Enforcing Context-Awareness and Privacy-by-Design in the Specification of Information Systems

Boris Shishkov[1,3(✉)] and Marijn Janssen[2]

[1] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria
[2] Faculty of Technology, Policy, and Management, Delft University of Technology, Delft, The Netherlands
M.F.W.H.A.Janssen@tudelft.nl
[3] Institute IICREST, Sofia, Bulgaria
b.b.shishkov@iicrest.org

Abstract. Networked physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity, allow for run-time acquisition of user data. This in turn can enable information systems which capture the "current" user state and act accordingly. The use of this data would result in *context-aware applications* that get fueled by user data (and environmental data) to adapt their behavior. Yet the use of data is often restricted by *privacy regulations and norms*; for example, the location of a person cannot be shared without given consent. In this paper we propose a design approach that allows for *weaving context-awareness and privacy-by-design into the specification of information systems*. This is to be done since the very early stages of the software development, while the *enterprise needs* are captured (and understood) and the *software features* are specified on that basis. In addition to taking into account context-awareness and privacy-sensitivity these two aspects will be balanced, especially if they are conflicting. The presented approach extends the "*Software Derived from Business Components*" (*SDBC*) approach. We partially demonstrate our proposed way of modeling, by means of a *case example* featuring *land border security*. Our proposed way of modeling would allow developers to smoothly reflect context and privacy features in the application design, supported by *methodological guidelines that span over the enterprise modeling and software specification*. Those features are captured as *technology-independent societal demands* and are in the end reflected in *technology-specific* (*software*) *solutions*. *Traceability* between the two is possible as well as *re-use* of modeling constructs.

Keywords: Enterprise modeling · Software specification
Context-awareness · Privacy

1 Introduction

We observe increasing public demands for resilient *Enterprises Information Systems* (*EIS*) that are not only effective and efficient but also compliant with societal (legislation-related) demands, in the fields of privacy, security, transparency, and so on

B. Shishkov (Ed.): BMSD 2017, LNBIP 309, pp. 87–111, 2018.
https://doi.org/10.1007/978-3-319-78428-1_5

[28]. EIS in turn often count on run-time environmental information, such that they are capable of adapting their behavior accordingly. Helpful in this respect are the latest *IoT* (*Internet-of-Things*) developments: networked physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity, allow for run-time acquisition of user data [15]. Hence, empowered by sensor technology and connectivity, EIS can "know" what is going on around and use this information for optimizing their internal processes, for maximizing the user-perceived effectiveness, and for analyzing relevant data. For this, EIS need support from *context-aware applications* (they are EIS-internal), for the sake of delivering to the end user services that are adequate with regard to his or her situation at the moment [30]; context-aware applications get fueled by user data from emails, chat messages, sensors, and so on. Nevertheless, those innovative features lead to an increased complexity with regard to the underlying software; this in turn often assumes new risks [15], including *risks that concern privacy* [28]: for example, the European data protection act requires that the location of a person cannot be shared without given consent. Hence, more advanced modeling methods and techniques may be necessary, especially in the area of EIS, such that: (i) *enterprise needs are aligned with software specifications*; (ii) *context-awareness is achieved but also balanced with privacy* (it is to be noted that because of the limited scope of this paper, we only focus on context-awareness as an optimization strategy and we only focus on privacy as a relevant societal demand). Thus, we have opted for an explicit consideration of context-awareness [30] and privacy [14].

We propose a design approach for *weaving context-awareness and privacy-by-design into the specification of information systems*. This is to be done since the very early stages of the software development, while the *enterprise needs are captured* (and understood) and the *software features are specified* on that basis. In addition to considering context-awareness and privacy-sensitivity these two aspects will be *balanced*, especially if they are conflicting.

In order to avoid starting from scratch, we have looked for a relevant existing *software specification approach* to extend further for the sake of accommodating context-awareness and privacy-by-design. Nevertheless, we could only consider an approach that effectively brings together *enterprise modeling* and *software specification* because context issues and privacy issues are to be captured from the enterprise environment but need to be reflected in software solutions. Actually, *enterprise engineering* alone is insufficiently capable of grasping the technical complexity of an EIS (and its reach outside through software services [31]), while a purely *software engineering* perspective would assume only superficial enterprise-specific domain knowledge [27]. 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* [26]. Further, we would only consider an approach that is consistent with the *Model-Driven Architecture – MDA* [22] that can be considered as a *de facto* standard. Finally, we consider important that the approach of

choice has sound *underlying theories*. In this regard, we have opted for considering the *SDBC* approach [27] reflected in previous work ("SDBC" stands for: "*Software Derived from Business Components*"), noting that: (i) SDBC effectively brings together (in a *component-based* way); (ii) SDBC is consistent with MDA; (iii) SDBC considers several underlying social theories, such as *Enterprise Ontology* [7] and *Organizational Semiotics* [21] as well as computing paradigms, such as *Service-Oriented Computing* [32]; also SDBC brings this all together through its *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*. Since we consider SDBC as an approach with such capabilities, we adopt SDBC in the current research. Further, staying consistent with MDA (see above), we assume a development process starting with computation-independent modeling and ending up with code generation.

Nevertheless, 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.

That is how an SDBC-rooted enterprise-modeling-driven software specification is improved, 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*.

Our proposed way of modeling would allow developers to reflect context and privacy features in the application design, supported by *methodological guidelines* that span over the *enterprise modeling* and *software specification*. Those features are captured as *technology-independent societal demands* and are in the end reflected in *technology-specific* (*software*) *solutions*. *Traceability* between the two is possible as well as *re-use* of modeling constructs.

The remaining of the current paper is organized as follows: In Sect. 2, we present several *basic concepts*. In Sect. 3, we provide the *problem conceptualization*. Section 4 is featuring *background information on context-awareness and privacy*, considering also *related work* respectively. In Sect. 5 we briefly *outline the SDBC approach* (also justifying its choice) and in Sect. 6 we present a proposal on how to *weave in context-awareness and privacy* in the software specification. In Sect. 7, we present a *motivating application scenario* in the public security domain, based on which we *partially demonstrate* our proposed way of modeling. Finally, in Sect. 8, we present the *conclusions*.

2 Basic Concepts

In order to effectively address the *enterprise-software alignment* and consider on top of that *context-awareness* and *privacy*, we need a *common conceptual background*. Hence, we present several relevant *basic concepts* in the current section, noting that 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 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 [26].

Taking this into account, we firstly present the *system* definition inspired by Bunge [3] and having fundamental importance in the SDBC modeling:

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 [26].

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 can fulfil many roles and one role can be fulfilled by many entities [26];
- 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* [7].

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*. The identification of business coMponents (featured in terms of *actors* and *transactions*) is hence considered an essential enterprise modeling task within SDBC.

Further elaboration of other relevant concepts will be presented in Sect. 5, when introducing the SDBC approach.

3 Problem Conceptualization

As suggested by the Introduction, the problem we are facing in the current paper concerns modeling situations in which context-awareness and/or privacy need to be adequately reflected in the (software) system functionalities. This points to two problem "components" (*context-awareness* and *privacy*), as visualized in Fig. 1.

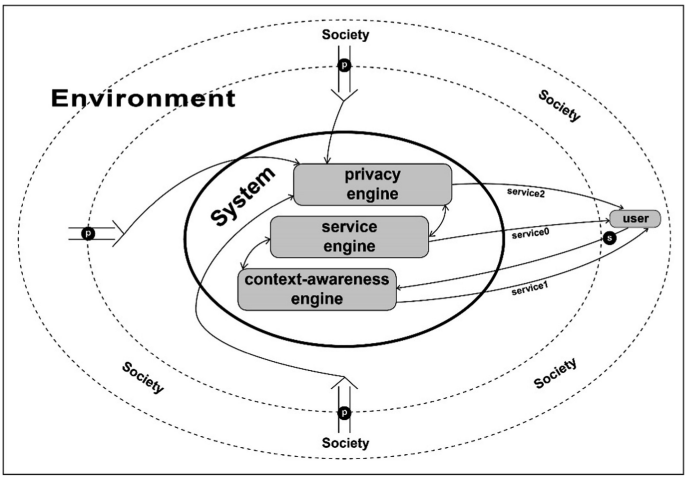

Fig. 1. Problem conceptualization

If we have a *system that is delivering services to a user* belonging to the system environment, then we may have the following *three situations*:

(i) *Situation "service0"*, as labelled in the figure: this is the *typical service provisioning*, when the (software) system is delivering a service to the user, *regardless of context and privacy demands*. An example of this is a ticket machine service – regardless of the user situation and of any specific privacy demands, the ticket machine is issuing tickets in the same way to any user in any situation.

(ii) *Situation "service1"* is when *the service is delivered in "versions"* in the sense that *depending on the situation of the user*, a corresponding service version is instantiated; the situation of the user is captured through sensors (see the black disk with "**s**" in the figure), as displayed in the figure. An example of this is an intelligent music playing service that may be adjusted not to play while the user is sleeping, to play tender music during morning hours, to play rhythmic music while the user is driving, and so on, assuming the possibility for capturing the user situation either through sensing (sensing that the user is in the car, for instance) and/or through a timer, or in another way.

(iii) *Situation "service2"* is featuring a *privacy-driven adaptation of the service delivery*, assuming that the system is receiving *privacy demands* from *Society* (see the black disks with "**p**" in the figure) that are to be "*translated*" into *functional solutions*. For example, security monitoring may need to be updated, driven by public demands, such that any captured visual information is to be destroyed if after some period of time, no incident has occurred.

Hence, if we assume that (i) is covered by current software specification approaches, such as SDBC, we consider challenging achieving (ii) and/or (iii), and if both are to be achieved – to resolve possible tensions.

Our proposed way of tackling this will be explained in the following sections.

4 Background and Related Work

As mentioned in the Introduction, in this section we address *context-awareness* and *privacy*, by providing brief introductions and considering *related work*.

4.1 Context-Awareness

The advances in wireless telecommunications and sensor technology, in combination with the capabilities of smart devices, have empowered IT systems to "know" what is going on with the end user while (s)he is utilizing corresponding services – this represents a *user perspective* in service delivery. Hence, *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 [1] through which body vital signs are captured, may appear to be at "normal state" and then, for example, vital signs are captured and recorded as archival information, or the person may appear to be in an "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 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 Fig. 2.

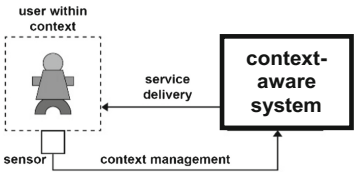

Fig. 2. A schematic representation of a context-aware system

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.

Context-aware systems actually deliver services to the user by means of ICT applications ("applications", for short). Hence, unlike "traditional" applications assuming that users would have common requirements independent of their context, **context-aware applications** are capable of *adapting their behavior to the situation of the user*. This is especially relevant to services delivered via *mobile devices*. Such applications are, to a greater or lesser extent, aware of the user context situation (for example: user is at home, user is traveling) and provide the desirable services corresponding to the situation at hand. This quality points also to another related characteristic, namely that context-aware applications must be able to *capture or be informed about information on the context of users, preferably without effort and conscious acts from the user part.*

Developing context-aware applications is hence not a trivial task and as above suggested, the following related challenges have been identified: (i) Properly deciding what physical context to sense and what high-level context information to pass to an application, and also bridging the gap between raw context data and high-level context information; (ii) Deciding which potential end-user context situations to consider and which ones to ignore; (iii) Modeling context-aware application behavior including switching between alternative behaviors [30].

The basic assumption underlying the development of context-aware applications is that *user needs are not static*, however partially dependent on the particular situation the user finds himself/herself in, as already mentioned. For example, depending on his/her current location, time, activity, social environment, environmental properties, or physiological properties, the user may have different interests, preferences, or needs with respect to the services that can be provided by applications.

Context-aware applications are thus primarily motivated by their **potential to increase user-perceived effectiveness**, i.e. to provide services that better suit the needs of the user, by *taking account of the user situation*. We refer to the collection of parameters that determine the situation of a user, and which are relevant for the application in pursue of user-perceived effectiveness, as *user context*, or *context* for short, in accordance to definitions found in literature [6].

As above-mentioned, context-awareness also implies that information on the user context must be captured, and preferably so without conscious or active involvement of the user. Although in principle the user could also provide context information by directly interacting with the application, one can assume that in practice this would be too cumbersome if not impossible; it would require deep expertise to know the relevant context parameters and how those are correctly defined, and furthermore be very time consuming and error-prone to provide the parameter specifications as manual input [30].

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* [37]. *Methodologies for architectural design* were proposed by researchers, such as: Context Toolkit – it aggregates context information [6], Context Modeling Language [11] & Model Driven Development (MDD), and UML- based approaches [2, 35] which mainly describe the *key steps and activities for modeling context-aware applications*; next to that: Contextual Elements Modeling and Management through Incremental Knowledge Acquisition (CEManTIKA) support the development of context-aware applications. Further, Vom Brocke et al. 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 [38]; 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 *concepts*, *networking aspects*, *middleware aspects*, *user-interface-related concerns*, *services*, and so on. Still, even such a wide consideration of context-aware applications has not yet inspired (in our view) a widely accepted agreement on the development of such 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.

4.2 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*. In this paper, we are focusing on privacy not only because it is one of the key values (e.g. [14]) 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) and then we will focus on privacy enforcement practices (related work) that are to be taken into account with regard to our case-driven modeling approach.

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** [24]. 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 concerning border control as privacy sensitive, referring to the Pearson's *privacy information classification* [24]:

- *Personally identifiable information*: information that can be used to identify an individual
  - *Data from records*: name, date of birth, bio-metrics, 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.

In studying RELATED WORK, we acknowledge that privacy enforcement is often difficult. ICT enables the creation of systems that ensure the privacy of data, which is called **`privacy-by-design`** [14]. 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* (such as data updating processes and procedures, access rights, decision-making responsibilities, and so on) and *technical aspects* (such as 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 are to be (adapted and) used. 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 border control information systems* - 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 [18].

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 [25, 40]. 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 PETs, **PIT**s (**Privacy-Invasive Technologies**) and privacy threats are also frequently examined in various domains [4, 13, 17, 34, 39].

Nevertheless, there is still limited insight in how enterprises can reduce privacy violation risks for open data in particular, and there is no uniform approach for privacy protection [16].

5 SDBC

In considering the **SDBC** approach in the current section, we will firstly provide *justification* with regard to our choice to base our modeling on that approach, secondly, we will discuss the *Design Science relevance* of SDBC, and finally, we will *briefly outline the approach*.

5.1 Justification

As studied by Shishkov [26], there are many other approaches/modeling languages, 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 related 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 [36] and is *consistent with MDA*.
- In previous work, SDBC has been *considered particularly in the border security application domain* [29].

For this reason, we have opted for adopting SDBC in the current work and in the following sub-section we will also justify the Design Science relevance of the approach.

5.2 Relevance to Design Science

In *Design Science research*, the **information systems research framework** proposed by Hevner et al. [12] has been widely accepted and applied in many IT artefact designs [23]. According to that framework, researchers develop an IT artefact, by considering the business needs and limits within an appropriate environment which consists of involved people, organizations and available technologies [12]. In such a design process, for the sake of supporting the design, researchers use: (i) existing knowledge bases (that include theories, models, and methods) as knowledge foundations and (ii) data analysis, measures, and validation criteria as methodologies. After having been developed, the IT artefact is to be evaluated and justified via analysis, case studies, experiments and/or simulation. New developed artefacts can also contribute to the knowledge base(s) accumulation.

Hence, referring to Design Science, we acknowledge that SDBC is essentially oriented towards a goal-driven modeling that relates to corresponding user needs and the modeling itself is justified by the capabilities (and limitations) of the corresponding entities contributing to the service deliveries. For this reason, we consider SDBC as *relevant in general* with regard to Design Science. Nevertheless, SDBC lacks powerful goal generation mechanisms and for that it needs support from other tools – for example, tools related to Artificial Intelligence [27]; anyway, this is left beyond the scope of the current paper.

5.3 Outline

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 integrating the two, 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 [26] for information on SDBC and we are reflecting the SDBC outline in Fig. 3.

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).

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 causalities concerning those inter-relations need to be modeled, such that we know what is required in order for something else to happen [32]. On that basis, the dynamics (the entities' behavior) is to be considered, featured by *transactions* (see Definition 2). 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 Sect. 2). Next to that, re-visiting such models could possibly inspire enterprise re-design activities, as shown in Fig. 3.

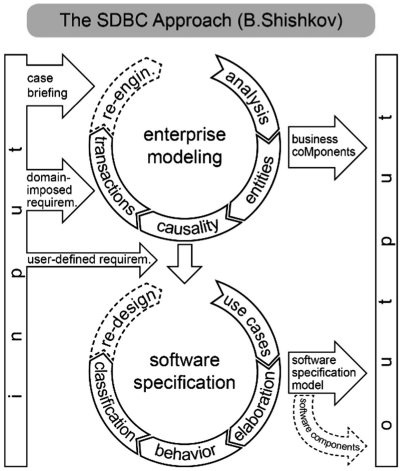

Fig. 3. SDBC - outline (Source: [28], p. 48)

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** [19] and the **Unified Modeling Language** – **UML** [36] but is also considered to be broadly accepted beyond RUP-UML [5, 8, 26]. The *use cases* are then elaborated, inspired by studies of Cockburn [5] and Shishkov [27], 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.

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 Fig. 3, corresponds to a particular ICT application being specified. **`Software components`** in turn 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* – this will be considered in the following section.

6 Weaving in Context-Awareness and Privacy

Considering the *problem conceptualization* (see Fig. 1), we are deriving three key demands with regard to the desired weaving of context-awareness and privacy:

- We need to be able to *capture user context to be used by the (software) system for its adapting services* delivered to the user;
- We need to be able to *capture public-values-related demands (such as privacy) and "translate" them into functional (software) solutions*;
- If fulfilling both privacy demands and context awareness would assume tensions (because of conflicting requirements), we need to be able to *resolve those tensions in a socially-responsible way*.

All those demands originate from the (enterprise) environment of the (software) system-to-be but require technology-specific (software) solutions. For this reason, we could neither position those demands as relevant to the enterprise modeling SDBC milestone nor can we position them as relevant to the software specification SDBC milestone; thus, they have to be positioned in between, as suggested by Fig. 4.

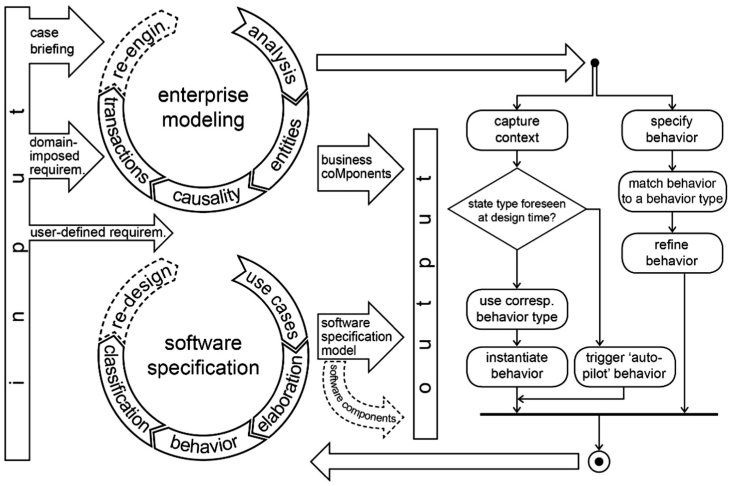

Fig. 4. Extending SDBC – weaving in context-awareness and privacy

Hence, as shown on the figure, the difference with the common SDBC modeling (with no specific needs for weaving context-awareness and privacy in the design) would be that the OUTPUT of Milestone 1 (Enterprise Modeling) is not the INPUT for Milestone 2 (Software Specification). Instead, the Milestone 1 output would have to undergo some transformations to become a Milestone 2 input: this is presented in the right part of Fig. 4, using the notations of the UML Activity Diagram [36] – the "start" point relates to the Milestone 1 output while the "end" point relates to the Milestone 2 input. What is going on between those two points and how is it justified?

- Two processes flow in parallel, one related to the desired *context-awareness enforcement* (left part of the Activity Diagram) and the other one – to the desired *privacy enforcement* (right part of the Activity Diagram).
- Those two parallel processes hold an *a-priori* equal importance but in the end, they reach a *synchronization bar* where BALANCING NORMS are to be implemented, as a final parameterization of the Milestone 2 input. For example, in the case of a security camera video-recording (the security system is assumed to be adapting to the monitoring circumstances as well as to be privacy-sensitive and not distribute facial information) if there is indication for real-time criminal activities, then according to a *balancing norm*, context-awareness should prevail over privacy and no privacy sensitivity would be observed towards those persons being monitored.
- As for the context-awareness process, it follows from the context-aware service delivery features, as introduced and discussed in the Introduction and in Subsect. 4.1, also assuming that *for each user state type that is of high occurrence probability, there is a corresponding system behavior type* that is prepared at *design time*. Then the first thing to be done is to *capture the user context* and if there is a system behavior type (prepared at design time) that corresponds to the user state

type (to which the captured context is pointing), then that behavior type is instantiated accordingly. Otherwise, "auto-pilot" behavior would have to be triggered that is guiding the system based on rules that are applied at run time.

- As for the privacy process, it follows from the privacy-related features as introduced and discussed in the Introduction and in Subsect. 4.2, also assuming that *for each situation type of high occurrence probability there is a corresponding system behavior type specified at design time, and there are corresponding privacy-related "instructions"*. Then it is only necessary to position the "current" system behavior with regard to a corresponding behavior type (for example: "camera surveillance while Police are chasing a criminal", "camera surveillance while a person is walking in a public area", and so on), such that it is known how the behavior instance would need to be refined.

Those methodological guidelines have been considered firstly by Shishkov et al. [28] and also in the current paper, mainly inspired by the relevance of weaving *context-awareness* and *privacy-by-design* in the (SDBC-driven) specification of software, considering dedicated studies addressing context-awareness and privacy. Still, our guidelines are not yet exhaustive and need further elaboration, especially as it concerns some modeling transformations (as it will be seen in the following section). Nevertheless, the partial validation (realized in terms of a case example) clearly demonstrates the adequacy of the proposed guidelines and their relevance. This will inspire further related research activities that would not only address the model transformations with regard to the context-privacy perspective considered in the current paper but would also broaden the public demands perspective, reaching as far as **`Value-Sensitive Design`** [9].

The above-mentioned illustrative case example will be considered in the following section.

7 Illustrative Example

As mentioned in the Introduction, we partially validate our proposed way of modeling (that touches upon the weaving of context-awareness and privacy) by means of an *illustrative example*, featuring **`land border security`**. In this section, we will firstly present the case briefing and then we will proceed with the security system modeling.

7.1 Case Briefing

Border control is one of Europe's biggest recent challenges, in the light of severe *sea border problems* in Greece and Italy in 2015–2017 [10] and *land border problems* in Bulgaria and Croatia, for example. This leads not only to *deadly incidents* for numerous migrants who undertake 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 tensions 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 EU efforts, aiming at improving security at the external borders of the European Union – for example: new border facilities are constructed along those borders, Police officers from some Western EU countries are sent to the South-Eastern EU borders to physically help, new organizational approaches and technical solutions are developed, and so on, as according to the European Union; all those efforts are directed towards stopping the illegal migration to the European Union 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 take 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* [20]. 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 [29]. We acknowledge this "duality" – *human entities* vs. *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 situation types accordingly. In realizing all this, we take as an example the situation at the *Bulgarian-Turkish land border* [29]; 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, such a facility can be overcome by 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 deployed in the border area, it would be physically impossible to guarantee police presence at any time anywhere along the border, over hundreds of kilometers. (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 plus artificial reasoning; based on this, it may generate contentful messages to be transmitted to corresponding human agents.

Taking the above into account, we argue that 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" with regard to (a) because:

- Within NS, all is just progressing according to *pre-defined rules* – hence, there is no need to adapt the system behavior 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 in turn *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 persons 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 run-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 gets 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.

7.2 Modeling the Border Security System

A logical starting point in our case-driven modeling is the "translation" of the *case briefing* 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-design (re-engineering) that is needed for the sake of making the considered enterprise system adequately supportable by ICT applications [7].

Because of the limited scope of this paper, we are not going in detail on how we analyze the case briefing and how we conduct such a partial enterprise re-design. 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-by-design. Hence, we move directly to the textual reflection of the case briefing, holding in itself re-design-driven and requirements-driven updates:

> Different situations may occur at the land border. Legislation 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. For the sake of brevity, we do not go in further detail here; still, for more information on those issues, interested readers are referred to [26].

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.

* 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**).

- **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 RECOMMENDATION) in order to establish their actions accordingly.

Thus, next to identifying **`entities`** (featuring actor-roles [7, 26]), 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 [7] – see Fig. 5:

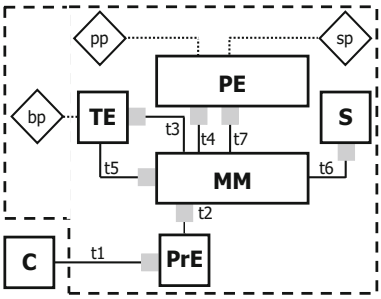

Fig. 5. Business entity model for the border security case

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 [26]. 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'.

Further, we have to make explicit the the **`causal relationships`** among the transactions, and given 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 Fig. 6, using the notations of UML Activity Diagram [36].

As can be 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 Fig. 6. Further, with regard to the SDBC modeling process, we have

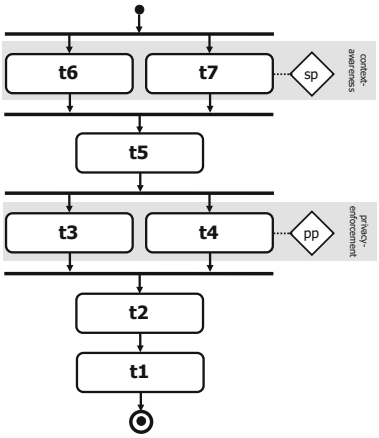

Fig. 6. Modeling the causal relationships among transactions

identified the *entity model* and the *causality relations*. What goes next are *transactions* (see Fig. 3) and with regard to this, we use the SDBC interpretation of the transaction concept – see Fig. 7.

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 it is seen from Fig. 7, 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

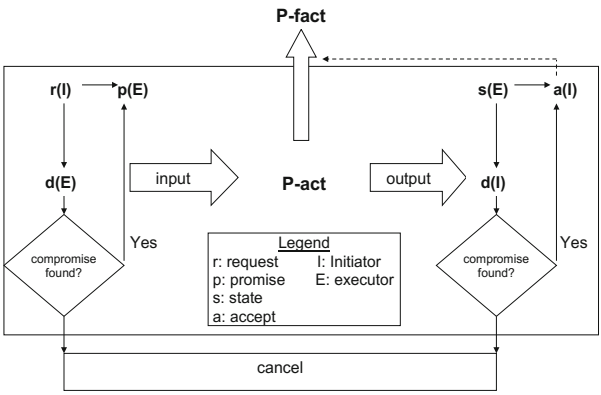

Fig. 7. The SDBC interpretation of the transaction concept (Source: [27], p. 70)

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 (co-ordination) *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 Fig. 6 and *elaborate* each transaction, using the *transaction pattern* presented in Fig. 7. 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 Fig. 8 (abstracting from declines and cancellations), featuring only part of the model depicted in Fig. 6, namely, focusing only on transactions **5**, **6**, and **7**:

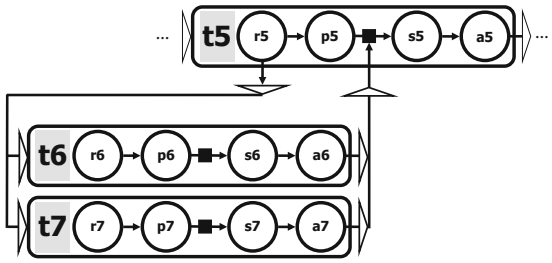

Fig. 8. Detailed behavior aspect model featuring transactions

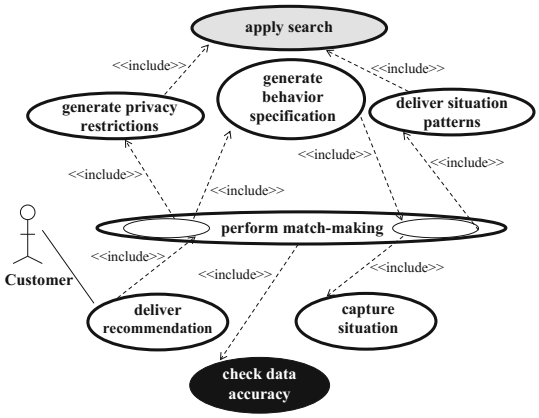

Fig. 9. Partial use case model for the border security case

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).

That 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 Fig. 3. For detailed information concerning the *derivation of use cases from transactions*, interested readers are referred to [26] – for the sake of brevity, we go directly to a partial *use case model*, derived on the basis of the 7 transactions (see Figs. 5 and 6). The model is depicted in Fig. 9.

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 [26, 33]. Those use cases usefully drive the alignment between *enterprise modeling* and *software specification*, guaranteeing that the software system-to-be is stemming from corresponding enterprise models.

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 [26]. 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 [27] where this is considered and justified by means of a case study. Still, we will consider (in future research) those issues with regard to the land border case example.

8 Conclusions

The contribution of the current paper concerns a proposed design approach that allows for smoothly reflecting context and privacy features (and tackling possible tensions among the two) in the application specification, supported by methodological guidelines that span over the enterprise modeling and software specification, fueled by the SDBC approach. We have partially demonstrated our way of modeling by means of a case example featuring the domain of land border security. Hence, we have not only contributed to the enterprise-software alignment research (addressing also the challenge of weaving context-awareness and privacy-by-design in the software specification) but we have also delivered a useful domain-specific study featuring an application domain where context-awareness and privacy have essential impact. As future research, we plan to consider a large-scale border security case study assuming software development activities as well as the consideration of other public values as well (next to privacy), such as transparency and accountability.

References

1. AWARENESS. Freeband AWARENESS Project (2008). http://www.freeband.nl
2. Ayed, D., Delanote, D., Berbers, Y.: MDD approach for the development of context-aware applications. In: Kokinov, B., Richardson, D.C., Roth-Berghofer, T.R., Vieu, L. (eds.) CONTEXT 2007. LNCS (LNAI), vol. 4635, pp. 15–28. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74255-5_2
3. Bunge, M.A.: Treatise on Basic Philosophy. A World of Systems, vol. 4. D. Reidel Publishing Company, Dordrecht (1979)
4. Burghardt, T., Buchmann, E., Böhm, K.: 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 (2008)
5. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Boston (2000)
6. Dey, A.K.: Understanding and using context. Pers. Ubiquit. Comput. **5**(1), 4–7 (2001)
7. Dietz, J.L.G.: Enterprise Ontology, Theory and Methodology, 1st edn. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-33149-2

8. Dietz, J.L.G.: Generic recurrent patterns in business processes. In: van der Aalst, W.M.P., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 200–215. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44895-0_14
9. Friedman, B., Hendry, D., Borning, A.: A survey of value sensitive design methods. Int. J. Found. Trends. Hum. Comput. Interact. **11**, 63–125 (2017)
10. FRONTEX: The website on the European Agency, FRONTEX (2018). http://frontex.europa.eu
11. Henricksen, K., Indulska, J.: Developing context-aware pervasive computing applications: models and approach. Perv. Mob. Comput. **2**, 37–64 (2006)
12. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
13. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: 1st International ACM Conference on Electronic Commerce, EC 1999. ACM (1999)
14. Hustinx, P.: Privacy by design: delivering the promises. Identity Inf. Soc. **3**(2), 253–255 (2010)
15. IoTDI 2nd International Conference on Internet-of-Things Design and Implementation. ACM/IEEE (2017)
16. Janssen, M., Van den Hoven, J.: Big and open linked data (BOLD) in government: a challenge to transparency and privacy? Gov. Inf. Q. **32**(4), 363–368 (2015)
17. Johnston, A., Wilson, S.: Privacy compliance risks for Facebook. IEEE Technol. Soc. Mag. **31**(2), 59–64 (2012)
18. Könings, B., Schaub, F., Weber, M.: Privacy and trust in ambient intelligent environments. In: Ultes, S., Nothdurft, F., Heinroth, T., Minker, W. (eds.) Next Generation Intelligent Environments, pp. 133–164. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-23452-6_4
19. Kruchten, P.: The Rational Unified Process, An Introduction. Addison-Wesley, Boston (2003)
20. LBS. LandBorderSurveillance, the EBF, LandBorderSurveillance Project (2012). http://ec.europa.eu
21. Liu, K.: Semiotics in Information Systems Engineering. Cambridge University Press, Cambridge (2000)
22. MDA. The OMG Model Driven Architecture (2018). http://www.omg.org/mda
23. Offermann, P., Blom, S., Schönherr, M., Bub, U.: Artifact types in information systems design science – a literature review. In: Winter, R., Zhao, J.L., Aier, S. (eds.) Global Perspectives on Design Science Research. DESRIST 2010. LNCS, vol. 6105, pp. 77–92. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13335-0_6
24. Pearson, S.: Taking account of privacy when designing cloud computing services. In: International Workshop on Software Engineering Challenges of Cloud Computing, ICSE 2009 (2009)
25. Seničar, V., Jerman-Blažič, B., Klobučar, T.: Privacy-enhancing technologies approaches and development. Comput. Stand. Interfaces **25**(2), 147–158 (2003)
26. Shishkov, B.: Enterprise Information Systems, A Modeling Approach, 1st edn. IICREST, Sofia (2017)
27. Shishkov, B.: Software specification based on re-usable business components (Ph.D thesis), 1st edition, TU Delft. Delft (2005)
28. Shishkov, B., Janssen, M., Yin, Y.: Towards context-aware and privacy-sensitive systems. In: 7th International Symposium on Business Modeling and Software Design, BMSD 2017. SCITEPRESS (2017)

29. Shishkov, B., Mitrakos, D.: Towards context-aware border security control. In: 6th International Symposium on Business Modeling and Software Design, BMSD 2016. SCITEPRESS (2016)
30. Shishkov, B., van Sinderen, M.: From user context states to context-aware applications. In: Filipe, J., Cordeiro, J., Cardoso, J. (eds.) ICEIS 2007. LNBIP, vol. 12, pp. 225–239. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88710-2_18
31. Shishkov, B., Van Sinderen, M.J., Tekinderdogan, B.: Model-driven specification of software services. In: IEEE International Conference on e-Business Engineering, ICEBE 2007. IEEE (2007)
32. Shishkov, B., Van Sinderen, M.J., Quartel, D.: SOA-driven business-software alignment. In: IEEE International Conference on e-Business Engineering, ICEBE 2006. IEEE (2006)
33. Shishkov, B., Dietz, J.L.G.: Deriving use cases from business processes, the advantages of DEMO. In: 5th International Conference on Enterprise Information Systems, ICEIS 2003. SCITEPRESS (2003)
34. Seigneur, J.-M., Jensen, C.D.: Trading privacy for trust. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) iTrust 2004. LNCS, vol. 2995, pp. 93–107. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24747-0_8
35. Simons, C., Wirtz, G.: Modeling context in mobile distributed systems with the UML. Vis. Lang. Comput. **18**(4), 420–439 (2007)
36. UML. The Unified Modeling Language (2017). http://www.uml.org
37. Vieira, V., Tedesco, P., Salgado, A.C.: Designing context-sensitive systems: an integrated approach. Expert Syst. Appl. **38**(2), 1119–1138 (2011)
38. Vom Brocke, J., Zelt, S., Schmiedel, T.: On the role of context in business process management. Inf. Manag. **36**(3), 486–495 (2016)
39. Weber, R.H.: The digital future - a challenge for privacy? Comput. Law Secur. Rev. **31**(2), 234–242 (2015)
40. Zhu, N., Zhang, M., Feng, D., He, J.: Access control for privacy protection for dynamic and correlated databases. In: International IEEE SmartCity Conference, SmartCity 2015. IEEE (2015)