al. 2006, Steel et al. 2005], surveys [Uzunov et al.

2012, Yskout et al. 2006] and technical reports [Blakley and Heath 2004, Dougherty et al. 2009], as well as in individual papers, covering access control models, remote authentication, filtering, XML encryption and others. However, as yet there is no single repository. The same remarks apply to analysis patterns and their secured versions (cf. [Fernandez and Yuan 2007]). The main drawback of the approach in this respect is that there is a lack of tool-support for pattern selection, implying that developers must choose patterns manually by navigating through the (text based) pattern catalogs. Although this is offset by the guidance provided in classification schemes, e.g. [Hafiz et al. 2007, VanHilst et al. 2009] and [Washizaki et al. 2009], simple tool-support would certainly be a beneficial feature, and is a planned addition to future developments of the methodology (cf. [Fernandez and Mujica 2010, Fernandez et al. 2011]).

3.4.2.2 Gutiérrez / Fernández-Medina / Piattini (PWSSec)

OUTLINE:

In a similar vein to PSecGCM ([see Section 3.4.1.1]), [Gutiérrez et al. 2005a, Gutiérrez et al. 2005b, Gutiérrez et al. 2006, Gutiérrez et al. 2009] present PWSSec – a comprehensive pattern-driven methodology for web-services (henceforth WS). The similarity in the approaches is understandable, since both methodologies were developed in the same research group (Alarcos, Universidad de Castilla-La Mancha). Like PSecGCM, PWSSec defines a rigorous and detailed software engineering process based on an incremental, iterative life-cycle model; PWSSec also uses a repository of re-usable artefacts, which helps to make the methodology scalable across different projects. Despite the similarities between PSecGCM and PWSSec, there are also a number of important differences. Firstly, PWSSec is exclusively tailored for web-services, and is not applicable to any other type of system, distributed or otherwise [Gutiérrez et al. 2009]. This makes the methodology very specific indeed, even though the overall approach has a number of interesting features applicable in wider settings. Secondly, PWSSec, uses a finer-grained approach, especially during requirements analysis, to determine security requirements, risks etc. Finally, PWSSec employs security patterns, making it more flexible and somewhat more approachable to non-experts.

DETAILED DESCRIPTION:

In PWSSec there are three main stages: WSSecReq (WS security requirements – Gutiérrez et al. 2005a, Gutiérrez et al. 2009]), WSSecArch (WS security architecture – Gutiérrez 2005b, Gutiérrez et al. 2006]) and WSSecTech (WS security technologies/standards – Rosado et al. 2006]), which roughly correspond to requirements analysis, design (or architecture development) and detailed design respectively.

Requirements analysis: The ultimate aim of the requirements analysis stage is to define the WS application's security requirements. Activities in this stage for the most part follow the standard course: determining assets, then threats on those assets to form a threat model, refinement of the threat model by considering attacks, analyzing risks for each attack and finally defining the security requirements. The threat model is based on attack trees [Schneier 1999], which are a very detailed approach to

enumerating threats. Each tree leaf is elaborated by considering attacks using misuse cases [Alexander 2003], which are subsequently mapped to corresponding countermeasures in the form of security use cases [Firesmith 2003]. Risk analysis is performed via a fine-grained risk allocation approach using percentages and any trade-offs are made to define and specify the final set of security requirements, which are carried over into the design stage.

Design: One of the most interesting aspects of PWSSec, as in PSecGCM, is the use of a reference security architecture (presented most fully in [Gutiérrez et al. 2005b]) during design. In the case of PWSSec, however, this architecture is of a somewhat different nature. Whereas the security reference architecture in [Rosado et al. 2011a] required integration with the software architecture, the reference architecture in [Gutiérrez et al. 2005b, Gutiérrez et al. 2006] makes use of security patterns, giving rise to a more flexible approach. In PWSSec, security requirements are satisfied by first identifying appropriate security patterns, whose range can, in theory, be arbitrary. This renders the approach more general and simultaneously allows for more detail and comprehensive coverage of security issues. The instantiated security patterns give rise to collections of (abstract) WS security components, which can be elaborated further until they are mapped to actual software components. Since security components are like any other WS components, they must be re-evaluated during another iteration through the requirements and analysis stage [Gutiérrez et al. 2005b].

Security patterns are selected and refined according to two levels of abstraction [Rosado et al. 2006], the first consisting of architectural security patterns, which cover more abstract security concerns affecting larger parts of a system's architecture; and the second consisting of design security patterns, which refine the latter by including specific implementation-related aspects. Design patterns can be mapped directly onto web-services standards and technologies, which can be seen as a final refinement stage for a particular security solution. The architectural security patterns are organized in a repository according to the requirements they address, together with corresponding design patterns, allowing for easier selection during the WSSecArch stage (architectural and design patterns) and the WSSecTech stage (technology-specific mapping).

Security patterns form one aspect of the PWSSec security reference architecture; the other aspect is the management of the resulting security components. During design, an application's (standard) constituent WS components are separated into security domains (termed zones) and assigned security management components called security kernels, with one single “master” kernel per zone. All communications pass through the master kernel for each zone, turning it into a type of reference monitor. The purpose of the kernels are twofold: firstly, they manage the WS security components in a zone; and secondly, they acts as a trusted centre or locus for all security functionality per zone. The last point makes kernels (and in particular the master kernel) a “one-stop shop” for all standard WS components, which can use the services they offer as indirections to the WS security components. Regarding implementation details, standard WS components must specify, essentially, “require” policies determining what security requirements they have (e.g. needing to use authentication), while WS security components must specify “provides” policies determining what security services they provide. At run-time, both these should be

registered with the kernels, which check whether an available WS software component exists that implements particular security functionality. PWSSec thus promulgates a programmatic security model.

In spite of the theoretical generality offered by the use of security patterns, [Gutiérrez et al. 2006] only concern themselves with communications security via the "QoP" (quality of protection) pattern system of [Rosado et al. 2006], and only instantiate the QoP pattern with one security kernel in one zone in their case study in [Gutiérrez et al. 2009].

Detailed design: During the detailed design stage, a set of WS-specific technologies and/or standards are identified to be implemented by the WS security components, which are subsequently mapped to concrete software units. In this respect, the security patterns from the earlier stages play a central role in the transition between architecture and detailed design, in conceptually mapping WS security components to specific WS technologies and/or standards. Traceability is thus kept from requirements to architecture to technologies.

DISCUSSION:

PWSSec is a comprehensive and highly detailed security methodology covering all the early stages of the SDLC. Precisely this level of detail, however, is PWSSec's main drawback. In the case study used to demonstrate the features of the methodology [Gutiérrez et al. 2009], the various activities generated over 200 detailed artefacts for a WS application with a single use case. Even though the application was comparatively large when measured in lines of code (more than 100 kloc), the utilization of the methodology for sizeable, complex systems with numerous use cases appears to be infeasible. Certainly, the scalability features offered by the methodology in terms of re-use of artefacts across different projects is a mitigating factor in this respect, however, they cannot reduce the high amounts of developer effort involved, even with the use of the accompanying CASE tool (UMLPWSSec). The requirement to re-evaluate (via further iterations) WS security components following the instantiation of security patterns is a further complication.

Overall, the methodology makes a good case for a highly specific approach to securing a particular type of distributed system, but the significant labour involved reduces its benefits, and may imply the necessity to automate certain activities or at least increase their granularity.

PWSSec is supported by UMLPWSSec, an extension to the Rational Rose 2000 CASE tool. The support offered by UMLPWSSec is essential in helping to automate the generation of templates for use cases, risks etc.

3.4.2.3 Delessy / Fernandez

OUTLINE:

[Delessy and Fernandez 2008] and [Delessy et al. 2008] present a pattern-driven (primary class) and model-driven (secondary class) methodology for securing SOA-based systems, adapting and building on the work of Fernandez and colleagues ([see Section 3.4.2.1]) for the design stage. The approach is based on two central ideas: the use of pattern maps to guide the selection of security patterns; and the application of MDA principles to SOA-specific models.

DETAILED DESCRIPTION:

Pattern maps are graphs of (security) patterns divided into different layers of abstraction. The patterns (nodes in the graph) on a given layer of abstraction are interrelated by collaboration or refinement relationships (edges) and are classified by a security goal, such as enforcing confidentiality, integrity etc. Patterns across layers are interrelated by a realization relationship, i.e. a pattern on a lower abstraction level realizes the related pattern on a higher level.

In Delessy and Fernandez's methodology, the pattern map consists of two layers, an abstract layer containing patterns with general SOA applicability, and a concrete layer, containing patterns for web-service security. Collectively, the set of patterns in the map at some given point in time represent the set of possible solutions available to developers, which currently address authentication, identity management, access control and web-service cryptographic and policy standards among others [see Delessy and Fernandez 2008] for details and references). While pattern maps have also been utilized in other contexts, e.g. [Fernandez et al. 2007], in the work of Delessy and Fernandez they take a central role in selecting security solutions.

Requirements analysis: Since the methodology of Delessy and Fernandez is based on the work of Fernandez and colleagues, the activities performed during the requirements analysis stage for threat enumeration and the determination of security requirements are identical (see [Section 3.4.2.1]).

Design: To support MDA transformations and the modeling of SOA-specific systems, [Delessy and Fernandez 2008] present a UML metamodel defining in detail all or most of the necessary SOA-specific semantic elements such as orchestration, services, roles and others. The metamodel is used during design to construct a conformant (primary), platform-independent model, guided by the analysis model from the previous development stages.

The process of introducing security into the resulting model proceeds as follows: (1) a fixed list of security requirements for SOA-based systems (in [Delessy and Fernandez 2008]) mapping requirements to policy classes is consulted to guide the initial selection of appropriate abstract security patterns, by mapping security requirements to a resulting policy and goal and finally to a related pattern from the SOA security pattern map; (2) selecting relevant patterns according to the relationships of the map, resulting in a decision tree [Fernandez et al. 2007b] that represents the set of chosen countermeasures as abstract and concrete patterns; (3) weaving the set of abstract security patterns into the platform-independent model in a manual process equivalent to pattern instantiation; (4) transforming the resulting security solutions by hand to the platform-specific level – along with the PIM to PSM transformation of the primary model – by specializing the abstract security patterns, according to the pattern map, to their related concrete web-services versions; (5) weaving the concrete patterns into the platorm-specific model as before. The PSM in particular conforms to a web-services metamodel which is not presented in any relevant publications.

The PIM-to-PSM transformatios are written in QVT, but there does not appear to be any tool-support at this stage, which implies that the weaving and transformation processes are manual.

DISCUSSION:
Overall, the intuitive guidance offered in selecting appropriate security solutions using the pattern map and the use of security patterns makes the approach easy to use. As for other methodologies based on security patterns, the range of possible security solutions is very large. However, the methodology currently lacks tool-support, which is essential if the model transformations are to be fully practical. The published information relating to the methodology is also somewhat sparse in detail, and in particular the last few steps of the security process described above are not demonstrated in full. Therefore, in its current state, the methodology presents a promising approach, but one that needs to be developed further for real-life use.

3.4.2.4 Other Methodologies Using Security Patterns

Other pattern-driven methodologies deserving mention (see also [Uzunov et al. 2012] for an overview from a pattern application perspective) include SODA [Meland and Jensen 2008], a security methodology "for the developer-on-the-street" in which patterns as well as general security guidelines and principles are presented to software developers via a web-based tool to guide them through the process of securing a system design. SODA borrows a number of ideas from process-driven methodologies like Microsoft's SDL, e.g. in its security life-cycle model: threat modeling, security reviews etc., except that security patterns are used as guides, thus reducing the amount of security expertise required.

[Schnjakin et al. 2009] present an approach to securing SOA-based (and in particular web-services based) systems using a "security advisor" CASE tool. The advisor helps developer map security goals to semi-formalized security patterns inspired by Schumacher's theoretical model [Schumacher 2003], which are subsequently transformed to security configurations in the underlying web-services platform using MDA principles. The set of patterns and possible security configurations used by the advisor must be determined by experts in a pre-configuration phase.

ISDF [Alkussayer and Allen 2010] proposes the use of security patterns alongside best practices in an integrated fashion throughout the whole SDLC, while basing the process phases and activities on Microsoft's SDL. Despite the use of patterns, ISDF is a very high-level approach, which implies that it suffers from all the draw-backs of a (general) code-based methodology with respect to specific systems.

Finally, [Scandariato et al. 2008] present a security pattern-driven methodology based on their previous work of cataloging and refining available security patterns in the open literature [Heyman et al. 2007, Yskout et al. 2006, Yskout et al. 2008]. The latter catalog is used as an inventory, which is further structured according to a decomposition of security objectives, relationships such as dependencies and conflicts, and a separation of design phases of a system (application architecture, application design and system-level). Security patterns from the inventory are used according with the latter phases to help introduce security properties into a system, based on the security objectives defined at the outset.

3.5 Agent-Driven Methodologies

Agent-driven methodologies are centered on the Agent-Oriented Software Engineering (AOSE) paradigm [Ciancarini and Wooldridge 2001, Jennings 2001]. AOSE is based on several fundamental concepts borrowed from AI and social theories, the most important of which is that of an agent – a system entity with strategic goals and intentionality, or, from another angle, an encapsulated piece of software "situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives" [Jennings 2001, Wooldridge 1997]. There is a close similarity in some respects between the agent-oriented and object-oriented paradigms, in that both aim to encapsulate some element of system functionality in a higher-level abstraction, and both aim at autonomy, even if in different ways. The main distinction lies in the fact that agents are pieces of (sometimes mobile) code that, notionally at least, have some reasoning logic enabling them to make decisions based on environmental stimuli, rather than passively reacting to method calls or messages. [Jennings 2001] argues that AOSE is well suited to developing distributed systems, due to the collaborative, autonomous nature of agents. In reality, the application of AOSE to distributed systems results in multi-agent systems (MAS), which can be considered specific types of distributed systems.

[Mouratidis et al. 2005] argue for the benefits of securing software using the AOSE paradigm, advancing the view that agents, with their intrinsic autonomy and intentionality (and hence goal-orientation), provide a suitable means to model high-level security concerns that can be propagated throughout the system. Although the argument itself does not stand to closer scrutiny (the same could be stated with respect to any paradigm), it is nevertheless true that AOSE allows for strategic goals to be captured early on, thereby allowing a goal-oriented approach to security that may be more natural for some developers.

A number of methodologies have been created for AOSE, however, very few incorporate security as a central concern (cf. [Mouratidis and Giorgini 2007a]). In this subsection we review Secure Tropos, an extension of the Tropos AOSE methodology, which integrates security throughout the early stages of the development life-cycle.

3.5.1 Mouratidis / Giorgini (Secure Tropos)

OUTLINE:

Secure Tropos [Mouratidis and Giorgini 2004, Mouratidis et al. 2005, Mouratidis and Griorgini 2007a, Mouratidis 2011] is an extension of the Tropos AOSE development methodology [Giunchiglia et al. 2003, Bresciani et al. 2004, Giorgini et al. 2004]. Since the AOSE paradigm is less familiar than other development paradigms, we will present the Tropos methodology in some detail below, before proceeding to review the extensions introduced by Secure Tropos.

DETAILED DESCRIPTION:

The Tropos methodology considers five development stages: early requirements, late requirements, architecture development, detailed design and implementation. Throughout each stage, Tropos uses the *i** modeling framework, a modeling language tailored for agent-oriented systems. The range of concepts that can be modeled by *i** include: actors (entities with strategic goals), which can be agents, roles (abstract

characterizations of behaviour) or positions (sets of roles); goals (an actor's strategic interests); soft-goals (goals without clear satisfiability criteria); tasks or plans (ways of doing something); resources (physical or informational entities without intentionality); capabilities (context-specific ability of an actor to select and execute a task for the fulfillment of a goal); and dependency relations – depender, dependum, dependee (inter-actor dependencies to attaining goals, executing tasks, and/or delivering resources). These concepts define the semantics of the modeling language; the syntax is comprised of a non-standard, custom graphical notation, which can be seen in all the Tropos-related publications.

According to [Mouratidis and Giorgini 2007a], "Tropos is based on the idea of building a model of the system that is incrementally refined and extended from a conceptual level to executable artefacts, by means of a sequence of transformation steps". Here transformation does not imply automated model-driven transformation, but the conceptual transformation from models at one development stage to models of the next stage. At the early requirements stage, stakeholders (i.e. participants or users of the system, entities in the system's environment etc.) together with their intentions are modeled as actors and goals respectively, to create a strategic dependency model, which is somewhat analogous to a conceptual model in standard software engineering. This model is augmented by a set of strategic rationale models, one for each actor, with the determination of how the actor's goals can be achieved (i.e. with the addition of tasks and sub-goals from the perspective of the specific actor and internal to that actor, using various analysis techniques, such as means-end and contribution analysis and AND/OR decomposition). During late requirements analysis, the software system itself, represented as an actor, is added to the previous models and the analyses performed to produce a rationale model for the system and to revise rationale models for other actors as needed. At this stage the functional and non-functional requirements of a system can be specified from the inter-actor dependencies. At the architecture development stage, the system's software architecture is elaborated using large-grained, agent-specific styles, and refined during detailed-design using agent-specific patterns [see Giorgini et al. 2004]). Throughout the architecture stage (as in the requirements analysis stages), all entities in the system are modeled as actors and the relationships between the entities are modeled as dependencies. In terms of functionality, architectural components (actors), which encapsulate elements of the system's functionality, can be added and new/existing ones can be recursively decomposed into sub-actors to which goals (of the original actor) are delegated. This is analogous to software components being refined into sub-components, which are assigned a portion of the larger component's functionality. Rationale models for each sub-actor are created as a result of analyzing the tasks and goals required to achieve the larger goal, as well as the dependencies with other actors (or sub-actors). The last steps in the architectural design stage consist in identifying capabilities needed by each actor [see Bresciani et al. 2004]) and assigning these capabilities to agents. The detailed design stage consists in further specifying each agent's capabilities and interactions with the help of Agent UML (AUML – [Huget 2004]).

Requirements analysis: Secure Tropos extends the Tropos methodology by introducing security constraints, which embody high-level policies and – during later

development stages – security requirements, placed on the system and its entities (actors); as well as by expanding Tropos' modeling language to allow for the modeling of concepts such as secure goals, secure tasks, secure resources, secure dependencies and secure capabilities. In addition to extending the modeling capabilities of Tropos, Secure Tropos introduces four custom security modeling and analysis activities. The first of these activities, security reference modeling, has the purpose of identifying the security needs of a system via threat and vulnerability analysis, the result of which is the determination of a set of security features (modeled as soft-goals), protection objectives (modeled as goals) and security mechanisms (modeled as tasks). This activity is not done with respect to the assets of an initial system model, but to the desired security features of the system. The second modeling activity, security constraints modeling, aims to model, decompose and assign security constraints to actors, whereby a set of secure goals are introduced into the system for the fulfilment of the constraints. The third modeling activity, secure entities modeling, is concerned with analyzing the secure goals introduced during constraint modeling, via, for example, means-end analysis, to determine a set of tasks and resources that can achieve the goals. Finally, the fourth modeling activity, secure capability modeling, seeks to identify a set of secure capabilities that can be assigned to agents in the system to satisfy the security constraints.

The modeling activities of Secure Tropos are used within its overall aims of identifying a system's security requirements at the early development stages; developing a viable design that satisfies those requirements; and verifying the security of the resulting system. This is achieved via a corresponding four step process [Mouratidis 2011], aligned with the (early and late) requirements analysis, architecture development and detailed design stages of Tropos.

During the early and late requirements stages, security reference, security constraints and secure entities modeling are employed alongside the Tropos strategic dependency and rationale models, to impose security constraints on the actors and their dependencies and to introduce secure entities (goals, tasks etc.) for the satisfaction of those constraints. The result of the requirements analysis stages is a set of models as well as functional, security and (other) non-functional requirements for the system.

Architecture development: During the architecture development stage, a suitable architectural style is selected, new actors are added and refined (i.e. recursively decomposed as per the Tropos approach) to elaborate the architecture; and secure capabilities are identified and subsequently assigned to actual agents. Architectural style selection occurs at the discretion of the developers, with the aid of a technique using fine-grained probability assignment to perform trade-offs (see [Mouratidis et al. 2005]). When elaborating the architecture, new security constraints can be introduced and analyzed as in previous stages, and an analysis of security criticality and complexity [Mouratidis and Giorgini 2004] for each actor can also be applied to determine if new actors need to be introduced in order to delegate overloaded functionality. Finally, a detailed verification is performed for the developed models, based on validation rule-sets, and for the design as a whole, based on such techniques as security attack scenarios [see Mouratidis and Giorgini 2007b]). Although realized quite differently, the use of attack scenarios in SecureTropos shares a similar purpose

to attack model composition in Georg et al.'s AORDD methodology (see [Section 3.2.2.1]).

The mapping between the conceptual models developed during the late requirements analysis stage and the architecture stage models is aided by the use of agent-specific security patterns [Mouratidis et al. 2003, Mouratidis et al. 2006a]. More specifically, the secure goals and secure tasks from the analysis models are seen as a set of security challenges guiding the selection of particular security patterns. The advantage of this is that the analysis models are conceptually mapped to known, predictable sets of actors (existing and new) as the result of the pattern instantiations, thereby reducing the risks associated with an ad-hoc introduction of design solutions. In spite of their usefulness, the existing set of agent-specific patterns addresses only a limited range of security concerns. The requirements for agent-specific patterns [Mouratidis 2006a] implies that standard collections of security patterns, e.g. [Schumacher et al. 2006] are not applicable and thus cannot be used. It is worth mentioning in this context that Mouratidis et al.'s proofs of completeness [Mouratidis et al. 2006a] – a better term would have been "closure" – based on the ideas of [Schumacher 2003], show that their existing agent-specific patterns cover a given set of security problems and do not introduce new problems, but not that new patterns are not needed.

Detailed design: During earlier versions of the methodology (even including its presentation in [Mouratidis 2009]), the detailed design stage followed Tropos and used AUML to model the agent interaction protocols. In [Mouratidis et al. 2006b] and [Mouratidis 2011], Secure Tropos is combined with UMLsec [Jürjens 2005], predominantly to consolidate the detailed design stage. In the combined version, the *i** models of the system are translated into UMLsec models at the architecture stage (after the initial architecture has been selected, patterns applied etc.), so that the further elaboration of the architecture and in particular the detailed design of individual components is performed using UMLsec. The resulting combined methodology, which can be described as Secure Tropos up to early architecture and UMLsec afterwards, has advantages and disadvantages relative to the complete Secure Tropos version. Firstly the mapping of (Secure) Tropos models to UMLsec models is not strictly one-to-one, which makes for a somewhat ad-hoc approach to converting Tropos goal-oriented models to models in UMLsec; secondly, the combined approach does not obviate the need for developers to learn Tropos notations, nor to think in terms of actors and dependencies during design to elaborate an initial version of the architecture, but adds the further requirement of learning UMLsec notations; and thirdly, the coherency of the approach, which used the same concepts, development approach and notation, is lost – contrary to the initial aims of Secure Tropos [Mouratidis et al. 2005] – and with it the relevance to AOSE. Conversely, the loss of relevance to AOSE implies that the combined approach is applicable to distributed software systems generally, as opposed to agent-based systems. Since Secure Tropos is so tightly bound to AOSE, employing specialized notation and modeling concepts, and since AOSE in the context of distributed systems is so tightly bound to multi-agent systems, the applicability of the methodology is otherwise severely limited to a specific domain. The combined Secure Tropos/UMLsec approach overcomes this limitation. Moreover, UML is a familiar

notation, and UMLsec follows standard software engineering paradigms, meaning that the design stages would be more approachable to a wider range of developers. Finally, UMLsec allows for the formal verification of interaction protocols, albeit at the cost of requiring greater expertise. In our opinion, the choice of which version of the methodology is actually superior is dependent on the problem domain and the project goals, as well as a trade-off between the existing approaches in that context, e.g. whether AOSE is the best choice of paradigm, whether the benefits of UMLsec outweigh coherency etc.

DISCUSSION:

Overall, Secure Tropos provides a very comprehensive security methodology that aims for high ease of use and flexibility. It differs significantly from other methodologies in its early requirement stages, predominantly due to the goal-oriented nature of AOSE, supporting the early introduction of security via high-level (informal) security constraints. Security expertise is still required for certain tasks (such as optimal assignment of secure capabilities), though less so in comparison to most other methodologies.

Secure Tropos is evaluated on a real-life case study [Mouratidis 2009] – eSAP, an electronic health-care system, which demonstrates the use of the methodology well. An independent application of the methodology appears in [Rojas and Mahdy 2011], which augments the approach with STRIDE threat modeling.

Secure Tropos is supported by the SecTro 2.0 CASE tool [Mouratidis 2011].

4 Evaluation

Just as it is difficult, perhaps even impossible, to produce perfectly secure software, it is also difficult to create a perfect security methodology. All the methodologies surveyed in [Section 3] have their own individual benefits and limitations, and may be more or less appropriate for practical use depending on the application, project demands, team skillsets and other factors. The real-life use of a methodology will very much depend, however, on the presence or absence of certain desirable characteristics.

4.1 Criteria

Based on Whyte and Harrison's study, taken as one of the most indicative reflections of the security needs in the software industry, we have formulated a set of criteria for the adoption of a security methodology in real-life scenarios. These criteria were subsequently compared and augmented where applicable using the more abstract requirement lists for a good security methodology presented by [Mouratidis 2009] and [Fernandez et al. 2011] to produce a definitive set, which we summarize below in Table 1.

Criteria numbers 1 to 5 (inclusive), represent essential criteria for industry projects, i.e. criteria that would indicate a low probability of adoption if they were not well satisified by a given methodology. In some instances, poor satisfaction of these criteria may indicate a specialized approach, for example, an approach focusing solely on the generation of security artefacts from a particular SDLC stage. Criteria numbers

6 to 10 (inclusive) represent important characteristics that can help to qualify risks associated with adopting a methodology. Criteria 11 and 12 represent desirable or useful features of a methodology, whose presence may or may not be valuable for a given project.

With respect to the taxonomy presented in [Section 2], criterion 1 is related to classification dimension *SDLC stages supported* [Section 2.2.6]; criterion 3 is related to classification dimensions *Specificity* [Section 2.2.2] and *Range of security properties* [Section 2.2.4]; criteria 4 and 5 are both related to classification dimension *Ease of use* [Section 2.2.7], where criterion 5 in particular is related to classification dimension *Tool-support* [Section 2.2.8]; criterion 6 is related to classification dimension *Modeling language and notation* [Section 2.2.3]; and criterion 8 is related to classification dimension *Use of formal methods and verification* [Section 2.2.5].

Criterion No.	Description
1	A security methodology should be end-to-end, across the whole system life-cycle (requirements analysis, design, implementation, testing, deployment, maintenance and even deposition); as well as the full spectrum of activities: modeling, producing test plans, coding etc. A methodology should attempt to address security threats as early as possible.
2	A security methodology should provide guidance on what security measure should be applied and where it should be applied; in particular, some form of early security assessment such as threat modeling should be an essential part of the approach.
3	A security methodology should be sensitive to the system or application: in particular, "security measures should be proportionate" and specific "to the nature of the application" with "a range of solutions, each equated to a different risk and cost" [Whyte and Harrison 2011].
4	A security methodology should not require high security expertise for its application, but should rely exclusively on the skillsets of the developers (predominantly based on software engineering). In particular, the current knowledge-base of developers, including University graduates with low exposure to security concepts, must be taken into account.
5	A security methodology should provide processes and tools to increase its ease of use and aid developer productivity. In particular, "(naïve) user-friendliness and automation are important" [Whyte and Harrison 2011] to reduce development time and potential errors.
6	A security methodology should support "security modelling on top of well-established model-languages such as UML" [Whyte and Harrison 2011], to reduce developer training and effort.
7	A security methodology should demonstrate validity through complete case studies to build confidence in its real-life value
8	A security methodology should utilize some means of assessment or verification to ensure the introduced security solutions correctly counter the relevant threats and improve the security of the system as a whole. The possibility of using "light-weight" formal methods for verification, even for parts of the system, is an advantage; however, excessive use of formal methods should be avoided.
9	A security methodology should not introduce disproportionate development overhead within the chosen software process
10	A security methodology should promote the use of common repositories or catalogs of security knowledge (relating to solutions, threats, vulnerabilities, standards etc.) to encourage the application of best practices, increase productivity and/or aid developer training.
11	A security methodology should take into account (i.e. be applicable to) third-party and legacy software
12	A security methodology should provide guidelines to balance security with other quality attributes

Table 1: Criteria for industry adoption of a security methodology

4.2 Analysis Results

Based on our detailed reviews of the surveyed methodologies, we have analyzed and evaluated the conformance of each individual methodology to the criteria above, summarizing the results in Table 2. To avoid excessive repetition and reduce space, we omit the detailed analysis for each methodology and provide only a final result in the table, in which: "Y" for "Yes" indicates that a methodology fully satisfies a criterion; "N" for "No" indicates that a methodology fails to satisfy a criterion; and "P" for "Part" indicates that a methodology satisfies a criterion in part. The integer in the top row refers to one of the numbered criteria above. It should be emphasized that while our assignment of values for each criterion per methodology are guided by our reviews, they nevertheless reflect subjective assessment. Moreover, the reviews themselves are often constrained by the amount and/or quality of published information. The merit of the results lies in revealing the current deficiencies in existing approaches and establishing trends – they should not be seen as an absolutely fair comparison between methodologies, something that no restricted evaluation system can achieve. In this respect, the analysis can be seen as being coarse-grained, and finer distinctions of which methodology satisfies a given criterion more than another requires examination of the relevant reviews.

4.3 Discussion

The results clearly show that, currently, no single methodology fully satisfies all requirements. The formal methodology of Ali et al. ([see Section 3.3.2]) satisfied the least number of criteria in the analysis, which was due to its complete reliance on formal methods and the lack of guidance offered throughout. SERENITY and PWSSec, which are among those to satisfy most criteria, are only applicable to AmI and WS-based systems in a very restrictive fashion. This should not be seen as an indicator suggesting that methodologies targeting specific system types (i.e. those with a high degree of specificity) are better able to satisfy the criteria than those of a more generic nature; it may, however, imply that the researchers were better able to tailor their approaches.

Perhaps not surprisingly, certain trends can be seen in the results, especially with respect to (primary) methodology paradigms. Pattern-driven methodologies, for example, fared well against criterion No. 10 (use of a repository of security knowledge), since patterns encapsulate security knowledge and best practices in one way or another, or, in the case of PSecGCM, a repository of solutions that were found successful allows re-use leading to increased productivity and good practice in subsequent projects. In fact, the general trend of the results showed that pattern-driven methodologies, even those in which this is a secondary paradigm (SECTET/SECTISSIMO, Georg et al.'s AORDD methodology) satisfied more criteria than others.

Methodology / Criterion No.	1	2	3	4	5	6	7	8	9	10	11	12
Model-based												
Jürjens (MBSE/UMLsec)	P	N	P	P	Y	Y	Y	Y	Y	N	Y	N
Model-driven (MDA/MDS)												
Lodderstedt/Basin/ Doeser (Model-Driven Security)	N	N	N	N	Y	Y	Y	Y	Y	N	N	N
Hafner/Breu/Memon (SECTET/ SECTISSIMO)	P	Y	Y	N	Y	Y	Y	N	Y	P	N	N
Lang/Schreiner (OpenPMF)	N	N	N	N	Y	Y	Y	N	Y	N	P	N
Gunawan/Hermann/ Kraemer (security-enhanced SPACE)	P	P	N	Y	Y	Y	N	N	Y	P	P	N
Sánchez and colleagues (ModelSec)	P	N	N	N	Y	Y	N	N	Y	N	N	N
Model-driven (AOSD)												
Georg/Ray/France (AORDD)	N	N	N	N	Y	Y	P	Y	N	P	P	Y
Mouheb and colleagues	N	P	N	Y	Y	Y	N	N	Y	P	P	N
Jakob/Loriant/ Consel (DiaAspect)	N	N	N	N	P	N	N	N	Y	N	P	N
Architecture-driven												
Ren/Taylor	N	N	N	N	Y	Y	Y	Y	Y	N	N	N
Ali/El-Kassas/ Mahmoud	N	P	N	N	N	N	N	P	P	N	N	N
Pattern-driven (components/architectures)												
Rosado/Fernández-Medina/Lopez (PSecGCM/ SecMobGrid)	P	Y	Y	N	Y	Y	Y	Y	Y	P	N	N
Schmidt/Hatebur/ Heisel (SEPP)	P	Y	P	Y	Y	P	Y	Y	Y	Y	N	N
SERENITY	P	P	Y	Y	Y	Y	Y	Y	Y	Y	N	N
Pattern-driven (security patterns)												
Fernandez and colleagues	P	Y	Y	Y	P	Y	N	Y	Y	Y	N	N
Gutiérrez/Fernández-Medina/Piattini (PWSSec)	P	Y	Y	P	Y	Y	Y	Y	N	Y	N	N
Delessy/Fernandez	P	Y	Y	Y	P	Y	P	Y	Y	Y	N	N
Agent-driven												
Mouratidis/Giorgini (Secure Tropos)	P	Y	Y	Y	Y	P	Y	Y	Y	N	N	P

Table 2: Conformance of security methodologies to criteria for industry adoption

Methodologies that did not satisfy the first five (essential) criteria well, such as MBSE/UMLsec, Model-Driven Security and others, indicate approaches that are tailored for helping security experts do a better job. Some of these methodologies may best realize their benefits when used as part of a larger security approach (e.g. together with another methodology, as is done with UMLsec and SecureTropos) capable of guiding developers in introducing security solutions. The latter include methodologies that mostly satisfied the essential criteria, and which aim at providing guidance to non-expert developers throughout larger portions of the SDLC.

Below we briefly comment on the analysis results for each criterion individually.

1. No methodology fully satisfied criterion No. 1 (end-to-end life-cycle support), which is perhaps the single most important criterion for real-life adoption, decidedly showing an area for further research and improvement. PSecGCM was the only approach to propose activities for the planning and maintenance stages; however, those stages, as well as implementation (especially for the reference architecture), did not receive the same attention as the core development stages.
2. Guidance in application of security solutions was predominantly satisfied by pattern-driven methodologies, due to the guidelines present in patterns. Mouheb et al.'s methodology satisfied the criterion insofar that aspects in that approach capture security knowledge, as well as where in the design the aspect can be applied; however, no threat modeling/assessment strategy is proposed. Similarly, SERENITY does not offer support to developers in determining the initial set of security requirements.
3. Some (most notably model-driven) methodologies supported the introduction of only one or two security properties (e.g. access control, secure communication), rendering them – despite the presence of promising and novel ideas – unrealistic for many practical situations. Methodologies relying on security patterns (fully) satisfied the criterion in a slightly relaxed form, since such patterns do not always specify the costs and associated risks with their use, even though, in theory, this is one of their essential aspects.
4. Perhaps the second-most important factor for real-life adoption of a methodology is criterion No. 4. Satisfaction was generally achieved by encapsulating security expertise in some form, and/or relying on tool-support, guidelines etc.
5. Some methodologies, such as that by Fernandez and colleagues, suffer from a lack of tool-support, even though they otherwise provide processes promoting simplicity and ease of use. In the case of Fernandez et al.'s methodology in particular, the high ease of use of the process can to a large degree compensate for the missing tools, but not entirely within the context of the proposed holistic approach (i.e. securing all layers of the software stack). Most model-driven methodologies offer good tool-support, as befits the MDE paradigm. MBSE/UMLsec, SECTET and the methodology of Lang and Schreiner make particularly strong use of tools and/or run-time software support.
6. Most methodologies make use of UML both for architectural modeling and for representing security solutions, and hence satisfied criterion No. 8 easily.

The methodology of Ren and Taylor uses Secure xADL, which is sufficiently simple to use and understand, and requires little extra effort from developers than using, for example, UML 2.x collaboration diagrams. SEPP, which uses problem frame diagrams during requirements analysis, and Secure Tropos, which uses the *i** modeling framework either throughout or, in the UMLsec variant, during the early development stages, both suffer from non-standard notation, which may be a usability hindrance.

7. Jürjens' MBSE/UMLsec and Lang and Schreiner's methodology stand out for having demonstrated real-life use in a number of industry scenarios. Other methodologies that completely satisfied the criterion were trialled on singular but full case studies developed from inception to deployment. Methodologies satisfying the criterion in part or not at all were presented in the literature using real-life examples and designs, but not full case studies, or were otherwise trialled on smaller aspects of a project.
8. Approaches to verification range from manual inspection and iteration of the process (PWSSec), generation of test cases (Fernandez et al.'s methodology) and utilizing attack scenarios (Secure Tropos) to semi-automated verification (Georg et al.'s AORDD approach). Most methodologies eschew the use of heavy formal methods, the notable exception being Ali et al.'s approach. Jürjens' MBSE/UMLsec, Model-Driven Security and the methodologies of Georg et al. all employ some form of formal techniques for verification. The verification in Model-Driven Security in particular, while based on OCL, is complex and unsuitable for non-experts. Secure Tropos can make use of formal methods from the Tropos methodology, as well as protocol verification from UMLsec (for the Secure Tropos/UMLsec variant). It is important to note that in all the cases above, the verification approaches generally target a restricted set of development aspects, e.g. a system's design, protocols, specific attack scenarios etc., and a restricted set of development stages.
9. Excessive overhead is generated in PWSSec by numerous iterations for verification and the creation of many (non-deliverable) development artefacts, as well as Georg et al.'s AORDD methodology by the (currently) iterative process of weaving attack aspects, mitigating aspects, deriving OCL from models for verification etc. SEPP satisfied this criterion on the provision that the formal specification stage is skipped, as did Model-Driven Security, on the provision that only a simple verification is performed, and that a new security modeling language is not required.
10. As already noted, use of a common repository of security knowledge was satisfied best by pattern-driven approaches and those utilizing some form of encapsulation of security solutions. Methodologies such as SECTET and PSecGCM partially satisfied the criterion, since their repositories require population of successful solutions, models etc. during an initial project. Other methodologies partially satisfying the criterion promote the use of security solution catalogs, but not with the purpose of being used as a common knowledge repository. It is worth noting that the catalogs or knowledge repositories discussed above are predominantly of security solutions. Common repositories of vulnerabilities such as the MITRE

Corporation's CVE/CWE, attack patterns and others are not employed in any of the discussed approaches, with the exceptions of SEPP, which makes use of a catalog of problem-domain patterns; PSecGCM and PWSSec, which store (security-related) requirements analysis artefacts in a repository for subsequent re-use; and the methodology of Fernandez et al., which proposes the use of a catalog of misuse patterns, currently only in its initial stages of development.

11. Only Jürjens' MBSE/UMLsec provides support for legacy software at both the design and implementation levels. Some model-driven methodologies, especially the AOSD ones, also partially satisfied this criterion in virtue of their ability to separate security code or models and introduce it into ready designs. However, understandably, the latter satisfaction is often reliant on the existence of a documented software architecture for a legacy system. Lang and Schreiner's OpenPMF, which is used for security enforcement, may support legacy systems at run-time; however, the use of OpenPMF may entail vendor lock-in issues.
12. Only Georg et al.'s AORDD methodology and Secure Tropos offered additional support for balancing security with other factors. In Secure Tropos, this occurs only during the architecture selection stage.

As a final remark, we should point out that the inability of any single methodology to satisfy all the criteria set out in the evaluation above, which reflect, essentially, industry requirements, should not suggest that the surveyed security methodologies are without merit, or that they cannot or should not be adopted. Indeed, employing any one of a number of the security methodologies reviewed to guide some part of the software process should produce better outcomes than ignoring security altogether. Considering once again the remark at the beginning of the evaluation, we should stress that there is no perfect methodology, and compromises will inevitably have to be made in both directions – for industry developers and security researchers.

5 Conclusion and Future Directions

Throughout this paper we have attempted to fill a growing gap in the literature by surveying and critically analyzing the state-of-the-art in (model-based) security methodologies for, or applicable to, both general and specific types of distributed systems. In doing so we have aimed to address the first of our questions posed in the Introduction ([Section 1]) – "why are security methodologies not better known?". Our detailed reviews can be seen as a step towards increasing awareness and appreciation of a large range of security methodologies among researchers and industry stakeholders.

Following the detailed survey, we proposed a number of criteria reflecting the characteristics security methodologies should possess to be adopted in real-life industry scenarios, and evaluated each methodology accordingly. The evaluation attempted to address the second major question posed in the Introduction – "why are security methodologies not receiving more attention from the industry?" – from a technical viewpoint.

Our evaluation showed that no single methodology fully satisfied all the criteria, suggesting that, as yet, an ideal security methodology tailored to industry needs does not exist. Although the lack of satisfaction in any given case by no means implies that a given approach cannot be adopted, it does indicate that a strong business case is missing. A number of factors contribute to this situation: high security expertise requirements, non-standard concepts and notations, partial coverage of the system life-cycle, excessive overheads, missing tool support and demonstrated credibility via case studies are just some of the main inhibitors towards more wide-spread practical use of otherwise valuable methodologies with a number of beneficial features. Of course, beneficial features alone do not necessarily add up to usable methodologies with industry-scale applicability. As a broad generalization, after all factors are weighed appropriately, the final outcome does indicate that more effort is required in advancing the current state-of-the-art in security methodologies as practical, usable approaches for securing real-life systems, distributed or otherwise.

The irony of the situation is that code-based approaches like Microsoft's SDL and OWASP's CLASP, while placing overwhelming emphasis on expertise and providing few (if any) concrete development mechanisms for introducing security at the early life-cycle stages, have – generally speaking – received greater acceptance in the industry. It may be seen as a supposition, but it would not be unfair to state that the main reasons for this are the weight of the organizations behind them, and the fact that such methodologies are, in a sense, complete and tailored for real-life use (within their paradigm, that is). Regarding the latter point, approaches that offer only a proof-of-concept, supporting one or two security solutions for example, will hardly receive the same acceptance. Regarding the former point, it should be noted that most of the methodologies surveyed in this paper originate from academia and not the industry. Indeed, many directly stem from, or are entirely within the confines of, PhD work (in the best sense), which would no doubt be a significant contributor to some of the negative factors enumerated in the paragraph above, and inevitably results in methodologies that can easily be classed as being “proof-of-concept”, rather than “production-ready”.

In most, if in not all, of these cases, there is little that can be done. Nevertheless, one can draw strong analogies between young start-up companies and methodologies originating from academia: just as a startup can be taken over by a larger company and its range and quality of products improved, so a security methodology can be elaborated and made more practical if adopted on industry projects and expanded by industry professionals in the appropriate setting. Of course, this leads to a closed circle that can only be broken by industry stakeholders taking a risk, on the one hand, and independent researchers in secure software engineering improving existing security approaches, on the other.

One important question that should be asked, and indeed has already been asked for development methodologies (see [Henderson-Sellers and Giorgini 2005, Ramsin and Paige 2008]), given the current state-of-the-art, is whether it is at all possible to produce a single or ideal, *fixed* methodology that can gain any good degree of industry adoption in the first place. While we have already considered quality factors that could help in this direction, one important factor that has not been mentioned is that not all methodologies are suitable for all purposes and scenarios: e.g. a web-services specific methodology may not be usable for systems in which web-services

play only a small, even if essential, role. In this sense, constructing *flexible* security methodologies, which are applicable to different types of systems (and in this context, different types of distributed systems) may be a promising approach. Currently, such methodologies do not exist in the published literature (cf. [Uzunov et al. 2012]). Taking this argument one step further, one could argue that a "one-size-fits-all" approach, even with a single flexible security methodology, would still be inadequate in a number of situations, and that what is required is an approach to *engineer* security methodologies, i.e. to construct (from scratch or from re-usable parts) and tailor methodologies on the fly for different project situations as required. This, of course, is not a new argument for development methodologies [Ramsin and Paige 2008]: a whole field called Situational Method Engineering (SME) exists to address such issues in that setting (see [Henderson-Sellers and Ralyté 2010]). First steps in this direction have already been taken in [Uzunov et al. sub1], which proposes a comprehensive approach to engineering security methodologies based on process patterns and meta-modeling. While this direction will almost certainly pose challenges with respect to industry adoption, especially since its source of inspiration – SME – has not received widespread industry attention as yet [Henderson-Sellers and Ralyté 2010, Low et al. 2010], the nature of these challenges may be different and perhaps lighter, inasmuch as security methodologies are something additional to a development process.

Besides constructing flexible methodologies or engineering security methodologies on the fly, our survey and analysis indicate a number of other important future directions, which are relevant for single, fixed methodologies, flexible methodologies and tailorable methodologies alike. We consider some of these directions below, and indicate how we have begun to address them in our own work where appropriate.

– *Extending the range of security solutions*: as noted in the evaluation ([Section 4]), one striking observation was the lack of a comprehensive catalog of security solutions in a number of methodologies, above all those within the model-driven paradigm. Simultaneously with this observation was another indicating that encapsulating security knowledge (using patterns, for example) is a beneficial feature. An important direction in this respect is not only to create or distill a wider range of solutions for developers to use, but also to encapsulate a wider range of security knowledge and guidance in each solution. In this sense, new security solutions types individually providing increased guidance and at the same time sufficient flexibility to allow freedom in application would be a valuable contribution towards improved security methodologies. Within the context of engineering methodologies in particular, it would also be beneficial if new solutions were methodology agnostic, and hence (re-)usable across different methodologies with different paradigms, allowing for even greater degrees of customizability to different project situations. The recently proposed security solution frames of [Uzunov et al. sub2], which organize security patterns vertically in hierarchies, and horizontally in related families, can be seen as an initial step in this direction. Other, related ideas include a consolidation of the relationships between classification schemes (e.g. [VanHilst et al. 2009] for patterns) and corresponding security solutions; extensions of existing solution types, such as the aspects of [Georg et al. 2009] and [Mouheb et al. 2009],

with both fixed and flexible parts; as well as the construction of hybrid solutions combining both design and implementation elements.

– *Encapsulation of other security knowledge*: encapsulating other forms of security knowledge such as threats, attacks, requirements etc. is another important direction following on the latter one. The problem frames of SEPP [Hatebur et al. 2007a, Schmidt 2010a] and the semantic analysis patterns of [Fernandez and Yuan 2000] are good examples of how knowledge can be captured at development stages earlier than design, however, the former are strongly tied to a particular development paradigm, while the latter are more domain-oriented and place heavier demands on developers in drawing analogies for extrapolation. The recently proposed threat patterns of [Uzunov and Fernandez sub1] are another contribution in this direction, encapsulating threats for networked and distributed systems in a fashion that is complementary to security solutions in general and also methodology agnostic.

– *Use of existing catalogs of security knowledge*: for example, CAPEC attack patterns, CWE vulnerability lists and others. The threat patterns mentioned previously can be realized using attack patterns, which can help construct penetration testing approaches [Uzunov and Fernandez sub1], however, little work has been done in this direction both by the current authors and by others.

– *Expansion of security activities and interpolation to other SDLC stages*: since a number of security methodologies suffered from a lack of (full) SDLC coverage as required by the evaluation criteria in [Section 4], interpolating methodologies to cover both earlier and later development stages is an important direction for future work. The incorporation of activities such as penetration testing, already mentioned above, or systematic attempts to evaluate the improvements in security via, for example, security metrics as in the work of [Heyman et al. 2008], is a related direction. In the context of engineering security methodologies, the aforementioned approach of [Uzunov et al. sub1] promotes this by allowing security process patterns for different activities to be instantiated, e.g. for penetration testing, for various types of verification etc. Other engineering approaches for development methodologies could offer similar or different advantages (cf. [Hurtado Alegría et al. 2011, Wagner et al. 2011]) – in all cases providing guidance in the incorporation of activities into a methodology – and their adaptation would also be a worthwhile pursuit, together with the development of new security-specific approaches.

– *Incorporation of security activities into distributed system specific development methodologies and integration*: a number of methodologies specific for certain types of distributed systems, such as SOA-based systems, not falling within the scope of this survey, e.g. SOMA [Arsanjani et al. 2008] and the work of [Dias et al. 2009], mention security but do not seriously consider its introduction in an integral way. An interesting (additional) future direction would be to integrate security engineering activities, perhaps inspired by some of the approaches reviewed here, into these and other existing development methodologies (cf. [Low et al. 2010]), including architecture-centric approaches such as ADD and QAW [Bass et al. 2003]. Following this direction leads to a consideration of (complementary) issues pertaining to the integration of security methodologies into different development methodologies, and a consideration of their mutual interaction (cf. [Uzunov et al. sub1]).

On a final note, we should point out that what has been discussed so far originates from a purely technical perspective on security methodologies, aiming at their

improvement. Adopting such methodologies in real-life scenarios also depends on various non-technical factors (see, for example, the study of [Whyte and Harrison 2010], [Mouratidis and Giorgini 2006], [Hein and Saiedian 2009]), which must be addressed. These include:

– *A code-centric view of security*: in many cases developers and managers ascribe to the view that security problems arise only during implementation and should be addressed predominantly at that level. While many vulnerabilities are indeed introduced during coding, a number of researchers and security practitioners have shown that design level problems are no less critical (see, for example, [Dowd et al. 2007, Hoglund and McGraw 2004, Jaquith 2002]) and that security features should be introduced early in the SDLC (cf. [Section 1]). The fact that many technical papers (white papers, development notes, practitioner books) are often written in a colloquial style and avoid more precise architectural diagrams (even UML) – with the idea that only words and code are understandable to developers – does nothing to improve the situation, which is, to a degree, further compounded by code-based methodologies such as Microsoft's SDL. The latter point accentuates the need in some cases for a paradigm shift, and in all cases for apposite developer education at least highlighting the scope of potential security problems, corresponding mitigation strategies, and the related benefits of model based approaches to security.

– *A limited emphasis on software only*: in a related vein, while sometimes developing a secure software application in isolation can be sufficient, often one must consider the system as a whole, i.e. the application together with its whole environment. This implies the consideration not only of all software levels (cf. [Fernandez 2003]), but also any supporting infrastructure, and entails the necessity for security to be considered in an integral way by all project roles.

– *Stubborn vision of security as an expense*: despite much research and many sound arguments for the converse, some companies continue to see the incorporation of security features during development as an extra expense, and do not appreciate the value of protecting their information, much less the value of the information about their customers (the recent security breaches of Sony's PlayStation Network and Zappos' customer database, both of which had large-scale effects, are just two pertinent examples).

In conjunction with the technical points discussed earlier, the latter points essentially conclude our attempt to answer the question of why security methodologies are not receiving more attention from the industry, and how the situation can be reversed.

Looking ahead into the future, the example of successful software development methodologies can certainly be emulated in the realm of secure software engineering, and this should be one of the major goals for new and existing security methodologies. The results and accompanying discussions in this survey, as well as the reviews and analyses of the methodologies themselves, can avail the latter task with clear indications of areas requiring improvement and important research directions, a number of which have been considered in this final section. Of course, there can be little doubt that our vision of the future is to some degree subjective, but we nevertheless believe that our concluding remarks can serve as a catalyst to further research in the area, and, ultimately, to foster a greater appreciation of the nature of some of the challenges which still need to be addressed.

Acknowledgements

We would like to acknowledge the careful reading and insightful comments by several of the *J.UCS* anonymous reviewers, which were indispensable in instigating us to make valuable improvements to our paper. We are especially grateful for the encouragement to expand the concluding part of the survey with "what we really thought" of the state-of-the-art, to be a little more provocative, and to indicate promising future directions as we see them.

References

[Alam et al. 2007] Alam, M., Breu, R., Hafner, M.: "Model-driven security engineering for trust management in SECTET"; *J. Softw.* 2(1), (2007), 47-59.

[Alberts and Dorofee 2002] Alberts, C., Dorofee, A.: "Managing Information Security Risks: The OCTAVE (SM) Approach*"*; 1st ed., Addison-Wesley Professional, (2002).

[Alexander 1979] Alexander, C.: "The Timeless Way of Building*"*; Oxford University Press, (1979).

[Alexander 2003] Alexander, I.: "Misuse cases: use cases with hostile intent"; *IEEE Software* 20(1), (2003), 58-66.

[Ali et al. 2009] Ali, Y., El-Kassas, S., Mahmoud, M.: "A rigorous methodology for security architecture modeling and verification"; In *HICSS09 42nd Hawaii International Conference on System Sciences 2009* IEEE, (2009), 1–10.

[Alkussayer and Allen 2010] Alkussayer, A., Allen, W.H.: "The ISDF Framework: Towards Secure Software Development"; *J. Inf. Process. Syst.* 6, (2010), 91-104.

[Anderson 2008] Anderson, R.J.: "Security Engineering: A Guide to Building Dependable Distributed Systems"; 2nd ed., (2008), Wiley.

[Arsanjani et al. 2008] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, K.: "SOMA: A method for developing service-oriented solutions"; *IBM Syst. J.* 47(3), (2008), 377-396.

[Basin et al. 2003] Basin, D., Doser, J., Lodderstedt, T.: "Model driven security for process-oriented systems"; In *Procs. of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, ACM Press, (2003), 100–109.

[Basin et al. 2006] Basin, D., Doser, J., Lodderstedt, T.: "Model Driven Security"; *ACM Trans. Softw. Eng. Methodol.* 15(1), (2006), 39–91.

[Basin et al. 2009] Basin, D., Clavel, M., Doser, J. & Egea, M.: "Automated analysis of security-design models"; *Inf. Softw. Technol.* 51(5), (2009), 815-831.

[Basin et al. 2011] Basin, D., Clavel, M., Egea, M.: "A decade of model-driven security"; In *Procs. of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT)*. Innsbruck, Austria, ACM, (2011), 1–10.

[Baskerville 1993] Baskerville, R.: "Information systems security design methods: implications for information systems development"; *ACM Comput. Surv.* 25(4), (1993), 375–414.

[Bass et al. 2003] Bass, L., Clements, P., Kazman, R.: "Software Architecture in Practice*";* 2nd ed., Addison-Wesley Professional (2003).

[Beck and Cunningham 1987] Beck, K., Cunningham, W.: "Using Pattern Languages for Object-Oriented Programs". *Technical Report No. CR-87-43*, (1987).

[Belapurkar et al. 2009] Belapurkar, A., Chakrabarti, A., Ponnapalli, H., Varadarajan, N., Padmanabhuni, S., Sundarrajan, S.: "Distributed Systems Security: Issues, Processes and Solutions"; Wiley (2009).

[Best et al. 2007] 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)*, Minneapolis, MN: IEEE Computer Society, (2007), 581–590.

[Blaimer et al. 2010] Blaimer, N., Bortfeldt, A., Pankratz, G.: "Patterns in Object-Oriented Analysis"; *Working Paper No. 451*, Faculty of Business Administration and Economics, University of Hagen, Germany, (2010).

[Blakley and Heath 2004] Blakley, B., Heath, C.: "Security Design Patterns". *Technical Guide*, The Open Group, (2004).

[Booch 1994] Booch, G.: "Object-Oriented Analysis and Design with Applications", Benjamin Cummings (1994).

[Braz et al. 2008] Braz, F.A., Fernandez, E.B., VanHilst, M.: "Eliciting Security Requirements through Misuse Activities"; In *Procs. 19th International Conference on Database and Expert Systems Applications (DEXA)*, IEEE, (2008), 328-333.

[Bresciani et al. 2004] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: "Tropos: An Agent-Oriented Software Development Methodology"; *Auton. Agents Multi-Agent Syst.* 8(3), (2004), 203-236.

[Breu et al. 2004] Breu, R., Burger, K., Hafner, M.: "Towards a systematic development of secure systems"; Fernández-Medina, L.J., Castro, E., Garcia-Villalba, J.C.H. (Eds.), *Inf. Syst. Secur. J. (Special Issue)* 13(3), (2004), 1-12.

[Breu et al. 2008] Breu, R., Hafner, M., Innerhofer-Oberperfler, F., Wozak, F.: "Model-Driven Security Engineering of Service Oriented Systems"; In Kaschek, R. et al. (Eds.), *Information Systems and e-Business Technologies*: Springer, Berlin Heidelberg, (2008), 59-71.

[Buschmann et al. 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: "Pattern-Oriented Software Architecture Volume 1: A System of Patterns*"*; Wiley, (1996).

[Cassou et al. 2009] Cassou, D., Bertran, B., Loriant, N., Consel, C.: "A generative programming approach to developing pervasive computing systems"; In *Procs. of the 8th International Conference on Generative Programming and Component Engineering (GPCE)*, Denver, Colorado, USA, (2009), 137-146.

[Ciancarini and Wooldridge 2001] Ciancarini, P., Wooldridge, M. (Eds.): "Agent-Oriented Software Engineering*"*; 1st ed., Springer, (2001).

[Clarke and Baniassad 2005] Clarke, S., Baniassad, E.: "Aspect-Oriented Analysis and Design: The Theme Approach"; Addison-Wesley Professional, (2005).

[Clavel et al. 2008] Clavel, M., Silva, V. da, Braga, C., Egea, M.: "Model-Driven Security in Practice: An Industrial Experience"; In I. Schieferdecker, A. Hartman (Eds.), *Model Driven Architecture – Foundations and Applications,* Springer, Berlin Heidelberg, (2008), 326–337.

[Cole 1990] Cole, R.: "A model for security in distributed systems"; *Comput. Secur.* 9(4), (1990), 319-330.

[Côté et al. 2008] Côté, I., Hatebur, D., Heisel, M., Schmidt, H., Wentzlaff, I.: "A Systematic Account of Problem Frames"; In *Procs. of EuroPLoP 2007*, (2008), 749-767.

[Dai and Cooper 2007] Dai, L., Cooper, K.: "A Survey of Modeling and Analysis Approaches for Architecting Secure Software Systems"; *Int. J. Netw. Secur.* 5(2), (2007), 187-198.

[Dashofy et al. 2005] Dashofy, E.M., Hoek, A. van der, Taylor, R.N.: "A comprehensive approach for the development of modular software architecture description languages"; *ACM Trans. Softw. Eng. Methodol.* 14(2), (2005), 199-245.

[De Win et al. 2009] De Win, B., Scandariato, R., Buyens, K., Grégoire, J., Joosen, W.: "On the secure software development process: CLASP, SDL and Touchpoints compared"; *Inf. Softw. Technol.* 51(7), (2009), 1152-1171.

[Dehlinger and Subramanian 2006] Dehlinger, J., Subramanian, N.V.: "Architecting Secure Software Systems Using an Aspect-Oriented Approach: A Survey of Current Research"; *Technical Report, Computer Science,* Iowa State University, (2006).

[Delessy and Fernandez 2008] Delessy, N.A., Fernandez, E.B.: "A pattern-driven security process for SOA applications"; In *Procs. of the 3rd International Conference on Availability, Security, and Reliability (ARES)*, Barcelona, Spain: IEEE Computer Society, (2008), 416-421.

[Delessy et al. 2008] Delessy, N., Fernandez, E.B., Larrondo-Petrie, M.M.: "A Pattern-Driven Process for Secure Service-Oriented Applications"; *FAU Technical Report (unpublished)*, (2008).

[Deng et al. 2003] Deng, Y., Wang, J., Tsai, J.J.P., Beznosov, K.: "An approach for modeling and analysis of security system architectures"; *IEEE Trans. Knowl. Data Eng.* 15(5), (2003), 1099-1119.

[Devanbu and Stubblebine 2000] Devanbu, P.T., Stubblebine, S.: "Software engineering for security: a roadmap"; In *Procs. Conference on the Future of Software Engineering*, ACM, (2000), 227–239.

[Dias et al. 2009] Dias, J.J.L., de Almeida, E.S., de Lemos Meida, S.R.: "A Systematic SOA-based Architecture Process"; In *Procs. of the 21st International Conference on Software Engineering, Knowledge Engineering (SEKE)*, Boston, Massachusetts: Knowledge Systems Institute Graduate School, (2009), 328-333.

[Dolinar et al. 2008] Dolinar, K., Fuchs, A., Klobucar, T., Llarena, R., Maña, A., Muñoz, A., Porekar, J., Rudolph, C., Saljic, S, Serrano, D.: "A3.D4.2 - Extended Set of S&D Patterns for Networks and Devices"; *Technical Report, A3.D4.2, SERENITY – 027587*, (2008).

[Dougherty et al. 2009] Dougherty, C., Sayre, K., Seacord, R.C., Svoboda, D., Togashi, K.: "Secure Design Patterns"; *Technical Report, CMU/SEI-2009-TR-010,* Carnegie-Mellon University, SEI, (2009).

[Dowd et al. 2007] Dowd, M., McDonald, J., Schuh, J.: "The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities"; 1st ed., Addison-Wesley Professional, (2007).

[Erl 2009] Erl, T.: "SOA Design Patterns"; 1st ed., Prentice Hall PTR, (2009).

[Fernandez and France 1995] Fernandez, E.B., France, R.B.: "Formal specification of real-time dependable systems"; In *Procs. of the 1st IEEE Int. Conf. on Eng. of Complex Comp. Systems (ICECCS)*, Fort Lauderdale, FL: IEEE Computer Society, (1995), 342-348.

[Fernandez and Hawkins 1997] Fernandez, E.B., Hawkins, J.C.: "Determining role rights from use cases"; In *Proceedings of the 2nd ACM Workshop on Role-based Access Control (RBAC)*, Fairfax, Virginia, US: ACM, (1997), 121–125.

[Fernandez and Nair 1998] Fernandez, E.B., Nair, K.R.: "An abstract authorization system for the Internet"; In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, IEEE, (1998), 310–315.

[Fernandez 1999] Fernandez, E.B.: "Coordination of security levels for Internet architectures"; In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications*, IEEE, (1999), 837–841.

[Fernandez and Yuan 2000] Fernandez, E.B., Yuan, X.: "Semantic analysis patterns"; In *Proceedings of the 19th International Conference on Conceptual Modeling, ER2000*. Salt Lake City, UT, (2000), 183-195.

[Fernandez and Pan 2001] Fernandez, E.B., Pan, R.: "A Pattern Language for Security Models"; In *Procs. of the 8th Conference of Pattern Languages of Programs (PLoP)*, (2001).

[Fernandez 2003] Fernandez, E.B.: "Layers and non-functional patterns"; In *Procs. of ChiliPLoP 2003*, Phoenix, (2003).

[Fernandez 2004] Fernandez, E.B.: "A methodology for secure software design"; In *Proceedings of the 2004 International Conference on Software Engineering Research and Practice (SERP)*, (2004), 21–24.

[Fernandez et al. 2005] Fernandez, E.B., Sorgente, T., Larrondo-Petrie, M.M.: "A UML-based Methodology for Secure Systems: The Design Stage"; In *Proceedings of the Third International Workshop on Security in Information Systems (WOSIS)*, Miami, FL, (2005).

[Fernandez and Larrondo-Petrie 2006] Fernandez, E.B., Larrondo-Petrie, M.M.: "Security patterns and secure systems design"; In *Proceedings of the 4th Latin American and Caribbean Conference for Engineering and Technology (LACCEI)*, Mayaguez, Puerto Rico, (2006).

[Fernandez et al. 2006a] Fernandez, E.B., Larrondo-Petrie, M.M., Sorgente, T., VanHilst, M.: "A methodology to develop secure systems using patterns"; In Giorgini, P., Mouratidis, H. (Eds.), *Integrating security and software engineering: Advances and future vision*, IDEA Press, (2006), 107-126.

[Fernandez et al. 2006b] Fernandez, E.B., VanHilst, M., Larrondo-Petrie, M.M., Huang, S.: "Defining Security Requirements through Misuse Actions"; In Ochoa, S.F., Roman, G.-C. (Eds.), *Advanced Software Engineering: Expanding the Frontiers of Software Technology*, Springer US, (2006).

[Fernandez and Larrondo-Petrie 2007] Fernandez, E.B., Larrondo-Petrie, M.M.: "Securing design patterns for distributed systems"; In Xiao, Y. (Ed.), *Security in Distributed, Grid, and Pervasive Computing*, Auerbach: CRC Press, (2007), 53-66.

[Fernandez and Yuan 2007] Fernandez, E.B., Yuan, X.: "Securing Analysis Patterns"; In *Proceedings of the 45th Annual Southeast Regional Conference*, Winston-Salem, North Carolina, ACM, (2007), 288–293.

[Fernandez et al. 2007a] Fernandez, E.B., Pelaez, J., Larrondo-Petrie, M.: "Attack Patterns: A New Forensic and Design Tool"; In Craiger, P., Shenoi, S. (Eds.), *Advances in Digital Forensics III*, Springer, New York, (2007), 345-357.

[Fernandez et al. 2007b] Fernandez, E.B., Cholmondeley, P., Zimmermann, O.: "Extending a Secure System Development Methodology to SOA"; In *Procs. of the 18th International Workshop on Database and Expert Systems Applications*, IEEE, (2007), 749-754.

[Fernandez et al. 2008] Fernandez, E.B., Washizaki, H., Yoshioka, N.: "Abstract security patterns"; *Procs. of the 15th Conference on Pattern Languages of Programs (PLoP)*, (2008).

[Fernandez 2009] Fernandez, E.B.: "Security patterns and a methodology to apply them"; In Spanoudakis, G., Maña, A., Kokolakis, S. (Eds.), *Security and Dependability for Ambient Intelligence*, Boston, MA: Springer Verlag, (2009), 37-46.

[Fernandez et al. 2009] Fernandez, E.B., Yoshioka, N., Washizaki, H.: "Modeling misuse patterns"; In *Procs. of the 2009 International Conference on Availability, Reliability and Security,* Fukuoka, Fukuoka Prefecture, Japan: IEEE Computer Society, (2009), 566-571.

[Fernandez and Mujica 2010] Fernandez, E.B., Mujica, S.: "Building Secure Systems: From Threats to Security Patterns"; In *Procs. of the 29th International Conference of the Chilean Computer Science Society (SCCC)*, IEEE, (2010), 66-70.

[Fernandez et al. 2011] Fernandez, E.B., Yoshioka, N., Washizaki, H., Jürjens, J., VanHilst, M., Pernul, G.: "Using security patterns to develop secure systems"; In Mouratidis, H. (Ed.), *Software Engineering for Secure Systems: Industrial and Research Perspectives*, IGI Global Group, (2011), 16-31.

[Fernandez and Uzunov 2012] Fernandez, E.B., Uzunov, A.V.: "Secure Middleware Patterns"; In Y. Xiang, J. Lopez, C.-C.J. Kuo and W. Zhou (Eds.), *Procs. of the 4th International Symposium on Cyberspace Safety and Security (CSS)*. Melbourne, Australia: LNCS 7672, Springer, Heidelberg, (2012), 470–482.

[Fernandez 2013] Fernandez, E.B.: "Security Patterns in Practice: Designing Secure Architectures Using Software Patterns"; Wiley, (2013).

[Fernández-Medina et al. 2009] Fernández-Medina, E., Jürjens, J., Trujillo, J., Jajodia, S.: "Model-Driven Development for secure information systems"; *Inf. Softw. Technol.* 51(5), (2009), 809-814.

[Firesmith 2003] Firesmith, D.: "Security Use Cases"; *J. Object Technol.* 2(3), (2003), 53-64.

[Flechais et al. 2003] Flechais, I., Sasse, M.A., Hailes, S.M.V.: "Bringing security home: a process for developing secure and usable systems"; In *Proceedings of the 2003 Workshop on New Security Paradigms*, ACM, (2003), 49–57.

[Flechais et al. 2007] Flechais, I., Mascolo, C., Sasse, M.A.: "Integrating security and usability into the requirements and design process"; *Int. J. Electron. Secur. Digit. Forensics* 1(1), (2007), 12–26.

[Fleurey et al. 2008] Fleurey, F., Baudry, B., France, R., Ghosh, S.: "A Generic Approach for Automatic Model Composition"; In Giese, H. (Ed.), *Models in Software Engineering,* Springer, Berlin Heidelberg, (2008), 7-15.

[Foster and Kesselman 2003] Foster, I., Kesselman, C.: "The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure"; 2nd ed., Morgan Kaufmann, (2003).

[France et al. 2002] France, R.B., Kim, D., Song, E., Ghosh, S.: "Patterns as precise characterizations of designs"; *Technical Report,* Colorado State University, (2002).

[France et al. 2004a] France, R., Ray, I., Georg, G., Ghosh, S.: "Aspect-oriented approach to early design modelling"; *IEE Proc. – Softw.* 151(4), (2004), 173-185.

[France et al. 2004b] France, R., Kim, D., Ghosh, S., Song, E.: "A UML-based pattern specification technique"; *IEEE Trans. on Softw. Eng.* 30(3), (2004), 193- 206.

[France and Rumpe 2007] France, R., Rumpe, B.: "Model-driven Development of Complex Software: A Research Roadmap"; In *Future of Software Engineering*, IEEE, (2007), 37-54.

[Frankel 2003] Frankel, D.S.: "Model Driven Architecture: Applying MDA to Enterprise Computing"; Wiley, (2003).

[Fu et al. 2006] Fu, Y., Dong, Z., He, X.: "An approach to web services oriented modeling and validation"; In *Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering (SOSE)*, ACM Press, (2006), 81.

[Gallego-Nicasio et al. 2009] Gallego-Nicasio, B., Muñoz, A., Maña, A., Serrano, D.: "Security patterns, towards a further level"; In *SECRYPT 2009 - International Conference on Security and Cryptography,* Milan, Italy: INSTICC Press, (2009), 349-356.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: "Design Patterns: Elements of Reusable Object-Oriented Software"; Addison-Wesley Professional, (1995).

[García et al. 2012] García, M., Llewellyn-Jones, D., Ortin, F., Merabti, M.: "Applying dynamic separation of aspects to distributed systems security: A case study"; *IET Softw.* 6(3), (2012), 231–248.

[Georg et al. 2002a] Georg, G., Ray, I., France, R.: "Using aspects to design a secure system"; In *Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, IEEE, (2002), 117–126.

[Georg et al. 2002b] Georg, G., France, R., Ray, I.: "Designing High Integrity Systems using Aspects"; In *Proceedings of the Fifth IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, (2002), 37-57.

[Georg et al. 2006] Georg, G., Houmb, S.H., Ray, I.: "Aspect-Oriented Risk Driven Development of Secure Applications"; In Damiani, E., Liu, P. (Eds.), *Data and Applications Security*: Springer Berlin Heidelberg, (2006), 282-296.

[Georg et al. 2009] 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), (2009), 846-864.

[Georg et al. 2010] Georg, G., Anastasakis, K., Bordbar, B., Houmb, S.H., Ray, I., Toahchoodee, M.: "Verification and Trade-Off Analysis of Security Properties in UML System Models"; *IEEE Trans. Softw. Eng.* 36(3), (2010), 338-356.

[Ghosh et al. 2002] Ghosh, A.K., Howell, C., Whittaker, J.A.: "Building Software Securely from the Ground Up"; *IEEE Softw.* 19(1), (2002), 14–16.

[Giorgini et al. 2004] Giorgini, P., Kolp, M., Mylopoulos, J., Pistore, M.: "The Tropos Methodology: An Overview"; In *Methodologies and Software Engineering for Agent Systems*, Kluwer Academic Press, (2004), 89-106.

[Giunchiglia et al. 2003] Giunchiglia, F., Mylopoulos, J., Perini, A.: "The Tropos Software Development Methodology: Processes, Models and Diagrams"; In Giunchiglia, F., Odell, J., Weiß, G. (Eds.), *Agent-Oriented Software Engineering III*, Springer, Berlin Heidelberg, (2003), 162-173.

[Gollmann 2011] Gollmann, D.: "Computer Security"; 3rd ed., Wiley, (2011).

[Gonzalez-Perez and Henderson-Sellers 2008] Gonzalez-Perez, C., Henderson-Sellers, B.: "Metamodelling for Software Engineering"; 1st ed., Wiley, (2008).

[Gregoire et al. 2007] Gregoire, J., Buyens, K., De Win, B., Scandariato, R., Joosen, W.: "On the Secure Software Development Process: CLASP and SDL Compared"; In *Third International Workshop on Software Engineering for Secure Systems (SESS: ICSE Workshops 2007)*, IEEE, (2007).

[Gritzalis et al. 1999] Gritzalis, S., Spinellis, D., Georgiadis, P.: "Security protocols over open networks and distributed systems: formal methods for their analysis, design, and verification"; *Comput. Commun.* 22(8), (1999), 697-709.

[Gunawan et al. 2009] Gunawan, L.A., Herrmann, P., Kraemer, F.A.: "Towards the Integration of Security Aspects into System Development Using Collaboration-Oriented Models"; In Ślęzak, D. et al. (Eds.), *Security Technology*, Springer, Berlin Heidelberg, (2009), 72-85.

[Gunawan et al. 2011] Gunawan, L.A., Kraemer, F.A., Herrmann, P.: "A tool-supported method for the design and implementation of secure distributed applications"; In *Proceedings of the 3rd International Conference on Engineering Secure Software and Systems (ESSoS)*, Madrid, Spain: Springer-Verlag Berlin / Heidelberg, (2011), 142–155.

[Gutiérrez et al. 2005a] Gutiérrez, C., Fernández-Medina, E., Piattini, M.: "Towards a Process for Web Services Security"; In *Proceedings of the 3rd International Workshop on Security in Information Systems (WOSIS)*, Miami, FL, INSTICC Press, (2005a), 298–308.

[Gutiérrez et al. 2005b] Gutiérrez, C., Fernández-Medina, E., Piattini, M.: "Web services enterprise security architecture: a case study"; In *Proceedings of the 2nd ACM Workshop On Secure Web Services (SWS)*, Fairfax, VA, USA, ACM, (2005b), 10–19.

[Gutiérrez et al. 2006] Gutiérrez, C., Fernández-Medina, E., Piattini, M.: "PWSSec: Process for Web Services Security"; In *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, (2006), 213–222.

[Gutiérrez et al. 2009] Gutiérrez, C., Rosado, D.G., Fernández-Medina, E.: "The practical application of a process for eliciting and designing security in web service systems". *Inf. Softw. Technol.* 51(12), (2009), 1712–1738.

[Hafiz et al. 2004] Hafiz, M., Johnson, R.E., Afandi, R.: "The security architecture of qmail"; In *Proceedings of the 11th Conference on Pattern Language of Programs (PLoP)*, Allerton, Illinois, (2004).

[Hafiz et al. 2007] Hafiz, M., Adamczyk, P., Johnson, R.E.: "Organizing security patterns"; *IEEE Softw.* 24(4), (2007), 52–60.

[Hafner et al. 2006] Hafner, M., Breu, R., Agreiter, B., Nowak, A.: "SECTET: an extensible framework for the realization of secure inter-organizational workflows"; *Internet Res.* 16(5), (2006), 491-506.

[Hafner et al. 2009] Hafner, M., Memon, M., Breu, R.: "SeAAS – A Reference Architecture for Security Services in SOA"; *J. Univers. Comput. Sci.* 15(15), (2009), 2916–2936.

[Hafner and Breu 2009] Hafner, M., Breu, R.: "Security Engineering for Service-Oriented Architectures"; 1st ed., Springer, (2009).

[Haley et al. 2008] Haley, C., Laney, R., Moffett, J., Nuseibeh, B.: "Security Requirements Engineering: A Framework for Representation and Analysis"; *IEEE Trans. Softw. Eng.* 34(1), (2008), 133–153.

[Hatebur et al. 2007a] Hatebur, D., Heisel, M., Schmidt, H.: "A security engineering process based on patterns"; In *Procs. of the 18th International Workshop on Database and Expert Systems Applications*, Regensburg, Germany: IEEE Computer Society, (2007), 734-738.

[Hatebur et al 2007b] Hatebur, D., Heisel, M., Schmidt, H.: "A Pattern System for Security Requirements Engineering"; In *Procs of the 2nd International Conference on Availability, Reliability and Security (ARES)*, IEEE Computer Society, (2007), 356-365.

[Hatebur et al. 2008] Hatebur, D., Heisel, M., Schmidt, H.: "Analysis and Component-based Realization of Security Requirements"; In *Procs. of the 3rd International Conference on Availability, Reliability and Security (ARES)*, IEEE, (2008), 195-203.

[Hatebur and Heisel 2009] Hatebur, D., Heisel, M.: "Deriving Software Architectures from Problem Descriptions"; In *Software Engineering 2009 – Workshopband,* GI, (2009), 383-392.

[Hatebur and Heisel 2010] Hatebur, D., Heisel, M.: "A UML profile for requirements analysis of dependable software"; In *Proceedings of the 29th International Conference on Computer Safety, Reliability and Security (Safecomp)*, Vienna, Austria: Springer, (2010), 317–331.

[Hatebur et al. 2011] Hatebur, Denis, Heisel, M., Jürjens, J., Schmidt, H.: "Systematic Development of UMLsec Design Models Based on Security Requirements"; In Giannakopoulou, D., Orejas, F. (Eds.), *Fundamental Approaches to Software Engineering*, Springer, Berlin Heidelberg, (2011), 232-246.

[He et al. 2002] He, X., Ding, J., Deng, Y.: "Model checking software architecture specifications in SAM"; In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Ischia, Italy: ACM, (2005), 271–274.

[Hein and Saiedian 2009] Hein, D., Saiedian, H.: "Secure Software Engineering: Learning from the Past to Address Future Challenges"; *Inf. Secur. J.: A Global Perspective*, 18(1), (2009), 8–25.

[Henderson-Sellers and Giorgini 2005] Henderson-Sellers, B., Giorgini, P.: "Agent-Oriented Methodologies", 1st ed., IGI Global, (2005).

[Henderson-Sellers and Ralyté 2010] Henderson-Sellers, B., Ralyté, J.: "Situational method engineering: state-of-the-art review"; *J. Univers. Comput. Sci. (J.UCS)* 16(3), (2010), 424–478.

[Heyman et al. 2007] Heyman, T., Yskout, K., Scandariato, R., Joosen, W.: "An Analysis of the Security Patterns Landscape"; In *Procs. 3rd International Workshop on Software Engineering for Secure Systems (SESS: ICSE Workshops 2007)*. Minneapolis, MN: IEEE, (2007).

[Heyman et al. 2008] Heyman, T., Scandariato, R., Huygens, C., Joosen, W.: "Using Security Patterns to Combine Security Metrics"; In *Procs. of the 7th International Conference on Availability, Reliability and Security (ARES)*. Barcelona, Spain: IEEE Computer Society, (2008), 1156–1163.

[Hoglund and McGraw 2004] Hoglund, G., McGraw, G.: "Exploiting Software: How to Break Code"; Addison-Wesley Professional, (2004).

[Houmb et al. 2011] Houmb, S.H., Georg, G., Petriu, D., Bordbar, B., Ray, I., Anastasakis, K., France, R.: "Balancing Security and Performance Properties During System Architectural Design"; In Mouratidis, H. (Ed.), *Software Engineering for Secure Systems*, IGI Global, (2011), 155-191.

[Howard and Lipner 2006] Howard, M., Lipner, S.: "The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software"; 1st ed., Microsoft Press, (2006).

[Huget 2004] Huget, M.P.: "Agent UML notation for multiagent system design"; *IEEE Internet Comput.* 8(4), (2004), 63- 71.

[Hurtado Alegría et al. 2011] Hurtado Alegría, J.A., Bastarrica, M.C., Quispe, A., Ochoa, S.F.: "An MDE approach to software process tailoring"; In *Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP)*. Waikiki, Honolulu, HI, USA: ACM, (2011), 43–52.

[Hutchinson et al. 2011] Hutchinson, J., Rouncefield, M., Whittle, J.: "Model-driven engineering practices in industry"; In *Proceeding of the 33rd International Conference on Software Engineering (ICSE)*, Waikiki, Honolulu, HI, USA: ACM, (2011), 633–642.

[ISO/IEC 7498-2 1989] ISO/IEC 7498-2: "Information Processing System - Open Systems Interconnection (OSI) - Basic Reference Model - Part 2: Security Architecture", (1989).

[Jaatun and Tøndel 2008] Jaatun, M.G., Tøndel, I.A.: "Covering Your Assets in Software Engineering"; In *Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES)*, Barcelona, Spain: IEEE Computer Society, (2008), 1172-1179.

[Jakob et al. 2009] Jakob, H., Loriant, N., Consel, C.: "An aspect-oriented approach to securing distributed systems"; In *Proceedings of the 2009 International Conference on Pervasive Services*. ACM, (2009), 21–30.

[Jaquith 2002] Jaquith, A.: "The security of applications: Not all are created equal"; *Research Report*, @stake, (2002), 1–12. Available at [Accessed 24 Mar 2011]: http://www.securitymanagement.com/archive/library/atstake_tech0502.pdf

[Jayaram and Mathur 2005] Jayaram, K., Mathur, A.P.: "Software engineering for secure software-state of the art: A survey"; *CERIAS Tech. Report 2005-67,* Purdue University, West Lafayette, (2005).

[Jennings 2001] Jennings, N.R.: "An agent-based approach for building complex software systems"; *Commun. ACM* 44(4), (2001) , 35–41.

[Jensen 1998] Jensen, C.: "Secure software architectures" In *Proceedings of the 8th Nordic Workshop on Programming Environment Research (WPER)*, Ronneby, (1998), 239–246.

[Jouve et al. 2008] Jouve, W., Palix, N., Consel, C., Kadionik, P.: "A SIP-Based Programming Framework for Advanced Telephony Applications"; In Schulzrinne, H., State, R., Niccolini, S. (Eds.), *Services and Security for Next Generation Networks (IPTComm 2008)*, Springer, Berlin Heidelberg, (2008), 1-20.

[Jürjens 2002a] Jürjens, J.: "UMLsec: Extending UML for Secure Systems Development"; In *Proceedings of the 5th International Conference on The Unified Modeling Language (UML)*, Springer-Verlag, (2002), 412–425.

[Jürjens 2002b] Jürjens, J.: "Using UMLsec and goal trees for secure systems development"; In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC)*, Madrid, Spain ACM, (2002), 1026–1030.

[Jürjens et al. 2002] Jürjens, J., Popp, G., Wimmel, G.: "Towards Using Security Patterns in Model-based System Development"; In *Procs. of EuroPLoP 2002*, Kloster Irsee, Germany, (2002).

[Jürjens 2005a] Jürjens, J.: "Secure Systems Development with UML"; Springer, (2005).

[Jürjens 2005b] 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)*, St. Louis, MO, USA ACM, (2005), 322–331.

[Jürjens 2006a] Jürjens, J.: "Model-Based Security Engineering with UML: Introducing Security Aspects"; In Boer, F.S. et al. (Eds.), *Formal Methods for Components and Objects*, Springer, Berlin Heidelberg, (2006), 64-87.

[Jürjens 2006b] Jürjens, J.: "Model-Based Security Engineering for Real"; In Misra, J., Nipkow, T., Sekerinski, E. (Eds.), *FM 2006: Formal Methods*, Springer, Berlin Heidelberg, (2006), 600-606.

[Jürjens et al. 2008] Jürjens, J., Schreck, J., Yu, Y.: "Automated analysis of permission-based security using UMLsec"; In *Proceedings of the Theory and Practice of Software (ETAPS), 11th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Budapest, Hungary: Springer-Verlag, (2008), 292–295.

[Jürjens 2009] Jürjens, J.: "Security and Dependability Engineering"; In Spanoudakis, G., Maña, A., Kokolakis, S. (Eds.), *Security and Dependability for Ambient Intelligence*, Boston, MA: Springer Verlag, (2009), 21-36.

[Kasal et al. 2011] Kasal, K., Heurix, J., Neubauer, T.: "Model-Driven Development Meets Security: An Evaluation of Current Approaches"; In *Procs. of the 44th Hawaii International Conference on System Sciences (HICSS)*. Kauai, HI, USA: IEEE, (2011), 1–9.

[Katt et al. 2010] Katt, B., Breu, R., Memon, M., Hafner, M.: "SECTET - Model driven Security of Service Oriented Systems based on Security-as-a-Service"; In *Presentation in Japan-Austria Joint Workshop on "ICT", October 18 - 19*, Tokyo, Japan, (2010). Available at: http://www.jst.go.jp/sicp/ws2010_austria/presentation/presentation_05.pdf (Accessed 2011)

[Khan and Zulkernine 2009] Khan, M.U.A., Zulkernine, M.: "A Survey on Requirements and Design Methods for Secure Software Development"; *Technical Report No. 2009–562, School of Computing,* Queen's University, Kingston, Ontario, Canada, (2009). Available at: http://research.cs.queensu.ca/TechReports/Reports/2009-562.pdf (Accessed: December 2010).

[Kiczales et al. 1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: "Aspect-oriented programming"; In Akşit, M., Matsuoka, S. (Eds.), *ECOOP'97 — Object-Oriented Programming*, LNCS 1241, Springer, Berlin Heidelberg, (1997), 220–242.

[Kraemer 2008] Kraemer, F.A.: "Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks"; PhD Thesis, Norwegian University of Science and Technology, (2008).

[Kraemer et al. 2009] Kraemer, F.A., Slåtten, V., Herrmann, P.: "Tool support for the rapid composition, analysis and implementation of reactive services"; *J. Syst. Softw.* 82(12), (2009), 2068–2080.

[Lang and Schreiner 2003] Lang, U., Schreiner, R.: "A Flexible, Model-Driven Security Framework for Distributed Systems"; In *Proceedings of the IASTED International Conference on Communication, Network, and Information Security (CNIS 2003)*, New York, USA, (2003).

[Lang and Schreiner 2004] Lang, U., Schreiner, R.: "OpenPMF Security Policy Framework for Distributed Systems"; In *Proceedings of the Information Security Solutions Europe (ISSE 2004) Conference,* Berlin, Germany, (2004).

[Lodderstedt et al. 2002] Lodderstedt, T., Basin, D., Doser, J.: "SecureUML: A UML-based modeling language for model-driven security"; In *UML 2002 - The Unified Modeling Language : 5th International Conference,* Dresden, Germany, Springer, (2002), 426–441.

[Lodderstedt 2003] Lodderstedt, T.: "*Model driven security from UML Models to Access Control Architectures*"; PhD Thesis, University of Freiburg, Germany, (2003). Available at: http://www.freidok.uni-freiburg.de/volltexte/1253/

[Low et al. 2010] Low, G., Mouratidis, H., Henderson-Sellers, B.: "Using a Situational Method Engineering Approach to Identify Reusable Method Fragments from the Secure TROPOS Methodology"; *J. Object Technol.* 9(4), (2010), 93–125.

[Lund et al. 2011] Lund, M.S., Solhaug, B., Stølen, K.: "Model-Driven Risk Analysis"; Berlin, Heidelberg: Springer Berlin Heidelberg, (2011).

[Maña et al. 2006] Maña, A., Sánchez-Cid, F., Serrano, D., Muñoz, A.: "Towards Secure Ambient Intelligence Scenarios"; In *Proceedings of the 18th International Conference on Software Engineering, Knowledge Engineering (SEKE)*, San Francisco, CA, (2006), 386-391.

[Maña and Pujol 2008] Maña, A., Pujol, G.: "Towards Formal Specification of Abstract Security Properties"; In *Proceedings of the 3rd International Conference on Availability, Reliability and Security*, Washington, DC, USA: IEEE Computer Society, (2008), 80–87.

[McGraw 2004] McGraw, G.: "Software security"; *IEEE Security & Privacy* 2(2), (2004), 80-83.

[McGraw 2006] McGraw, Gary: "Software Security: Building Security In"; Addison-Wesley Professional, (2006).

[Mekerke et al. 2002] Mekerke, F., Georg, G., France, R.: "Tool Support for Aspect-Oriented Design"; In Bruel, J.-M., Bellahsene, Z. (Eds.), *Advances in Object-Oriented Information Systems*, Springer, Berlin Heidelberg, (2002), 280-289.

[Meland and Jensen 2008] Meland, P.H., Jensen, J.: "Secure software design in practice"; In *Procs. of the 3rd International Conference on Availability, Reliability and Security (ARES)*. IEEE, (2008), 1164–1171.

[Memon et al. 2008] Memon, M., Hafner, M., Breu, R.: "SECTISSIMO: A Platform-independent Framework for Security Services"; In *Modeling Security Workshop In Association with MODELS 2008*, Toulouse, France, (2008).

[Memon 2011] Memon, M.: "Security Modeling with Pattern Refinement for a Security-as-a-Service Architecture"; PhD Thesis, University of Innsbruck, Austria, (2011).

[Menzel and Meinel 2009] Menzel, M., Meinel, C.: "A Security Meta-model for Service-Oriented Architectures"; In *Procs. of the 2009 IEEE International Conference on Services Computing (SCC)*, IEEE, (2009), 251-259.

[Menzel et al. 2009] Menzel, M., Thomas, I., Meinel, C.: "Security Requirements Specification in Service-Oriented Business Process Management"; In *Procs. of the 2009 International Conference on Availability, Reliability and Security (ARES)*, IEEE, (2009), 41-48.

[Menzel and Meinel 2010] Menzel, M., Meinel, C.: "SecureSOA"; In *Procs. of the 2010 IEEE International Conference on Services Computing*, IEEE Computer Society, (2010), 146-153.

[Menzel et al. 2010] Menzel, M., Warschofsky, R., Meinel, C.: "A Pattern-Driven Generation of Security Policies for Service-Oriented Architectures"; In *Procs. of the 2010 IEEE International Conference on Web Services*, IEEE, (2010), 243-250.

[Milojicic et al. 2002] Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: "Peer-to-Peer Computing"; *Technical Report, HPL-2002-57,* HP Labs, (2002).

[Moriconi et al. 1997] Moriconi, M., Qian, X., Riemenschneider, R.A., Gong, L.: "Secure software architectures"; In *Proceedings of the 1997 IEEE Symposium on Security and Privacy,* IEEE, (1997), 84–93.

[Mouheb et al. 2009] Mouheb, D., Talhi, C., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: "Weaving security aspects into UML 2.0 design models"; In *Proceedings of the 13th Workshop on Aspect-Oriented Modeling*, Charlottesville, Virginia, USA: ACM, (2009), 7–12.

[Mouheb et al. 2010] Mouheb, D., Talhi, C., Nouh, M., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: "Aspect-Oriented Modeling for Representing and Integrating Security Concerns in UML"; In Lee, R. et al. (Eds.), *Software Engineering Research, Management and Applications 2010*, Springer, Berlin Heidelberg, (2010), 197-213.

[Mouratidis et al. 2003] Mouratidis, H., Giorgini, P., Schumacher, M.: "Security Patterns for Agent Systems"; In *Procs. of the 8th European Conference on Pattern Languages of Programs (EuroPLoP)*, (2003).

[Mouratidis and Giorgini 2004] Mouratidis, H., Giorgini, P.: "Analysing Security in Information Systems"; In *Procs. of the 2nd International Workshop on Security In Information Systems (WOSIS)*, Porto, Portugal, (2004).

[Mouratidis et al. 2005] Mouratidis, H., Giorgini, P., Manson, G.: "When security meets software engineering: a case of modelling secure information systems"; *Inf. Syst.* 30(8), (2005), 609-629.

[Mouratidis and Giorgini 2006] Mouratidis, H., Giorgini, P.: "Integrating Security and Software Engineering: Advances and Future Vision"; 1st ed., IGI Global, (2006).

[Mouratidis et al. 2006a] Mouratidis, H., Weiss, M., Giorgini, P.: "Modelling Secure Systems Using An Agent Oriented Approach and Security Patterns"; *Int. J. Softw. Eng. Knowl. Eng.* 16(4), (2006), 471-498.

[Mouratidis et al. 2006b] Mouratidis, H., Jürjens, J., Fox, J.: "Towards a Comprehensive Framework for Secure Systems Development"; In Dubois, E., Pohl, K. (Eds.), *Advanced Information Systems Engineering*, Springer, Berlin Heidelberg, (2006), 48-62.

[Mouratidis 2007] Mouratidis, H.: "Secure Information Systems Engineering: A Manifesto"; Int. J. Electron. Secur. Digit. Forensics, 1(1), 27–41.

[Mouratidis and Giorgini 2007a] Mouratidis, H., Giorgini, P.: "Secure Tropos: A Security-Oriented Extension of the Tropos methodology"; *Int. J. Softw. Eng. Knowl. Eng.* 17(2), (2007), 285-309.

[Mouratidis and Giorgini 2007b] Mouratidis, H., Giorgini, P.: "Security Attack Testing (SAT)--testing the security of information systems at design time"; *Inf. Syst.* 32(8), (2007), 1166-1183.

[Mouratidis 2009] Mouratidis, H.: "Secure Tropos: An Agent Oriented Software Engineering Methodology for the Development of Health and Social Care Information Systems"; *Int. J. Comput. Sci. Secur.* 3(3), (2009), 241-271.

[Mouratidis and Jürjens 2010] Mouratidis, H., Jürjens, J.: "From Goal-Driven Security Requirements Engineering to Secure Design"; *Int. J. Intell. Syst.* 25(8), (2010), 813-840.

[Mouratidis 2011] Mouratidis, H.: "Secure Software Systems Engineering: The Secure Tropos Approach" (Invited Paper); *J. Softw.* 6(3), (2011), 331-339.

[Myagmar et al. 2005] Myagmar, S., Lee, A., Yurcik, W.: "Threat Modeling as a Basis for Security Requirements"; In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability (StorageSS)*. ACM Press, (2005), 94–102.

[Naqvi and Riguidel 2004] Naqvi, S., Riguidel, M.: "Security architecture for heterogeneous distributed computing systems"; In *Procs. of the 38th Annual International Carnahan Conference on Security Technology,* IEEE, (2004), 34-41.

[Oladimeji et al. 2007] Oladimeji, E.A., Supakkul, S., Chung, L.: "A model-driven approach to architecting secure software"; In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering,* Boston, USA, (2007), 535–551.

[OWASP 2011] OWASP: "Comprehensive, Lightweight Application Security Process (CLASP)"; (2011), Available at: http://www.owasp.org/index.php/Category:OWASP_CLASP_Project (Accessed 2011).

[Popp et al. 2003] Popp, G., Jürjens, J., Wimmel, G., Breu, R.: "Security-Critical System Development with Extended Use Cases"; In *Proceedings of the 10th Asia-Pacific Software Engineering Conference Software Engineering Conference (APSEC)*, IEEE Computer Society, (2003), 478–487.

[Ramsin and Paige 2008] Ramsin, R., Paige, R.F.: "Process-centered review of object oriented software development methodologies"; *ACM Comput. Surv.* 40(1), (2008), 1–89.

[Ray et al. 2004] Ray, I., France, R., Na, L., Georg, G.: "An aspect-based approach to modeling access control concerns"; *Inf. Softw. Technol.* 46(9), (2004), 575-587.

[Reddy et al. 2006] Reddy, Y.R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., McEachen, N., Song, E., Georg, G.: "Directives for Composing Aspect-Oriented Design Class Models"; In Rashid, A., Aksit, M. (Eds.), *Transactions on Aspect-Oriented Software Development I*, Springer, Berlin Heidelberg, (2006), 75-105.

[Ren and Taylor 2005] Ren, J., Taylor, R.N.: "A Secure Software Architecture Description Language"; In *Procs. of the Workshop on Software Security Assurance Tools, Techniques, and Metrics, in conjunction with the 20th IEEE/ACM Int. Conf. on Automated Software Engineering*. Long Beach, California, USA, (2005).

[Ren et al. 2005] Ren, J., Taylor, R., Dourish, P., Redmiles, D.: "Towards an architectural treatment of software security: a connector-centric approach"; *ACM SIGSOFT Softw. Eng. Notes* 30(4), (2005), 1–7.

[Ren 2006] Ren, Jie: "*A Connector-Centric Approach to Architectural Access Control*"; PhD Thesis, University of California Irvine, (2006).

[Reznik et al. 2007] Reznik, J., Ritter, T., Schreiner, R., Lang, U.: "Model Driven Development of Security Aspects"; *Electron. Notes Theor. Comput. Sci.* 163(2), (2007), 65-79.

[Rodriguez et al. 2009] Rodriguez, L.C., Mora, M., Vergas-Martin, M., O'Connor, R., Alvarez, F.: "Process Models of SDLCs: Comparison and Evolution"; In M.R. Syed, S.N. Syed (Eds.), *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications,* IGI Global, (2009), 76–89.

[Rojas and Mahdy 2011] Rojas, D.M., Mahdy, A.M.: "Integrating Threat Modeling in Secure Agent-Oriented Software Development"; *Int. J. Softw. Eng.* 2(3), (2011), 23-36.

[Rosado et al. 2006] Rosado, D.G., Gutiérrez, C., Fernández-Medina, E., Piattini, M.: "Security patterns and requirements for internet-based applications"; *Internet Res.* 16(5), (2006), 519-536.

[Rosado et al. 2008] Rosado, D.G., Fernández-Medina, E., López, J., Piattini, M.: "PSecGCM: Process for the development of Secure Grid Computing based Systems with Mobile devices"; In *Procs. of the 3rd International Conference on Availability, Reliability and Security (ARES),* IEEE, (2008), 136–143.

[Rosado et al. 2009a] Rosado, D.G., Fernández-Medina, E., López, J., Piattini, M.: "Obtaining Security Requirements for a Mobile Grid System"; *Int. J. Grid High Perform. Comput.* 1(3), (2009), 1-17.

[Rosado et al. 2009b] Rosado, D.G., Fernández-Medina, E., López, J.: "Applying a UML Extension to Build Use Cases Diagrams in a Secure Mobile"; In Heuser, C., Pernul, G. (Eds.), *Advances in Conceptual Modeling - Challenging Perspectives*, Springer Berlin / Heidelberg, (2009), 126-136.

[Rosado et al. 2010a] Rosado, D.G., Fernández-Medina, E., López, J., Piattini, M.: "Analysis of Secure Mobile Grid Systems: A Systematic Approach"; *Inf. Softw. Technol.* 52(5), (2010), 517-536.

[Rosado et al. 2010b] Rosado, D.G., Fernández-Medina, E., López, J., Piattini, M.: "Developing a Secure Mobile Grid System through a UML Extension"; *J. Univers. Comput. Sci.* 16(17), (2010), 2333-2352.

[Rosado et al. 2011a] Rosado, D.G., Fernández-Medina, E., López, J.: "Security Services Architecture for Secure Mobile Grid Systems"; *J. Syst. Arch.* 53(3), (2011), 240-258.

[Rosado et al. 2011b] Rosado, D.G., Fernández-Medina, E., López, J., Piattini, M.: "Systematic Design of Secure Mobile Grid Systems"; *J. Netw. Comput. Appl.* 34(4), (2011), 1168-1183.

[Sachitano et al. 2004] Sachitano, A., Chapman, R.O., Hamilton, J.A.: "Security in software architecture: a case study;" In *Proceedings of the 5th Annual IEEE SMC Information Assurance Workshop,* IEEE, (2004), 370-376.

[Sánchez et al. 2009] Sánchez, Ó., Molina, F., García-Molina, J., Toval, A.: "ModelSec: A Generative Architecture for Model-Driven Security"; *J. Univers. Comput. Sci.* 15(15), (2009), 2957-2980.

[Sánchez-Cid and Maña 2008] Sánchez-Cid, F., Maña, A.: "SERENITY Pattern-Based Software Development Life-Cycle"; In *Proceedings of the 19th International Conference on Database and Expert Systems Application*, Washington, DC, USA: IEEE Computer Society, (2008), 305–309.

[Sánchez-Cid et al. 2009] Sánchez-Cid, F., Maña, A., Spanoudakis, G., Serrano, D., Muñoz, A.: "Representation of Security and Dependability Solutions"; In G. Spanoudakis, A. Maña, S. Kokolakis (Eds.), *Security and Dependability for Ambient Intelligence*, Springer, (2009), 69-96.

[Scandariato et al. 2008] Scandariato, R., Yskout, K., Heyman, T., Joosen, W.: "Architecting software with security patterns"; *CW Reports, volume CW515, Department of Computer Science,* KU Leuven, (2008).

[Schmidt 1995] Schmidt, D.C.: "Using design patterns to develop reusable object-oriented communication software"; *Commun, ACM* 38, (1995), 65–74.

[Schmidt 2006] Schmidt, D.C.: "Model-Driven Engineering"; *Comput.* 39(2), (2006), 25-31.

[Schmidt 2010a] Schmidt, H.: "A Pattern and Component-Based Method to Develop Secure Software" (PhD Thesis); Baden-Baden, Germany: Deutscher Wissenschafts-Verlag, (2010).

[Schmidt 2010b] Schmidt, H.: "Threat- and Risk-Analysis During Early Security Requirements Engineering"; In *Procs. of the 5th International Conference on Availability, Reliability, and Security (ARES)*, IEEE, (2010), 188-195.

[Schmidt et al. 2011] Schmidt, H., Hatebur, D., Heisel, M.: "A Pattern-Based Method to Develop Secure Software"; In Mouratidis, H. (Ed.), *Software Engineering for Secure Systems: Industrial and Research Perspectives*, IGI Global, (2011), 32-74.

[Schmidt and Jürjens 2011a] Schmidt, H., Jürjens, J.: "Connecting Security Requirements Analysis and Secure Design Using Patterns and UMLsec"; In Mouratidis, H., Rolland, C. (Eds.), *Advanced Information Systems Engineering*, Springer, (2011), 367-382.

[Schmidt and Jürjens 2011b] Schmidt, H., Jürjens, J.: "UMLsec4UML2 - adopting UMLsec to support UML2"; *Technical Report 838,* Technical University of Dortmund, (2011), Available at: http://hdl.handle.net/2003/27602.

[Schneier 1999] Schneier, B.: "Attack Trees"; *Dr. Dobb's Journal*, (1999).

[Schneider 1999] Schneider, E.A.: "Security architecture-based system design"; In *Proceedings of the 1999 Workshop on New Security Paradigms (NSPW)*, ACM Press, (1999), 25-31.

[Schnjakin et al. 2009] Schnjakin, M., Menzel, M., Meinel, C.: "A pattern-driven security advisor for service-oriented architectures"; In *Proceedings of the 2009 ACM Workshop on Secure Web Services (SWS),* ACM Press, (2009), 13-20.

[Schreiner and Lang 2008] Schreiner, R., Lang, U.: "Protection of complex distributed systems"; In *Proceedings of the 2008 Workshop on Middleware Security (MidSec),* ACM Press, (2008), 7-12.

[Schumacher 2003] Schumacher, M.: "Security Engineering with Patterns: Origins, Theoretical Models, and New Applications"; 1st ed., Springer, (2003).

[Schumacher et al. 2006] Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., Sommerlad, P.: "Security Patterns: Integrating Security and Systems Engineering"; Wiley Series on Software Patterns, 1st ed., Wiley, (2006).

[SecTro 2011] SecTro: "SecTro automated modeling tool"; (2011), Available at: http://sectro.securetropos.org/ (Accessed 2011).

[Serrano et al. 2008] Serrano, D., Maña, A., Sotirious, A.D.: "Towards Precise Security Patterns"; In *Procs. of the 19th International Conference on Database and Expert Systems Application (DEXA),* Turin, Italy: IEEE, (2008), 287–291.

[Serrano et al. 2009a] Serrano, D. Maña, A., Llarena, R., Gallego-Nicasio, B., Li, K.: "SERENITY Aware System Development Process"; In Spanoudakis, G., Maña, A., Kokolakis, S. (Eds.), *Security and Dependability for Ambient Intelligence*, (2009), 165–179.

[Serrano et al. 2009b] Serrano, D., Ruiz, J.F. Muñoz, A., Maña, A., Armenteros, A., Gallego-Nicasio, B.: "Development of Applications Based on Security Patterns"; In *Procs. of the 2nd International Conference on Dependability*, Athens, Glyfada, Greece: IEEE, (2009), 111–116.

[Shaw 1996] Shaw, M.: "Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status"; In *Studies of Software Design,* Springer, (1996), 17–32.

[Shin and Gomaa 2007] Shin, M.E., Gomaa, H.: "Software requirements and architecture modeling for evolving non-secure applications into secure applications"; *Sci. Comput. Program.* 66(1), (2007), 60-70.

[Spanoudakis et al. 2009] Spanoudakis, G., Maña, A., Kokolakis, S. (Eds.): "*Security and Dependability for Ambient Intelligence*"; Springer, (2009).

[Steel et al. 2005] Steel, C., Nagappan, R., Lai, R.: "Core Security Patterns: Best Practices and Strategies for J2EE(TM), Web Services, and Identity Management"; Prentice Hall, (2005).

[Straw et al. 2004] Straw, Greg et al.: "Model Composition Directives"; In Baar, T. et al. (Eds.), In *UML 2004 - The Unified Modeling Language. Modelling Languages and Applications*. Springer, Berlin Heidelberg, (2004), 84-97.

[Swiderski and Snyder 2004] Swiderski, F., Snyder, W.: "Threat Modeling"; 1st ed., Microsoft Press, (2004).

[Talhi et al. 2009] 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), (2009), 103–122.

[Taylor et al. 2010] Taylor, R.N., Medvidovic, N., Dashofy, E.M.: "Software Architecture: Foundations, Theory, and Practice"; Wiley, (2010).

[Tryfonas et al. 2001] Tryfonas, T., Kiountouzis, E., Poulymenakou, A.: "Embedding security practices in contemporary information systems development approaches"; *Inf. Manag. Comput. Secur.* 9(4), (2001), 183-197.

[Uzunov and Fernandez sub1] Uzunov, A.V., Fernandez, E.B.: "An Extensible Pattern-based Library and Taxonomy of Security Threats for Distributed Systems"; *submitted for publication*, (n.d.).

[Uzunov et al. 2012] Uzunov, A.V., Fernandez, E.B., Falkner, K.: "Securing distributed systems using patterns: A survey"; *Comput. Secur.* 31(5), (2012), 681–703.

[Uzunov et al. sub1] Uzunov, A.V., Falkner, K., Fernandez, E.B.: "A Comprehensive Pattern-Oriented Approach to Engineering Security Methodologies"; *submitted for publication*, (n.d.).

[Uzunov et al. sub2] Uzunov, A.V., Fernandez, E.B., Falkner, K.: "A Software Engineering Approach to Authorization in Distributed, Collaborative Systems using Security Patterns and Security Solution Frames"; *submitted for publication*, (n.d.).

[VanHilst et al. 2009] VanHilst, M., Fernandez, E.B., Braz, F.: "A Multi-dimensional Classification for Users of Security Patterns"; *J. Res. Pract. Inf. Technol.* 41(2), (2009), 87-97.

[Vaquero-Gonzalez et al. 2009] Vaquero-Gonzalez, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: "A break in the clouds: towards a Cloud definition"; *ACM SIGCOMM Comput. Commun. Rev.* 39(1), (2009), 50-55.

[Villarroel et al. 2005] Villarroel, R., Fernández-Medina, E., Piattini, M: "Secure information systems development – a survey and comparison"; *Comput. Secur.* 24(4), (2005), 308-321.

[Wagner et al. 2011] Wagner, R., Fontoura, L.M., Fontoura, A.B.: "Using security patterns to tailor software process"; In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Miami, FL, USA: Knowledge Systems Institute Graduate School, (2011), 672–677.

[Washizaki et al. 2009] Washizaki, H, Fernandez, E.B., Maruyama, K., Kubo, A., Yoshioka, N.: "Improving the Classification of Security Patterns"; In *Procs. of the 2009 International Workshop on Database and Expert Systems Applications*, (2009), 165-170.

[Whitmore 2001] Whitmore, J.J.: "A method for designing secure solutions"; *IBM Syst. J.* 40(3), (2001), 747-768.

[Whyte and Harrison 2011] Whyte, B., Harrison, J.: "State of Practice in Secure Software: Experts' Views on Best Ways Ahead"; In Mouratidis, H. (Ed.), *Software Engineering for Secure Systems*, IGI Global, (2011), 1-14.

[Wooldridge 1997] Wooldridge, M.: "Agent-based software engineering"; *IEEE Proc .Softw.* 144(1), (1997), 26-37.

[Yoder and Barcalow 1997] Yoder, J., Barcalow, J.: "Architectural patterns for enabling application security"; In *Procs. of the 4th Conference on Patterns Languages of Programs (PLoP)*, Monticello, Illinois, (1997).

[Yskout et al. 2006] Yskout, K., Heyman, T., Scandariato, R., Joosen, W.: "A system of security patterns"; *CW Reports vol. CW469, Dept. of Computer Science*, KU Leuven, (2006).

[Yskout et al. 2008] Yskout, K., Heyman, T., Scandariato, R., Joosen, W.: "Security patterns: 10 years later"; *CW Reports vol. CW514, Dept. of Computer Science,* KU Leuven, (2008).

[Yu et al. 2003] Yu, H., He, X., Gao, S., Deng, Y.: "Formal Software Architecture Design of Secure Distributed Systems"; In *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, (2003), 450-457.

[Yu et al. 2004] Yu, H., He, X., Deng, Y., Mo, L.: "A formal approach to designing secure software architectures"; In *Procs. of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE)*, IEEE Computer Society, (2004), 289-290.

[Yu et al. 2005] Yu, H., Liu, D., He, X., Yang, L., Gao, S.: "Secure software architectures design by aspect orientation"; In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, IEEE, (2005), 47- 55.

[Zhang et al. 2010] Zhang, Q., Cheng, L., Boutaba, R.: "Cloud computing: state-of-the-art and research challenges"; *J. Internet Serv. Appl.* 1(1), (2010), 7-18.