Towards Context-Aware and Privacy-Sensitive Systems

Boris Shishkov[1,3], Marijn Janssen[2] and Yi Yin[2]
[1]*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria*
[2]*TBM - Delft University of Technology, Jaffalaan 5, 2628 BX Delft, The Netherlands*
[3]*IICREST, 53 Iv. Susanin Str., 1618 Sofia, Bulgaria*
b.b.shishkov@iicrest.org, {M.F.W.H.A.Janssen, Y.Yin}@tudelft.nl

Keywords: Enterprise Modeling, Software Specification, Context-Awareness, Privacy, Land Border Security.

Abstract: Current global trends push *enterprises* to be increasingly efficient and flexible, and at the same time compliant with legislation regarding privacy, security, and transparency. The latest *IoT* (Internet-of-Things) developments offer opportunities for enterprises but at the same time those developments lead to an increased complexity with regard to the underlying software, this in turn leading to new risks. Hence, more advanced modeling methods and techniques may be necessary, especially in the area of *enterprise information systems* (often featured currently by enterprise-aligned *IoT*-enabled software systems), such that both the enterprise needs are captured (and understood) and software features are specified accordingly. We need a common modeling ground for this, allowing us to properly align *enterprise modeling* and *software specification*. Such a common ground can be co-created by enterprise engineers and software engineers, featuring: (a) technology-independent enterprise models rooted in social theories; (b) technology-specific software models rooted in computing paradigms. An approach is needed on top of that because such an integrated enterprise-software modeling requires to be greased by modeling guidelines and notations, such that adequate modeling generations and transformations are possible. This means that taking as input unstructured business information, we should be able to usefully apply a modeling and design process, such that we come through enterprise models and reach as far as the specification and implementation of software. We argue that an existing approach has those capabilities, namely the approach *SDBC* (Software Derived from Business Components). Hence, we adopt *SDBC* in the current research. Further, we have opted for an explicit consideration of *context-awareness* and *privacy*, claiming their relevance with regard to some current *IoT*-related demands (mentioned above). Nevertheless, it has not yet been extensively studied how *SDBC* can be used for modeling software systems with requirements on those properties. For this reason, we aim at enriching the *SDBC*-rooted enterprise-modeling-driven software specification, by weaving in *context-awareness* and *privacy enforcement*. We partially demonstrate our proposed way of modeling, by means of a case example featuring *land border security*.

1 INTRODUCTION

Current global trends push *enterprises* to be increasingly efficient and flexible, and at the same time compliant with legislation regarding privacy, security, and transparency. The latest *IoT* (*Internet-of-Things*) developments offer opportunities for enterprises but at the same time those developments lead to an increased complexity with regard to the underlying software, this in turn leading to new risks (IoTDI, 2017). Hence, more advanced modeling methods and techniques may be necessary, especially in the area of *EIS* - *Enterprise Information Systems* (often featured currently by enterprise-aligned *IoT*-enabled software systems), such that both the enterprise needs are captured (and understood) and software features are specified accordingly; it is important to bring together enterprise modeling and software specification since an *enterprise engineer* alone is insufficiently capable of grasping the technical complexity of an *EIS* (and its reach outside through software services), while a *software engineer* would have only superficial enterprise-specific domain knowledge (Shishkov, 2005). We need a common modeling ground for this, allowing us to properly align *enterprise modeling* and *software specification*. Such a common ground can be co-created by enterprise engineers and software engineers, featuring: (a) *technology-independent enterprise models* rooted in social theories; (b)

technology-specific software models rooted in computing paradigms. An approach is needed on top of that because such an integrated enterprise-software modeling requires to be greased by *modeling guidelines and notations*, such that adequate modeling *generations* and *transformations* are possible. This means that taking as input unstructured business information, we should be able to usefully apply a modeling and design process, such that we come through enterprise models and reach as far as the specification and implementation of software. We argue that an existing approach has those capabilities, namely the approach *SDBC* - Software Derived from Business Components (Shishkov, 2017). Hence, we adopt *SDBC* in the current research. SDBC is consistent with the *Model-Driven Architecture – MDA* (MDA, 2017) that features a life cycle starting with computation-independent modeling and ending up with code. Further, we have opted for an explicit consideration of *context-awareness* (Shishkov and Van Sinderen, 2008) and *privacy* (Hustinx, 2010), claiming their relevance with regard to some current *IoT*-related demands (mentioned at the beginning of this section). Nevertheless, it has not yet been extensively studied how *SDBC* can be used for modeling software systems with requirements on those properties. For this reason, we aim at enriching the *SDBC*-rooted enterprise-modeling-driven software specification, by *weaving in context-awareness and privacy enforcement*. We partially demonstrate our proposed way of modeling, by means of a *case example* featuring *land border security* (Shishkov and Mitrakos, 2016).

In this paper, for the sake of brevity, we are limiting our focus to the *CIM* generation (*Computation-Independent Models (CIM)* point to the highest level of abstraction in *MDA*), noting that:

- *SDBC* is capable of adequately reflecting a *CIM* input into lower-level software specifications;
- It is at this highest level of abstraction where context-awareness and privacy are to be weaved in, bringing together both an enterprise perspective and a software perspective.

The remaining of the current paper is organized as follows: In Section 2, we consider the *SDBC*-rooted enterprise-modeling-driven specification of software, by: (a) motivating the choice of *SDBC* over other approaches, inspired by relevant features and strengths of the *SDBC* approach; (b) coming through several important SDBC modeling constructs, limiting ourselves to those ones that are actually used in the case example; (c) addressing the SDBC design method. In Section 3, we address context-awareness and privacy, and also the challenge of weaving them in the software specification, and we address related work as well. In Section 4, we present a motivating application scenario in the public security domain (and particularly, in the area of land-border security), in which different situations are specified, such as the monitoring of the border, the detection of illegal crossings, and so on. In Section 5, we put forward our case-driven modeling where we further elaborate our ideas concerning the specification of context-aware and privacy-sensitive software systems. Finally, in Section 6, we present the conclusions.

2 BACKGROUND

In this section: we will firstly motivate the choice of SDBC; secondly, we will consider several important modeling constructs; thirdly, we will focus on the design process.

2.1 SDBC Modeling

SDBC is a software specification approach (consistent with *MDA*) that covers the early phases of the software development life cycle and is particularly focused on the derivation of *software specification models* on the basis of corresponding (re-usable) *enterprise models*. *SDBC* is based on three key ideas: (i) The software system under development is considered in its *enterprise* context, which not only means that the *software specification models* are to stem from corresponding *enterprise models* but means also that a deep understanding is needed on real-life (*enterprise*-level) processes, corresponding roles, behavior patterns, and so on. (ii) By bringing together two disciplines, namely *enterprise engineering* and *software engineering*, *SDBC* pushes for applying *social theories* in addressing *enterprise-engineering*-related tasks and for applying *computing paradigms* in addressing software-engineering-related tasks, and also for bringing the two together, by means of sound methodological guidelines. (iii) Acknowledging the essential value of re-use in current software development, *SDBC* pushes for the identification of re-usable (generic) *enterprise engineering building blocks* whose *models* could be reflected accordingly in corresponding *software specification models*. We refer to (Shishkov, 2017) for information on *SDBC* and we are reflecting the *SDBC* outline in Figure 1.

As the figure suggests, there are two *SDBC* modeling milestones, namely enterprise modeling (first milestone) and software specification (second milestone). The first milestone has as input a case

briefing (the initial (textual) information based on which the software development is to start) and the so called domain-imposed requirements (those are the domain regulations to which the software system-to-be should conform).

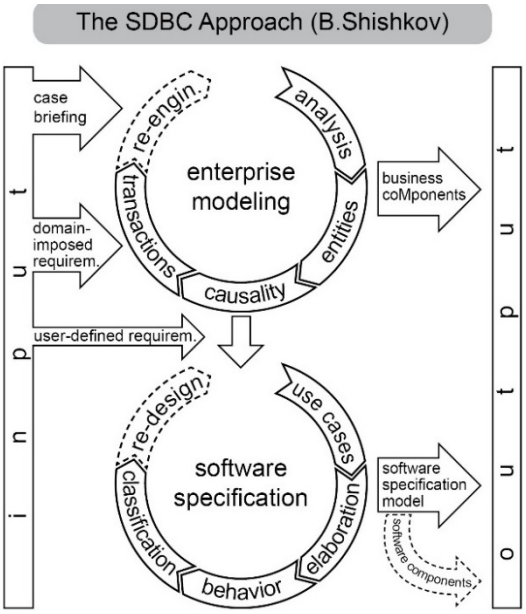

Figure 1: SDBC – outline.

Based on such an input, an analysis should follow, aiming at structuring the information, identifying missing information, and so on. This is to be followed by the identification (supported by corresponding social theories) of enterprise modeling entities and their inter-relations. Then, the causality concerning those inter-relations needs to be modeled, such that we know what is required in order for something else to happen (Shishkov et al., 2006). On that basis, the dynamics (the entities' behavior) is to be considered, featured by *transactions* (to be addressed in the following sub-section). This all leads to the creation of *enterprise models* that are elaborated in terms of *composition*, *structure*, and *dynamics* (all this pointing also to corresponding *data* aspects) – they could either feed further software specifications and/or be 'stored' for further use by enterprise engineers. Such enterprise models could possibly be reflected in corresponding *business coMponents* (see Sub-section 2.2). Next to that, re-visiting such models could possibly inspire enterprise re-engineering activities, as shown in Figure 1.

Furthermore, the second milestone uses as input the *enterprise model* (see above) and the so called user-defined requirements (those requirements reflect the demands of the (future) users of the software system-to-be towards its functioning).

That input feeds the derivation of a use case model featuring the software system-to-be. Such a software specification starting point is not only consistent with the *Rational Unified Process - RUP* (Kruchten, 2003) and the *Unified Modeling Language – UML* (UML, 2017) but is also considered to be broadly accepted beyond RUP-UML (Cockburn, 2000; Dietz, 2003; Shishkov, 2017). The *use cases* are then elaborated, inspired by studies of Cockburn (2000) and Shishkov (2005), such that software behavior models and classification can be derived accordingly. The output is a *software specification model* adequately elaborated in terms of *statics* and *dynamics*. Applying de-composition, such a model can be reflected in corresponding software components, as shown in the figure. Such an output could inspire software engineers to propose in a future moment software re-designs, possibly addressing new requirements.

As studied by Shishkov (2017), there are many other modeling approaches, some widespread and widely used. What justifies our considering particularly *SDBC* is the following:

- *SDBC* is neither addressing only enterprise modeling nor is it addressing only software specification; instead, the approach brings both together which is important if one needs to reflect sophisticated (legislative) requirements in complex software architectures.
- *SDBC* is not only limited to general guidelines and proposed modeling notations but it is also a method in the sense that different modeling activities are carried out in a specific order – this is to ensure that the software system being modeled is well-aligned with the business needs.
- *SDBC* is empowering re-usability and traceability which are considered essential with regard to software development in general.
- *SDBC* is aligned with the *UML* notations representing a *de facto* standard notation for specifying software (UML, 2017) and is consistent with *MDA*.
- In previous work, *SDBC* has been considered particularly in the border security application domain (Shishkov and Mitrakos, 2016).

For this reason, we have opted for adopting *SDBC* in the current work.

2.2 Concepts and Modeling Constructs

There are numerous concepts and modeling constructs underlying *SDBC*. For the sake of brevity

however, we will only address some of them in the current sub-section, especially those ones that are considered relevant to the challenge of weaving context-awareness and privacy-enforcement in land-border-security-related software specifications. For more related information on *SDBC*, interested readers are referred to (Shishkov, 2017).

Taking this into account, we firstly present the *system* definition inspired by Bunge (1979) and having fundamental importance with regard to both *SDBC* milestones:

DEFINITION 1 Let T be a nonempty set. Then the ordered triple $\sigma = <C, E, S>$ is **system** over T if and only if C (standing for *Composition*) and E (standing for *Environment*) are mutually disjoint subsets of T (i.e. $C \cap E = \varnothing$), and S (standing for *Structure*) is a nonempty set of active relations on the union of C and E. The system is *conceptual* if T is a set of conceptual items, and *concrete* (or material) if $T \subseteq \Theta$ is a set of concrete entities, i.e. things.

Inspired by the *system* definition, we focus particularly on **enterprise systems** since a (border-security) software system would inevitably operate in an enterprise surrounding (comprising (organizational) entities, business processes, regulations, and so on) and we consider an *enterprise system* as being composed of *human entities* collaborating among each other through *actions*, driven by the *goal* of delivering *products/services* to entities belonging to the environment of the system. As for an **EIS**, it is also composed of *human entities* (they are often backed by *ICT* (Information and Communication Technology) applications as well as by technical and technological facilities) but the *EIS* goal is to support informationally a corresponding *enterprise system*. This is functionally reflected in the collection, storage, processing, and exchange (or distribution) of data among users within or between enterprises, or among people within wider society (Shishkov, 2017).

Further, it is important to present the *SDBC* units of modeling and in this regard, it is to be noted that essentially, *SDBC* is focusing on the ENTITIES to be considered and their INTER-RELATIONS. It is desired to be able to model entities and relations abstractly (no matter if enterprise entities or software entities are concerned), and also to be able to specialize such models accordingly, in an enterprise direction or in a software direction. For this:

- We consider actors (combination of the *actor-role* and the *entity* fulfilling the *role*) since often one *entity* type can fulfil many *role* types and one *role* type can be fulfilled by many *entity* types (Shishkov, 2017).
- We consider a generic interaction pattern (featuring the *transaction* concept – see Definition 2) that is claimed to be helpful in modeling any real-life interaction in an enterprise/software context:

DEFINITION 2 A **transaction** is a finite *sequence of coordination acts* between two actors, concerning the same *production fact*. The actor who starts the transaction is called the *initiator*. The general objective of the initiator of a transaction is to have something done by the other actor, who therefore is called the *executor* (Dietz, 2006).

Hence, *enterprise modeling* and *software specification* are both being approached by those two essential concepts: ACTOR and TRANSACTION. Thence, a **business process** is viewed as a *structure of* (*connected*) *transactions* that are executed in order to fulfil a starting transaction and a **business component** is viewed as an *enterprise sub-system* that *comprises exactly one business process*. Further, a complete (by this we mean elaborated in terms of structure, dynamics, and data) **model** of a *business component* is called a **business coMponent**. That is why Figure 1 is featuring the identification of *business coMponents* as an essential enterprise modeling task within *SDBC*. Said otherwise, **THE FIRST SDBC MILESTONE** is about the identification of *business coMponents* featured in terms of *actors* and *transactions*.

Further, in bringing together the first milestone of *SDBC* and the second one, we need to be aware of possible *granularity* mismatches. The *enterprise modeling* is featuring *business processes* and corresponding *business coMponents* but this is not necessarily the level of granularity concerning the *software components* of the system-to-be. With this in mind, AN ICT APPLICATION is considered as matching the granularity level of a *business component* – an **ICT application** is an *implemented software product* realizing a particular *functionality* for the benefit of entities that are part of the composition of an *enterprise system* and/or a (corresponding) *EIS*. Thus, the label 'software specification model' as presented in Figure 1, corresponds to a particular ICT application being specified. Hence, **software components** are viewed as *implemented pieces of software*, which represent *parts of an ICT application*, and which *collaborate among each other* driven by the *goal* of *realizing the functionality of the application* (functionally, a **software component** is a *part*

of an ICT application, which is *self-contained*, *customizable*, and *composable*, possessing a *clearly defined function and interfaces* to the other parts of the application, and which can also *be deployed independently*). Hence, a ***software coMponent*** is a *conceptual specification model* of a *software component*. Said otherwise, **THE SECOND SDBC MILESTONE** is about the identification of *software coMponents* and corresponding *software components*.

In this paper, we will only address the *business-coMponent* identification and its reflection in a *use case model* featuring the specification of the *ICT application*-to-be, weaving in *context-awareness* and *privacy-enforcement* accordingly.

2.3 Design Method

SDBC assumes four modeling perspectives, namely: *Structural perspective* that reflects entities and their relationships; *Dynamic perspective* that reflects the overall business process and corresponding to this – the states of each entity, evolving accordingly; *Data perspective* that reflects the information flows across entities and within the business process; *Language-action perspective* (Dietz, 2006) that reflects real-life human communication and the expression of promises, commitments, etc. as also relevant to the challenge of soundly building an exhaustive enterprise model. In this, *SDBC* is grounded in line with: (a) *Enterprise engineering* and in particular, *enterprise ontology* and *organizational semiotics* (Dietz, 2006; Liu, 2000); (b) *Software engineering* and in particular, *model-driven engineering* and *component-based development* (Shishkov et al., 2007; Shishkov, 2017). Next to that, *software specification models* derived by applying *SDBC*, can be further updated to accommodate *service-orientation* (Shishkov et al., 2006).

Further, *SDBC* comes through several key modeling outputs (Shishkov, 2017):

1. Building a **`business entity model`** is the first major challenge requiring the fulfilment of many inter-related analysis/modeling tasks including: information structuring, gathering of additional information, reflection of the domain-imposed requirements, decision about the system boundary (determining the modeling focus), identification of actor-roles, capturing of their inter-relations, and so on, in concert with *Definition 1* and *Definition 2*. Hence, the resulting model shows the system boundary, the entities (actor-roles) that are inside the system and the relevant entities that remain outside the system boundary, the relations among those entities (featuring potential *transactions*), the related data aspects, and the *initiator-executor* positioning as according to *Definition 2* and Dietz (2006).
2. Deriving a corresponding **`causality model`** (Shishkov et al., 2006) abstracting from entities and only featuring the dependencies among corresponding transactions, such that it becomes clear how the realization of one transaction would possibly depend on the completion of another one(s).
3. Making an elaboration in terms of **`transactions and underlying communicative acts`** (Dietz, 2006; Shishkov, 2017), such that it becomes clear how the causalities (see above) are dynamically realized.
4. This all represents an adequate basis for **`deriving use cases`**, as studied by Shishkov and Dietz (2003).

For the sake of brevity, we limit ourselves to this partial outline of the *SDBC* design method and its underlying modeling / specification process – we only cover issues that are relevant to the current case-driven research and for the rest, interested readers are referred to (Shishkov, 2017).

In the following section, we will address *context-awareness* and *privacy*, and study the potentials for their incorporation in the *SDBC* driven specification of software.

3 CONTEXT-AWARENESS AND PRIVACY

In this section, we will firstly address *context awareness* and secondly, we will address *privacy*.

3.1 Context-awareness

Referring to *Definition 1*, **`context-awareness`** is about the system *environment*. The *system user* (using what the *system* is delivering) may comprise one or more entities belonging to the *environment* – each of them (or they all) could consume different *services* (or they could consume together one *service*). Further, not all entities belonging to the *environment* should necessarily be parts of the *user* since it might be that the *system* needs to collaborate with other entities from the *environment* (different from the *user*), such that the *system* is capable of delivering the requested products and/or services to the *user*.

Hence, a *user perspective* is needed in order to capture such a delivery of a product and/or a service

(we call this *service*, for short). Further, it is often that the *service* delivered to the *user* is to be adapted to the situation of the *user*. For example, a person wearing a body-area network (AWARENESS, 2008) through which body vital signs are captured, may appear to be in 'normal state' and then, for example, vital signs are captured and recorded as archival information, or the person may appear to be in 'emergency state' and then help would need to be urgently arranged. Thus, one kind of *service* would be needed at *normal state* and another kind of *service* would be needed at *emergency state*. For this reason, the *system* (or a corresponding system-internal EIS or ICT *application*) should be able to: (i) *identify the situation of the user*; (ii) *deliver a service to the user, which is suited for the particular situation*. This is illustrated in Figure 2:

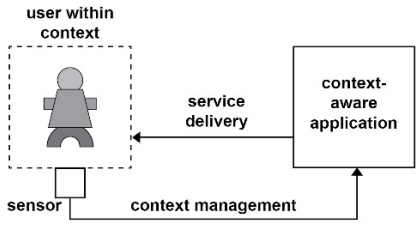

Figure 2: Schematic representation of a context-aware application (Shishkov and Van Sinderen, 2008).

As it is seen from the figure, a *service* is delivered to the *user* and the *user* is considered within his or her **context**, such that the *service* is adapted on the basis of the *context state* (or *situation*) the *user* finds himself/herself in. That state is to be somehow *sensed* and often technical devices, such as sensors, are used for this purpose. In the current paper, we do not go into discussing *sensor technology* in detail and for this reason, by **sensor** we broadly mean the technical or other facility that helps establishing the *user situation*. As mentioned before, it might be an *EIS* delivering the *service* to the *user* or it might be that just one ICT *application* (for example) as part of the *EIS* is delivering the *service* – no matter whether the former or the latter, we call it **context-aware application** in the current paper. Hence, a *context-aware application* adapts its behavior, in delivering *service*(*s*) to the *user*, based on the actual *context state* of the *user*, which *context state* is captured by *sensors* and corresponding information is sent to the *context-aware application* accordingly.

Nevertheless, the raw sensor data is of limited value unless it is reflected in higher-level context information that can be reasoned about.

It is also to be known how the application would 'move' from one behavior to another, when the user situation (*context state*) changes.

Summarizing the above, a *context-aware system* can be seen as concerning a sequence of *actions* that achieve: **S** (*sensing* and *capturing*), **I** (*interpretation* and *state derivation*), **w** (*switching*), and **P** (*provisioning*), as shown in Figure 3:

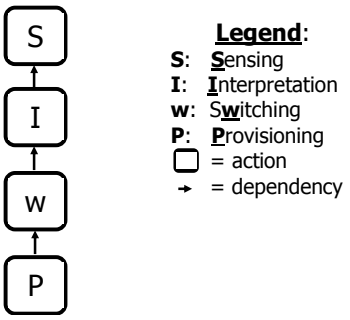

Figure 3: Simplified view on a context-aware system (Shishkov, 2017).

With regard to **S**: The system should be able to *sense context* and *capture this context as context information*. With regard to **I**: the system should be able to *interpret* the captured context information and *derive higher-level context information* that would be used to identify context state changes (those changes are to trigger in turn changes in the system behavior). With regard to **w**: the system should be able to *handle the switching between its alternative behaviors*. With regard to **P**: the system should be able to *provide services* covering different possible context states.

This is obviously a simplified model, since each of the actions represents a potentially complex process, and the dependencies between those normally involve multiple instances of information exchange and triggering.

Based on the above background, we propose the following way of weaving context-awareness in the SDBC-rooted enterprise-modeling-driven software specification: (i) particular user *context state types* are foreseen at *design time* and 'stored' in a reference bank; (ii) corresponding system *behavior types* are specified at *design time*; (iii) the *current user context state* is captured (see above) and matched to the bank of state types; (iv) if there is a match, then a *corresponding behavior type is instantiated* accordingly, otherwise, the system switches to '*auto-pilot*' in order to deliver a behavior in an exceptional situation. This is illustrated in Figure 4, using the notations of UML Activity Diagram (UML, 2017):

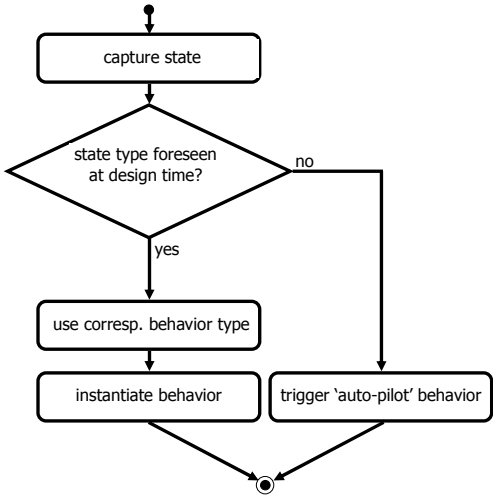

Figure 4: Weaving context-awareness in the software specification.

As it is seen from the figure, firstly the situation of the user (the user context state) is to be captured (as discussed already, this is usually done, facilitated by sensors). Secondly, it is important to establish whether or not the captured situation (state) is an instance of one of the situation (state) types foreseen at design time – if not, the system should switch to 'auto-pilot' behavior mode (by this we mean run-time rules-based behavior adaptation to the environment). If there is a match between the captured situation and a corresponding situation type foreseen at design time, then a corresponding behavior type (specified at design time) should be instantiated accordingly.

This way of weaving in *context-awareness* has been proposed inspired by the observation that: (i) there are high-occurrence-probability context state types suitable for consideration at design time; (ii) there are low-occurrence-probability context state types that are unpredictable and for this reason better addressable at run time.

In studying **RELATED WORK**, we have considered context-aware application practices. Due to the complexity and importance of handling *context-awareness*, many studies have tried to investigate different ways of developing context-aware applications. Many context modeling techniques have been created to enumerate and represent context information (Vieira et al., 2011). Many methodologies for architectural design were also proposed by researchers, such as: *Context Toolkit* (Dey, 2001) which aggregates context information, *Context Modeling Language* (Henricksen and Indulska, 2006) and *Model Driven Development* (MDD) and *UML- based approaches* (Ayed et al., 2007; Simons and Wirtz, 2007) which mainly describe the key steps and activities for modeling context-aware applications, the *Contextual Elements Modeling and Management through Incremental Knowledge Acquisition* (CEManTIKA) that supports the building of context-aware applications. Further, Jan vom Brocke, Sarah Zelt and Theresa Schimiedel have proposed a framework which consists of 4-dimensional factors to be considered in the design of context-aware applications, including 1) application goals, 2) characteristics of the process, 3) internal organizational specifications where context-aware applications are implemented, 4) the broader or external environment in which context-aware applications are built (Vom Brocke et al., 2016). Those factors can be used as guidelines when designing a context-aware application. In general, many current research projects are focusing on the development of context-aware applications, touching upon concept, networking aspects, middleware aspects, user-interface-related concerns, services, and so on. Still, this increasing attention has not been enough to inspire a widely accepted agreement on the development of context-aware applications. Hence, it is still a question how to weave context-awareness in the specification of software, and the current paper offers some contribution in this direction.

3.2 Privacy

As mentioned already, with regard to the (software) system-to-be, we are not only aiming at context-awareness but we are also willing to weave in values, such as privacy and transparency, for example. Particularly in this paper, we are focusing on **`privacy`** not only because it is one of the key values, as according to Hustinx (2010) but also because it is highly relevant with regard to the land-border security application domain addressed in the paper. Hence, in the remaining of the current sub-section, we will firstly discuss privacy in general (still assuming a border security focus), then we will present our view on how to weave in privacy enforcement in the specification of software, and finally we will particularly focus on privacy enforcement practices (related work) that are to be taken into account with regard to our case-driven modeling approach.

3.2.1 The Privacy Concept

Although the boundaries and specific contents of privacy vary significantly in different countries, the main definition of *information privacy* includes '**`the right to be left alone`**' and '**`control of`**

information about ourselves' (Pearson, 2009). Data can have various needs of privacy, whereas some information should always be opened to create transparency, other information should not be shared without proper authorization.

Although there is much information claimed to be *privacy-sensitive*, we consider the following information in border control as privacy sensitive information by using Pearson's privacy information classification (Pearson, 2009):

- *Personally identifiable information*: information that can be used to identify an individual:
  - = *Data from records:* name, date of birth, biometrics, address, social security number, and so on;
  - = *Surveillance data*: images, video, voice, and so on;
  - = *Secondary data*: bank account number, credit card number, phone number, social media network ID, and so on;
- *Demographical information*: sex, age group, race, health status, religion, education, and so on;
- *Usage data*:
  - = *Networking*-related data: mobile phone history data, Internet access point data, computer log files, and so on;
  - = *Recorded online activities*: messenger records, contribution to social websites, and so on;
  - = *Travel data*: ticketing / boarding pass data, reservations, cancellations, and so on;
- *Unique device identities*: any information that might be uniquely traceable to a device, e.g. IP address, device fabric number, Radio Frequency Identity (RFID) tags, and so on.

3.2.2 Weaving in Privacy Enforcement

Taking into account the privacy concept and the privacy-sensitivity issue, we propose the following way of weaving privacy enforcement in the SDBC-rooted enterprise-modeling-driven software specification: (i) when specified, a behavior instance is to be matched against a bank of pre-defined behavior types, such that it is clear what kind of behavior is that and what corresponding (pre-defined) privacy-related restrictions are to be weaved in; (ii) based on this, the behavior instance is to be refined accordingly. This is illustrated in Figure 5, using the notations of UML Activity Diagram:

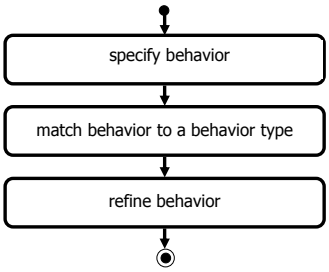

Figure 5: Weaving privacy enforcement in the software specification.

Hence, once specified, a behavior instance is to be refined in terms of privacy-driven restrictions.

3.2.3 Privacy Enforcement Practices – Privacy-by-Design and Related Work

Enforcement of privacy is often difficult. ICT enables the creation of systems that ensure the privacy of data, which is called **privacy-by-design** (Hustinx, 2010). *Privacy-by-design* has received attention within organizations as a way to always ensure that privacy is protected. *Privacy-by-design* suggests integrating *privacy* requirements into the design specifications of systems, business practices, and physical infrastructures. In the ideal situation, data is collected in such a way that privacy cannot be violated. This requires that both *governance aspects* (data updating processes and procedures, access rights, decision-making responsibilities, and so on) and *technical aspects* (encryption, access control, anonymization, and so on) are covered.

Since privacy enforcement solutions differ in different contexts, some general principles to guide the *privacy-by-design* can be used or sometimes must be compiled. For instance, the principles stated in Article 5 of the EU General Data Protection Regulation, need to be carefully considered, including: **lawfulness**, **fairness and transparency**, **purpose limitation**, **data minimization**, **accuracy**, **storage limitation**, **integrity and confidentiality**, and **accountability**. However, some principles would often be *in conflict with the characteristics of implemented information systems for border control*. For instance, the continuous collection of surveillance image data is against the principle of purpose limitation. Therefore, technical solutions should be a *trade-off* between privacy and (border-control-related) benefits (Könings et al., 2016).

Technical solutions regarding *privacy enforcement* would in general refer to **PET** – **Privacy-Enforcement Technologies**.

Those technologies assume secure communication and data storage by encryption, access control and auditing, anonymization of on-line activity, detection of privacy violators, and so on (Seničar et al., 2003; Zhu et al., 2015). Since *PET* can only partially address *privacy*-related problems, they need to be combined with **information governance features** in order to create comprehensive *privacy*-enforcement mechanisms.

Besides ***PET***s, ***PIT***s (**Privacy-Invasive Technologies**) and privacy threats are also frequently examined in various domains (Burghardt et al., 2008; Huberman et al., 1999; Johnston and Wilson, 2012; Seigneur and Jensen, 2004; Weber, 2015).

Nevertheless, there is still limited insight on how enterprises can *reduce privacy violation risks* for open data in particular, and there is no uniform approach for *privacy protection* (Janssen and Van den Hoven, 2015).

4 APPLICATION SCENARIO

Border control is one of Europe's biggest recent challenges, in the light of severe sea border problems in Greece and Italy in 2015-2016 (FRONTEX, 2017) and land border problems in Bulgaria and Croatia, for example. This leads not only to deadly incidents for numerous migrants who realize illegal sea/land border crossings in severe (weather) conditions but also to allowing terrorists mixed with regular migrants land on Europe's territory. According to many reports of the European Union - EU (www.europa.eu), this uncontrolled migration to Europe is causing societal tension and is stimulating extreme political views. Further, even though illegal migration to Europe is mainly fueled by smuggling channels, it is partially 'facilitated' by technical / organizational weaknesses at the EU external borders. In this paper, we abstract from the former and focus on the latter. Such a focus has been justified by numerous (current) efforts within the EU, aiming at improving security at its external borders – for example, new border facilities are constructed along those borders, police officers from some Western EU countries are sent to the Eastern EU borders to physically help, new organizational approaches and technical solutions are discussed, and so on (Ref.: www.europa.eu); all those efforts are directed towards *stopping* the illegal migration to the EU and it is widely agreed that any migrant should legally approach an EU border point where (s)he would be treated according to the laws and values of the EU.

In that sense, we consider an application scenario which concerns the EU **land border** control (our focus is particularly on the external EU borders) and this is about **monitoring** and **reaction to violations**. Fulfilling this assumes human actions because security-related decisions are always human-centric (LBS, 2012). Still, in what they are doing, border police officers receive useful technical support, assuming various channels: infrared images, visible images, proximity sensors, and so on, followed by some kind of intelligent data fusion algorithms (Shishkov and Mitrakos, 2016). We acknowledge this 'duality' – **human entities** and **technical entities**, and acknowledge as well the need to orchestrate this 'whole' in a sound way, allowing for objectivity and capability with regard to any situation that is possible to occur. Hence, we are approaching typical situations in this regard, and also the corresponding desirable reactions to those situations. Hence *context-awareness* is relevant with respect to land border security. Further, realizing that, the above-mentioned technology requires, among other things, IT-based services to recognize people (i.e. biometrics), we acknowledge the need for a special treatment of those issues as far as *privacy* is concerned because it is justified to distribute personal details of a terrorist but it is not justified to distribute personal details of anybody. We thus identify and approach some *privacy-sensitive* situations accordingly. In realizing all this, we take as example the situation at the Bulgarian-Turkish land border (Shishkov and Mitrakos, 2016); nevertheless, we abstract from many location-specific details in order to reach findings that are generic and widely applicable.

Monitoring the land border is a continuous process where: (i) *There is a (wired) border fence that is supposed to obstacle illegal migrants to get in*; still, this facility can be overcome using a ladder or by just destroying the wire. (ii) *There are border police officers who are patrolling (possibly using vehicles)*; still, no matter how many border police officers are sent to the border, it would be physically impossible to guarantee police presence at any time anywhere along the border, over hundreds of kilometres. (iii) *There are sensors and other (smart) devices*, as mentioned above; they are realizing surveillance; we assume the possibility that a device would perform local processing + artificial reasoning – based on this, it may generate contentful messages to be transmitted to corresponding human agents.

There are two main situation types at any point along the border, namely: (a) Normal Situation (NS); (b) Alarm Situation. We realize that both *context-*

awareness and *privacy enforcement* are 'under control' in (a) because:

- Within NS, all is just progressing according to pre-defined rules – hence, there is no need to adapt the system behavior with regard to surrounding context.
- Following pre-defined rules would also assume adequate treatment of privacy-sensitive data (for example: the border police officers are also monitored but it is not allowed to distribute their facial information).

What is more interesting thus is what is done in the case of (b) where *context-awareness* and related *privacy enforcement* are crucial.

Approaching (b) and taking into account the case information, we define three situation types concerning migrants possibly attempting to illegally cross the land border outside an official border crossing point: **1. Human-Triggered Alarm Situation (HTAS)**: *when a border police officer faces an attempt of one or more persons to illegally cross the border*. Then the officer can do ONE of three things, namely: 1.1. Try to physically stop the migrants from crossing, following the corresponding EU regulations; 1.2. Connect to colleagues and ask help; 1.3. Activate particular devices for taking pictures and video of the violators. It is important to note that in this situation, the person in charge has full decision-making capacity. **2. Device-Triggered Alarm Situation (DTAS)**: *when a device is 'alarmed' by anything and there is no border police officer on the spot*. Then, there are two possibilities: 2.1. The detecting device is 'passive' in a sense that the (video) information it is transmitting, is received in real-time and straightforwardly 'used' by a distant officer who intervenes, generating a decision and corresponding actions; 2.2. The detecting device is 'active' in a sense that based on information coming from at least several sensing units, the information is filtered and automated reasoning is performed, based on which a 'hypothesis' on what is happening is generated by the device and sent to corresponding human agents. **3. Outage Situation (OS)**: *when any unexpected (power, performance, or other) outage occurs*, not necessarily assuming illegal border crossing at the same time. This calls for urgent system recovery both in human and technical respect.

5 MODELING THE BORDER SECURITY SYSTEM

A logical starting point in our case-driven modeling is the 'translation' of the case briefing (see Section 4) into better structured information that would be featuring the original business reality and corresponding domain-imposed requirements. As it is well-known, this often assumes (partial) enterprise re-engineering such that the enterprise system being approached is adequately supportable by ICT / software applications (Dietz, 2006).

For the sake of brevity, we are not going in detail on how we analyze the case briefing and how we conduct such (partial) enterprise re-engineering. Moreover, this is not directly related to the main challenges addressed in the current paper, namely: the enterprise-IT alignment, with incorporation of context-awareness and privacy enforcement. Hence, we move directly to the textual reflection of the case briefing, holding in itself re-engineering-driven and requirements-driven updates:

Different situations may occur at the land border. Law requires that each situation type is addressed conforming to corresponding normative acts. This points to an exhaustive list of situation types that have to be pre-defined and stored in a corresponding 'bank' - we consider them as subclasses with regard to a Class 'Situation': Subclass 'NS', Subclass 'HTAS', Subclass 'DTAS', Subclass 'OS', and so on (see the previous section). Hence, we should have pre-defined accordingly legislation-driven behavior types - we consider them as subclasses with regard to a Class 'Behavior': Subclass 'Behavior 1', Subclass 'Behavior 2', and so on. Said otherwise, we should have behavior subclasses corresponding to respective situation subclasses. This means that any particular situation occurring at the land border is to be positioned as an instance of one of the situation subclasses, such that a system behavior is prescribed accordingly, by instantiating a corresponding behavior subclass. In order to achieve this, it is necessary that: FIRSTLY, the situation instance is captured; SECONDLY, the captured situation instance is positioned as relevant to a particular situation subclass; THIRDLY, a corresponding behavior subclass is identified and instantiated accordingly. This represents **CONTEXT-AWARENESS**: the system behavior depends on the situation at hand (in this, we abstract from the 'auto-pilot' option - see Figure 4). Further, it is necessary that privacy-driven restrictions are identified, corresponding to

the behavior subclass, leading to a refinement of the instantiated behavior. This represents **PRIVACY-ENFORCEMENT**: the system behavior is refined to accommodate relevant privacy requirements.

Hence, this refined case briefing appropriately reflecting the business needs, is our starting point. *SDBC* has particular strengths on further structuring such information: *actor-roles* are methodologically identified as well as corresponding *transactions*, and so on. Because of the limited scope of this paper, we do not go in further detail here; still, for more information on those issues, interested readers are referred to (Shishkov, 2017).

The entities (featuring *actor-roles*) are:

- **S** (*Sensor*); **S** is capturing the occurring situations (situation instances), for example: "all looks normal during night time", "two persons are hanging over the border fence", "one person is running next to the patrolling vehicle", and so on, to give just several examples; in this, **S** is supported by sensing devices, sensor networks, cameras, data fusion engines, and so on.
- **PE** (*Pattern Engine*); **PE** is linked to two pattern banks, namely: 'sp' and 'pp' – they hold the subclass specifications ('sp' featuring situations and 'pp' featuring privacy-driven restrictions). Hence, **PE** is capable of providing such information as reference.
- **MM** (*Match-Maker*); **MM** is matching an instance to a subclass, for example: matching a situation instance captured by **S** to a subclass from Bank 'sp'.
- **TE** (*Task Engine*); **TE** is generating a desired system behavior description (a task), by instantiating accordingly a behavior subclass (the bank that holds the subclass specifications featuring behaviors is 'bp') corresponding to a respective situation subclass.
- <*comment*> For the sake of enforcing privacy, it is necessary to match each prescribed desired system behavior to corresponding privacy-driven restrictions stored in Bank 'pp'; Thus, **MM** should do a match, based on a prescribed behavior instance (delivered by **TE**) and privacy patterns (delivered by **PE**). </*comment*>
- **PrE** (*Privacy Engine*); **PrE** delivers a refined behavior recommendation accordingly.
- **C** (*Customer*); **C** is hence fulfilled by the corresponding border police officer(s) and/or other team member(s) using such a task specification (as RECOMMENDA-TION) in order to establish their actions accordingly.

Thus, next to identifying **`entities`** (featuring actor roles (Dietz, 2006; Shishkov, 2017)), we are to also identify corresponding **`transactions`** (see *Definition 2*): this we present as the Border Security Business Entity Model, expressed using notations inspired by DEMO (Dietz, 2006) – see Figure 6.

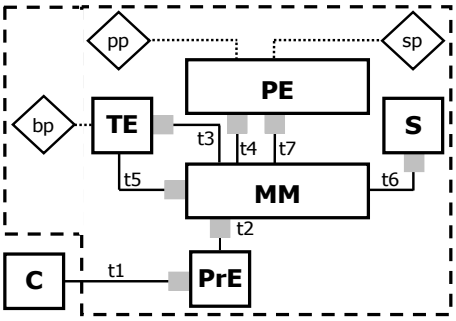

Figure 6: Business entity model for the border security case.

On the figure, the identified entities are presented in named boxes, while the small grey boxes, one at the end of each connection indicate the executor entity (Shishkov, 2017). The connections indicate the need for interactions between entities in order to achieve the business objective of recommendation generation – in our case, those interactions reflect **`transactions`**. Hence, with each connection, we associate a single transaction (**t**): **C**- **PrE** (**t1**), **PrE**-**MM** (**t2**), and so on. As for the delimitation, C is positioned in the environment of the recommendation generation system, and **PrE**, **MM**, **TE**, **PE**, and **S** together form the system, where we have included as well the three data banks mentioned above, namely: 'bp', 'pp', and 'sp'.

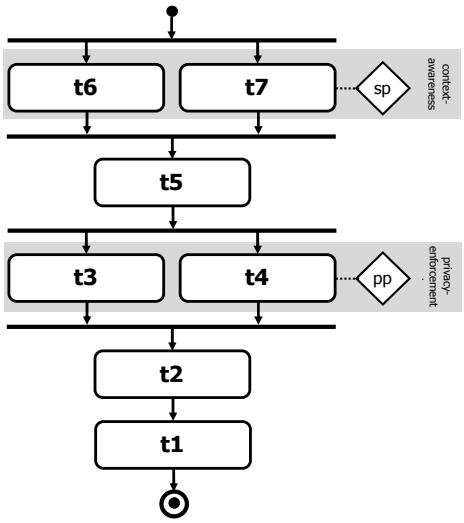

Figure 7: Modeling the causal relationships among transactions.

Further, we have to be explicit about the **causal relationships** among the transactions, and considering the business entity model, we establish that in order for **PrE** to deliver a refined task specification as a recommendation to **C**, it needs input from **MM** that in turn needs input from **TE** and **PE**. Further, in order for **TE** to deliver a desired system behavior description, it needs input from **MM** that in turn needs input from **S** and **PE**. Those causal relationships are presented in Figure 7, using the notations of *UML* Activity Diagram (UML, 2007).

As it is seen from the figure: (a) capturing a situation instance and considering corresponding situation patterns (viewed as subclasses) go in parallel firstly; (b) secondly goes a match between the two that establishes the relevant subclass (featuring situations) corresponding to a respective behavior pattern; (c) the behavior specification and consideration of relevant privacy-driven restrictions go in parallel thirdly; (d) fourthly goes a match between the two, that establishes the relevant privacy-driven restrictions with regard to the considered behavior; (e) finally, the refined behavior specification is delivered to **C** in the form of recommendation.

Hence, *context-awareness* and *privacy* are incorporated through corresponding modeling 'building blocks' featuring transactions 6+7 and 3+4, respectively, as suggested by Figure 7. Further, with regard to the *SDBC* modeling process, we have identified the *entity model* and the *causality relations*. What goes next are *transactions* (see Figure 1) and with regard to this, we use the *SDBC* interpretation of the transaction concept – see Figure 8:

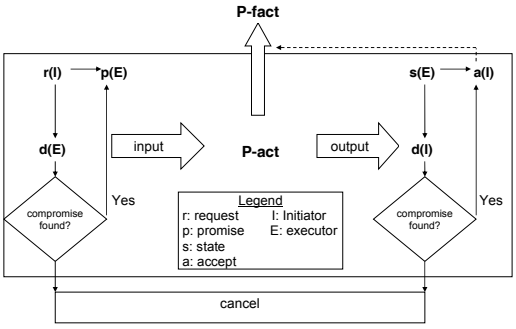

Figure 8: The *SDBC* interpretation of the transaction concept (Shishkov, 2017).

SDBC interprets the *transaction* concept as centered around a particular *production fact* (see *Definition 2*). The reason is that the actual output of any enterprise system represents a set of *production facts* related to each other. They actually bring about the useful value of the business operations to the outside world and the issues connected with their creation are to be properly modeled in terms of structure, dynamics, and data.

However, considering also the corresponding *communicative aspects* is important. Although they are indirectly related to the *production facts*, they are to be positioned around them. *SDBC* realizes this through its interpretation of the *transaction* concept. As seen from Figure 8, the *transaction* concept has been adopted, with a particular stress on the *transaction*'s output – the *production fact*. The order phase (left side of the figure) is looked upon as input for the *production act*, while the result phase (right side of the figure) is considered to be the *production act*'s output. The dashed line shows that a *transaction* could be successful (which means that a *production fact* has been successfully created) only if the *initiator* (the one who is initiating the *transaction*) has accepted the *production act* of the other party (called *executor*). As for the (coordination) *communicative act types*, grasped by an *SDBC transaction*, they are also depicted in the figure. The *initiator* expresses a request attitude towards a proposition (any *transaction* should concern a proposition – for example, a shoe to be repaired by a particular date and at a particular price, and so on). Such a *request* might trigger either promise or decline - the *executor* might either *promise* to produce the requested product (or service) or express a *decline* attitude towards the proposition. This expressed decline attitude actually triggers a discussion (negotiation), for example: 'I cannot repair the shoe today, is tomorrow fine?... and so on'. The discussion might lead to a compromise (this means that the *executor* is going to express a *promise* attitude towards an updated version of the proposition) or might lead to the *transaction*'s cancellation (this means that no *production fact* will be created). If the *executor* has expressed a *promise* attitude regarding a proposition, then (s)he must bring about the realization of the *production act*. Then the result phase follows, which starts with a statement expression by the *executor* about the requested proposition that in his/her opinion has been successfully realized. The *initiator* could either accept this (expressing an accept attitude) or reject it (expressing a decline attitude). Expressing a *decline* attitude leads to a discussion which might lead to a compromise (this means that finally the *initiator* is going to express an *accept* towards the realized *production act*, resulting from negotiations that have taken place and compromise reached) or might lead to the *transaction*'s cancellation (this means that no *production fact* will be created). Once the realized *production act* is accepted the corresponding

production fact is considered to have appeared in the (business) reality.

Hence, one could 'zoom in' with regard to any of the transactions depicted in Figure 7 and elaborate each transaction, using the transaction pattern presented in Figure 8. This actually means modeling transactions at **two different abstraction levels**. At the highest abstraction level, the *transaction* is represented as a single action which models the *production fact* that is enabled. At a lower abstraction level, the *transaction*'s *communicative aspects* are modeled conforming to the *transaction* pattern. The transaction's *request (r)*, *promise (p)*, *state (s)*, *accept (a)*, *decline*, and the *production act* are modeled as separate actions. This is illustrated in Figure 9 (abstracting from declines and cancellations), featuring only part of the model depicted in Figure 7, namely, focusing only on transactions 5, 6, and 7:

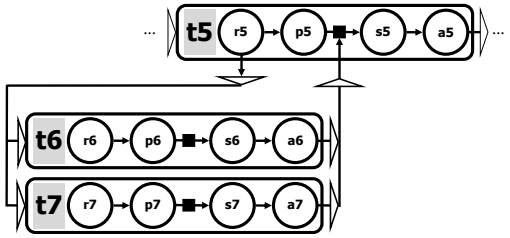

Figure 9: Detailed behavior aspect model featuring transactions.

As it is seen from the figure, in order for **t5** to be realized, both the realization of **t6** and the realization of **t7** are to be fulfilled. Hence, upon requesting **t5** and before the promise, it is necessary that **t6** and **t7** are initiated. If realized successfully, both *transactions*' output is necessary for the delivery of the *production act* of **t5** (the *production acts* are depicted as *black boxes* in the figure).

This is how *transactions* are elaborated.

In summary, such an *enterprise modeling* featuring *entities* (and data aspects) and corresponding *causal relationships* as well as the *behavior elaboration* of respective *transactions*, represents an adequate basis for specifying software on top of it.

We now move to the specification of software - the derivation of *use cases* is the first challenge – see Figure 1. For detailed information concerning the *derivation of use cases from transactions*, interested readers are referred to (Shishkov, 2017) – for the sake of brevity, we go directly to a partial *use case model*, derived on the basis of the 7 transactions (see Figure 6 and Figure 7). The model is depicted in Figure 10:

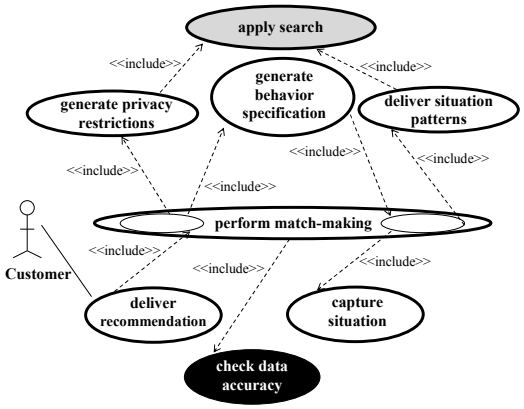

Figure 10: Partial use case model for the border security case.

As it is seen from the figure, all *use cases*, except for the ones backgrounded in black and grey, correspond to respective transactions: the SYSTEM's DELIVERY OF RECOMMENDATION (assuming behavior refinement) to CUSTOMER includes MATCHING between: (i) BEHAVIOR SPECIFICATION and (ii) PRIVACY RESTRICTIONS. In turn, (i) includes MATCHING between (iii) CAPTURED SITUATION and (iv) A SITUATION PATTERN (this matching allowing to identify the right behavior pattern to consider).

Those are the so called **essential use cases** – the ones straightforwardly reflecting transactions (Shishkov, 2017). Those *use cases* usefully drive the alignment between *enterprise modeling* and *software specification*, guaranteeing that the software system-to-be is stemming from enterprise models. Then all enterprise models would be helpful accordingly with regard to the further software specification and elaboration, based on the *use case model*.

Nevertheless, next to the *essential use cases*, we have also: (a) **informational use cases**, reflecting informational issues (not essential); (b) **use cases reflecting user-defined requirements** with regard to the software system-to-be (Shishkov, 2017). An example for (a) is the use case APPLY SEARCH - delivering situation patterns and generating privacy restrictions are essential business tasks requiring in turn *informational activity*, namely: searching through the corresponding data banks. An example for (b) is the use case CHECK DATA ACCURACY - it may be required by the user that upon match-making, the accuracy of corresponding data is checked. Those two *use cases* are only to illustrate (a) and (b). Because of the limited scope of this paper, we have only considered

a partial *use case model*, aiming at being explicit on the enterprise-software alignment that in turn builds upon the weaving of context-awareness and privacy at the enterprise modeling level.

For this reason, we are not going to address in the current paper the *elaboration of use cases* as well as the further software specification reflected in *behavior+states modeling* and *classification*. Interested readers are referred to (Shishkov, 2005) where this is considered and justified by means of a case study.

6 CONCLUSIONS

In this paper, we have considered in general the alignment between *enterprise modeling* and *software specification*, fueled by the *SDBC* Approach. In particular, we have addressed the challenge of weaving *context-awareness* and *privacy* in the enterprise models, such that context-awareness and privacy are then reflected accordingly in the specification of software. We have partially demonstrated our way of modeling by means of a case example featuring *land border security*. Hence, the contribution of the paper is two-fold: (i) We have contributed to the research concerning enterprise-software alignment, by studying how particular desired values (such as context-awareness and privacy) can be methodologically reflected in the specification of software. (ii) We have directed the current research to the border security application domain where context-awareness and privacy are of great importance, especially if they could be reflected in the functionalities of the (technical) systems facilitating the border control. As future research, we plan to consider a large-scale border security case study assuming software development activities.

REFERENCES

AWARENESS, 2008. *Freeband AWARENESS Project*. http://www.freeband.nl.

Ayed, D., Delanote, D., Berbers, Y., 2007. MDD Approach for the Development of Context-Aware Applications. In *International and Interdisciplinary Conference on Modeling and Using Context*.

Bunge, M.A., 1979. *Treatise on Basic Philosophy*, vol. 4, A World of Systems, D. Reidel Publishing Company. Dordrecht.

Burghardt, T., Buchmann, E., Böhm, K., 2008. Why do Privacy-Enhancement Mechanisms Fail, After All? A Survey of Both, the User and the Provider Perspective. In *Workshop W2Trust*, in conjunction with *IFIPTM*.

Cockburn, A., 2000. *Writing Effective Use Cases*, Addison-Wesley.

Dey, A. K., 2001. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1), 4-7.

Dietz, J.L.G., 2006. *Enterprise Ontology, Theory and Methodology*, Springer. Heidelberg, 1st edition.

Dietz, J.L.G., 2003. Generic Recurrent Patterns in Business Processes. In *BPM'03, International Conference on Business Process Management*, LNCS. Springer.

FRONTEX, 2017, the website on the European Agency, FRONTEX: http://frontex.europa.eu.

Henricksen, K., Indulska, J., 2006. Developing Context-Aware Pervasive Computing Applications: Models and Approach. *Pervasive and Mobile Computing*.

Huberman, B. A., Franklin, M., Hogg, T., 1999. Enhancing Privacy and Trust in Electronic Communities. In EC '99, 1st Int. ACM Conf. on Electronic Commerce. ACM.

Hustinx, P., 2010. Privacy by Design: Delivering the Promises. In *Identity in the Information Society 3(2)*. Springer.

IoTDI, 2017. *2nd International Conference on Internet-of-Things Design and Implementation*. ACM/IEEE.

Janssen, M., Van den Hoven, J., 2015. Big and Open Linked Data (BOLD) in Government: A Challenge to Transparency and Privacy? *Government Information Quarterly*, 32(4).

Johnston, A., Wilson, S., 2012. Privacy Compliance Risks for Facebook. *IEEE Technology and Society Magazine*, 31(2).

Könings, B., Schaub, F., Weber, M., 2016. Privacy and Trust in Ambient Intelligent Environments. In *Next Generation Intelligent Environments: Ambient Adaptive Systems*. Springer.

Kruchten, P., 2003. *The Rational Unified Process, An Introduction*, Addison-Wesley.

LBS, 2012. LandBorderSurveillance, the EBF, LandBorderSurveillance Project: http://ec.europa.eu.

Liu, K., 2000. *Semiotics in Information Systems Engineering*, Cambridge University Press. Cambridge.

MDA, 2017. *The OMG Model Driven Architecture*. http://www.omg.org/mda.

Pearson, S., 2009. Taking Account of Privacy when Designing Cloud Computing Services. In *ICSE'09, International Workshop on Software Engineering Challenges of Cloud Computing*.

Seničar, V., Jerman-Blažič, B., Klobučar, T., 2003. Privacy-Enhancing Technologies - Approaches and Development. *Computer Standards & Interfaces*, 25(2).

Shishkov, B., 2017. *Enterprise Information Systems, A Modeling Approach*, IICREST. Sofia, 1st edition.

Shishkov, B., 2005. *Software Specification Based on Re-usable Business Components* (PhD Thesis), TU Delft. Delft, 1st edition.

Shishkov, B., Mitrakos, D., 2016. Towards Context-Aware Border Security Control. In *BMSD'16, 6th International Symposium on Business Modeling and Software Design*. SCITEPRESS.

Shishkov, B., Van Sinderen, M.J., 2008. From User Context States to Context-Aware Applications. In *ICEIS'07 – Revised Selected Papers, LNBIP 12*. Springer.

Shishkov, B., Van Sinderen, M.J., Tekinderdogan, B., 2007. Model-Driven Specification of Software Services. In *ICEBE'07, IEEE International Conference on e-Business Engineering*. IEEE.

Shishkov, B., Van Sinderen, M.J., Quartel, D., 2006. SOA-Driven Business-Software Alignment. In *ICEBE'06, IEEE International Conference on e-Business Engineering*. IEEE.

Shishkov, B., Dietz, J.L.G., 2003. Deriving Use Cases from Business Processes, The Advantages of DEMO. In *ICEIS'03, 5th International Conference on Enterprise Information Systems*. SCITEPRESS.

Seigneur, J. M., Jensen, C. D., 2004. Trading Privacy for Trust. LNCS, Vol. 2995, incl. subseries Lecture Notes in Artificial Intelligence. Springer.

Simons, C., Wirtz, G., 2007. Modeling Context in Mobile Distributed Systems with the UML. *Visual Languages & Computing*, 18(4).

UML, 2017. *The Unified Modeling Language*. http://www.uml.org.

Vieira, V., Tedesco, P., Salgado, A. C., 2011. Designing Context-Sensitive Systems: An Integrated Approach. *Expert Systems with Applications*, 38(2).

Vom Brocke, J., Zelt, S., Schmiedel, T., 2016. On the Role of Context in Business Process Management. *Information Management*, 36(3).

Weber, R. H., 2015. The Digital Future – A Challenge for Privacy? *Computer Law & Security Review*, 31(2).

Zhu, N., Zhang, M., Feng, D., He, J., 2015. Access Control for Privacy Protection for Dynamic and Correlated Databases. In SmartCity'15, International IEEE SmartCity Conference. IEEE.