Towards a Framework for the Extension and Visualisation of Cyber Security Requirements in Modelling Languages

Curtis L. Maines[1], Bo Zhou[2], Stephen Tang[3] and Qi Shi[4]
Department of Computer Science
Liverpool John Moores University
Liverpool, L3 3AF, United Kingdom
Email: c.l.maines@2011.ljmu.ac.uk[1], b.zhou@ljmu.ac.uk[2], s.o.tang@ljmu.ac.uk[3], q.shi@ljmu.ac.uk[4]

***Abstract*—Every so often papers are published presenting a new extension for modelling cyber security requirements in Business Process Model and Notation (BPMN). The frequent production of new extensions by experts belies the need for a richer and more usable representation of security requirements in BPMN processes.**

In this paper, we present our work considering an analysis of existing extensions and identify the notational issues present within each of them. We discuss how there is yet no single extension which represents a comprehensive range of cyber security concepts. Consequently, there is no adequate solution for accurately specifying cyber security requirements within BPMN.

In order to address this, we propose a new framework that can be used to extend, visualise and verify cyber security requirements in not only BPMN, but any other existing modelling language. The framework comprises of the three core roles necessary for the successful development of a security extension. With each of these being further subdivided into the respective components each role must complete.

***Keywords*—security framework; modelling language; BPMN; business process**

I. Introduction

Business Process Model and Notation (BPMN) can be used to graphically represent business processes and their component relationships in a common standard between organisations [2], [12].

BPMN fulfils the requirement of visually representing business processes and is now the industry standard for their modelling [3], [17]. Nevertheless, even though security directly affects the functionality of business processes, BPMN has no support for specifying cyber security requirements [15], [16]. Current BPMN security extensions have made attempts, but they are being constructed unsystematically, without any empirical evidence to support their choice of concepts [8] or notational design.

The objective of this paper is to propose a new framework for visualising and verifying comprehensive cyber security requirements within not only BPMN, but any modelling language. Unlike current approaches, our framework is built upon existing literature and empirical evidence. From our evaluation of existing extensions we are able to identify the problem areas and propose a new set of requirements for avoiding them. Along with these and existing theories, we are able to propose a framework that can be used for extending any modelling language with security requirements.

In Section II, we analyse some existing BPMN security extensions identifying the notational issues within each one. From these reviews we are able to create a requirements specification in Section III to act as a foundation for our proposed framework. In Section IV, we explain our framework detailing why each component is compulsory for a successful solution. Section V finishes with concluding and future works.

II. BPMN Security Extensions

For existing extensions, the Physics of Notations (PoN) principles defined by Moody [11] are used for their evaluation. This is to ensure a scientific approach is kept in their assessment. These principles are frequently used in the evaluation of modelling languages including that of BPMN [5], [14]. These principles are as follows: semiotic clarity, perceptual discriminability, semantic transparency, complexity management, cognitive integration, visual expressiveness, dual coding, graphic economy, and cognitive fit [11]. Due to lack of space, we are unable to include definitions in this paper. For any uncertainties please refer to the respective paper.

We have already evaluated most existing extensions against the PoN in previous papers [9], [10] and therefore will be brief in their assessment for this one. Of the nine principles, we do not consider cognitive integration in our assessment. We feel this is a principle that can only be fairly evaluated against a standalone language. It is difficult for an extension to meet this requirement if the parent language does not.

Although low in number of concepts, Rodriguez et al. [15] extension covers a broad range of the cyber security domain. The symbols within their notation from left-to-right in Fig. 1 represent: non-repudiation, attack harm detection, integrity, privacy and access control.

Overall, this extension managed to satisfy just one graphic economy of the eight principles (disregarding cognitive integration). With the passed principle being solely down to the fact that the notation is symbol deficit. The extension feels negligent of modelling language design and as such

Fig. 1. Rodriguez *et al.* security notation

fails to act as a suitable solution for specifying cyber security requirements in BPMN.

Saleem et al. [16], slightly newer, security extension takes a different approach notation-wise, but still yields several issues. In Fig. 2 from left-to-right, the symbols represent confidentiality, integrity and availability respectfully; the core concepts of cyber security [13].

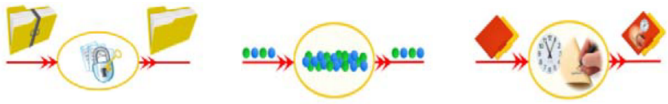

Fig. 2. Saleem *et al.* security notation

Of the eight principles, this extension satisfies four (perceptual discriminability, semantic transparency, graphic economy and cognitive fit) of them. Although failing in comprehensiveness and complexity management. If we focus on the design of the symbols themselves, this notation is one of the closest to meeting Moodys requirements [11].

Salnitri et al. SecBPMN2 [17] notation can be seen in Fig. 3. From left-to-right the concepts are as follows: integrity, authenticity, accountability, non-repudiation, auditability, confidentiality, privacy, binding of duties, separation of duties, availability and non-delegation.

Fig. 3. SecBPMN2 security notation

This extension is also able to satisfy four of the eight principles (perceptual discriminability, semantic transparency, graphic economy and cognitive fit). Labda et al. security notation [7] can be seen in Fig. 4. The symbols from left-to-right represent: access control (allow), access control (prevent), access control (limited), separation of tasks, binding of tasks, user consent, necessity to know (high), necessity to know (medium) and necessity to know (low).

Fig. 4. Labda *et al.* security notation

Overall, this extension managed to satisfy two (perceptual discriminability and graphic economy) of the eight PoN principles. Koh and Zhous notation [6] (see Fig. 5) is similar to that of Salnitri et al. Both opting for a circular shape with some form of icon inside. The underlying semantics for this notation are security task, authentication, access control, authorisation, harm protection, encrypted message, non-repudiation and secure communication respectively left-to-right. The final three symbols represent confidentiality, integrity and availability. With the stars visualising the required level for each concept.

Fig. 5. Koh and Zhou security notation

Overall, Koh and Zhou are able to satisfy just two (dual coding and graphic economy) of the eight PoN principles. The perceptually discriminability between the symbols themselves and that of BPMN is rather poor. Along with the perverse icon design and the relationships used to link security and BPMN elements. This notation is far from an adequate security extension to BPMN.

From this review, of the five extensions evaluated we can see that none are able to satisfy an adequate number of Moodys principles. Moody discussed how there are trade-offs amongst the principles and satisfying one may have a negative effect on another. Nevertheless, certain principles such as complexity management should always be achieved, especially in software engineering [11]. There is need for an extension that is not only comprehensive to the domain but can also satisfy the most optimal number of principles.

III. Requirements Specification

The PoN can be used for the scientific evaluation of a modelling language as well as design requirements for the creation of one [11]. Nevertheless, having identified the issues encountered with current extensions, we are able to define our own set of requirements for avoiding them in the future.

A. Comprehensiveness

The reason for specifying this requirement is due to the construct deficit current extensions suffer from. Although some extensions appear comprehensive to the domain, this is only relative to the deficit many of the others suffer from. In fairness to several extensions, they do state their focus on a particular area of security and as such their paucity may be excused. Nevertheless, needing multiple extensions to specify requirements for the same domain is very poor usability and will likely lead to the extensions being dismissed altogether. From the standpoint of an ideal solution the best approach is to include a comprehensive range of constructs which gives the modeller the ability to restrict the domain coverage as they see fit.

B. Coherence

In 2003, Donner said in a short journal article far too much security terminology is vaguely defined [4]. Although this

statement is almost 15 years old, from the review of existing extensions, it is clear to see how relevant it still is today. There is still much discrepancy surrounding various concept meanings and uses.

As such, we believe there should also be a principle which advises the explicit definition of a concept within a language. This ensures that any future modellers or readers of a diagram will always have a common archive of definitions should they require reminding. Current approaches are placing too much trust in the coherent definition of concepts from external sources such as the web. This requirement will ensure everyone is working in accordance with the same concept understanding.

C. Structure

The inclusion of a new domain in any existing language, will almost always cause an unparalleled increase in complexity (especially cyber security). Therefore, to try and overcome this issue, we propose the principle of structure. This links closely to Moodys principle of hierarchy. In the PoN however, Moody discusses using hierarchy as a form of complexity management to collapse diagrams into more manageable high level elements [11]. In this instance, we are targeting the notation symbols themselves, and their own collapsing of parent and child concepts.

D. Graphical Framework

From the review of existing extensions we are able to see that only two are able to satisfy half of Moodys principles (excluding cognitive integration). Although the PoN is not entirely based on the graphical design of the notation. It is fairly safe to generalise that the more principles satisfied the better the design of the symbols.

Therefore, we propose that a developer should first define a graphical framework for the symbols within their notation. One which uses the principles as defined in the PoN as a core foundation. This not only ensures the satisfaction of the principles, but also allows any future users to add to the notation as they see fit without effecting any pre-existing design in place.

E. Complexity Management

Complexity management is potentially the single reason current security extensions have the issues they do. When viewing current extensions from a complexity management perspective, it is likely the authors struggled to develop a visual solution in which they could be more comprehensive to the domain. Therefore they opted for construct deficit opposed to poor complexity management. Of course, without directly questioning each author we cant indefinitely confirm this was the case. However, there is no disputing that all extensions either suffer from complexity management or are on the threshold of doing so. For this reason, we see this principle as the most important of the PoN and therefore duplicate it into our own requirement specification.

F. Modelling Tool Functionality

When developing a security extension, most authors will typically expand on an already existing tool such as Microsoft Visio. The majority of tools usually include functionality for the creation or inclusion of custom notation to assist with this process. If the authors simply wanted to extend a language with same domain notation, utilising current software is the logical choice. However, extending a language with a new domain is more extensive than previous authors have given credit. It is not simply a case of specifying a few concepts then creating symbols for them. There are a lot of other variables which must be considered (the objective of this section being to highlight these). Moreover, the tool which is used for creating and reading diagram instances requires just as much attention.

Although there may be an existing tool or framework which allows for the complexity managed specification of different domain elements across an existing language. This principle is defined to highlight the possibility that this may not be the case and a new tool featuring new functionality may be needed to satisfy the aforementioned requirements along with the PoN.

IV. Proposed Framework

This leads onto the core novelty of our work, to overcome the aforementioned issues and satisfy the specified requirements. A framework for the extension and visualisation of cyber security requirements in modelling languages, see Fig. 6.

As previously stated, little attention has been given to notation design or tool functionality. The majority of authors focus all their efforts on semantics. Although we agree that semantics play a vital role in the creation of any modelling language (or extension), it is not the only role. The following section discusses our proposed framework, detailing the importance of each component to ensuring a complete solution is achieved

A. Key Roles

Our framework consists of three core roles; language architect, application developer and end user. All of which, are necessary for the successful design, implementation and utilisation of a security extension. The roles are structured vertically in the framework and represent the strict chronological order of development from bottom-to-top. (Similar to that of a brick wall). For the most part, this rule is also true when reading the framework from left-to-right. Regardless of this however, each component is numbered to reiterate the order that should be followed when creating an extension.

B. Language Architect

The language architect represents the first and most important role in the development process. As implied, their role is very similar to that of a software architect. They are responsible for the scope of each domain to include within the solution, along with the symbol design and visualisation approach. The importance of this role compared to the others,

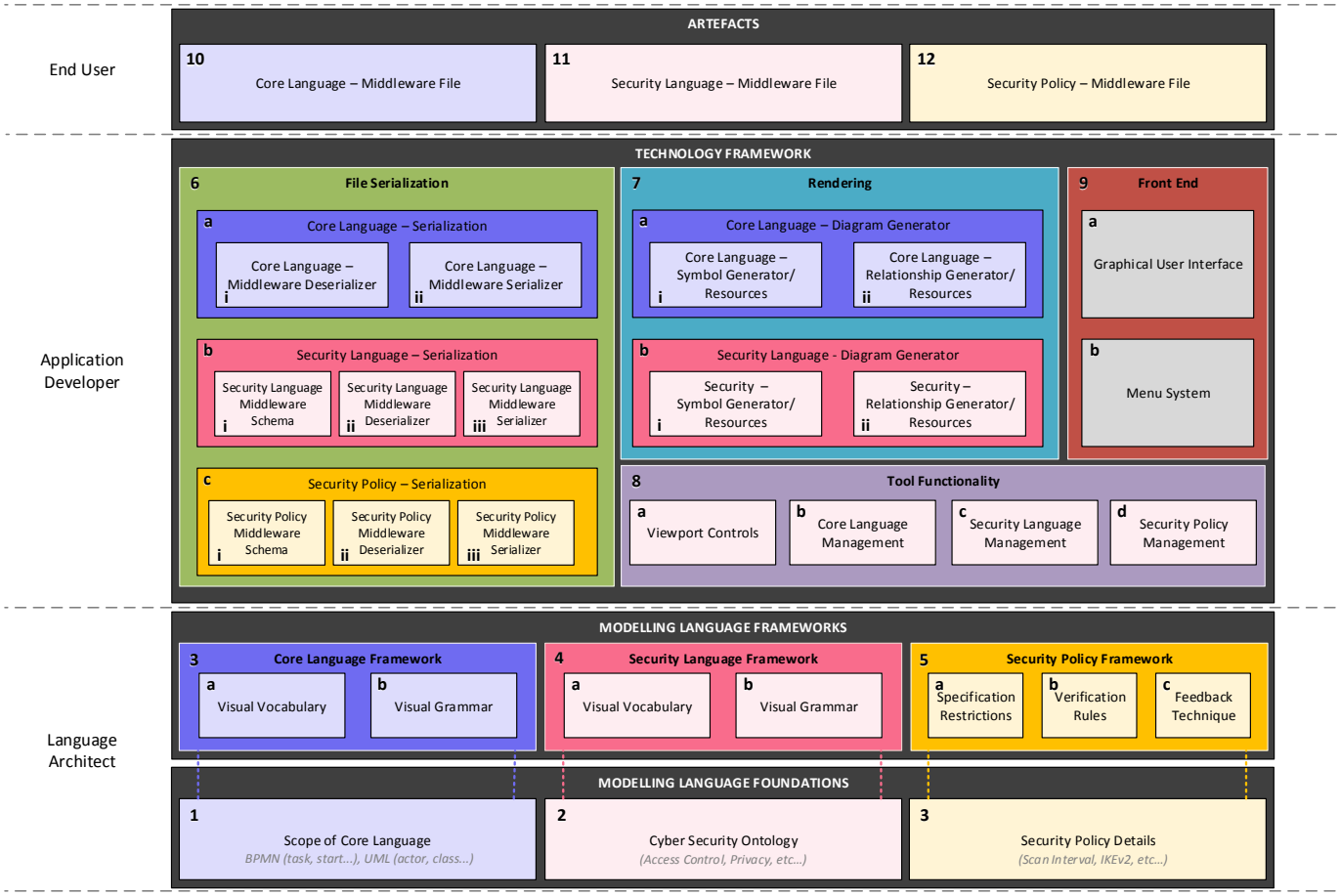

Fig. 6. Framework for the Extension and Visualisation of Cyber Security Requirements in Existing Modelling Languages

comes from the number of principles that can be satisfied at this stage of the framework.

1) Modelling Language Foundations: As the framework is based around the extension of an already existing language (BPMN in this case). We must take into consideration the restrictions of the parent language right from the beginning.

Scope of Core Language - At this stage of the framework, this is primarily to do with the scope of the core language. Naturally, incorporating the entire language is the most robust decision. However, when considering larger languages such as BPMN, which contain 171 constructs [5]. Feasibility decisions have to be made. Of course, this component of the framework represents the acknowledgement of the core language scope, rather than the strict requirement of reducing it. Depending on the intended use of the security extension will decide on the scope of the core language to include.

This component also requires the language architect to specify what elements of the core language will support security requirements. For example, in the case of BPMN, one may choose to only allow security specification on task elements and nothing else. Either way, a decision must be made to later inform the restrictions the application developer will implement into the tool.

Cyber Security Ontology - As stated by Moody [11], to satisfy the principle of semiotic clarity there must be a one-to one mapping between modellable constructs and ontological concepts. Given that we are creating a language in this instance and want to ensure adequate domain coverage. We will need an ontology of cyber security requirements to assess against the final notation to determine whether or not this principle was satisfied. Aforementioned, we have developed such an ontology in a previous publication [9], consisting of 79 security requirements and split into six areas (*access control, privacy, integrity, accountability, attack/harm detection and prevent* and *availability*). Adhering to this ontology will ensure the extension is comprehensive to the cyber security domain.

Security Policy Details - Although the framework is still usable without the inclusion of policy validation. Given the imposed regulations from governments and in-house standards, the ability to check a diagrams compliance with various policies at such an early stage in the Software Development Lifecycle (SDL) has to be utilised [1]. Specifying security at design can have several benefits. However, these can be negated if the specification later fails compliance checks due to inefficiencies. Therefore, at this stage of the framework we include a component to define the scope of the security

policies.

2) Modelling Language Frameworks: As previously stated, utilising the functionality of existing software does not always provide the best results. They are very restricted in what they can do, and typically only work for same domain extension. Hence why we specified a requirement to consider the functionality of the extension end tool. Taking this into account, we must acknowledge the strong likelihood of current software being unable to effectively extend existing languages with security requirements.

Core Language Framework - Therefore it is essential that the language architect defines the rules of the core language to later ensure the application developer implements them correctly into a new tool. Moody states that all visual notations are made up of three core components [11]. Visual vocabulary, which can also be described as a set of graphical symbols. Visual grammar, which defines the compositional and relationship rules of the languages, and finally visual semantics or symbol definitions. As visual semantics are covered in the previous section, this component focuses on just visual vocabulary and grammar. That being, at this stage of the framework the language architect must specify in explicit detail what symbol represents each concept within the core language and what form of visualisation is required to ensure the composition and relationship rules are adhered to.

Security Language Framework - Contrastingly, the security language will require much more attention at this stage as no language yet exists. It is not simply a case of extracting information from an existing source. The end goal of the framework has little reliance on what methods of symbolic design and visual representation are chosen here as all approaches will output solutions with identical functionality. However, the effectiveness of these solutions and their ability to satisfy the PoN, our own requirements and be an appealing, usable extension are very heavily dependent on the decisions made here. Visual vocabulary and grammar represent two of the key areas almost every extension has thus far failed to successfully implement. They are also where a large majority of our requirements specification can be achieved.

Security Policy Framework - The policy language framework isn't as visual as the core and security frameworks, dealing largely with back-end checking opposed to visual feedback. Therefore the requirements are slightly different. For this component, the language architect must decide on specification restrictions or validation rules as well as the method of verification against a diagram. These will then be able to define what feedback technique can be used to inform the end user of any discrepancies between the diagram and policy.

C. Application Developer

The previous section was based around the visual and conceptual construction of the languages. This portion of our framework focuses on the technical requirements (or technology framework) for creating the application.

1) File Serialization: One of the key technologies required for the development of a security extension (or software in general) is some form of middleware file for saving and loading data, and consequently, the functionality to serialize and deserialize these files from within the tool.

Core Language Serialization - As discussed earlier, whether a tool is created as an add-on or a standalone application the core language will require a rendering engine. Therefore, it will also require some form of data file to hold the diagram information. Given that the core language has already been created, there will already be existing schemas for these files which can be sourced from other software. The tool simply requires the functionality to serialize and deserialize these.

Security Language Serialization - Once again, the security language is not as straightforward as the core language as no solution yet exists. Therefore we not only require serialize and deserialization here but also the creation of a schema for the data file.

Security Policy Serialization - The security policy data file is similar to that of the security language, also requiring a schema.

2) Rendering: This component feeds directly from the visual vocabulary of the modelling language frameworks. The middleware files contain the necessary data to draw the languages but without some form of rendering tool this data is useless.

Core Language Diagram Generator - If the tool is a standalone application (or loads a middleware file from an external engine), it will require the ability to generate the symbols and relationships for the core language.

Security Language Diagram Generator - Likewise, the security language will also require the same functionality. Although this is something an end user takes for granted. If we are to generate a new tool, a rendering engine represents a significant component for doing so.

3) Tool Functionality: The tool functionality component represents the key stage of the development process. This is effectively where the engine for the application is implemented.

Viewport Controls - Before implementing any of this functionality however, our framework first advises to create the necessary tools for controlling the viewport. Depending on the size of the diagram, both as a developer debugging the application, or as an end user. Without the ability to traverse a diagram, it is difficult to confirm elements have been placed correctly.

Core Language Management - Similar to the renderer, the management components feed directly from the visual grammar elements of the previous section. Representing the core libraries that are required for drawing a diagram as per the language architects restrictions. The application developer must ensure there is functionality to either render and/or select core language elements. Without this, there will be no way of relating parent language elements with security requirements.

Security Language Management - Depending on the designs specified by the language architect will decide on the workload required for this component. Security language management

represents the implementation of the security language and thereby the core functionality of the extension itself. The rendering component as mentioned, is a vital constituent to this. However, this element deals directly with the implementation of the necessary functionality for specifying security, as well as the placement of these requirements onto a diagram. The key objective here being to ensure that an end user is strictly limited to the visual grammar approach specified by the language architect in the previous section.

Security Policy Management - The implementation of policy verification and the feedback it generates, although different to the previous two components, still feeds from the security policy framework of the previous section. At this stage, the application developer is required to implement the necessary functionality for carrying out the tasks specified by the language architect.

4) Front End: Finally, the last component of the technology framework is the front end of the application, otherwise referred to as the Graphical User Interface (GUI). The main requirement of these components, is to ensure that a usable interface is created which allows the end user to access and utilise the functionality from the previous section.

D. End User

Aforementioned, the end user represents the third and final role for the design, implementation and utilisation of a security extension. As expected, the end user has little responsibility in terms of the design and implementation of the extension. However, they do represent the end goal, and although they are not compulsory for the production of the end product. When creating diagrams and using the extension, they will be producing the saved data, middleware files discussed earlier. Those of which, are compulsory for the application to even load a diagram. Therefore, their role is quite significant.

V. Conclusions and Future Works

Current research surrounding the visualisation of cyber security requirements within BPMN has so far been very limited in terms of design rationale. Authors have typically focused heavily on their choice of semantics and put very little thought into the notation design. (Ironic considering not one extension even manages a comprehensive set of concepts.) Totally disregarding the complexity issues associated with adding a new extension into BPMN.

Throughout this paper we highlighted the key requirements an extension need meet to be considered both comprehensive to the domain and usable as a modelling language.

From these requirements we were able to create a framework not only for BPMN, but for any other modelling language requiring an extension for visualising security requirements. Although this framework is only theoretical at this stage it is based on the analysis and contribution of existing literature. Therefore, we are confident in its ability to satisfy the aforementioned requirements.

Nevertheless, our immediate future work involves the true testing of our framework by creating our own security extension which uses BPMN as a case study. This will then provide us with the necessary evidence to prove (or disprove) our frameworks ability to create a comprehensive, complexity managed security specification and verification extension for modelling languages.

References

[1] W. Arsac, L. Compagna, G. Pellegrino, and S. E. Ponta. Security Validation of Business Processes via Model Checking. *Lecture Notes in Computer Science*, 6542(213471):29–42, 2011.

[2] P. Bocciarelli and A. D'Ambrogio. A BPMN Extension for Modeling Non Functional Properties of Business Processes. In *TMS-DEVS '11 Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 160–168, Boston, 4th - 7th April 2011.

[3] M. Chinosi and A. Trombetta. BPMN: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, January 2012.

[4] M. M. Donner. Toward a security ontology. *IEEE Security & Privacy*, 1(3):6–7, 2003.

[5] N. Genon, P. Heymans, and D. Amyot. Analysing the Cognitive Effectiveness of the BPMN 2 . 0 Visual Notation. *Software Language Engineering, Springer LNCS*, pages 377–396, 2010.

[6] S. S. Koh and B. Zhou. BPMN security extensions for healthcare process. In *13th IEEE International Conference on Dependable, Autonomic and Secure Computing*, Liverpool, 26th - 28th October 2015.

[7] W. Labda and P. Sampaio. Modeling of Privacy-Aware Business Processes in BPMN to Protect Personal Data. In *29th ACM Symposium on Applied Computing*, pages 1399–1405, Gyeongju, 24th - 28th March 2014.

[8] M. Leitner, M. Miller, and S. Rinderle-Ma. An Analysis and Evaluation of Security Aspects in the Business Process Model and Notation. In *2013 International Conference on Availability, Reliability and Security*, pages 262–267, Regensburg, 2nd - 6th September 2013.

[9] C. L. Maines, D. Llewellyn-Jones, S. Tang, and B. Zhou. A cyber security ontology for BPMN-security extensions. In *13th IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 1756–1763, Liverpool, 26th - 28th October 2015.

[10] C. L. Maines, B. Zhou, S. Tang, and Q. Shi. Adding a Third Dimension to BPMN as a means of Representing Cyber Security Requirements. In *2016 International Conference on Developments of E-Systems Engineering (DeSE)*, Liverpool, 31st - 2nd September 2016.

[11] D. Moody. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, Nov.-Dec. 2009.

[12] OMG. *Business Process Model and Notation [Online]*, 2015, (accessed: 15-02-2016). Available: http://www.bpmn.org/.

[13] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Prentive Hall PTR, 4 edition, 2006.

[14] G. Popescu and A. Wegmann. Using the Physics of Notations Theory to Evaluate the Visual Notation of the Systemic Enterprise Architecture Methodology. In *16th IEEE Conference on Business Informatics*, pages 166–173, Geneva, 14th - 17th July 2014.

[15] A. Rodríguez, E. Fernández-Medina, and M. Piattini. A BPMN extension for the modeling of security requirements in business processes. *IEICE Transactions on Information and Systems*, 90(4):745–752, April 2007.

[16] M. Q. Saleem, J. B. Jaafar, and M. F. Hassan. A Domain-Specific Language for Modelling Security Objectives in a Business Process Models of SOA Applications. *International Journal on Advances in Information Sciences and Service Sciences, AICIT*, 4(1):353–362, January 2012.

[17] M. Salnitri, F. Dalpiaz, and P. Giorgini. Modeling and Verifying Security Policies in Business Processes. In *Enterprise, Business-Process and Information and Information Systems Modeling, Springer LCBIP*, volume 17, pages 200–214, 2014.