Three Categories of Context-Aware Systems

Shishkov, Boris; Larsen, John Bruntse; Warnier, Martijn; Janssen, Marijn

Published in:
Business Modeling and Software Design

Link to article, DOI:
10.1007/978-3-319-94214-8_12

Publication date:
2018

Document Version
Peer reviewed version

Three Categories of Context-Aware Systems

Shishkov, Boris; Larsen, John Bruntse; Warnier, Martijn; Janssen, Marijn

DOI
10.1007/978-3-319-94214-8_12

Publication date
2018

Document Version
Accepted author manuscript

Published in
Proceedings of Business Modeling and Software Design - 8th International Symposium, BMSD 2018

Citation (APA)
Shishkov, B., Larsen, J. B., Warnier, M., & Janssen, M. (2018). Three Categories of Context-Aware Systems. In Proceedings of Business Modeling and Software Design - 8th International Symposium, BMSD 2018 (Vol. 319, pp. 185-202). (Lecture Notes in Business Information Processing; Vol. 319). Springer Verlag. https://doi.org/10.1007/978-3-319-94214-8_12

Three Categories of Context-Aware Systems

Boris Shishkov[1,4], John Bruntse Larsen[2], Martijn Warnier[3], Marijn Janssen[3]

[1] Institute of Mathematics and Informatics
Bulgarian Academy of Sciences, Sofia, Bulgaria
[2] Department of Applied Mathematics and Computer Science
Technical University of Denmark, Lyngby, Denmark
jobla@dtu.dk
[3] Faculty of Technology, Policy, and Management
Delft University of Technology, Delft, The Netherlands
M.E.Warnier@tudelft.nl, M.F.W.H.A.Janssen@tudelft.nl
[4] Institute IICREST, Sofia, Bulgaria
b.b.shishkov@iicrest.org

Abstract. With regard to context-aware systems: some optimize system-internal processes, based on the context state at hand; others maximize the user-perceived effectiveness of delivered services, by providing different service variants depending on the situation of the user; still others are about offering value-sensitivity when the society demands so. Even though those three perspectives cover a broad range of currently relevant applications there are no widely accepted and commonly used corresponding concepts and terms. This is an obstacle to broadly understand, effectively integrate, and adequately assess such systems. We address this problem, by considering a (component-based) methodological derivation of technical (software) specifications based on underlying enterprise models. That is because context states are about the enterprise environment of a (software) system while the delivery of context-aware services is about technical (software) functionalities; hence, we need a perspective on both. We consider the SDBC (Software Derived from Business Components) approach that brings together enterprise modeling and software specification. On that basis: (a) We deliver a base context-awareness conceptualization; (b) We partially align it to agent technology because adapting behaviors to environments assumes some kind of proactivity that is only fully covered by agent systems, in our view. We partially illustrate our proposed conceptualization and particularly - the agent technology implications, by means of a case example featuring land border security.

Keywords: Modeling; System design; Context-awareness; SDBC; AORTA

1 Introduction

Whenever a group of *entities* collectively realize a *goal*, we consider them to belong to a **`system`** [17]. An **`adaptable system`** has the ability to adjust to new conditions [8]. We argue that an essential feature of adaptable systems is **`context-awareness`** [22] – this is *adjusting the system behavior depending on the situation at hand* (**context state**). Our observation is that **`context-aware systems`** are all about adjusting "something" to the *context state*; however, what is adjusted differs: (i) Some context-aware systems optimize system-internal processes based on the context state at hand [3,12], for example: *regulating the electro-consumption of home appliances for the sake of keeping the overall building consumption within some boundaries*. (ii) Other context-aware systems maximize the user-perceived effectiveness of delivered services, by

providing different *service variants* depending on the situation of the user [1], for example: *treating a distantly monitored patient in one way when his/her condition is normal and in another way in case of emergency*. (iii) Still other context-aware systems are about offering value-sensitivity when the society demands so [7], for example: *in the case of supporting judiciary processes, different levels of transparency are to be provided to different categories of stakeholders*. Not claiming exhaustiveness, we argue that those three context-awareness perspectives cover a broad range of currently relevant applications, while corresponding system categorizations are observed to be missing in literature, especially as it concerns real-life (business) processes. Hence, we propose and elaborate (in the following section) the above **`categorization`**.

Further, there are *no widely accepted and commonly used concepts and terms* with respect to (i), (ii), and (iii). This is considered to be an obstacle with regard to *broadly understanding*, *effectively integrating*, and *adequately assessing* such systems. We address this problem, by taking a (component-based-) technical / software design perspective, assuming a desired *methodological derivation* of the **`technical`** (**`software`**) **`specifications`** on the basis of underlying **`enterprise models`**. That is because *context states* are about the *enterprise environment* of a (software) system while the delivery of *context-aware services* is about *technical* (*software*) *functionalities*; hence, we need a perspective on both. Stepping on *previous work*, we consider the **`SDBC`** (**`Software Derived from Business Components`**) approach that brings together *enterprise modeling* and *software specification* [17,18,20]. On that basis:

- We deliver a base context-awareness conceptualization (inspired by the current discussion and the analysis carried out in Section 2) that is claimed to hold for all (i), (ii), and (iii).
- We partially align that conceptualization to agent technology [11,27] because adapting behaviors to environments is considered to assume some kind of pro-activity that is only fully covered by agent systems, in our view.

We partially *illustrate* our proposed conceptualization and particularly - the agent technology implications, by means of a *case example* featuring **`land border security`**.

The remaining of the current paper is organized as follows: Section 2 elaborates and discusses the above-mentioned context-awareness perspectives (*optimizing internal processes, maximizing the user-perceived effectiveness, being value-sensitive*). In Section 3, we: (a) briefly *outline the SDBC approach* that gives the general methodological guidelines we follow; (b) elicit (in line with SDBC) the *base context-awareness system concepts*, referring to the analysis in Section 2; (c) enrich those concepts from the *AORTA* (*agent-technology*) *perspective*; (d) distill on that basis a *meta-model* that is considered a key contribution of this paper and propose guidelines refinements. Assuming the SDBC methodological guidelines and referring to a real-life case study that is featuring the usage of drones in support of land border security, we partially illustrate in Section 4 the meta-model and the AORTA framework influence, emphasizing the applicability with regard to all three categories of context-aware systems, as discussed above. We justify the adequacy of the delivered contribution in Section 5, by analyzing *related work*. Finally, in Section 6 we *conclude* the paper.

2 System Behavior Perspectives

Referring to the notions considered in the Introduction, we will firstly elaborate on the context-driven optimization of system-internal processes (Sub-section 2.1), secondly – on the context-driven maximization of the user-perceived effectiveness (Sub-section 2.2), and finally, on the context-driven value-sensitivity (Sub-section 2.3). It is often that the context-awareness is enabled by sensor technology [21] allowing to “know” what is happening around; alternatively, there should be other ways of “sensing” the environment [1]. As it concerns

(i), (ii), and (iii) – see the previous section – this counts for all of them. Further, considering the essence of their underlying system behaviors, we use the following labels: SELF-MANAGING CONTEXT-AWARE SYSTEM for (i); USER-DRIVEN CONTEXT-AWARE SYSTEM for (ii), and VALUE-SENSITIVE CONTEXT-AWARE SYSTEM for (iii). Finally, even though most often context-aware systems are "sensitive" to changes in the system environment, it is also possible that the "sensitivity" is towards internal issues (things happening inside the system (not in the environment) may trigger either internal optimizations, or changes in the services delivered to the user, or a reconsideration of the "covered" values). In this paper, we are not restrictive with regard to the "sensitivity" (whether it concerns the system or the environment).

2.1 Self-Managing Context-Aware Systems (SMCAS)

SMCAS' context-awareness is directed towards **`internal (system) optimization purposes`** [16]. Such **`autonomic`** solutions [12] are proposed as a way to reduce the cost of maintaining complex systems, and to increase the human ability to manage these systems properly, by automating (part of) their working. In essence, **`self-managing`** system can be characterized by a **`feedback loop mechanism`** that allows them to optimize their working based on input from the environment. For the basic *feedback loop*, the system receives *input from the environment* (monitor) and can change its behavior which in turn has an *effect on the environment* (effector). In **`Autonomic Computing`** [10] this basic loop is extended into *four components*, resulting in the **`MAPE Cycle`**, see Figure 1 (left). Next to the "monitor" and "effector" phases, the system internally has an analyze phase that *processes environmental input* and a plan phase that *changes the internal and external working of the system*.

Taking this one step further, an **`autonomic system`** can manage another system (the **`managed system`**) by placing it in the *MAPE Cycle* as shown in Figure 3 (right). Placed in such a configuration, the *autonomic* and *managed* systems together form a **`SELF-MANAGING SYSTEM`** or *self-adaptive system* [14].

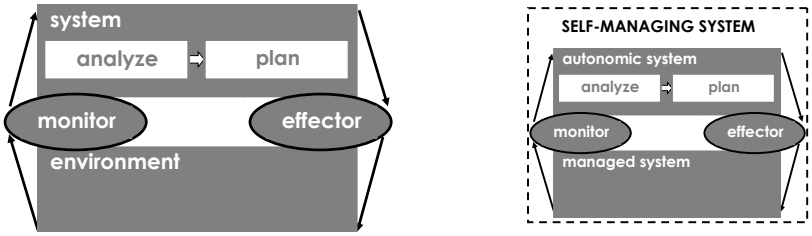

Fig. 1. *M*onitor-*A*nalyze-*P*lan-*E*ffect (MAPE) cycle (left); A *managed system* and an *autonomic system* that together form a *self-managing system* (right)

Internally *self-managing systems optimize their behavior based on inputs to and outputs from the managed system*. Such state updates can range from simple **if-then rules** (for example: *if the temperature is below zero degrees, then pre-heat the car*) to more sophisticated approaches such as those based on **machine learning techniques** (e.g., *neural networks* or *inference engines that determine the best action based on a large internal knowledge base*). Thus, the main **`objective of self-managing systems`** is to optimize their internal working based on inputs from the environments.

2.2 User-Driven Context-Aware Systems (UDCAS)

The UDCAS' context-awareness is directed towards the **maximization of the external (user) satisfaction** [1]. Hence, such systems should be able to: (i) identify the situation of the user (possibly through *sensors*); (ii) deliver a service to the user, that is suited for the particular situation, as illustrated in Figure 2 (left).

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. UDCAS actually deliver services to the user by means of ICT (Information and Communication Technology) applications [17] (**applications**, for short). Hence, unlike "traditional" applications assuming that users would have common requirements independent of their context, *user-driven 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). Hence, 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 *user-driven 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** [18]. Hence, a basic *assumption* underlying the development of *user-driven 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*.

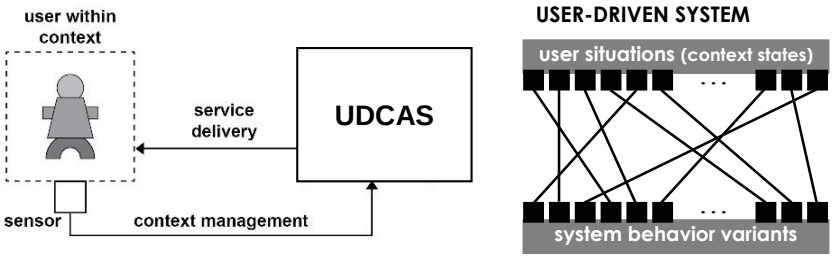

Fig. 2. The UDCAS vision (inspired by [18,22]): a schematic representation (left); Context-states-driven system behavior variants (right)

User-driven context-aware applications are thus primarily motivated by their **potential to increase the 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 [5].

Finally, UDCAS are hence about delivering **behavior variants** to corresponding user situations (**context states**), as illustrated by Figure 2 (right). The idea is that for each *context state*, the system has a *behavior variant*. Nevertheless, this would not be always realistic because if the possible *context states* are too

many (for example: tens or hundreds), the effort for "preparing" (at design time) *behavior variants* would be huge. For this reason, *statistical data analysis* and *probability studies* are needed, as studied in [17], for establishing the **`context states of high occurrence probability`** (for example, during a normal working day, an employee would most probably be either *at work* or *at home*, or *traveling*, and it is not very likely that the employee is somewhere else, especially during day hours). Hence all other (possibly hundreds) low probability *context states* would not need to be addressed at design time; instead, a "collecting" *context state* (for example, labelled: OTHER) may be considered at design time, assuming a more *generic behavior algorithm*, such that the behavior is "tuned" at real time (possibly in a *rule-based way*). This would of course lead to lower quality-of-service which is nevertheless justified since it would be very rare that the OTHER behavior variant is triggered. Thus, this is a matter of "trade-off" between *quality-of-service* and *resources*.

2.3 Value-Sensitive Context-Aware Systems (VSCAS)

VSCAS' context-awareness is directed towards a **`sensitivity to public values`** [7]. *Public values* (**`values`** for short) like *privacy* and *data protection*, *security*, *accountability*, *integrity* and *provenance*, and *sustainable data storage*, often need to be incorporated in the functionalities of *information systems*. Hence, the first question to be answered is: How do values relate to requirements (we mean particularly *non-functional requirements* because values are essentially *non-functional*)? In answering this question, we refer to [19] who argue that values are desires of the general public (or public institutions / organizations that claim to represent the general public), that are about properties considered societally valuable, such as *respecting the privacy of citizens* or *prohibiting polluting activities*. Even though values are to be broadly accepted (that is why they are **`public`**), they may concern individuals (for example: considering privacy). Hence, put broadly, values concern the *societal expectations with regard to the way services should be delivered* and with regard to the above question: *values* are *desires* or *goals*, not *requirements*. Values are abstract and not directly related to an enterprise or software system, as opposed to requirements. Moreover, values are construct by and for society and not by and for the enterprise domain in which a specific system will be used. Those domains may overlap but are not the same. Values that are adopted as goals by an enterprise would therefore impact the requirements on a system that the enterprise wants to introduce in order to realize its goals. For this reason, *the impact of values cannot be limited to non-functional requirements*. It is therefore considered important to clearly distinguish *values* from *requirements* and acknowledge the limitations of requirements engineering with regard to the development of value-sensitive (software) systems.

Hence, the development of systems should take into account the objective, the user needs, but also the operating and societal context. In this way *values* can be used as a *guidance for making choices when developing systems*; looking at possible *tensions* among values is an issue as well. Thus, we have identified challenges in several directions:

Firstly, values are normative by nature and different stakeholders might prefer different values; also, values might differ among countries and cultures.

Secondly, even though different societies may agree on a value at high-level, their cultural and other differences may "push" for different value "realizations". This may result in different operationalizations and implementations of the same value.

Thirdly, values may be conflicting to each other (for example: fulfilling two values at the same time may be impossible). Searching for criminals might lead to violating the privacy of innocent people, for example.

Thus, in considering VSCAS, it is important to be aware that **`different context states may assume the consideration of different values`**, which in turn would mean **`different system functionality variants`**. Nevertheless, this goes beyond a mapping just between *values* and *non-*

functional requirements and would assume a *broader consideration of the software functionalities specification*.

Overall: *computing power becomes larger*, *wireless telecommunications are advancing*, and *sensor technology is developing fast* [17]; this allows for *ubiquitous network connectivity* and numerous *capabilities of smart devices*, as a basis for developments in several directions: (a) **Systems that are traditionally designed for one specific situation and task can be augmented to become "smarter", being able to operate in complex environments**. (b) **Systems are empowered to "sense" what is going on inside them and also what is going on with the end user while (s)he is utilizing corresponding services**. This concerns the system-internal processes, the way services are delivered to users, and the way values are considered, as discussed in the current section. In the following section, we will present our *proposed way of modeling context-aware systems*, taking into account those *three system behavior perspectives*.

3 Proposed Modeling and Design

Furthering previous work, we consider SDBC (see Section 1) as the approach of choice (generally) because of several reasons: its *strengths in aligning enterprise modeling and software specifications*, its *component-orientation* and support for *re-use*, and its *previous use for specifying context-aware and privacy-sensitive systems* [17,18,20]. We will thus briefly introduce SDBC in this section. We will then present *the main concepts* we consider, making sure that they are consistent with SDBC and relevant to the three categories of context-aware systems, as considered in the previous section; we will also provide a *conceptual enrichment from an agent-technology perspective* (see Section 1). We will then reflect this in a *meta-model* derived accordingly. And in the end, we will *narrow the design scope* (in the above context) to only touch upon issues that concern the key features of the considered three categories of context-aware systems.

3.1 SDBC

SDBC is an *approach* (consistent with *MDA* [15]) that is 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-to-be is considered in its enterprise context, which means that the *software specification models* are to stem from corresponding *enterprise models*; this means in turn 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 (*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 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 [17] for information on SDBC and we are reflecting the SDBC outline in Figure 3.

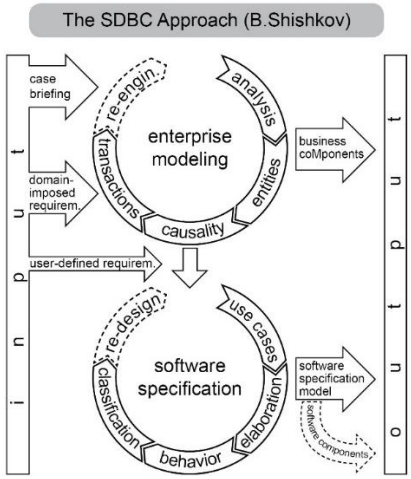

Fig. 3. SDBC - outline (Source: [20], p. 48)

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 [23]. On that basis, the dynamics (the entities' behavior) is to be considered, featured by *transactions* [17]. 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`** (models of *business components* [17]). Next to that, re-visiting such models could possibly inspire enterprise re-design activities, as shown in Figure 3.

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`** [13] and the **`Unified Modeling Language – UML`** [25] but is also considered to be broadly accepted beyond RUP-UML [17]. The *use cases* are then elaborated, inspired by studies of Cockburn [4] and Shishkov [17], 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 be inspiration for proposing in the future 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) enterprise information system. Thus the label **`software specification model`**, as presented in Figure 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; the second SDBC milestone is about the identification of software coMponents and corresponding software components.

3.2 Conceptualization

In this sub-section, we will firstly consider the basic concepts aligned with SDBC and the system behavior perspectives, as considered in the previous section.

BASIC CONCEPTS (GENERAL)

SYSTEM: A collection of elements possibly interacting with each other, driven by the purpose of delivering a service to another entity or group of entities.

SUB-SYSTEM: A system part identified based on a functional decomposition with regard to the system (hence, a system can optimize itself, by optimizing corresponding sub-systems).

ENVIRONMENT: Anything not belonging to a system belongs to the system environment
Part of the environment concerns those environmental entities that are assumed to have some interaction with the system.

ENTITIES: Composition elements with regard to a system / environment
- Human vs Artificial entities;
- Passive (a passive entity is an entity that only performs actions when another entity
interacts with it) vs Autonomous (an autonomous entity is an entity that performs actions on its own initiative) entities;
- Sensing (capturing context data in support of the system's service delivery) vs Actuator (causing changes in the environment on behalf of the system) entities.

ROLE: The state of carrying out certain objectives
- The entity-role combination is labelled "actor-role".

ACTOR: An autonomous entity that can enact a role.

USER: An actor that the system services.

ACTION: Something done by an entity
- A sequence of actions, occurring between two entities that are collaborating in support of the service delivery, is labelled "inter-action".

OBJECTIVE: The motive behind the service delivery.

REGULATIONS: Reflection of the existing norms that have impact on the service delivery, prescribing what is allowed in some situations, forbidden in others, and so on.

VALUES: Reflection of the public perception towards what is important regarding the service delivered by the systems*
* We address values that are shared among environmental human entities.

BASIC CONCEPTS (FEATURING SYSTEM ADAPTATION DIMENSIONS)

SELF-MANAGING BEHAVIOR: Context-driven enforcement of syst.-internal optimizations.
USER-DRIVEN BEHAVIOR: Context-driven service adaptation based on the user situation.
VALUE-SENSITIVITY: Service adaptation inspired by values, delivered through the operationalization of values.

Wishing to enrich those concepts from an agent-technology perspective (see the Introduction) **AORTA** is partially considered as a meta-model for **enabling agents to perform organizational reasoning** based on the *OperA* model [6] for agent organizations [11]. *Organizational reasoning* enables an agent to identify the objectives it is expected to solve, the other agents that it depends on for solving those objective, and whether an action is permitted, obliged, allowed or forbidden. In doing so the social expectations are made explicit through the AORTA concepts, which is considered an advantage when creating and reiterating the design of agents. Those *concepts* are shown below in the form of logical predicates:

CONCEPT	MEANING
role(Role, Objs)	Role is the name of a role and Objs is a set of main objectives of that role.
obj(Obj, SubObjs)	Obj is an objective that has SubObjs as a set of sub-objectives.
dep(Role_1, Role_2, Obj)	Role_1 depends on Role_2 in order to complete Obj.
rea(Ag, Role)	Agent Ag enacts Role.
cond(Role,Obj,Deadline,Cond)	When the condition Cond holds, Role is obliged to complete Obj before the objective Deadline.
obl(Ag, Role, Obj, Deadline)	Agent Ag is obliged to enact Role to complete Obj before Deadline.
viol(Ag, Role, Obj)	Agent Ag enacting Role has violated the obligation to complete Obj.

A *role* defines a *role* in the *organization*, and the *primary objectives* associated with that *role*. By *enacting a role*, an *agent* announces to other *agents* in the *organization* that it takes upon itself that *role* and thus that other *agents* can expect it to work towards solving the *objectives* associated with that *role*. It is also possible to make an explicit notion for how an *objective* can be *decomposed* into *sub-objectives*, and how a *role* may depend on other *roles* to solve an *objective*. Finally, *normative statements* are supported that define *conditions* on how *agents* should solve an *objective* in a certain *context*.

AORTA divides organizational reasoning into **three phases**: obligation check (OC), option generation (OPG) and action execution (AE). In the OC-phase the agent uses its beliefs to *check if obligations are activated*, *satisfied* or *violated*, and updates its beliefs accordingly. Following that it *generates possible organizational options* in the OPG-phase, which it then considers in the AE-phase when *deliberating an action*. Figure 4 visualizes such a reasoning "component".

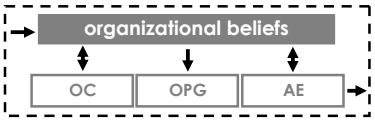

Fig. 4. Reasoning component

Further, in specializing some of the basic concepts (see above) from an agent-technology perspective (consistent with AORTA), we consider the following:

- Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control of their actions and internal state;
- Social "ability": agents interact with other agents (and possible humans) via some kind of agent communication language;
- Reactivity: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of those combined), and respond in a timely fashion to changes that occur in it;
- Pro-activeness: agents do not simply act in response to their environment, they are also capable of exhibiting goal-driven behavior, by taking the initiative.

Hence, we define an **`agent`** as an **`entity that is capable of operating autonomously, interacting with other agents, perceiving and reacting to changes in the environment, and taking actions toward solving an objective`**.

Next to that, it is considered common to have **`agent systems`** where *an agent represents a larger group of agents*; such an agent is often tasked with *coordinating* the other agents in the group (such tasks can be captured with the notions of *role* and *objective*, with *agents enacting roles*). Thus, by applying the *agent* concept, we could conveniently capture the notions of *autonomy*, *social ability*, *reactivity* and *pro-activeness* that we find useful with regard to the challenge of modeling complex (context-aware) systems where it is not straightforward understanding how every system part works in detail.

Finally, we present below the enrichment (from an **`agent-technology perspective`**) with regard to the basic concepts that were presented above:

SYSTEM: A *system* is represented through the notion of an *agent* that enacts a designated system *role*. The *role* specifies the *objectives* (**`obj`**) of the system, reflecting its *purpose*. These *objectives* can be decomposed into *sub-objectives* by *obj*-statements. We decompose the *objective* of the *system* into *sub-objectives*, matching the *objectives* of the *sub-systems*. We show the *dependencies* (**`dep`**) between the *system* and its *sub-systems* by *dep*-statements.

SUB-SYSTEM: We represent individual *sub-systems* by *agents* enacting *roles*, specifying the *sub-objectives* of the system that they address. We also make explicit the dependency between the *system* and its *sub-systems* by *dep*-statements.

ENVIRONMENT: We represent the parts of the *environment* that we assume interact with the *system* by *agents*. An **`environmental agent`** may enact a *role*, meaning that the *system* can expect it to carry out a certain *objective*, but this is not required.

ENTITIES: Inspired by the agent notion as considered in [Wolldr95], we represent entities in general by **`agents`** and **`roles`**

- **`human`** vs **`artificial`** entities – no distinction; an *agent* can represent a *human* and implement a *human behavior model* but from a model perspective, *human* and *artificial agents* are indistinguishable;
- **`passive`** vs **`autonomous`** entities; *agents* that implement *proactive behavior* models are inherently capable of *autonomous behavior*, whereas *agents* that

implement *reactive behavior* models are suited for representing *passive* entities;
- **`sensing`** vs **`actuator`** entities; we have *roles* dedicated to *sensing* and *actuator* such that we represent these entities by *agents* enacting those *roles*.

ROLE and ACTOR-ROLE: We represent *roles* and their designated *objectives* by *role-statements*, and the *entity-role combination* by *rea-statements*, stating that an *agent* enacts a *role*.

ACTOR: We represent an *actor* by its designated *agent*.

USER: The *user* is a specific *agent* that plays the *role* of *user*. For *context-aware systems*, the system *agent* depends on the *user* to deliver its *service* and for gaining *feedback*. The *user role* may have certain *objectives* but this is not required.

ACTION and INTERACTION: An *agent* is capable of performing *actions*, and in the **`OPG`**-phase, it *generates options* based on its *beliefs*. It then *deliberates the action* to take in the **`AE`**-phase. We represent the concept *interaction* by *dep*-statements, meaning that multiple *agents* are collaborating in solving an *objective*.

OBJECTIVE: We represent *objectives* primarily by *role*- and *obj*- statements.

REGULATIONS: We represent *regulations* by *condition* statements, stating *situations* where *obligations* to carry out *objectives* get activated. In the **`OC`**-phase, the *agent* updates itself for current active *obligations* and violated *obligations* with *obl*- and *viol*- statements.

VALUES: *Values* are reflected in the *regulations* expressed by *cond*-predicates.

3.3 Proposed Meta-Model

After having introduced our *concepts* (in synch with SDBC), we have derived a **`meta-model`** accordingly, considered a key contribution of the current paper. The meta-model is presented in Figure 5, using the notations of UML – Class Diagram [25].

As it is seen in the figure, we consider a **`system`** and its **`environment`**. Both are composed of numerous **`entities`** which in turn can be **`components`** (*non pro-active*) or **`agents`** (*pro-active* and *intelligent*). One *entity* (an *agent*, for example) can enact many different **`roles`** (and in this research, we limit ourselves to *four role categories*, namely: **user**, **`sensor`**, **`actuator`**, and **`processor`**) that are restricted by corresponding **`rules`** and are subject of **regulations**. A *regulation* in turn is composed of many *rules* and is affecting not only the *roles* but the *system* as a whole.

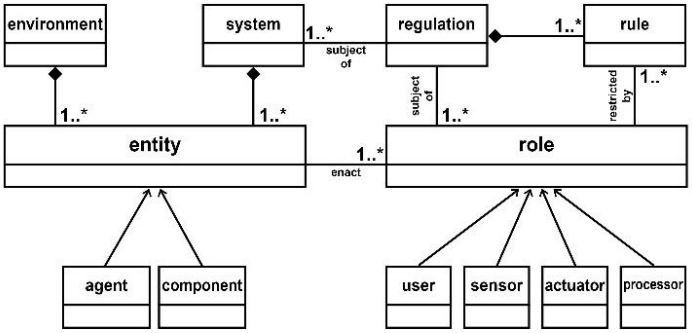

Fig. 5. Proposed meta-model

Since we are taking particularly an agent perspective, we consider it important **matching roles to their corresponding executing agents** because it is not for sure that *anybody* would have the right capabilities of fulfilling a *role*.

With regard to what an agent is capable of doing, *AORTA* takes a discrete approach to capabilities: **the capabilities of an agent are defined as the set of states that an agent can achieve**. More concretely, an agent is defined as the tuple (**α**, **MS**, **AR**, **F**, **C**, **μ**) where **α** is the *name of the agent*, **MS** is its *"mental" state*, **AR** are its *rules of reasoning*, **F** is a s*et of transitioning functions* (Action × MS → MS), **C** are its *capabilities* and **μ** is a *"mailbox"* for incoming messages from and outgoing messages to other agents.

When considering a role enactment, an *agent checks whether its capabilities overlap the objectives of that role* (if not, an *enactment* would be impossible). One can *split a role into two roles* where one defines what objectives are expected of an agent enacting the "partial" role and what is expected of an agent enacting the "full" role.

As mentioned in the Introduction, we propose at the end of this section *refined guidelines* (inspired by SDBC and the meta-model).

3.4 Refined Design Guidelines

In following SDBC (see Figure 3) and implementing the meta-model (see Figure 5), we propose refined design guidelines, and we limit ourselves only to considering the *strengths of agent technology with regard to the three categories of context-aware systems addressed* in the current paper.

The *three system behavior perspectives* (*SMCAS*, *UDCAS*, and *VSCAS* – see Section 2) considered in combination, referring to the *meta-model*, inspires a **proposed design vision**; according to it, a *system* uses an **AORTA Engine** considered for doing all three things simultaneously, as it is illustrated in Figure 6.

As the figure suggests, the **user** is situated in the **environment** and the **system** uses **sensors** to receive info from the *environment*, including the *user*, and **actuators** to manage **sub-systems**, realizing *internal optimizations*. The **AORTA Engine** allows the *system* to conform to **rules** and **regulations** (*regulations* are encoded, reflecting the *values* the *system* should consider). The *AORTA Engine* checks the input from the **Analysis Engine** to identify *obligated*, *forbidden* or *allowed actions*, and what *organizational influence* those actions have (enacting a *role* involves notifying other *entities*). The output is provided as input to the **Planner** that in turn decides the *action* to be taken

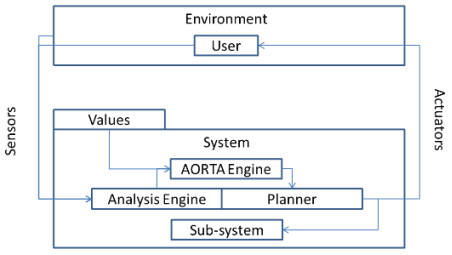

Fig. 6. A system combining features of SMCAS, UDCAS, and VSCAS

4 The Border Security Case

Unmanned aircraft (for example, **drones**) are a way to facilitate the surveillance along land borders, as according to a case example considered in [21].

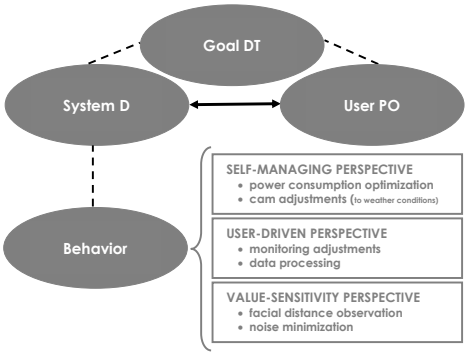

Fig. 7. Exemplifying the SMCAS, UDCAS, and VSCAS perspectives

Drones are capable of carrying and running infrared (and regular) surveillance cameras and based on their input, scarce capacity can be allocated. The goal is to find illegal activities. Despite drones' advantages, there can be some unintended effects: Drones can make noise and scare ordinary people who are passing the borders in a legitimate way, often at check-points; custom officials and/or police officers might be scared and/or disgraced by drones. Next to that, (video) recording activities might be violating the privacy of people. Hence, **it should be avoided that**: (i) drones come too close to legitimate activities; (ii) data that is not needed is recorded and shared. This asks for balancing between the *internal objectives of finding illegal activities* and the *societal demands* that are two-fold: it is needed *to avoid drones getting too close to people* and it is also needed *to ensure their privacy*.

We hence make the following explicit: (A) We do address **SYSTEM D** – *a drone system*, featuring a drone flying over a border and supporting border police officers; (B) *System D* is delivering services to **USER PO** – *the border police officers* who are patrolling the border; (C) The goal in context is: **GOAL DT**, featuring the *detection of illegal activities* in general (trespassers, in particular). This is reflected in Figure 7.

As also seen from the figure, we have identified six case-specific behavior lines, two per each of the system behavior perspectives:

[POWER CONSUMPTION OPTIMIZATION] [SMCAS PERSPECTIVE]: When a drone is flying, it is exchanging information with the ground station – the drone is sending sensor data to the station that is in turn generating instructions. Hence, it is at the ground station where the flight is controlled such that it is guaranteed that the drone mission is completed. This requires that the power consumption is optimized, such that the drone has enough power to complete the mission and get back.

[CAM ADJUSTMENTS] [SMCAS PERSPECTIVE]: When a drone is flying, it counts on its cameras, such that it is capable of adjusting its flight accordingly. Hence, those cameras need to be adjusted when weather conditions change.

[MONITORING ADJUSTMENTS] [UDCAS PERSPECTIVE]: When a drone is flying, it is realizing its mission to support border police officers (the "user") and for this, it should adjust its monitoring to mainly cover those areas that are not close to where police officers

are.
[DATA PROCESSING] [USCAS PERSPECTIVE]: When a drone is flying, raw sensor data is processed by the drone itself and/or by the ground station, such that higher-level context information is derived and delivered to border police officers.
[FACIAL DISTANCE OBSERVATION] [VSCAS PERSPECTIVE]: When a drone is flying, it is to act in a privacy-sensitive way, which means that without explicit instructions, the drone should not (video) capture people's facial information or if this happens, the photo/video material should be blurred accordingly.
[NOISE MINIMIZATION] [VSCAS PERSPECTIVE]: When a drone is flying, it should avoid noise-polluting residential environments, which means that if this wouldn't be mission-critical, the drone should avoid approaching residential areas.

Acting adequately with regard to all those perspectives is considered important because it wouldn't be acceptable neither "sacrificing" the drone, nor compromising the mission, nor disregarding values. Nevertheless, it is sometimes impossible to satisfy all this. For example: If from a SMCAS perspective, the drone should immediately turn back (with indications of insufficient power to go on) but from a UDCAS perspective, the drone should go on to approach an area of interest, then what is the solution? Resolving such TENSIONS is considered non-trivial because we argue that "universal rules" cannot work in cases like this. Thus, it is a matter of *sophisticated prioritization* to come to the "right" solution. If, for example, a border control mission is on and a noise pollution is observed, then if the mission is just a routine surveillance, probably the noise-pollution should be avoided but if the mission is targeting trespassers, then the neighborhood silence may be sacrificed and for sure the residents would understand and appreciate such actions. For this reason, the AORTA drone mission modeling is to be complemented by a "prioritization scheme" which is not shown in the current section, for the sake of brevity.

What we are demonstrating is the OVERALL model (simplified to what Figure 7 suggests) where all three perspectives are "super-imposed" (such that *tensions* could be straightforwardly identified) – this is presented in Table 1.

Table 1: Exemplifying the overall behavior, by applying AORTA.

role(*drone, {reportTrespassing(Trespasser, Location, Time), optimizeVision(Camera), optimizePower}*)
role(*officer,{apprehend(Trespasser)}*)
obj(*reportTrespassing(Trespasser, Location, Time), {patrol(Location,Time), record(Trespasser, Location, Time)}*)
dep(*drone, officer, reportToOfficer)}*)
dep(*drone, camera, optimizeVision)}*)
cond(*camera, lightSensing, optimizeVision, clearWeather /\ daytime*)
cond(*camera, darknessSensing, optimizeVision, cloudyWeather*)
cond(*drone, keepDistance, record, recordingFace*)
cond(*drone, reduceNoise, patrol, ¬alarm*)

5 Related Work

AORTA is based upon the *OperA* meta-model for *agent organizations*. *MOISE+* [26] is another meta-model for *agent organizations* that is well known in *AI* (*Artificial Intelligence*)-communities, and which is implemented in the agent-programming platform *JaCaMo* [2]. With regard to *agent-based AI systems*: "*Pepper*" is a *robot* working together with *humans* in a *socially acceptable way*. *Pepper* perceives its environment through *visual*

sensors and can also receive *input through a touch screen*. *Pepper* is able to *make simple gestures*, *speak*, and *move* around on wheels. *Pepper*'s behavior is programmed using the agent-based language *GOAL* [9] in which an *agent* is specified in terms of a *mental state* with *personal info*. In contrast, we propose using *AORTA* in which an *agent maintains personal information and organizational information separately*. This relates to *user-driven context-aware systems* where for example *Body-Area Networks* are implemented to *capture vital signs* from the body of a patient from distance, such that this information is processed and communicated in an intelligent way [1] while *self-managing systems* were widely used in *home appliances* that adjust their operation with regard to some *goal* that concerns a unit, such as neighborhood [24]. This all concerns *values* that are *abstract* and *non-functional* and that is how they are considered to date [7], not aligning them adequately with the processes that concern the design of software. We believe that the current work is a step forward in approaching methodologically *context-aware systems*, in general and particularly – the three categories of context-aware systems addressed in this paper.

6 Conclusions

Furthering previous research that touches upon the enterprise-modeling-driven software specification, we have particularly addressed context-aware (software) systems in the current paper. We have identified and studied three categories of context-aware systems, featuring context-based system behavior adaptations that concern the optimization of system-internal processes, the maximization of the user-perceived effectiveness, and the consideration of relevant public values. Super-imposing those three system behavior perspectives and in synch with previous work (the SDBC approach), we have identified concepts and we have enriched them from an agent-technology perspective (in line with the AORTA framework), reflecting this in a meta-model that is considered a major contribution of the paper. Inspired by the meta-model, we have provided a partial refinement of the SDBC design guidelines, taking an agent technology perspective and focusing on the three above-mentioned system behavior perspectives, and we have partially illustrated this by means of a case example featuring land border security. As future work we plan to CONSOLIDATE our SDBC-AORTA-driven proposal into one dedicated design approach that is specific to context-aware systems.

Acknowledgements

This work is supported by: (i) the *TU Delft - Delft Pilot project*; (ii) *Technical University of Denmark* and the *PDC A/S project*. We would like to thank *Jeroen van den Hoven* for his support and guidance.

References

1. AWARENESS, 2008. Freeband AWARENESS Project. http://www.freeband.nl.
2. Boissier O., Bordini R.H., Hübner J.F., Ricci A., Santi A. (2013) Multi-Agent Oriented Programming with JaCaMo. Sci. Comput. Program. 78, 6.
3. Brun Y. et al. (2009) Engineering Self-Adaptive Systems through Feedback Loops. In: Cheng B.H.C., de Lemos R., Giese H., Inverardi P., Magee J. (eds) Software Engineering for Self-Adaptive Systems. Lecture Notes in Computer Science, vol 5525. Springer, Berlin, Heidelberg.

4. Cockburn, A., 2000. Writing Effective Use Cases, Addison-Wesley.
5. Dey A.K. (2001) Understanding and Using Context. In: Personal and Ubiquitous Computing, 5(1), 4-7.
6. Dignum v. (2004) A Model for Organizational Interaction: Based on Agents, Founded in Logic. PhD Thesis, Utrecht University.
7. Friedman B., Hendry D.G., Borning A. (2017) A Survey of Value Sensitive Design Methods. In A Survey of Value Sensitive Design Methods , 1, Now Foundations and Trends, 2017, pp.76-
8. Google Dictionary, 2018. The website of Google Dictionary: http://www.google.com.
9. Hindriks, K.V. (2009) Programming Rational Agents in GOAL. In El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H. (eds) Multi-Agent Programming: Languages, Tools and Applications, Springer, 119–157.
10. Huebscher M.C., McCann J.A. (2008) A Survey of Autonomic Computing - Degrees, Models, and Applications. ACM Comput. Surv. 40, 3, Article 7.
11. Jensen, A.S., Dignum, V., Villadsen, J (2017) A Framework for Organization-Aware Agents. Auton. Agent. Multi-Agent Syst. 31(3), 387–422.
12. Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. Computer, 36(1), 41-50.
13. Kruchten, P., 2003. The Rational Unified Process, An Introduction, Addison-Wesley.
14. Mahdavi-Hezavehi S., Avgeriou P., Weyns D. (2016) A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems With Multiple Quality Requirements. In: Mistrik I., Ali N., Kazman R., Grundy J., Schmerl B. (eds) Managing Trade-offs in Adaptable Software Architectures, 1st Edition. Elseiver Inc.
15. MDA, 2018. The OMG Model Driven Architecture. http://www.omg.org/mda.
16. Muehl G., Werner M., Jaeger M.A., Herrmann K., Parzyjegla H. (2007) On the Definitions of Self-Managing and Self-Organizing Systems. Communication in Distributed Systems - 15. ITG/GI Symposium, Bern, Switzerland, 2007, pp. 1-11.
17. Shishkov, B., 2017. Enterprise Information Systems, A Modeling Approach, IICREST. Sofia.
18. Shishkov B., Janssen M. (2018) Enforcing Context-Awareness and Privacy-by-Design in the Specification of Information Systems. In: Shishkov B. (eds) Business Modeling and Software Design. BMSD 2017. Lecture Notes in Business Information Processing, vol 309. Springer, Cham.
19. Shishkov B., Mendling J. (2018) Business Process Variability and Public Values. In: Shishkov B. (ed) Proc. of BMSD 2018 - 8th Int. Symposium on Business Modeling and Software Design. Lecture Notes in Business Information Processing, vol 319. Springer, Cham.
20. Shishkov B., Janssen M., Yin Y. (2017) Towards Context-Aware and Privacy-Sensitive Systems. In BMSD'17, 7th International Symposium on Business Modeling and Software Design. SCITEPRESS
21. Shishkov B., Mitrakos D. (2016) Towards Context-Aware Border Security Control. In BMSD'16, 6th International Symposium on Business Modeling and Software Design. SCITEPRESS
22. Shishkov B., van Sinderen M. (2008) From User Context States to Context-Aware Applications. In: Filipe J., Cordeiro J., Cardoso J. (eds) Enterprise Information Systems. ICEIS 2007. Lecture Notes in Business Information Processing, vol 12. Springer, Berlin, Heidelberg.
23. 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.
24. Shishkov B., Warnier M., Van Sinderen M. (2010) On the Application of Autonomic and Context-aware Computing to Support Home Energy Management. In Proc. ICEIS 2010 - the 12th Int. Conference on Enterprise Information Systems; 8-12 June 2010; Funchal, PT. SCITEPRESS, Setúbal
25. UML, 2018. The Unified Modeling Language. http://www.uml.org.

26. Van Riemsdijk M.B., Hindriks K., Jonker C. (2009) Programming Organization-Aware Agents. In: Aldewereld H., Dignum V., Picard G. (eds) Engineering Societies in the Agents World X. ESAW 2009. Lecture Notes in Computer Science, vol 5881. Springer, Berlin, Heidelberg
27. Wooldridge M., Jennings N.R. (1995) Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, vol. 10, no. 2. pp. 115–152.