Integrating security mechanisms into embedded systems by domain-specific modelling

Maria Vasilevskaya, Linda Ariani Gunawan, Simin Nadjm-Tehrani and Peter Herrmann

Linköping University Post Print

N.B.: When citing this work, cite the original article.

Original Publication:

Maria Vasilevskaya, Linda Ariani Gunawan, Simin Nadjm-Tehrani and Peter Herrmann, Integrating security mechanisms into embedded systems by domain-specific modelling, 2014, Security and Communication Networks, (7), 12, 2815-2832.
http://dx.doi.org/10.1002/sec.819
Copyright: Wiley: 12 months
http://eu.wiley.com/WileyCDA/

Postprint available at: Linköping University Electronic Press
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-95637
RESEARCH ARTICLE

Integrating security mechanisms into embedded systems by domain-specific modelling

Maria Vasilevskaya\textsuperscript{1}\textsuperscript{*}, Linda Ariani Gunawan\textsuperscript{2}, Simin Nadjm-Tehrani\textsuperscript{1}, Peter Herrmann\textsuperscript{2}

\textsuperscript{1} Department of Computer and Information Science, Linköping University, Linköping, Sweden
\textsuperscript{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 get 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 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 a) assets automatically elicited from the functional models, b) domain security knowledge captured by the security expert, and c) 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. Copyright © 0000 John Wiley & Sons, Ltd.

KEYWORDS

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

\textsuperscript{*}Correspondence
Department of Computer and Information Science, Linköping University, Sweden.
E-mail: maria.vasilevskaya@liu.se

Received...

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 in order 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].

While 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 due to the fact that 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.
Since 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 domain-specific modelling 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 while 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 modelling 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 Vasilevkaya 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 above 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, i.e., 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, while 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 security knowledge, while 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 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]. TSN 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 done by Trusted Sensor Modules (TSMs) consisting of a computing platform and physical sensors. The acquired measurement data is 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. Since 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 since, 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 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, e.g., 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 while 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 since 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` (see 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 is received when the buffer is empty, it is emitted immediately; otherwise, it is buffered. The pin `next` is used to get 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 is sent to the other entity as illustrated by the edge crossing a partition border. In the receiver partition, the data is 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, while a negative acknowledgment is issued in case the received measurements have not passed the validation test executed by the `db` block. Thus, the token emitted via the 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 is 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 (like security) as a set of concepts and the relations among those concepts.

A number of ontologies for information security have been proposed, e.g., Herzog et al. [8] and Fenz and Ekelhart [19]. We adopt the ontology presented by Herzog et al., since it builds upon classic components concepts of 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, i.e., 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 be a threat of several vulnerabilities; a threat threatens assets with respect to some security goals; 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, e.g., confidentiality, integrity, authorisation, and anonymity.

2.4. MARTE profile

MARTE (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 (GRM) package contains general concepts required for modelling of an execution platform, e.g., computing, storage, and communication resources. These general terms are further refined in two other packages, namely Hardware Resource Modeling (HRM) and Software Resource Modeling (SRM). Besides, MARTE contains packages for analysis of time and performance as well as allocation of application functions onto resources. Each concept, e.g., 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, i.e., 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 in order 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 defense strategy. Assets are the “objects of value”, in a system and which need to be protected. In our context, they can be data stationary residing on a physical component or data in transit being transmitted between different components. Other types of assets, e.g., algorithms or Intellectual Properties (IPs), may also be considered and introduced. The protection of an asset leads to the fulfillment of a particular security goal.
after an attack. For the security goal like protecting its confidentiality, integrity or availability. The countermeasures introduced below follow a certain defence strategy, e.g., 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 while 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, e.g., 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, e.g., 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], since 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, e.g., 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 in order to fulfil the confidentiality and integrity goals. Functional and platform models are related to the concrete SBB 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 to use our ontology and to support security experts in capturing domain-specific security knowledge, we represent the ontology as a UML model, i.e., as a class diagram (see Figure 6). Since 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 above 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, i.e., to create DSSMs, while 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 DataIn-Transit. The preserved relations are implements (between ConcreteSBB and AbstractSBB), protects (between Asset and AbstractSBB), and creates (between ConcreteSBB and Asset). The is-a relation from DataStationary and DataIn-Transit 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 \texttt{mFunctional}, \texttt{mPlatform}, and \texttt{stdCompliance} properties of the \texttt{ConcreteSBB} class respectively. Finally, the \texttt{requiredGoal} property in the association class \texttt{creates} represents the relation \texttt{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 due to the fact that the triple asset, security goal, and defence strategy already captures 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 \texttt{externalDSSM} in the \texttt{creates} association class. The \texttt{externalDSSM} property refers to other DSSMs. The use of 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: \texttt{StoredMeasurement} that represents energy measurements stored on a device; \texttt{CollectorToServerMsr} that represents energy measurements sent from a collector device to an operator server; \texttt{SensorToMeterMsr} that represents energy measurements sent from a sensor to a metering device. These assets may be protected by five abstract SBBS: \texttt{Secure storage} and \texttt{Tamper evident seal} that provide confidentiality and integrity for the \texttt{StoredMeasurement} asset; \texttt{Cipher} that provides confidentiality for the \texttt{StoredMeasurement} and \texttt{CollectorToServerMsr} assets; \texttt{Digital signature} that provides integrity and authentication for the \texttt{CollectorToServerMsr} asset; \texttt{Anomaly detection} that provides integrity for the \texttt{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 \texttt{DES} and \texttt{AES} concrete SBBS implement the \texttt{Cipher} abstract SBB. These concrete SBBS have a pair of functional models, which are Arctis blocks for encryption (\texttt{AES_Encryption} and \texttt{DES_Encryption} respectively) and decryption (\texttt{AES_Decryption} and \texttt{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, e.g., 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 “acknowledgment”, while 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: \texttt{Creation of functional models for concrete SBBS} and \texttt{Creation of a DSSM}. The order of these activities is undefined, since 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 \texttt{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 \texttt{mFunctional} slot of the \texttt{ConcreteSBB} class depicted in Figure 6).

Thus, the outcome of this activity is definition of the \texttt{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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{The UML presentation of our ontology, adapted from [9] adding the platform and standard compliance elements.}
\end{figure}
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, i.e., 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 one 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

![Figure 7. A fragment of the metering DSSM](image-url)
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 assert 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 ontology with concrete SBBs that actually refer to the same real world element. 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, e.g., 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] while 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 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.

![Figure 8. Creation of DSSMs and SBBs](image)

![Figure 9. Registration of concrete SBBs (the user interface)](image)

### 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 (see 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 (see Section 4.1). Note that if a required DSSM is not found, this step should be preceded by creation of a DSSM (see 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 (see Section 4.2), which results 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, e.g., 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 regards 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 above process is compliant with the industrial development needs that rest on traceability of requirements within the developed system. Specifically, this is done 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: (1) a DSSM that is relevant for a system under development is identified and selected; (2) 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 exemplify the Association with DSSMs step with our use case presented in Section 2.

![Figure 11. Association of the selected DSSM with the system elements (the user interface)](image)

Since 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 \texttt{c: Collector} in the Trusted Sensor Module Collector (TSMC) to \texttt{db: Db Handler} in the operator server (see 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.

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 domain-specific language). 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 (see Figure 13) and their application logic (see 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, i.e., kindy : V → KV and kindE : E → KE, where KV = \{operation, merge, join, fork, decision, local, collaboration, other\} and KE = \{object, control\}.
- The set ON of all object nodes of a given activity, i.e., the data stored in the system and transported within the data flow tokens.
- Two functions returning an object flowing to (resp. from) a given node through an edge, i.e., inO : E × V → ON and outO : V × E → ON.
- A function mapping a given a asset to a class from our ontology, i.e., class : A → KA, where KA = \{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, i.e., part : V → 2P, 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, i.e., source : E → V and target : E → V respectively.
- A function inner : A → 2A that assigns to an activity the set of its inner activities, i.e., the activities referring to the local and collaborative blocks that it contains. For instance, the transfer handler block contains the reactive buffer as its only inner block (see Figure 4).
- A tuple A ⊑ ON × V × E that uniquely identifies an asset.

\[ e \in E, \text{kind}_E(e) = \text{object}, \]
\[ \{\text{kind}_V(\text{source}(e))\} \cup \{\text{kind}_V(\text{target}(e))\} \]
\[ \cap \{\text{operation, local, collaboration}\} \neq \emptyset, \]
\[ \text{part(source(e))} \cap \text{part(target(e))} \neq \emptyset. \]

**R1:**
\[
\begin{align*}
\text{Asset}_1, \text{Asset}_2 & \in A : \\
\text{asset}_1 & = \{\text{out}_O(\text{source}(e), e), \text{source}(e), e\}, \\
\text{class(Asset}_1) & = \text{stationary}, \\
\text{asset}_2 & = \{\text{in}_O(e, \text{target}(e)), \text{target}(e), e\}, \\
\text{class(Asset}_2) & = \text{stationary}.
\end{align*}
\]

**R2:**
\[
\begin{align*}
e & \in E, \text{kind}_E(e) = \text{object}, \\
\text{kind}_V(\text{source}(e)) & \in \{\text{operation, merge, decision, fork, join, local, collaboration}\}. \\
\text{part(source(e))} & \cap \text{part(target(e))} = \emptyset
\end{align*}
\]

**Figure 13.** Rules for asset identification

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...
function traverseBlocks (Activity A) /
∀e ∈ E :
   R1, R2
∀a ∈ inner(A) :
     traverseBlocks(a)

Figure 14. A function to traverse a functional system model

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.

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

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 CollectorToServerMsr (CtS) the following set of security properties is retrieved:

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

The next three steps (step 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 (see Section 2.4). The modelling is done 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 SPI (Serial Peripheral Interface) and the SPI bus. Finally, 10/100Mbps NIC is used to connect a TSMC to a communication channel (LAN). Below, 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, e.g., 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.

4. Associate each data in transit asset with some resource (i.e., components annotated with the
Table II. Association of the assets with the platform components

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

Table III. Unacceptable threats and violated security goals

<table>
<thead>
<tr>
<th>Platform component</th>
<th>Threat</th>
<th>Violated security goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAND Flash</td>
<td>injection</td>
<td>integrity</td>
</tr>
<tr>
<td>LAN</td>
<td>eavesdropping</td>
<td>confidentiality</td>
</tr>
</tbody>
</table>

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 to the security goal violated by the threat (see Table III), which targets a platform component associated with an asset of the considered security property (see 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:

- **SP₁**: [StoredMeasurement, Integrity, Prevention]
- **SP₂**: [CollectorToServerMsr, Confidentiality, Prevention]

**SP₁** is formulated due to the knowledge about the existence of an injection threat that potentially violates integrity of the NAND Flash component (see Table III) used in association of the StoredMeasurement asset (see Table II). Similarly, **SP₂** is identified due to the presence of an eavesdropping threat that violates confidentiality of the LAN component (see Table III), which is used in association of the CollectorToServerMsr asset (see Table II).

### 4.3. Search for concrete SBBs

The set of identified security properties to be satisfied, which are **SP₁** and **SP₂** 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 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 (from Figure 6). Values in the square brackets denote parameters of the query.

Execution of the above query for **SP₁** and **SP₂** 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, etc. 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’s assume that a system engineer decides to use the AES concrete SBB to satisfy **SP₂**. 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 fulfill 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 done 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, since 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 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 done by arranging their instances before and after the block t: Transfer Handler as shown in Figure 16.

---

¹ The concrete SBBs that start with the suffix “SecFutur” are currently under development within the SecFutur [4] European project.
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 in order 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. While 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 domain-specific modelling 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 creation of a Domain-Specific Language (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, while Domain-Specific Modelling (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 practise 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 domain-specific modelling 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: (1) general security knowledge through ontologies, (2) efficient model-based development through reuse of domain-specific models, (3) 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, Herrmit, and Arctis. Our approach has been illustrated using fragments of a model for a non-trivial case study, i.e., 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


30. OMAP3530; [www.ti.com](http://www.ti.com), last visited February 2013.


34. Ontology Language Manchester Syntax; [www.w3.org/TR/owl2-manchester-syntax/](http://www.w3.org/TR/owl2-manchester-syntax/), last visited April 2013.


