SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2014; **7:**2815–2832
Published online 27 June 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.819

SPECIAL ISSUE PAPER

Integrating security mechanisms into embedded systems by domain-specific modelling

Maria Vasilevskaya[1]*, Linda Ariani Gunawan[2], Simin Nadjm-Tehrani[1] and Peter Herrmann[2]

[1] Department of Computer and Information Science, Linköping University, Linköping, Sweden
[2] Department of Telematics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

ABSTRACT

Embedded devices are crucial enablers of the Internet of Things and become increasingly common in our daily life. They store, manipulate and transmit sensitive information and, therefore, must be protected against security threats. Due to the security and also resource constraint concerns, designing secure networked embedded systems is a difficult task. Model-based development (MBD) is promoted to address complexity and ease the design of software intensive systems. We leverage MBD and domain-specific modelling to characterise common issues related to security and embedded systems that are specific to a given application domain. Security-specific knowledge relevant for a certain application domain is represented in the form of an adapted information security ontology. Further, the elements of the ontology are associated with security building blocks modelled with the MBD method SPACE. The selection of relevant security building blocks is based on (i) assets automatically elicited from the functional models, (ii) domain security knowledge captured by the security expert and (iii) the platform adopted by the embedded system engineer. A tool is developed to support the steps supporting this methodology and help to bridge between the security and embedded systems domains. We illustrate our approach with a case study from the smart metering domain. © 2013 The Authors. *Security and Communication Networks* published by John Wiley & Sons, Ltd.

KEYWORDS
model-based engineering; security engineering; security ontology; domain-specific modelling; embedded systems; smart metering

***Correspondence**
Maria Vasilevskaya, Department of Computer and Information Science, Linköping University, Linköping, Sweden.
E-mail: maria.vasilevskaya@liu.se

1. INTRODUCTION

The development of secure embedded systems software is a difficult task, but the difficulties can be significantly alleviated by applying model-based engineering. In particular, the use of models enables an integration of functional and security aspects already in the early development stages, which is necessary to capture security flaws as soon as possible [1].

Another possibility to make model-based engineering more attractive for the development of security-enhanced embedded systems is to support a better reuse of functionality created for prior systems, which reduces the engineering effort significantly [2]. We consider domain-specific modelling (DSM) [3] as one way to achieve such reuse.

With this technique, a certain application domain (e.g. smart metering) is analysed for functionalities and concepts that reappear in various realisations. This results in creation of sub-models specifying these recurring functions, which can be (re-)used when a particular system in the application domain is developed [3].

Although the exploitation of the DSM principles traditionally results in creating domain-specific languages (DSLs), we connect these methods to ontologies. In particular, we define a dedicated security language for capturing security knowledge in different application domains. This language describes both possible security breaches that are common for a certain application domain and existing practical solutions for these breaches. Further, this knowledge is stored and is available for reuse. The SecFutur project [4] has concluded that such an approach is especially valuable for systems built of embedded devices because an embedded systems engineer is typically not an expert in security, although security solutions need to be selected and tailored for a given application domain or platform.

Because networked embedded devices can often be characterised by their intensive communication and their collaborative nature, we leverage the engineering technique SPACE [5] and its tool set Arctis [6], which make the composition of interactive software from modelled components possible. For instance, the tool set currently offers 52 different sub-models to facilitate the development of apps for Android devices, from basic system access via user interface management, location handling, communication, audio, sensor management to barcode scanning [7]. So, a system model of an Android app can be effectively created by selecting suitable sub-models and combining them. This paper extends the reach of this compositional approach by integrating it into domain-specific security modelling and applying it to security concerns in networked embedded systems.

The main contribution of this paper is to exploit the concept of DSM to support the protection of systems by defining processes for security and embedded system engineers to describe and reuse security knowledge specific for a considered domain and to incorporate security mechanisms in the context of formal design modelling. The paper details these elements as follows:

- Definition of two processes. One enables a security engineer to capture domain-specific security knowledge, whereas the other facilitates an embedded system engineer to reuse the captured knowledge by an embedded system engineer at the design stage.
- Identification of a subset of the information security ontology presented by Herzog *et al.* [8] with focus on security-enhanced embedded systems development. The subset is further refined with the notion of security building blocks (SBBs) that represent available implementations of security mechanisms.
- Definition of the concept of domain-specific security models (DSSMs). A DSSM is a unified modeling language (UML) object diagram that supports the use of the above-mentioned ontology by security engineers to capture security knowledge, and by developers to utilise the knowledge when incorporating security mechanisms into an embedded system design.
- Enhancement of the method for asset elicitation presented by Vasilevskaya *et al.* [9]. In particular, we extend this method with additional steps to utilise the information about a system execution platform.
- Development of a MagicDraw [10] plug-in to support the previously described processes. This plug-in integrates MagicDraw functionality to capture the domain-specific security knowledge, the HermiT ontology reasoner [11] to explore various suitable SBBs, the Arctis [6] tool set to integrate the SBBs into a system model and Acceleo [12] to support transformation of UML-based DSSMs into the ontology format.

While we are reusing existing security and software engineering solutions, we believe that combining them in this manner is novel. By narrowing down the scope of the existing concepts to those relevant for secure embedded systems, we bridge the gap between the two expert communities, that is, embedded systems and security experts, so that integration of security aspects in embedded systems development is supported. In particular, our work is consistent with practical guidelines that are provided by software engineering life cycle standards such as ISO 12207 [13]. We make the design phase of the process more systematic when security and platform related choices are considered. Initial validation of the proposed approach has been done by industrial partners of the SecFutur project [4]. In this paper, we demonstrate our approach on a smart metering infrastructure that is currently under development in industry.

The overview of the approach is presented in Figure 1. Embedded system engineers create both functional and execution platform models of a system. The functional system model will thereafter be extended with security features using the domain-specific security knowledge gathered as the DSSMs and SBBs, whereas the execution platform system model is utilised to analyse known security breaches guiding the selection of SBBs. To create the DSSMs, the related security knowledge of particular application domains have to be captured. We propose that security engineers are best equipped to perform this. Thus, a DSSM acts as a suitable vehicle for the communication between the two expert groups.

The rest of this paper is organised as follows. In Section 2, we describe our case study and provide the necessary background. The elements of the system from this case study will be used to exemplify the new concepts, method and tools in later sections. Section 3 defines the process of capturing and storing the domain-specific

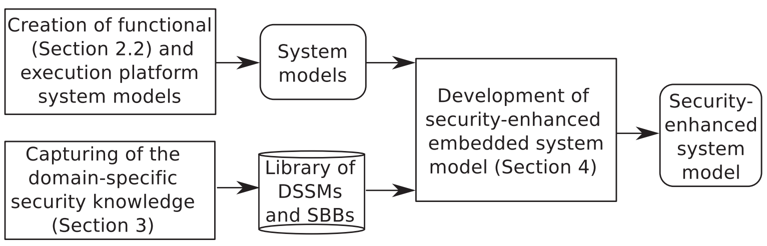

Creation of functional (Section 2.2) and execution platform system models
System models
Capturing of the domain-specific security knowledge (Section 3)
Library of DSSMs and SBBs
Development of security-enhanced embedded system model (Section 4)
Security-enhanced system model

Figure 1. An overview of the domain-specific secure system development.

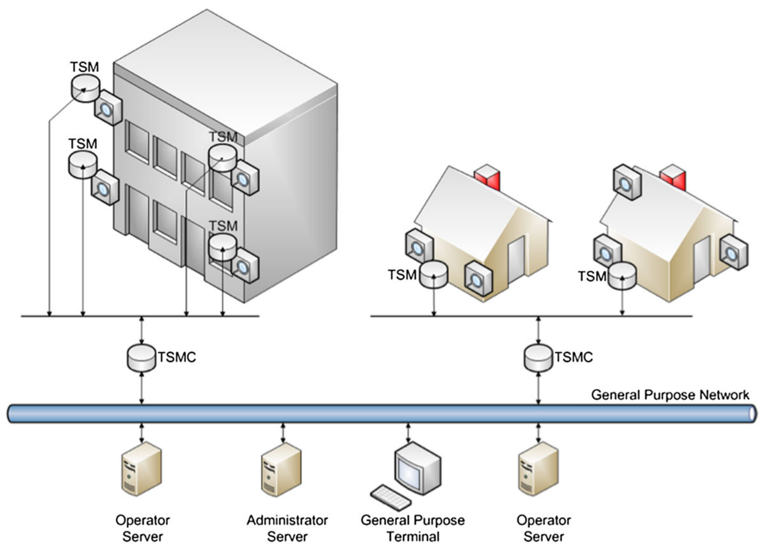


Figure 2. Trusted sensor network system overview.

security knowledge, whereas Section 4 explains the use of the knowledge to integrate protection mechanisms into embedded system designs. Thereafter, we provide a summary of some related work followed by conclusions.

2. CASE STUDY AND BACKGROUND

This section begins by presenting the case study that we use throughout this paper. Thereafter, we introduce the engineering method SPACE [5], a security ontology adopted in our approach, and the modelling and analysis of real-time embedded systems (MARTE) profile [14] used to model an execution system platform.

2.1. Smart metering application

Figure 2 depicts an infrastructure called trusted sensor network from the smart metering domain. This case study is provided by the company MixedMode for the SecFutur project [4]. Trusted sensor network is built of a set of metering devices, database servers, client applications and a communication infrastructure. The main goal of this system is to measure energy consumption at households and to associate measurements with the clients' data for billing purposes.

The actual measurement is carried out by trusted sensor modules (TSMs) consisting of a computing platform and physical sensors. The acquired measurement data are transferred via a local bus from each TSM to a trusted sensor module collector (TSMC). All measurements collected by TSMCs are eventually sent to an operator server through a general purpose network. The overall specification of this case study consists of 11 main scenarios. In our earlier work [9], we have used a scenario involving TSM and TSMC communication as a running example. In this paper, we continue developing this case study and focus on the measurement data transfer from TSMCs to an operator server. Note that the TSMC is also an embedded device, similar to TSM but with more functionality. That is, TSMC and TSM are functional modules that are implemented on the same physical platform. Because the collected data are highly sensitive, there are obvious confidentiality and integrity concerns to be considered.

2.2. Model-based engineering with SPACE

To develop secure networked embedded systems, we employ the model-based engineering method SPACE [5] together with its tool support Arctis [6]. Applications are composed of *building blocks* that can specify local behaviour as well as the interaction between several distributed entities. This specification style enables a rapid application development because, on average, more than 70% of a system specification comes from reusable building blocks provided in domain-specific libraries [2]. In turn, this strategy helps to reduce the expertise required in developing cross domain applications. Moreover, the formal semantics of the specification (see Kraemer and Herrmann [15]) makes it possible to model check relevant system properties (e.g. that the building blocks are

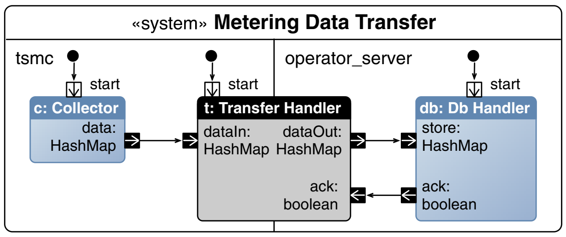


Figure 3. Functional system model of the measurement transfer scenario.

correctly integrated into activities) [6]. This is a property of the tool set that we propose to leverage.

Similar to functional building blocks, security mechanisms can be expressed as self-contained building blocks, for example, by modelling the SBBs mentioned in the introduction. Additionally, SPACE has been already used for encapsulating security functionality in a form of building blocks [16] with validation of their correct integration [17]. Finally, the recent work of Gunawan and Herrmann [18] enables compositional verification of security properties for SPACE models.

Figure 3 depicts the measurement transfer scenario modelled in SPACE. It is a UML activity consisting of two partitions, namely, *tsmc* and *operator_server*, modelling the respective entities in our case study. The activity is composed of three building blocks that are connected with some 'glue logic' through pins on their frames. The building block *c: Collector* models periodic collection of measurement data from TSMs handled by the same TSMC. Block *db: Db Handler* encapsulates the behaviour to store the data in a database of the operator, whereas block *t: Transfer Handler* manages the communication between the two components that, as will be described later, buffer data, send it and resend it in the case of a negative acknowledgement. Block *c* and *db* are local blocks because they specify local behaviour in an entity. In contrast, block *t* is a collaborative block as it also describes interaction between two entities. The three blocks (*c*, *db* and *t*), further, refer to activity diagrams that define their detailed internal behaviour as exemplified for block *t* in Figure 4.

The Petri net-like semantics of the activities models behaviour as control and object flows of tokens between the nodes of an activity via its edges. When the system starts, a token flows from each of the initial nodes (•) following the edges of the activity. In the application in Figure 3, all three inner blocks are started in the initial step. Then, periodically, the collector block emits a token containing an object of type *HashMap* through its pin *data*. This object maps TSM identifiers to measurement values at a particular time. As depicted by the outgoing edge from pin *data* of block *c*, the object is forwarded to block *t* and further to block *r: Reactive Buffer* via its pin *add* (Figure 4). This buffering block, which is taken from one of the Arctis libraries, is used to buffer measurement data that may arrive when other data is being sent but not yet acknowledged. If data are received when the buffer is empty, it is emitted immediately; otherwise, it is buffered. The pin *next* is used to obtain subsequent data. Following the outgoing edge of pin *out* of block *r*, a copy of measurement data is stored temporarily in variable *temp* by the operation *set temp*. Thereafter, the token flows through a merge node (◇), and data are sent to the other entity as illustrated by the edge crossing a partition border. In the receiver partition, the data are forwarded out of the block which, according to Figure 3, is stored in a database by block *db*.

Block *db: Db Handler* in Figure 3 will emit a token via pin *ack* containing either a positive or a negative acknowledgement. A positive acknowledgement corresponds to successful transfer of the measurement data, whereas a negative acknowledgement is issued in case the received measurements have not passed the validation test executed by the *db* block. Thus, the token emitted via pin *ack* flows further inside block *t* and, as depicted in Figure 4, reaches a decision node (◇). A positive acknowledgement leads the token flows through the outgoing edge labelled with *true*. Thereafter, operation *delete* that removes the data stored in variable *temp* is called and subsequent data, if any, is retrieved from buffer *r*. A negative acknowledgement moves the token through the edge labelled with *false*. In this case, the previously sent data are retrieved from variable *temp* and sent again.

Note that as a result of applying the SPACE method (i.e. building a system as composition of reusable building blocks), the models used in two different scenarios can share a lot of commonalities. In particular, we have reused some elements from our earlier study [9] that focuses on the TSM-TSMC communication when extending the work with TSMC-server communication.

2.3. Ontology engineering

Any ontology represents knowledge in a particular domain (such as security) as a set of concepts and the relations among those concepts.

A number of ontologies for information security have been proposed, for example, Herzog *et al.* [8] and Fenz and Ekelhart [19]. We adopt the ontology presented by Herzog *et al.*, because it builds upon classic concepts of

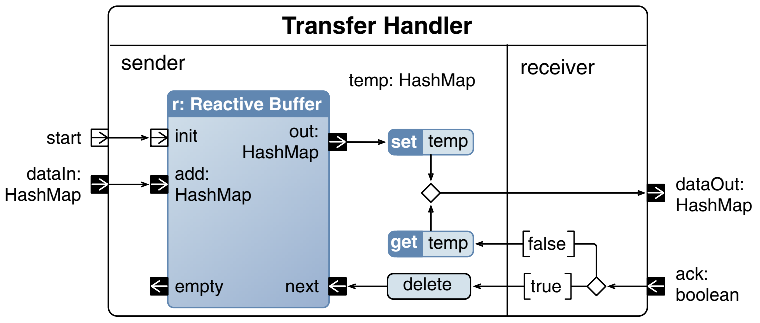


Figure 4. Detailed behaviour of the Transfer Handler block.

risk analysis and allows us to focus on the desired properties of a system rather than on threat modelling that tends to vary depending on the deployment environment of a system. Threat models are indeed included in this ontology, but in our work, we emphasise on security-related assets that essentially can be identified within a functional system model. We consider the notion of threat only when the execution platform for a system is elaborated. The section continues with a brief description of this ontology.

The core of Herzog *et al.*'s ontology [8] consists of six classes. Four of them are concepts related to risk analysis, that is, asset, vulnerability, threat and countermeasure. The remaining two classes are security goal and defence strategy. Relations between these concepts are defined as follows: an asset can *have* several vulnerabilities, a threat *threatens* assets with respect to some security goals and a countermeasure *protects* assets with respect to security goals by means of defence strategies.

In the ontology, diverse classifications of countermeasures, assets, threats and vulnerabilities relevant for information security are introduced. The security goal and defence strategy classes are described by a set of instances. Six instances are defined for the defence strategy class, namely, correction, deflection, detection, deterrence, prevention and recovery. Fifteen instances are defined for the security goal class, for example, confidentiality, integrity, authorisation and anonymity.

2.4. MARTE profile

Modeling and analysis of real-time embedded systems is an extension of the UML language, which is encapsulated in a UML profile. It contains a rich set of concepts to describe platform-dependent resources of embedded systems. We use it in our work to create models of an execution system platform.

This profile consists of several related packages. The general resource modeling package contains general concepts required for modelling of an execution platform, for example, computing, storage and communication resources. These general terms are further refined in two other packages, namely hardware resource modelling and software resource modeling. Besides, MARTE contains packages for analysis of time and performance as well as allocation of application functions onto resources. Each concept, for example, type of a platform resource, is represented as a stereotype [20] that can be used to annotate different types of UML diagrams (e.g. class and deployment) creating models of a system platform.

Experience of applying this profile to industrial cases has been reported by different researchers and practitioners, including a recent account presented by Iqbal *et al.* [21]. Moreover, as the need for analysis of resource-constrained system models becomes more apparent, new techniques are emerging to enable performance analysis based on MARTE models, including simulation [22] and analytical approaches [23].

3. DOMAIN-SPECIFIC SECURITY KNOWLEDGE

Capturing and storing of the domain-specific security knowledge is an essential step to enable its further reuse. For this task, we employ an ontology and UML [20], in particular, class and object diagrams. In the following, we explain how these two technologies are employed in our approach. Then, we explain the process of DSSM creation, that is, capturing of domain-specific security knowledge, followed by a security engineer.

3.1. Ontology development

The ontology developed by Herzog *et al.* [8] is a general information security ontology. We use an ontology to define the formal semantics of a language that assists solving specific tasks within our process (namely, capturing and use of domain-specific security knowledge). Some concepts used in our process are refinements of those introduced by Herzog *et al.* The main point of departure arises to introduce new concepts, which are security property, SBB, and domain, as depicted in Figure 5. Nevertheless, the use of the Herzog *et al.* ontology has served its purposes outlined by the authors [8], namely as a learning material about the structure of information security and as a framework for developing new detailed security taxonomies. We proceed to describe our adapted ontology when used to support the processes of capturing and using *domain-specific* security knowledge.

In our ontology, we reuse three basic concepts introduced by Herzog *et al.* [8], namely asset, security goal and defence strategy. *Assets* are the 'objects of value', in a system and which need to be protected. In our context, they

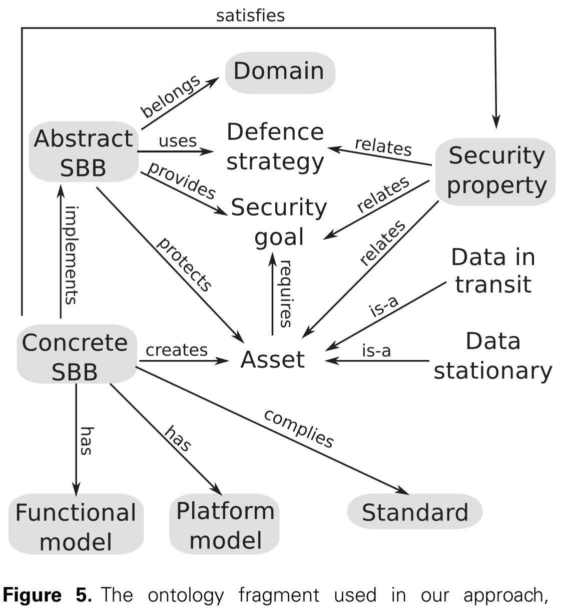


Figure 5. The ontology fragment used in our approach, adapted from [8] and [9].

can be *data stationary* residing on a physical component or *data in transit* being transmitted between different components. Other types of assets, for example, algorithms or intellectual properties, may also be considered and introduced. The protection of an asset leads to the fulfilment of a particular *security goal* such as protecting its confidentiality, integrity or availability. The countermeasures introduced later in the text follow a certain *defence strategy*, for example, preventing attacks or recovering after an attack. For the *security goal* as well as the *defence strategy*, we reuse all the terms (i.e. individuals) defined in the ontology of Herzog *et al.*

In addition to these elements, our ontology introduces new concepts, which are shown as grey rounded corner boxes in Figure 5. Two of them are *abstract* and *concrete security building blocks* replacing the notion of a countermeasure used by Herzog *et al.* [8]. These refinements enable us to distinguish between more general countermeasures represented by the abstract SBBs and their implementations specified as concrete SBBs. For example, an abstract SBB might refer to a cryptographic hash function as a general method to provide integrity, whereas the different realisations of the hash function (e.g. SHA-1, MD2 or MD5) implemented as a piece of code or hardware are each described by a concrete SBB. With respect to the resource limits of embedded systems, it is important to note that the implementations may have different resource footprints. Each concrete SBB has some *functional model* and *platform model*. The latter may be considered as description of platform components that are required by a concrete SBB for execution. Further, a concrete SBB can comply to some standard, for example, if it has passed some certification. Another concept introduced by our ontology is the notion of *security property* aggregating asset, security goal and defence strategy. Finally, we enrich the ontology with the concept of a *domain* that represents an application domain, for example, the smart metering domain.

The relations in our ontology are defined as follows. Like the countermeasures in the Herzog *et al.* ontology, an abstract SBB *protects* an asset, *provides* some security goals and *uses* some defence strategies. We have modified the *protects* relation for security goal and defence strategy used by Herzog *et al.* [8], because we find that the *provides* and *uses* relations reflect more precisely their semantics within our process. In addition to these three relations, an abstract SBB *belongs to* some application domain. A concrete SBB *implements* an abstract SBB but, in turn, may *create* certain assets itself, for example, encryption keys, that have to be protected as well. Therefore, both the assets in the core system and the assets created to realise the concrete SBBs *require* security goals. For example, the keys in some implementation of a public key cryptography mechanism have to be protected to fulfil the confidentiality and integrity goals. Functional and platform models are related to the concrete SBB concept with the *has* relation. The last relation of the concrete SBB concepts is *complies* that relates it to the standard concept. This covers common requirements in engineering of networked embedded systems. In the metering domain, for example, a system will have to fulfil legal calibration requirements following a standard. Finally, a security property *relates* assets, security goals and defence strategies, which in the following sections will be referred to by triplets `[asset, security goal, defence strategy]`.

3.2. UML representation of the ontology

To help system engineers use our ontology and to support security experts in capturing domain-specific security knowledge, we represent the ontology as a UML model, that is, as a class diagram (Figure 6). Because a DSSM is effectively an instantiation of the ontology, we specify it in the form of an instance of this class diagram. The security knowledge captured by each DSSM is used to extend our ontology presented previously with a corresponding set of axioms on relations and individuals. This enables us to use the ontology reasoning services to obtain security-relevant information. In other words, the class diagram in Figure 6 serves as a language dedicated to capture knowledge by security engineers, that is, to create DSSMs, whereas the ontology in Figure 5 is a formalism for this language [3].

The class diagram in Figure 6 consists of five classes and three relations, which are direct mappings of the elements of our ontology. The preserved classes are *Asset*, *AbstractSBB*, *ConcreteSBB*, *DataStationary* and *DataInTransit*. The preserved relations are *implements* (between *ConcreteSBB* and *AbstractSBB*), *protects* (between *Asset* and *AbstractSBB*) and *creates* (between *ConcreteSBB* and

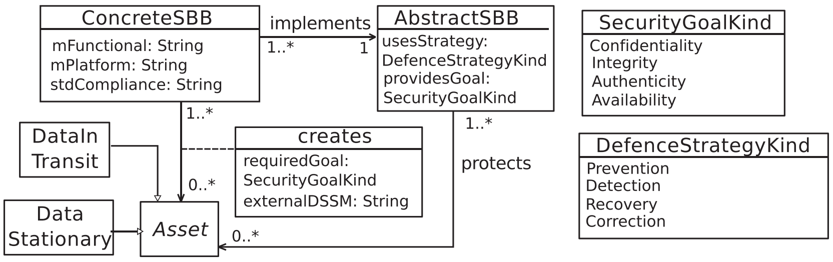


Figure 6. The unified modelling language presentation of our ontology, adapted from [9] adding the platform and standard compliance elements.

Asset). The *is-a* relation from *DataStationary* and *DataInTransit* to *Asset* in the ontology is modelled in the form of generalisations.

Other elements of the ontology are specified in a different way. Instances of the security goal and defence strategy classes are represented as the enumerations *SecurityGoalKind* respective *DefenceStrategyKind*. The *uses* relation between the abstract SBB and defence strategy classes in our ontology are represented by the property *usesStrategy* in the class *AbstractSBB*. Likewise, the *providesGoal* property in *AbstractSBB* replaces the *provides* relation between the abstract SBB and security goal classes of our ontology. Similarly, the functional model, platform model and standard concepts related to the concrete SBB concept are represented as the *mFunctional*, *mPlatform* and *stdCompliance* properties of the *ConcreteSBB* class, respectively. Finally, the *requiredGoal* property in the association class *creates* represents the relation *requires* between the created asset and the security goal classes of the ontology.

In contrast, the security property and domain concepts of the ontology are not directly represented in the UML model. This is because the triple asset, security goal and defence strategy already capture the notion of security property. Therefore, such a security property can be directly extracted from a DSSM. Analogously, the domain concept from our ontology is represented by the name of a DSSM (i.e. the object diagram).

Besides, the UML class diagram in Figure 6 has one additional property, which does not exist in the ontology, namely *externalDSSM* in the *creates* association class. The *externalDSSM* property refers to other DSSMs. The use of

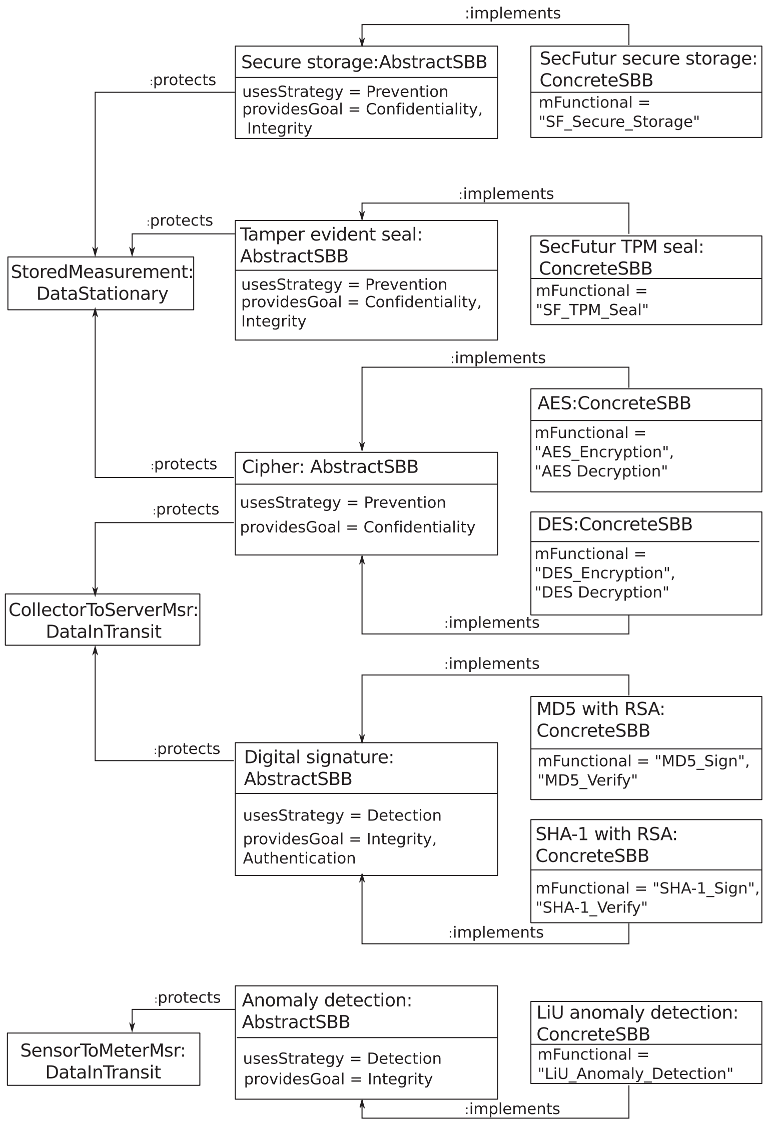

:implements
Secure storage:AbstractSBB
usesStrategy = Prevention
providesGoal = Confidentiality, Integrity
SecFutur secure storage: ConcreteSBB
mFunctional = "SF_Secure_Storage"
:protects
Tamper evident seal: AbstractSBB
usesStrategy = Prevention
providesGoal = Confidentiality, Integrity
SecFutur TPM seal: ConcreteSBB
mFunctional = "SF_TPM_Seal"
StoredMeasurement: DataStationary
AES:ConcreteSBB
mFunctional = "AES_Encryption", "AES Decryption"
Cipher: AbstractSBB
usesStrategy = Prevention
providesGoal = Confidentiality
DES:ConcreteSBB
mFunctional = "DES_Encryption", "DES Decryption"
CollectorToServerMsr: DataInTransit
MD5 with RSA: ConcreteSBB
mFunctional = "MD5_Sign", "MD5_Verify"
Digital signature: AbstractSBB
usesStrategy = Detection
providesGoal = Integrity, Authentication
SHA-1 with RSA: ConcreteSBB
mFunctional = "SHA-1_Sign", "SHA-1_Verify"
Anomaly detection: AbstractSBB
usesStrategy = Detection
providesGoal = Integrity
SensorToMeterMsr: DataInTransit
LiU anomaly detection: ConcreteSBB
mFunctional = "LiU_Anomaly_Detection"

Figure 7. A fragment of the metering domain-specific security model.

this property is needed if a created asset requires a building block, which belongs to another (known) domain.

A given DSSM is essentially an instance of the UML class diagram depicted in Figure 6. As an example, we depict in Figure 7 the UML diagram for a small fragment of the metering DSSM used for our measurement transfer scenario from Section 2. This DSSM contains three assets: *StoredMeasurement* that represents energy measurements stored on a device, *CollectorToServerMsr* that represents energy measurements sent from a collector device to an operator server and *SensorToMeterMsr* that represents energy measurements sent from a sensor to a metering device. These assets may be protected by five abstract SBBs: *Secure storage* and *Tamper evident seal* that provide confidentiality and integrity for the *StoredMeasurement* asset, *Cipher* that provides confidentiality for the *StoredMeasurement* and *CollectorToServerMsr* assets, *Digital signature* that provides integrity and authentication for the *CollectorToServerMsr* asset and *Anomaly detection* that provides integrity for the *SensorToMeterMsr* asset. Seven concrete SBBs implement these abstract SBBs. Each abstract SBB in Figure 7 is supported by one or two implementations. In general, one abstract SBB can be implemented by several concrete SBBs. For example, the *DES* and *AES* concrete SBBs implement the *Cipher* abstract SBB. These concrete SBBs have a pair of functional models, which are Arctis blocks for encryption (*AES_Encryption* and *DES_Encryption*, respectively) and decryption (*AES_Decryption* and *DES_Decryption*, respectively).

3.3. The process to create DSSMs and SBBs

In this section, we explain the proposed process for DSSM creation. The starting point of creation of DSSMs is to decide on a domain. The DSM theory inherently leaves the notion of a domain as a flexible notion. Hence, it is up to security engineers to decide what kind of domain a DSSM will describe.

In this paper, we consider application domains (e.g. metering devices, set-top boxes, banking access terminals, etc.), which can be characterised by a different set of assets and a specialised set of security solutions (i.e. concrete SBBs). Such domains can be in some relations, for example, domains can overlap or one domain can be a part of another domain. Note that the closer a selected domain is tailored to a type of a system, the more specialised and detailed solutions it contains (i.e. the set of assets and concrete SBBs). For example, both communication and metering DSSMs may be applied for our scenario described in Section 2.1, but obviously the communication DSSM will contain such general assets as ‘message’ and ‘acknowledgement’, whereas metering DSSM operates with ‘measurement’ as an asset. We continue describing the proposed process for DSSM creation.

The process of DSSM creation is depicted in Figure 8. It starts with two activities: *Creation of functional models for concrete SBBs* and *Creation of a DSSM*. The order of these activities is undefined, because it does not play a significant role in our process. Thus, if Arctis models of the considered concrete SBBs exist (e.g. they are available in the Arctis library of building blocks [2]), the corresponding activity can be omitted. The *Creation of a DSSM* activity includes definition of assets, abstract SBBs with their goals and strategies, and concrete SBBs omitting definition of a functional model (i.e. the *mFunctional* slot of the *ConcreteSBB* class depicted in Figure 6). Thus, the outcome of this activity is the definition of *the DSSM with place holders for concrete SBBs*. Note that these two activities can be extended with the third activity of creation of platform models for concrete SBBs to populate the *platform model* concept of the ontology in Figure 5. However, this goes beyond the contributions of this paper and is subject of work elsewhere [24].

The next activity is *Registration of concrete SBBs* that allows binding the created functional models of concrete SBBs (i.e. Arctis models) with the corresponding elements of the DSSM, that is, with the *mFunctional* slot. Once all concrete SBB instances of the created DSSM have been bound with the functional (Arctis) models, the DSSM can be registered.

The main outcome of the *Registration of the DSSM* activity is an ontology (described in Section 3.1) enriched with knowledge captured by the DSSM. To differentiate between the ontology depicted in Figure 5 and an evolving ontology, which is constantly extended with knowledge captured by DSSMs, we refer to the latter as the *enriched ontology*. The task of updating the enriched ontology with the knowledge captured by a newly created DSSM is implemented as transformation of elements of a DSSM and their relations into corresponding set of axioms on classes, relations and individuals. Afterwards, these axioms are added into the enriched ontology. For example, all objects of the metering DSSM in Figure 7 are added as individuals of the corresponding concepts of the ontology in Figure 5. Thus, we support the constant update of the original ontology in Figure 5 with knowledge tailored to a particular domain and captured by security experts.

Here, an important question of maintaining consistency of the enriched ontology arises. In particular, an obvious problem with such updating is pollution of the ontology with concrete SBBs that have different names but refer to the same implementation[†]. The unique name assumption of an ontology says that entities with different names refer to different elements of the real world. The OWL language has two constructs to express this assumption, namely, *owl:sameAs* or *owl:differentFrom*, that asserts that two or more given entities refer to the same or to different elements of the real world, respectively. We use the latter construct each time, when a new concrete SBB is added into the enriched ontology. However, it may be the case that security engineers will populate the enriched

[†] The trivial case, i.e. two entities with the same names, is not possible.

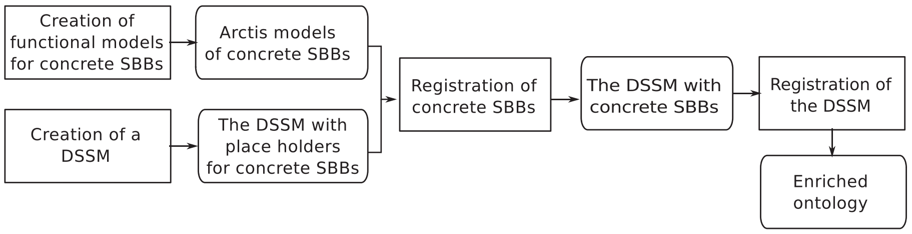
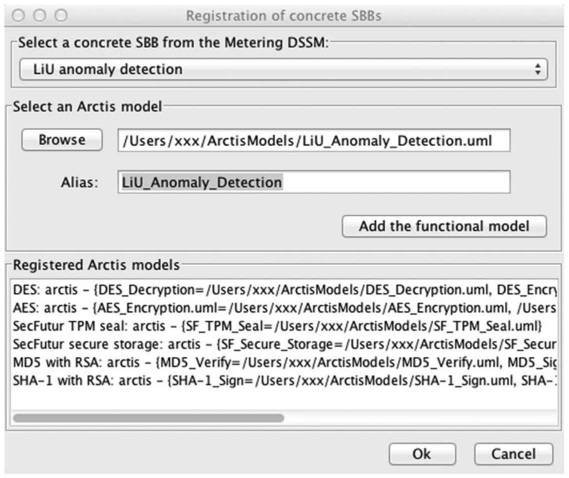

Creation of functional models for concrete SBBs
Arctis models of concrete SBBs
Creation of a DSSM
The DSSM with place holders for concrete SBBs
Registration of concrete SBBs
The DSSM with concrete SBBs
Registration of the DSSM
Enriched ontology

Figure 8. Creation of domain-specific security models (DSSMs) and security building blocks (SBBs).

ontology with concrete SBBs that actually refer to the same implementation (i.e. to the same Arctis model in our case). This situation can be resolved by the *owl:sameAs* construct that states that two or more individuals refer to the same element of the real world. However, some additional tool support may be needed to ensure that two (or more) concrete SBBs under different names refer to actually the same implementation. Techniques from the area of model comparison or model diff can be employed to calculate the difference between two models that, in our case, represent functional and platform models of considered concrete SBBs. For example, Selonen [25] and Bendix and Emanuelsson [26] have written a survey about existing model comparison methods for UML models. Besides, a set of tools exist to implement model comparison, for example, EMF Compare [27] and SiDiff [28]. The exploitation of these techniques goes beyond the scope of the current paper. However, we consider it as a further enhancement of our tool support. In the rest of this section, we outline a developed tool to support the process of DSSM creation depicted in Figure 8.

As it was mentioned earlier, DSSMs are created in the MagicDraw tool [10], whereas functional models of concrete SBBs are created in the Arctis tool [6]. To bind these two tools and to support the process of DSSM creation, we have developed a MagicDraw plug-in. In particular, this plug-in assists the creation of DSSMs by supporting the following activities of the process depicted in Figure 8:

- Creation of a DSSM: the plug-in prepares an environment for a security engineer, that is, it creates a MagicDraw project and loads the class diagram depicted in Figure 6.
- Registration of concrete SBBs: the plug-in provides an interface (shown in Figure 9) for binding concrete SBB elements of the DSSM with the corresponding Arctis models.
- Registration of a DSSM: the plug-in executes transformation of the DSSM to a set of axioms and adds them into the enriched ontology. Additionally, the plug-in can be used to upload the created DSSM to a library (local or public) for its further use.


Registration of concrete SBBs

Select a concrete SBB from the Metering DSSM: LiU anomaly detection

Select an Arctis model: /Users/xxx/ArctisModels/LiU_Anomaly_Detection.uml

Alias: LiU_Anomaly_Detection

Registered Arctis models

```
DES: arctis – {DES_Decryption=/Users/xxx/ArctisModels/DES_Decryption.uml, DES_Encry
AES: arctis – {AES_Encryption.uml=/Users/xxx/ArctisModels/AES_Encryption.uml, /Users
SecFutur TPM seal: arctis – {SF_TPM_Seal=/Users/xxx/ArctisModels/SF_TPM_Seal.uml}
SecFutur secure storage: arctis – {SF_Secure_Storage=/Users/xxx/ArctisModels/SF_Secur
MD5 with RSA: arctis – {MD5_Verify=/Users/xxx/ArctisModels/MD5_Verify.uml, MD5_Sig
SHA-1 with RSA: arctis – {SHA-1_Sign=/Users/xxx/ArctisModels/SHA-1_Sign.uml, SHA-
```

Figure 9. Registration of concrete security building blocks (SBBs) (the user interface).

4. DOMAIN-SPECIFIC SECURITY PROCESS

In this section, we describe the proposed process of integrating security aspects into formal functional models of a networked embedded system, using domain-specific security knowledge captured by DSSMs. Refining the part *Development of security-enhanced embedded system model* in the overview in Figure 1, a schematic description of the proposed approach is depicted in Figure 10.

The process starts from the functional and platform models of an embedded system created with Arctis (Section 2.2) and MagicDraw, respectively. In particular, to create platform models, we use a class diagram annotated with the corresponding MARTE stereotypes as described in Section 2.4. Thereafter, a suitable DSSM is selected, and its elements are associated with the components of the Arctis system model (Section 4.1). Note that if a required DSSM is not found, this step should be preceded by creation of a DSSM (Section 3). That is, it could be postponed until the suitable DSSM is created.

The step *Association with DSSMs* is followed by the step *Enhanced asset elicitation* (Section 4.2), which results

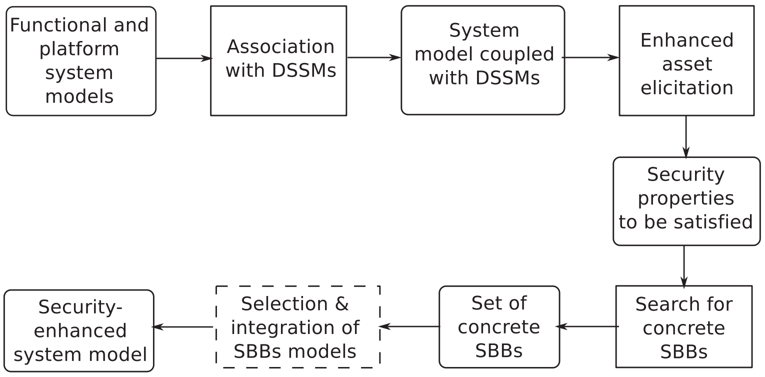

Functional and platform system models
Association with DSSMs
System model coupled with DSSMs
Enhanced asset elicitation
Security properties to be satisfied
Search for concrete SBBs
Set of concrete SBBs
Selection & integration of SBBs models
Security-enhanced system model

Figure 10. Domain-specific security process for developing security-enhanced embedded systems.

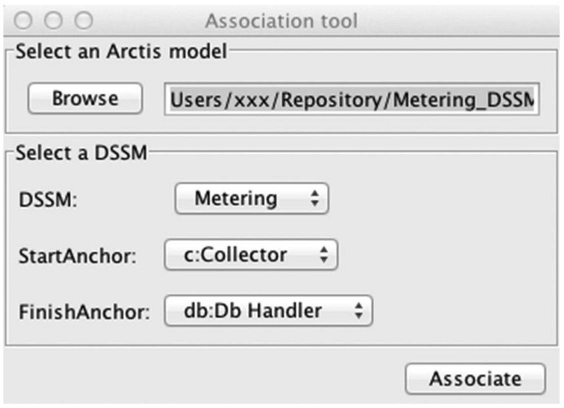
in a list of security properties to be satisfied. Afterwards, concrete SBBs that satisfy the identified security properties are inferred from the enriched ontology. Thereafter, the related set of Arctis models are fetched, for example, from the Arctis libraries.

Due to the existence of different concrete SBBs, often various ways to secure the functional system model are possible, which may differ with regard to a range of criteria. For example, an engineer needs to ensure that a system under consideration will still perform the required functionality when security mechanisms are incorporated [17]. Additionally, when dealing with embedded systems, one needs to investigate how the added security mechanisms affect the consumption of crucial resources. Compliance of considered concrete SBBs to some standard can also affect a decision taken by a system engineer. Extensive set of other possible criteria to be considered is proposed by Georg *et al.* [29]. Thus, to find the best solution, one needs to carry out analysis of desired criteria and compare different alternatives. Subsequently, a set of Arctis blocks that satisfy those criteria are integrated into the functional system model as it is reflected by the *Selection&integration of SBBs* step. The dashed line for this step in Figure 10 clarifies that the detailed development of this step is not a contribution of this paper and subject of ongoing work.

The previous process is compliant with the industrial development needs that rest on traceability of requirements within the developed system. Specifically, this is carried out by the link created between the concrete SBBs and security properties through the abstract SBBs and assets.

4.1. Association with security domain knowledge

The goal of the step *Association with DSSMs* in the proposed process is twofold: (i) a DSSM that is relevant for a system under development is identified and selected and (ii) bounds of a system (its functional model), where the knowledge (captured by a selected DSSM) should be applied, are established. Figure 11 depicts an interface of the developed plug-in to support this step. We exem-



Select an Arctis model

Users/xxx/Repository/Metering_DSSM

Select a DSSM

DSSM: Metering

StartAnchor: c:Collector

FinishAnchor: db:Db Handler

Figure 11. Association of the selected domain-specific security model (DSSM) with the system elements (the user interface).

plify the *Association with DSSMs* step with our use case presented in Section 2.

Because the case study is a smart metering application, an embedded system engineer selects the metering DSSM from the library of DSSMs (i.e. step (1)). Hence, the association is based on the matching of the system and security domains. Thereafter, those parts of the system containing data to be protected are identified (i.e. step (2)). For brevity, we discuss only the protection of the metering data that, in the functional system model, flows from block *c: Collector* in the trusted sensor module collector (TSMC) to *db: Db Handler* in the operator server (Figure 3). Thus, these two blocks are the starting respective end points of the object flow to be protected. Therefore, the identifiers of these two blocks are assigned to the fields *StartAnchor* and *FinishAnchor*, respectively, of our tool.

4.2. Enhanced asset elicitation

In this section, we further develop the asset elicitation technique initially presented by Vasilevskaya *et al.* [9]. Figure 12 outlines six steps in this enhanced asset elicitation technique.

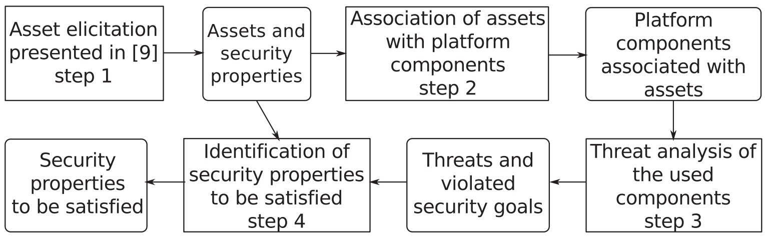

Asset elicitation presented in [9] step 1
Assets and security properties
Association of assets with platform components step 2
Platform components associated with assets
Threat analysis of the used components step 3
Threats and violated security goals
Identification of security properties to be satisfied step 4
Security properties to be satisfied

Figure 12. Outline of the enhanced asset elicitation technique.

This technique allows to identify assets existing in a system and captured by the associated DSSM using both functional and platform system models. In Section 4.2.1, we demonstrate a set of rules applied to a functional model introduced in our earlier work [9], here presented in a more concise form. Then, we explain an enhancement of this asset elicitation technique utilising the platform description information in Section 4.2.2.

4.2.1. Rules for asset elicitation on the functional model.

The starting point (step 1 in Figure 12) of the enhanced technique in Figure 12 is elicitation of assets from a functional model of a system employing the method presented by Vasilevskaya *et al.* [9]. Recall that this method allows identifying assets within a functional model and their classification according to the ontology. Here, the considered classes are 'data stationary' and 'data in transit'. This task is achieved by applying the rules **R1** and **R2** presented in Figure 13 to Arctis models. We have implemented this functionality in a tool called *Asset analyser* [9]. Afterwards, an engineer complements this classification according to an associated DSSM. However, it is worth noting that the latter task can potentially be automated given a system modelling language closely tailored to a domain (i.e. a DSL). Note that we use the label "Not an asset" to mark those proposed (by applying the rules) assets that can not be matched to any of assets from the associated DSSM. Hence, they should not be considered for the further analysis. We now proceed to explain our rules (Figure 13) and their application logic (Figure 14).

According to the SPACE semantics [15], an activity is a directed graph with a set of activity nodes V and their connecting edges E. Figure 13 presents the two identification rules **R1** and **R2**. This is a refined presentation of the seven rules proposed earlier [9]. These rules use the following functions:

- Two functions mapping an activity node and edge to their particular types, that is, $kind_V : V \to K_V$ and $kind_E : E \to K_E$, where $K_V = \{operation, merge, join, fork, decision, local, collaboration, other\}$ and $K_E = \{object, control\}$.
- The set ON of all object nodes of a given activity, that is, the data stored in the system and transported within the data flow tokens.

$$\textbf{R1:}\ \frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ (\{kind_V(source(e))\} \cup \{kind_V(target(e))\}) \\ \quad \cap \{operation, local, collaboration\} \neq \emptyset, \\ part(source(e)) \cap part(target(e)) \neq \emptyset \end{array}}{\begin{array}{l} \exists asset_1, asset_2 \in A : \\ asset_1 = \langle out_O(source(e), e), source(e), e\rangle, \\ class(asset_1) = stationary, \\ asset_2 = \langle in_O(e, target(e)), target(e), e\rangle, \\ class(asset_2) = stationary \end{array}}$$

$$\textbf{R2:}\ \frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ kind_V(source(e)) \in \{operation, merge, \\ \quad decision, fork, join, local, collaboration\}, \\ part(source(e)) \cap part(target(e)) = \emptyset \end{array}}{\begin{array}{l} \exists asset \in A : \\ asset = \langle out_O(source(e), e), source(e), e\rangle, \\ class(asset) = transit \end{array}}$$

Figure 13. Rules for asset identification.

```
function traverseBlocks (Activity A)
    $\forall e \in E$ :
        $R1, R2$
    $\forall a \in inner(A)$ :
        traverseBlocks($a$)
```

Figure 14. A function to traverse a functional system model.

- Two functions returning an object flowing to (resp. from) a given node through an edge, that is, $in_O : E \times V \to ON$ and $out_O : V \times E \to ON$.
- A function mapping a given asset to a class from our ontology, that is, $class : A \to K_A$, where $K_A = \{transit, stationary\}$ and A is the set of assets constructed from elements of the set ON.
- A function mapping a node to a set of partitions, which it belongs to, that is, $part : V \to 2^P$, where P is a set of all partitions of a given activity diagram.
- Two functions that return the source and target nodes of a given edge, that is, $source : E \to V$ and $target : E \to V$ respectively.
- A function $inner : A \to 2^A$ that assigns to an activity the set of its inner activities, that is, the activities referring to the local and collaborative blocks that

Table I. Results of asset elicitation [9].

Asset	DSSM classification
data, *dataIn*, *dataOut*, *store*, *out*, *add*, two *temp* (to and from the *set* operation), *temp* (from the *get* operation),	StoredMeasurement (Data Stationary)
ack (from the *db* block), *ack* (to the *t* blocks)	Not an asset (Data Stationary)
temp (from the merge node)	CollectorToServer-Msr (Data in Transit)
ack (that goes from the decision node to the operations *get*), *ack* (that goes from the decision node to the operation *delete*)	Not an asset (Data in Transit)

it contains. For instance, the transfer handler block contains the reactive buffer as its only inner block (Figure 4).

- A tuple $A \triangleq ON \times V \times E$ that uniquely identifies an asset.

The rule **R1** guarantees that if there is an object edge whose source and target nodes are of the kinds *operation*, *local* or *collaboration* and both the source and the target nodes belong to the same partition, then there are two data stationary assets. These assets correspond to an object outgoing from the source node and an object incoming into the target node. The rule **R2** is applied to an object edge that crosses the border of two partitions, which corresponds to the data in transit concept.

The application of the rules **R1** and **R2** to a functional system model is outlined by the function `traverseBlocks` depicted in Figure 14. Recall that each activity A is represented by a pair of nodes and edges (V, E). First, the function `traverseBlocks` traverses all edges E of this activity and applies the rules **R1** and **R2** to find the assets to be protected. In particular, application of this logic to the model in Figure 3 identifies six stationary assets using rule **R1**. For example, rule **R1** applied to the edge from block c to t returns the *data* and *dataIn* stationary assets.

Thereafter, the function is recursively applied for the activities of the inner blocks detailing the behaviour. For the metering data transfer, these are blocks c, t and db. For example, in the activity of the transfer handler block in Figure 4, the algorithm finds one stationary asset *add* due to an object edge that goes from pin *dataIn* to pin *add* of block r. Further, use of rule **R2** results in identification of three data in transit assets: the *temp* asset that flows from the merge node to pin *dataOut*, and two *ack* assets that flow from the decision node to the *get* and *delete* operations.

Table I summarises the results of applying this technique to our scenario described in Section 2. For those assets that have duplicate names (i.e. *ack* and *temp*), we have added their location information in brackets.

4.2.2. Refinement of elicited assets utilising the platform model.

Once the classification of assets is given a set of corresponding security properties (in the form `[asset, security goal, defence strategy]`) can be retrieved from the enriched ontology. This technique employs the HermiT reasoner as mentioned earlier. For example, for those assets that are classified as *Collector-ToServerMsr* (CtS) the following set of security properties is retrieved:

```
[CtS, Confidentiality, Prevention]
[CtS, Integrity, Detection]
[CtS, Authentication, Detection]
```

The next three steps (steps 2–4 in Figure 12) allow identifying which security properties must be considered as those security properties to be satisfied given a platform model. A platform model of a TSMC device for our scenario is depicted in Figure 15. We use a class diagram annotated with stereotypes of the hardware resource modelling package of the MARTE profile (Section 2.4). The modelling is carried out in the MagicDraw tool [10].

The main component of the TSMC platform is the OMPA3530 board [30]. This board includes computing elements (C64x+ DSP and ARM Cortex-A8), storage elements (NAND Flash and LPDDR), communication interfaces (I2C, SDIO, and 10/100Mbps NIC), a daughter card and the 3.5" VGA/QVGA touch screen LCD display. The daughter card is used to connect the ADE7758 sensor [31] via serial peripheral interface and the serial peripheral interface bus. Finally, 10/100 Mbps NIC is used to connect a TSMC to a communication channel (LAN). Later, we proceed to explain and to exemplify steps 2–4 (in Figure 12) that utilise the platform description information for asset elicitation.

Step 2 requires association of the elicited assets with available platform resources, for example, communication, computing and storage components. In general, any platform components that are involved in operations with assets should be mentioned during this association. In particular, we provide the following basic guidelines:

(1) Associate each data stationary asset with a computing unit that operates on it (i.e. components annotated with the *HwProcessor* stereotype and its subclasses) and a memory unit that stores it (i.e. components annotated with the *HwMemory* stereotype and its subclasses).
(2) Associate each data in transit asset with a communication channel that is used to transmit this asset (i.e. components annotated with the *HwMedia* stereotype and its subclasses) and to two interfaces on the sender and receiver ends (i.e. the *HwEndPoint* stereotype).
(3) Associate each data stationary asset with some resource (i.e. components annotated with the *HwResource* or *HwDevice* stereotypes) that contains computing and memory units, which operate with the asset and store it respectively.

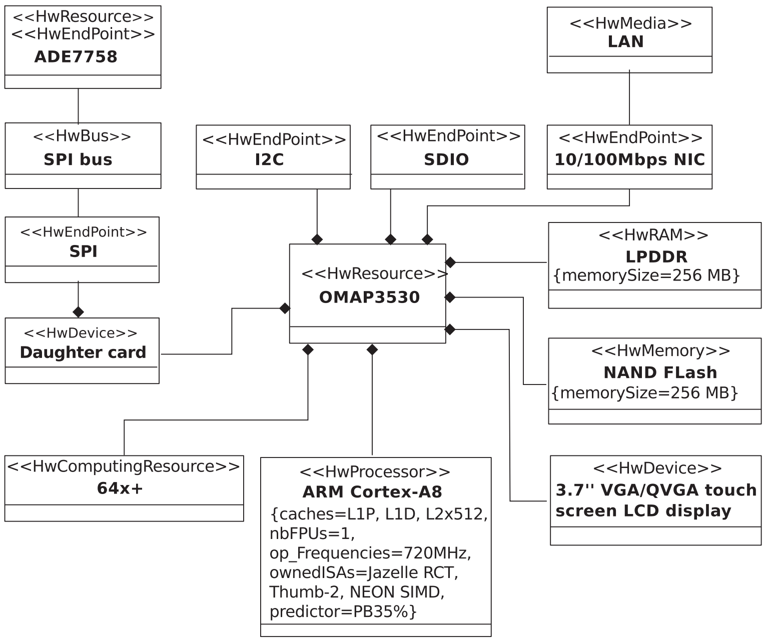


Figure 15. Platform model for a trusted sensor module collector device.

Table II. Association of the assets with the platform components.

Asset	Classification	Association
data	StoredMeasurement	ADE7758
dataIn, *dataOut*, *out*, *add*	StoredMeasurement	[NAND Flash, ARM Cortex-A8]
two *temp* (to and from the *set* operation),	StoredMeasurement	[LPDDR, ARM Cortex-A8]
temp (from the *get* operation) *temp* (from the merger node)	CollectorToServerMsr	[OMAP3530: 10/100Mbsp NIC,LAN, DBHost: 10/100Mbsp NIC]

(4) Associate each data in transit asset with some resource (i.e. components annotated with the *HwResource* or *HwDevice* stereotypes) that contains sender and receiver interfaces and a communication channel, which are involved in transmission of the asset.

Table II demonstrates association of assets with the components of the TSMC platform depicted in Figure 15. The *data* asset is associated with the ADE7758 component (the third rule of our guideline). All other data stationary assets (rows 2 and 3 in Table II) are associated with the NAND Flash or LPDDR components (i.e. memory units) and ARM Cortex-A8 component (i.e. computing unit) as it is instructed by the first guideline. Finally, following the second guideline, the data in transit assets (row 4 in Table II) are associated with 10/100 Mbps NIC components of a TSMC device and of the operator server host (not shown in Figure 15) and onto LAN, which is used as a communication channel.

Step 3 of the enhanced asset elicitation technique in Figure 12 involves an analysis of known threats given the components of the associated platform. This task may imply collaboration between security and system engineers, but the use of threat repositories can facilitate this task. For example, an engineer can query an ontology such as the one presented by Herzog *et al.*, [8] potentially extended with other expert knowledge. In our case, we use knowledge acquired within the SecFutur project employing the CORAS method [32], combined with results of threat analysis for embedded system platforms published by Ravi *et al.* [33]. Table III shows the identified threats and potentially violated security goals.

The last step of our enhanced asset elicitation technique (step 4 in Figure 12) identifies a set of security properties to be satisfied. The identification algorithm is implemented as follows. The security goal of each earlier retrieved security property (step 1 in Figure 12) is compared with the security goal violated by the threat (Table III), which targets a platform component associated with an asset of the considered security property (Table II). Now, if the security goal of the security property is equal to the security goal potentially violated by the threat, then this security property is added to the set of security properties to be satisfied. In our scenario, we have extracted the following set of security properties to be satisfied:

- SP_1: `[StoredMeasurement, Integrity, Prevention]`
- SP_2: `[CollectorToServerMsr, Confidentiality, Prevention]`

SP_1 is formulated because of the knowledge about the existence of an injection threat that potentially violates integrity of the NAND Flash component (Table III) used in association of the StoredMeasuement asset (Table II). Similarly, SP_2 is identified because of the presence of an eavesdropping threat that violates confidentiality of the LAN component (Table III), which is used in association of the CollectorToServerMsr asset (Table II).

Table III. Unacceptable threats and violated security goals.

Platform component	Threat	Violated security goal
NAND Flash	Injection	Integrity
LAN	Eavesdropping	Confidentiality

4.3. Search for concrete SBBs

The set of identified security properties to be satisfied, which are SP_1 and SP_2 for the metering scenario, is used to find a set of concrete SBBs. This procedure has already been described earlier [9]. In this section, we refine this procedure with a cycle detecting algorithm. Furthermore, we exemplify the results of a concrete SBB integration into the functional system model.

Concrete SBBs for a particular domain, which are described within a DSSM, are retrieved from the enriched ontology executing the following query:

```
ConcreteSBB
and satisfies only [SecurityProperty]
and belongsTo only [Domain]
```

This query is formulated with the Manchester syntax [34], where *and* and *only* are the syntax keywords denoting *sets intersection* and *universal quantifier*, respectively. The rest of words in the query are names of corresponding concepts and relations (Figure 6). Values in the square brackets denote parameters of the query.

Execution of the previous query for SP_1 and SP_2 retrieves two concrete SBBs for the StoredMeasurement asset (namely *SecFutur secure storage* and *SecFutur TPM seal*[‡]) and two concrete SBBs for the CollectorToServerMsr asset (namely AES and DES). Hence, due to existence of several alternatives to secure the considered scenario, an engineer needs to carry out additional analysis to select one concrete SBB for each security property to be satisfied. For example, this analysis may include investigation of the resource overhead introduced by concrete SBBs (exploiting the platform models of concrete SBBs), their effect on the original functionality of a system (exploiting the functional models of concrete SBBs), cost of concrete SBBs' integration and so on. These needs are captured by our process in Figure 10 as the *Selection&integration of concrete SBBs* step. The work on formalisation of some of these criteria is currently ongoing and not in the scope of this paper. For illustration purposes, let us assume that a system engineer decides to use the AES concrete SBB to satisfy SP_2. As a result, the system engineer is directed towards a pair of Arctis blocks, namely *AES Encryption* and *AES Decryption*.

As mentioned in Section 3.1, integrating a concrete SBB may create new assets as expressed by the *creates* and *requires* relations in our ontology. Hence, a further search of concrete SBBs (i.e. a recursive application of the above-mentioned query) is needed to fulfil the security goals required for these new assets. For this query, the security property is composed of the created asset and its required security goal, namely the attribute *requiredGoal* in Figure 6. The search is carried out within the domain specified by the *externalDSSM* attribute in Figure 6. As a result, search of concrete SBBs will continue until all security goals of all created assets are fulfilled. Alternatively, this search will lead to an empty set of SBBs and identify that a vulnerability remains in terms of an unprotected asset.

Note that such an approach can lead to a cycle, because an ontology reasoner exhaustively searches for any concrete SBB in a DSSM that satisfies the security property. For example, the query can return the same concrete SBB that has invoked it, if this concrete SBB satisfies the same security property required by its created asset. To handle

[‡] The concrete SBBs that start with the suffix 'SecFutur' are currently under development within the SecFutur [4] European project.

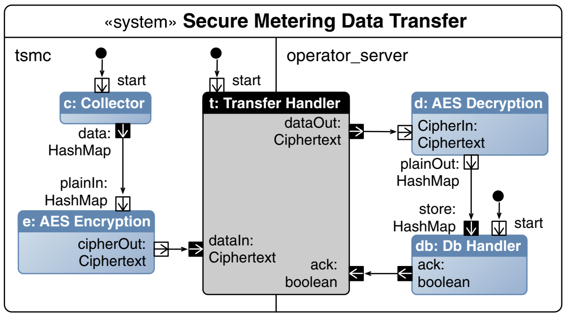


Figure 16. Adapted model protecting the transfer of measurement data.

such occurrences, we have developed an algorithm that detects and resolves such cycle conditions.

This algorithm is based on constructing a directed graph while the search for concrete SBBs goes on:

- Create a node for each found concrete SBB and asset.
- Create a directed edge from a concrete SBB to an asset if the concrete SBB creates the asset.
- Create a directed edge from an asset to a concrete SBB if the concrete SBB protects the asset.

Then, we use a cycle detection algorithm (one based on identification of backward edges during execution the depth-first search algorithm [35]) to detect cycles in the constructed graph. If there are alternative paths ignoring (by removing) the detected cycles, the search continues. Otherwise, the engineer is notified of the remaining unprotected asset.

A nice property of the previously selected Arctis blocks (i.e. the AES pair) is that they already contain a protection of the keys such that no new assets are created. Thus, we can directly continue with the integration of these blocks. Integration of the AES blocks (encryption and decryption) is easily carried out by arranging their instances before and after the block *t: Transfer Handler* as shown in Figure 16.

5. RELATED WORK

The challenge of integrating security for various types of systems has been addressed by several approaches, for instance, security patterns, security aspects, SecureUML, AVATAR and UMLsec. Security patterns [36] capture security solutions for common security challenges. Each pattern describes a solution in a human-readable text, which is sometimes augmented by UML diagrams to help developers understand the pattern better. Hamid *et al.* [37] enforce the notion of security (and dependability) patterns with formal validations. The aspect-oriented paradigm identifies security as a crosscutting concern [38]. To deal with this concern, security is encapsulated as aspects that are weaved into the main specification. SecureUML [39] deals with design and verification of role-based access control systems. Pedroza *et al.* [40] propose a SysML-based environment AVATAR to model and verify safety, authenticity and confidentiality properties. UMLsec [1] is a UML profile that is used to incorporate security-related information such as fair exchange and secure communication links in various UML diagrams and hence facilitates the assignment of security requirements and their implementation. Compared with our method in which security-related knowledge is captured by DSSMs, those approaches still require in-depth security knowledge by the system developer. In contrast, we provide a bridge between security domain experts and embedded domain experts.

With respect to integrating security into embedded systems, a number of model-based approaches were proposed. Ruiz *et al.* [41] use a technique based on threats. Attacks and threats are modelled to describe the capabilities of intruders to do harm in a system. From the threat model, security properties are identified. As opposed to Ruiz *et al.* [41], our approach is based on formal modelling. Hamid *et al.* [42] attempt to model trust properties as reusable patterns for specific domains. Although that work shares our aspirations for reusability, in this work, we consider security concerns rather than trust relations. Similar to our work, Eby *et al.* [43] adopt a DSM approach. They propose to integrate a security analysis language (SAL) into a domain-specific modelling language (DSML) for embedded systems. However, they focus on security of information flows. Saadatmand and Leveque [44] develop a method for incorporating security aspects into the ProCom component models. The authors consider two security goals, namely confidentiality and authentication and use annotations to identify those parts of a system model where integration of security aspects is needed. In contrast to this work, we have developed the enhanced asset elicitation technique to identify vulnerable parts of a system avoiding manual tagging of a system model.

Ontologies [45] are widely used to represent knowledge as a set of domain concepts and their relations. Similarly, the conceptual description of a domain through metamodelling is performed while creating a DSL [3]. Both technologies suggest a range of benefits. For example, one of the main benefit of the ontology technology is its automated reasoning services, whereas DSM enjoys wider adoption in development environments and tools. Therefore, it is not a surprise that researchers try to find a logical synergy to exploit advantages of both of these technologies [46]. For example, Walter *et al.* [47] in their recent work employ ontologies to improve the practice of DSM. The authors have developed a framework for DSL that relies on the ontology reasoning services (e.g. the inconsistency checker) to guide a designer and to validate incomplete structural domain models. In our work, we combine the ontology and DSM technologies to assist development of security-enhanced system models from different application domains. Additionally, we employ DSM to constantly populate the used ontology with the domain-specific security knowledge.

Reuse (of code and models) has been a major area of activity in software engineering, specially in the context of product lines [48]. When security is considered as a feature, then our approach can be complementary to the tools in that area by collecting knowledge about various features documented through SBBs.

Finally, to place this work in the context of our own earlier work [9], the description of the DSSM-ontology based approach is new in this paper. The platform model as an extension of the ontology, and the integrating plug-in for MagicDraw extends the earlier asset analyser tool to include all but one of the steps in Figure 10. An early draft version of this work appeared as a deliverable 4.2 in the SecFutur project [4].

6. CONCLUSION

In this paper, we have extended the reach of DSM and tool-supported model-based engineering to the sphere of the development of security-enhanced embedded networked applications. Our approach builds on the basic premise that models are viable means of communication for expert knowledge. The work flow that we provide combines earlier isolated islands of expertise: (i) general security knowledge through ontologies, (ii) efficient model-based development through reuse of domain-specific models and (iii) systematic integration of functional and extra-functional components with well-defined semantics and tool support.

We support the proposed process with the developed MagicDraw plug-in that integrates outcomes of different tools used for different purposes, namely MagicDraw, HermiT and Arctis. Our approach has been illustrated using fragments of a model for a nontrivial case study, that is, a real smart metering system currently under industrial development. Ontologies have been demonstrated to be capable of dealing with large data sets in a scalable manner [49]. Moreover, we believe that the proposed work flow is engineer-friendly. This is a hypothesis that is currently evaluated extensively in the ongoing European project SecFutur [4].

Further work will explore other important extra-functional requirements that should be considered during selection of security building blocks to be integrated, which will be consequently incorporated in our process and tool support. In particular, current work on utilisation of the platform-related information of concrete SBBs (introduced as the platform model concept), to support making an informed decision on its integration into a system is in progress. Also, refinement of the asset concept to address different levels of importance will be a direction for future works. Our endeavour follows the emerging trend of systematic treatment of security in embedded systems. There is still a long way before getting this vision into every day practice, but we believe that our work makes a small step on this path.

REFERENCES

1. Jürjens J. *Secure System Development with UML.* Springer-Verlag: Berlin Heidelberg, 2005.
2. Kraemer FA, Herrmann P. Automated encapsulation of UML activities for incremental development and verification, *International Conference on Model Driven Engineering, Languages and Systems (MoDELS),* LNCS, Denver, CO, USA, Springer, 2009; 571–585.
3. Kelly S, Tolvannen JP. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, Inc.: Hoboken, New Jersey, 2008.
4. *The SecFutur project: design of secure and energy-efficient embedded systems for future Internet application*. www.secfutur.eu, last visited May 2013.
5. Kraemer FA. Engineering reactive systems: a compositional and model-driven method based on collaborative building blocks. *Ph.D. Thesis*, Norwegian University of Science and Technology, August 2008.
6. Kraemer FA, Slåtten V, Herrmann P. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software* 2009; **82**(12): 2068–2080.
7. Kraemer FA. Engineering android applications based on UML activities, *Model Driven Engineering Languages and Systems (MODELS),* LNCS, Wellington, New Zealand, Springer Berlin / Heidelberg, 2011; 183–197.
8. Herzog A, Shahmehri N, Duma C. An ontology of information security. In *Journal of Techniques and Applications for Advanced Information Privacy and Security*. IGI Global: Hershey, Pennsylvania, USA, 2007; 1–23.
9. Vasilevskaya M, Gunawan LA, Nadjm-Tehrani S, Herrmann P. Security asset elicitation for collaborative models, *Model-Driven Security Workshop (MDSec) in Conjunction with MoDELS, ACM Digital Library (DL)*, Innsbruck, Austria, 2012; 7–13.
10. *MagicDraw*. www.magicdraw.com, last visited May 2013.
11. *HermiT Reasoner*. www.hermit-reasoner.com, last visited February 2013.
12. *Acceleo*. www.eclipse.org/acceleo/, last visited February 2013.
13. ISO/IEC 12207:2008. *Systems and software engineering – Software life cycle processes*.
14. Object Management Group. *UML Profile for MARTE: modeling and analysis of real-time embedded systems, version 1.1*, June 2011. Document number: formal/2011-06-02.
15. Kraemer FA, Herrmann P. Reactive semantics for distributed UML activities. In *Formal Techniques*

for Distributed Systems (FMOODS/FORTE), LNCS. Springer-Verlag: Berlin, Heidelberg, 2010; 17–31.

16. Gunawan LA, Herrmann P, Kraemer FA. Towards the integration of security aspects into system development using collaboration-oriented models. In *International Conference on Security Technology (SecTech)*, Ślęzak D, hoon Kim T, Fang WC, Arnett KP (eds), Communications in Computer and Information Science. Springer: Berlin Heidelberg, 2009; 72–85.
17. Gunawan LA, Kraemer FA, Herrmann P. A tool-supported method for the design and implementation of secure distributed applications. In *Engineering Secure Software and Systems (ESSoS)*, LNCS. Springer-Verlag: Berlin, Heidelberg, 2011; 142–155.
18. Gunawan LA, Herrmann P. Compositional verification of application-level security properties. In *Engineering Secure Software and Systems (essos)*, LNCS. Springer–Verlag: Berlin Heidelberg, 2013; 75–90.
19. Fenz S, Ekelhart A. Formalizing information security knowledge, *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, ACM, 2009; 183–194.
20. Object Management Group. *Unified modeling language: superstructure, version 2.4.1*, August 2011. Document number: formal/2011-08-06.
21. Iqbal MZ, Ali S, Yue T, Briand L. Experiences of applying uml/marte on three industrial projects, *Model Driven Engineering Languages and Systems, (MoDELS)*, Innsbruck, Austria, Springer, 2012; 642–658.
22. Robert T, Perrier V. *CoFluent methodology for UML*. cofluent design, white paper 2010.
23. Petriu DC. *Model Driven Engineering for Distributed Real-Time Systems: MARTE modelling, Model Transformations and Their Usages*, chap. Software Model-based Performance Analysis. Wiley Online Library: London: ISTE, 2010.
24. Vasilevskaya M, Nadjm-Tehrani S. *Support for cross-domain composition of embedded systems using MARTE models*. Submitted 2013.
25. Selonen P. A review of UML model comparison approaches, *Nordic Workshop on Model Driven Engineering*, Ronneby, Sweden, 2007.
26. Bendix L, Emanuelsson P. Diff and merge support for model based development, *Workshop on Comparison and Versioning of Software Models (CVSM)*, Leipzig, Germany, ACM, 2008; 31–34.
27. *EMF Compare*. www.eclipse.org/emf/compare/, last visited April 2013.
28. *SiDiff*. http://pi.informatik.uni-siegen.de/sidiff/, last visited May 2013.
29. Georg G, Anastasakis K, Bordbar B, Houmb SH, Ray I, Toahchoodee M. *Verification and Trade-Off Analysis of Security Properties in UML System Models*. IEEE Press: USA, NJ, 2010.
30. *OMAP3530*. www.ti.com, last visited Feburary 2013.
31. *ADE7758: poly phase multifunction energy metering ic with per phase information*. www.analog.com, last visited Feburary 2013.
32. Braber F, Hogganvik I, Lund MS, Stølen K, Vraalsen F. *Model-Based Security Analysis in Seven Steps – a Guided Tour to the CORAS Method*. Springer-Verlag: Berlin and Heidelberg, 2007.
33. Ravi S, Raghunathan A, Chakradhar S. Tamper resistance mechanisms for secure embedded systems, *International Conference on VLSI Design*, Mumbai, India, IEEE, 2004; 605–611.
34. *Ontology Language Manchester Syntax*. www.w3.org/TR/owl2-manchester-syntax/, last visited April 2013.
35. Cormen TH, Stein C, Rivest RL, Leiserson CE. *Introduction to Algorithms*, 2nd edn. The MIT Press and McGraw-Hill Book Company: Cambridge, MA, USA, 2001.
36. Schumacher M, Fernandez-Buglioni E, Hybertson D, Buschmann F, Sommerlad P. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons: West Sussex, England, 2005.
37. Hamid B, Gürgens S, Jouvray C, Desnos N. Enforcing S&D pattern design in RCES with modeling and formal approaches, *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, Wellington, New Zealand, 2011; 319–333.
38. Georg G, Ray I, Anastasakis K, Bordbar B, Toahchoodee M, Houmb SH. *An aspect-oriented methodology for designing secure applications*, 2009.
39. Lodderstedt T, Basin D, Doser J. SecureUML: a UML-based modeling language for model-driven security, *International Conference on the Unified Modeling Language (UML)*, Dresden, Germany, 2002; 426–441.
40. Pedroza G, Apvrille L, Knorreck D. AVATAR: a SysML environment for the formal verification of safety and security properties, *IEEE International Conference on New Technologies of Distributed Systems (NOTERE)*, 2011.
41. Ruiz JF, Harjani R, Mana A, Desnitsky V, Kotenko I, Chechulin A. A methodology for the analysis and modeling of security threats and attacks for systems of embedded components, *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Pisa, Italy, IEEE Computer Society, 2012; 261–268.
42. Hamid B, Desnos N, Grepet C, Jouvray C. Model-based security and dependability patterns in RCES – the TERESA approach, *International Workshop on Security and Dependability for Resource Constrained*

Embedded Systems (S&D4RCES), Vienna, Austria, ACM, 2010.

43. Eby M, Werner J, Karsai G, Ledeczi A. Integrating security modeling into embedded system design, *IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS)*, Tucson, Arizona, USA, 2007; 221–228.
44. Saadatmand M, Leveque T. Modeling security aspects in distributed real-time component-based embedded systems, *International Conference on Information Technology: New Generations (ITNG)*, Las Vegas, Nevada, USA, 2012; 437–444.
45. Chandrasekaran B, Josephson JR, Benjamins VR. What are ontologies and why do we need them? In *Intelligent Systems*. IEEE: NJ, USA, 1999; 20–26.
46. Gasevic D, Djuric D, Devedzic V. *Model Driven Engineering and Ontology Development*, 2nd edn. Springer-Verlag: Berlin Heidelberg, 2009.
47. Walter T, Parreiras FS, Staab S. An ontology-based framework for domain-specific modeling. *Software & Systems Modeling* 2012: 408–422, DOI 10.1007/s10270-012-0249-9,.
48. Clements P, Northrop L. *Software Product Lines: Practices and Patterns*, The SEI Series in Software Engineering. Addison–Wesley Longman Publishing Co., Inc.: Boston, MA, USA., 2001.
49. Urbani J, Maassen J, Drost N, Seinstra F, Bal H. Scalable RDF data compression with MapReduce. *Concurrency and Computation: Practice and Experience* 2013: 24–39, DOI: 10.1002/cpe.2840,.