From User Context States to Context-Aware Applications

Boris Shishkov and Marten van Sinderen

University of Twente, Department of Computer Science, Enschede, The Netherlands
{b.b.shishkov,m.j.vansinderen}@ewi.utwente.nl

Abstract. In many cases, in order to be effective, software applications need to allow sensitivity to user context state changes. This implies however additional complexity associated with the need for applications' adaptability (being capable of capturing context, interpreting it and reacting on it). Hence, we envision 3 'musts' that, in combination, are especially relevant to the design of context-aware applications: (i) At the business level, the different possible context states of the user must be properly identified and modeled; (ii) Both at the business level and application level, the corresponding desirable behaviors must be identified and modeled, as well as the overall behavior which represents the required adaptability in terms of valid switches between desirable behaviors; (iii) The models at the business level and application level must be aligned, i.e. the application models should represent proper solutions with respect to functionality and adaptability needs expressed at the business level. In this work, we address the mentioned challenges, by furthering the development of a business-application-alignment approach, extending it to cover context-awareness. We illustrate our achieved results by means of a small example. It is expected that this research contribution will be relevant and useful with respect to the challenge of aligning business modeling and software design.

Keywords: Business modeling, Application modeling, Context-aware applications, Context states, SOA, MDA, LAP.

1 Introduction

In developing a software application, the designer should take into account not only the user requirements but also the characteristics of the environment or situation in which the user will interact with the application [13, 14]. This sometimes leads to the identification of different possible states – referred to as (user) *context states*, where by *context* is meant 'the interrelated conditions in which something exists' [7]. For example, possible user context states could be "user is at home", "user is travelling", and "user is at work". Hence, sensitivity to context changes is sometimes essential for the effectiveness of applications, usually referred to as *Context-Aware* (CA) *applications* [10,16]. It should be decided therefore which are the relevant context states to be considered by the application. Further, the application should be capable of deriving the context states and performing the desirable behaviors corresponding to these states. Deriving context states involves sensing the user environment and transforming the sensed raw data into context information which is useful to the

J. Filipe, J. Cordeiro, and J. Cardoso (Eds.): ICEIS 2007, LNBIP 12, pp. 225–239, 2008.

application. Such context information would allow the application to recognize a context state change and react on this by switching to the corresponding desirable behavior. This capability of an application does constitute a quality known as *adaptability*.

All this implies complex design. We envision *3 'musts'* that, in combination, are especially relevant to the design of CA applications:

(i) At the business level, the different possible context states of the user must be properly identified and modeled;
(ii) Both at the business level and application level, the corresponding desirable behaviors must be identified and modeled, as well as the overall behavior which represents the required adaptability in terms of valid switches between desirable behaviors;
(iii) The models at the business level and application level must be aligned, i.e. the application models should represent proper solutions with respect to functionality and adaptability needs expressed at the business level.

By *business modeling* we mean the modeling of business-level *entities* as well as their corresponding *relations* and *behaviors*. The desirable application behaviors must (logically) be appropriate refinements of the business-level behaviors. Business modeling and *application modeling* are considered to address different levels of abstraction, and thus form separate design activities, each one focusing on the concerns appropriate to the level at hand. These activities could be carried out in any order, or could be (partially) done in parallel; however the results of these modeling activities should be consistent according to the required business-application alignment. For example, one could model firstly entities and behaviors at the business level, which concern a computation- and technology-independent 'view' on business processes, and secondly, one could model entities and behaviors at the application level, which concern application functionality views that are inevitably computation-dependent and to some extent technology platform independent (and technology-rooted at the same time). Bridging the gap between these abstraction levels and achieving an adequate business-application alignment is partially considered in this paper and more thoroughly approached in previously reported work [14].

The desirable context-awareness and context-driven adaptability of applications imply, as mentioned before, the necessity for adequate capturing and processing of context information, and reaction on context changes. Although an application would be required to react on context changes at real time, those changes should be foreseen at design time, so that proper desirable application behaviors are prescribed.

This *design preparation* is the main focus of the current paper. In particular, we further the development of a business-application-alignment approach [13], by extending it to cover context-awareness. This relates to another issue addressed in the paper, namely *consistency* - claimed to be important in the business-application alignment [12, 13]. Consistency is a desired relationship between models that address separate concerns, for instance business and application concerns [1]. We illustrate our achieved results, by means of a small example.

We adopt *service-orientation* [1, 8] as a preferred architectural style for organizing systems (to be reflected in our models) - this decision has been motivated and inspired by previously achieved results [14]. This implies that we aim at system structures

where functionality is only accessible through services, hiding how each of the services is implemented, and where services can be published and discovered through corresponding service descriptions. Service-orientation helps to speed up the development of business-aligned application models (through the composition of services), and also to flexibly utilize advanced technological platforms for their implementation (through the independence of services from implementations) [15].

The paper's outline is as follows. Section 2 motivates further our proposed design views and also introduces the concepts/theories and methods that we use. Section 3 introduces a case study that is used in the next sections to detail and illustrate the different phases of our approach. Section 4 and Section 5 present the business and application modeling activities, respectively. Finally, Section 6 contains the conclusions and outlook for future work.

2 Modeling Approach

During business and application modeling, the notions of business/software *system* and *environment* [2] should be explicitly considered. A system and its environment are both composed of entities which could fulfill different *roles*. Entities are constrained in their behavior, corresponding to the roles they fulfill [11]. A system integrates entities' behaviors, which results in an overall external behavior (or service) provided to the system's environment; the entities in the system's environment that interact with the system (in accordance to the service) are often referred to as service users or *users*, for short.

A service provisioning needs to appear sometimes in different 'versions', depending on the user context state. Said otherwise, for one user context state, the system should provide one type of external behavior to the user while for another state, another behavior is to be provided. Hence, context state changes trigger changes in the system behavior [6], including changes in the number and composition of entities involved in the service provisioning. Based on these basic considerations concerning the design of CA applications, we identify a number of challenges. Among them are:

- The application should be able to *sense* context and capture this context as context information;
- The application should be able to *interpret* the captured context information and derive higher-level context information, in particular – user context state changes, as triggers to alternative behaviors;
- The application should be able to handle the *switching* between its alternative behaviors;
- The application should be able to *provide* services covering all possible context states.

As illustrated in Figure 1, a CA application can be seen as concerning a sequence of 'actions' that achieve: **S** (sensing and capturing), **I** (interpretation and state derivation), **W** (switching), and **P** (provisioning), respectively, as explained above. This is obviously a simplified model, since each of the actions represents a potentially complex process, and the dependencies between these normally involve multiple instances of information exchange or triggering.

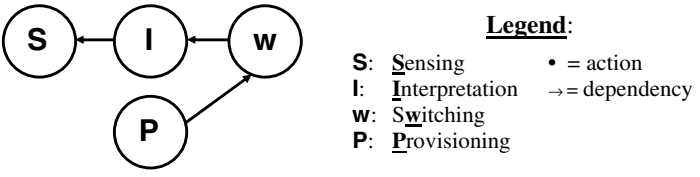

Fig. 1. Simplified view on a CA application

In the following, we largely ignore Sensing (supported by sensors and context sources, for example) and Interpretation (supported by aggregation and inference, for example) – they are addressed in related work [16]. Further, we pay little attention to Switching between alternative application behaviors; this is positioned as future research.

Hence we focus here on the modeling of alternative desirable behaviors (as needed by the user in corresponding context states) and their consequent realization by an application. We face thus the gap between domain-driven requirements on the application service, or its alternative behaviors, on one hand, and the technology-rooted application realization of this service, on the other hand. In order to properly address this, we need to consider different aspects of consistency:

- Correspondence between user context states and the business model (concerning the desirable – business level – application service and how this affects business entities);
- Consistency between the business model and the application model (concerning the application realization in terms of – application level – services to be assigned to application entities);
- Consistency between dynamic aspects (behavior) and corresponding static (entity) aspects of the business/application models.

Figure 2 illustrates these consistency aspects (designated by dashed lines).

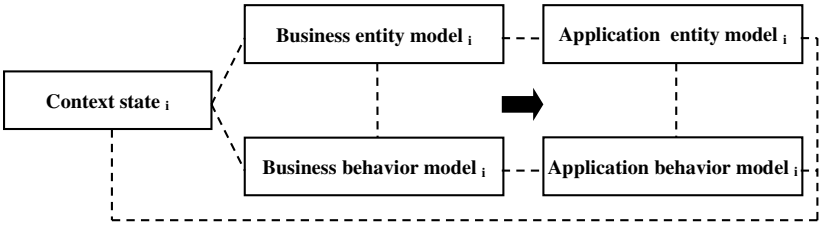

Fig. 2. Consistency aspects in the application design process

As shown in the figure, the models considered in the application design process depend on context states (i.e., different states correspond to different behaviors and possibly different entities involved). Two aspect models are considered, namely entity and behavior models. Furthermore, models are refined in the design process, starting with abstract business level models and ending up with more concrete application

models (as suggested by the black arrow in Figure 2), through gradually increasing consideration of computational and technology platform concerns. Two fundamental modeling phases and milestones are distinguished, namely the business modeling phase, which leads to a business model, and the application modeling phase, which leads to an application model.

We model a behavior as a set of related *events*; each event corresponds to a unit of behavior, which is indivisible at the abstraction level at which it is defined. We distinguish two types of events, viz. *action* (performed by a single entity) and *interaction* (performed by two or more entities, in cooperation). An interaction is expressed as two or more connected interaction contributions that represent the participation of the involved entities.

Our modeling approach adopts the abstractions introduced by the *Model Driven Architecture – MDA* [3, 9], by considering: (i) business modeling from a computation-independent perspective (no decisions are made with respect to the - complete or partial - automation of business processes), and (ii) application modeling from a technology platform independent perspective (even though the applications are technology-rooted, no decisions are made with respect to the specific technological platform on which the application is implemented). The consideration of such specific technological platforms is left beyond the scope of this paper; for a discussion on web-services-based technology solutions, readers are referred to [8].

Further, the mentioned adoption of service-orientation, affects our modeling in a way that we are mainly interested in external behaviors (services) [17]. We hence could arrive at a service model from two directions: either by identifying services from business level requirements or by abstracting from available technology solutions. We claim that both directions are possible; nevertheless, the former is probably always needed. This is the case since a business-requirements-driven service model would possess the right restrictions, whose fulfillment (in application design) guarantees that the application is not only feasible from a technical point of view but also useful from the business (user) point of view.

With respect to the modeling of real-life-level business requirements, we consider a theoretically-rooted approach, namely the *Language-Action Perspective – LAP* [12], possessing strengths in modeling real-life interactions. LAP distinguishes between two types of activities - *production acts* and *coordination acts*, and two types of roles that an entity could fulfill - *initiator* and *executor*. The initiator initiates an interaction and the executor delivers the required *production fact*. This is accompanied however by coordination acts which could be *request*, *promise*, *state*, *accept*, and *decline*, and which together with the production act form a *generic interaction* (GI) *pattern* that concerns real-life communication/coordination [2, 11, 12]. Complex interactions can in most cases be represented in terms of such patterns.

The GI pattern specifies that the initiator initiates an interaction, by making a request which could be either taken or declined by the executor. If taken, it should be fulfilled by the executor, by performing a production act and delivering the corresponding production fact. If the executor has declined the request, he and the initiator enter a negotiation. A negative negotiation result leads to interaction's failure; if the result is positive, i.e. they find a compromise, the executor must make commitment of delivering the 'updated' desired result. As for the production act, it is responsibility of the executor. This act however does not mark the interaction's

completion; a result delivery is subject to announcement (explicit or implicit) by the executor. The result is to be 'evaluated' by the initiator who may accept it (interaction completed) or not (interaction not completed and negotiation starts). A negative negotiation result leads to interaction's failure; if a compromise is found then the interaction is to reach completion.

3 The Health-Care Scenario

We will describe and illustrate (in Section 4 and Section 5) the different modeling phases, supported by a *health-care scenario* (outlined below), inspired by a broader case that has been studied in [16].

In the scenario, we consider patients who are suffering from conditions that are characterized by occasional occurrences of undesired effects; an example of this is epilepsy. For this reason, such patients need help from caregivers each time when symptoms occur.

We distinguish two situations: *Situation 1* – the traditional institutional-care situation, and *Situation 2* – the situation in which patients are no longer bound to an institution like a hospital, but receive mobile care through monitoring and/or treatment realized from distance, using advanced technology.

Situation 1. In approaching the traditional institutional-care situation, we identify the role of *Caregiver* (fulfilled by a medical doctor or a medical nurse) who provides help to patients. In this help provisioning, the Caregiver receives support from medical workers who fulfill the following roles: *Triager* (the allocator of treatment to patients), *Trend Synthesizer* (the first checker of the patient's condition), *Processor* (the examiner of the patient's symptoms), *Analyst* (the patient history analyzer), and *Advisor* (the rules-supported generator of advice to the Caregiver). Furthermore, we distinguish between two possible states that are relevant to this care provisioning, namely: *State 1* (*'not too busy'*) - some doctors are immediately available to provide help, and *State 2* (*'very busy'*) - all doctors are occupied or have scheduled appointments (within half an hour, for example). In **State 1**, the Caregiver (in particular, a doctor) helps a patient if the patient had been directed by the Triager. In order to give a proper direction to the patient, the Triager must have received input from the Trend Synthesizer who in turn must have checked (beforehand) the patient's condition, for which the Trend Synthesizer needs two inputs, one coming from the Processor and another one – coming from the Analyst. The Processor provides information resulting from a conducted examination of the patient's symptoms (for example, a consideration of vital signs, such as blood pressure and blood sugar). The Analyst delivers conclusions derived from the medical history of the patient. In **State 2**, it is desirable (if possible) to minimize the work directed to doctors and to replace them (in some cases) by nurses (such a replacement could happen nevertheless only if the patient had been directed by the Triager and the Advisor had provided sufficient instructions that allow the nurse to give adequate care). As for the delivery of instructions, the Advisor needs input from the Triager who in turn needs input similar to the one concerning State 1.

Situation 2. In approaching the technology-facilitation-driven situation, we identify the same roles and interactions as described in Situation 1, and they are involved in

the same scenario. The difference however is that those who fulfill the roles of Triager, Trend Synthesizer, Processor, Analyst, and Advisor, are not human beings, they are components belonging to a distributed software application; it runs on a number of devices, supporting the doctors and nurses in their help provisioning.

Section 4 considers (CA) business modeling that is relevant to Situation 1. Section 5 outlines, based on this, the specification of an application that could run on (advanced) devices, adequately fulfilling the corresponding requirements, as suggested in the above paragraph that concerns Situation 2.

4 Business Modeling

In achieving the first modeling milestone (as according to Section 2) we come through the following 3 *sub-phases*.

The *Context analysis sub-phase*, approaching the possible context states and corresponding desirable behaviors, includes: (i) study of the possible context states and their occurrence probabilities; (ii) discovery of useful context parameters whose values indicate the occurrence of particular states.

The *Structural (static) modeling sub-phase* includes the identification of: (i) business system(s) relevant to each desirable behavior; (ii) relevant entities belonging to the system/environment - for each of the system 'versions'; (iii) relations between entities, representing interaction abilities that concern only two-entity interactions (see Section 2) - for each of the system 'versions'; (iv) the entities' Initiator/Executor roles in the relations - for each of the system 'versions'; (v) proper rules that define the 'switch' between different desirable behaviors. All this builds up a Business entity model.

The *Behavior modeling and Service identification sub-phase* concerns the modeling of entities' integrated interaction behavior, abstracting from interaction contributions. Being concerned with different levels of abstraction and elaboration, the modeling evolves as follows: (i) the system's external behavior is firstly modeled, considering the system as a 'black box'; (ii) the system's internal behavior is disclosed on this basis (relevant interactions are modeled as well as the way the interactions relate to each other); (iii) units of composite behaviors are identified by grouping interactions (putting together the coordination acts, following the GI pattern), arriving therefore at a service model. For more elaborations on these steps and on the related conformance justification, readers are referred to [13, 14].

4.1 Context Analysis Sub-phase

As mentioned already, the context analysis should come through an occurrence probability study as well as a parameter-value study.

Occurrence probabilities. Deciding about states, the designer is sometimes inevitably driven by subjective judgments that are hardly supportable by rules: How a situation is perceived? What behaviors can be expected? Further, the designer must often make pragmatic decisions – ignoring, for example, states that usually do not occur (although they might occur). In our view, besides such subjective decisions, there are steps which in general help to adequately approach the context analysis

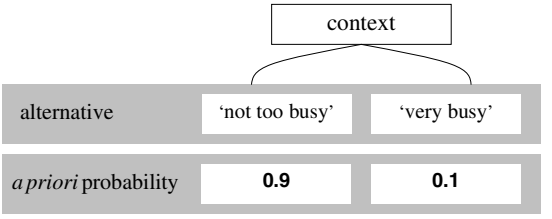

Fig. 3. Two context-state alternatives

challenge. These steps concern the consideration of *random variables*. Exploring their probabilities, allows us to apply *statistical analysis*, including *hypotheses testing* and *parameters estimation* [4].

Considering just possible outcomes is sometimes not enough in approaching a phenomenon; we might need to refer to an outcome in general. This is possible if we have a random variable and we study the occurrence probability of the outcomes.

As concerns the Health-Care Scenario, we have there exactly two possible states, namely: 'not too busy' and 'very busy'. We consider the random variable **Y** with respect to these outcomes. **Y** would be a *discrete random variable* [4] since it may take on only a countable number of distinct values (in our case two). Provided the number of possible distinct values is exactly two, we have the case of *a priori probabilities* of each of the alternative outcomes (one of these probabilities can be calculated by deducting the other one from **1**).

According to a conducted study, whose details are omitted for brevity, the *a priori* probability of the first of the mentioned possible outcomes is **0.9**. The *a priori* probability of the second alternative outcome is therefore **0.1**.

Hence, our context states represent the 'not too busy' and 'very busy' alternatives, with *a priori* probabilities **0.9** and **0.1**, respectively, as illustrated in Figure 3.

Knowing the *occurrence probability* of each outcome helps in deciding which to be the 'default' desirable external behavior and also what could be ignored (if anything).

Parameters and Values. In order to prescribe how to recognize each of the states (two in our case), we assume that the state at a particular moment is recognizable through observing the values of appropriate *parameters*. If we have **n** parameters appropriate to our scenario and if each of them has certain possible *values*, then each values combination would point to a particular state.

For brevity, we exemplify with just two parameters, namely $\mathbf{p}_1$ and $\mathbf{p}_2$:

- $\mathbf{p}_1$ is about the ratio between the number of patients and the number of doctors at a moment, and is with just three possible values: $\mathbf{v}_{11}$ (the number is less than **1**), $\mathbf{v}_{12}$ (it is exactly **1**), and $\mathbf{v}_{13}$ (it is more than **1**);
- $\mathbf{p}_2$ concerns the particular moment – *normal* or *not* ('not' would be during night-time, for example), and has just two possible values, respectively for 'normal' and 'not' (not normal), namely $\mathbf{v}_{21}$ and $\mathbf{v}_{22}$.

There are six possible value ($\mathbf{p}_1$,$\mathbf{p}_2$) combinations, namely $\mathbf{v}_{11}$.$\mathbf{v}_{21}$, $\mathbf{v}_{11}$.$\mathbf{v}_{22}$, $\mathbf{v}_{12}$.$\mathbf{v}_{21}$, $\mathbf{v}_{12}$.$\mathbf{v}_{22}$, $\mathbf{v}_{13}$.$\mathbf{v}_{21}$ and $\mathbf{v}_{13}$.$\mathbf{v}_{22}$. Driven by additional domain analysis, omitted here for

	parameters' values' combinations
first alternative	$v_{11}.v_{21}$, $v_{11}.v_{22}$, $v_{12}.v_{21}$, $v_{12}.v_{22}$, $v_{13}.v_{21}$
second alternative	$v_{13}.v_{22}$

Fig. 4. Context state recognition

brevity, we determine that only the last combination is validly corresponding to the **0.1**-*probability* alternative (the 'Second' alternative); thus all the rest of the combinations correspond to the **0.9**-*probability* alternative (the 'First' alternative), as depicted in Figure 4.

Hence, knowing the values of the two parameters (the values could be captured using sensors for example), one could actually 'sense' the context state at a particular moment.

4.2 Structural Modeling Sub-phase

We omit the *business-entity-model*-derivation steps concerning each of the two desirable behaviors (the ones corresponding to the 'First alternative' state and the 'Second alternative' state) as well as decisions on which are the relevant entities and how they are related to each other. We omit all this not only because the *SDBC* approach is exhaustive about it, possessing capabilities to transform unstructured case information into a business model [11, 12], but also because the consideration of such early-business-analysis-related issues would actually shift the focus from the business-application alignment (addressed in this paper as main challenge).

Hence, we 'arrive' directly at the Business entity model for the *Health-Care* (**HC**) *case* (Figure 5); the model is expressed using a diagramming technique, inspired by *DEMO* [11]. The identified entities are presented in named boxes – these are *Caregiver* (**C**; *D/N* – fulfilled by a doctor/nurse), *Triager* (**T**), *Trend Synthesizer* (**TS**), *Processor* (**P**), *Analyst* (**A**), and *Advisor* (**Adv**), while the small grey boxes, on one end of each connection, indicate the executor role of the connected entities. The lines that connect entities, indicate the need for interactions between those entities, in order

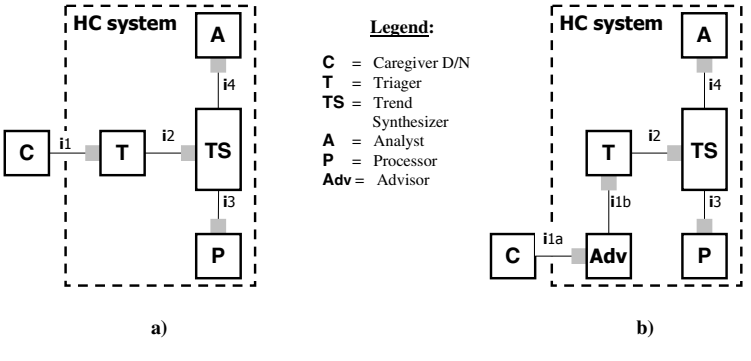

Fig. 5. Business entity model for the **HC** case

to achieve the objective of delivering a health-care service; with each such 'connection' we associate a single interaction, as follows: **C**(*CaregiverD*)-**T** (**i**1), **T**-**TS** (**i**2), **TS**-**P** (**i**3), and **TS**-**A** (**i**4). As for the delimitation, **C** is positioned in the environment of the health-care (**HC**) system, and **T**, **TS**, **P**, and **A** together form the system. Through **i**1, the **HC** system is related to its environment (represented by **C**). Thus, from the perspective of **C**, there is no difference between the system and **T**. All this concerns the 'First alternative' state, as depicted in the left part of the figure, labeled 'a)'.

In the 'Second alternative' state (the 'b)' model), an Advisor (**Adv**) is envisioned 'between' **C** (*CaregiverN*) and **T** (interaction **i**1 is replaced by two interactions, namely **i**1*a* and **i**1*b*).

For brevity, we will consider further only the 'First-alternative' state model since it represents a sufficient base for us to discuss the business-application alignment. As for modeling a transition from one desirable behavior (corresponding to a state) to another, this can be done (in our view) using Semiotic norms [5], and is positioned as future research (see Section 2).

4.3 Behavioral Modeling and Service Identification Sub-phase

We decide firstly on the external behavior of the **HC** system, at a high level of abstraction, and then we move to the abstraction level that concerns the internal behavior of **HC**.

With respect to the external behavior model, it should envision the interaction between the *Caregiver* (**C**) and the system (**HC**), and is represented by a single action (expressed by an oval) in Figure 6-a).

Regarding the internal behavior model, it should reflect the interactions between the entities of the system, as exhibited in Figure 6-b). This model shows how the interaction **i**1 (between the *CaregiverD* **C** and the *Triager* **T**) is made dependent on other interactions (**i**2, **i**3 and **i**4). The black box indicates that the results of both **i**3 and **i**4 are necessary for the triggering of **i**2. Such models can be extended further (e.g., with attributes) and interested readers could find more on this issue in [13].

We need to further elaborate this model, in order to achieve a service specification that allows for a better 'link' to relevant real-life aspects. As already mentioned, we apply the LAP-driven *GI pattern* in enriching our behavior model. We thus consider the coordination acts *request* (**r**), *promise* (**p**), *state* (**s**), and *accept* (**a**). We also follow the interaction-interaction triggering 'mechanism' (as in the LAP theory): if the initiator of one interaction requests something and if the executor promises to

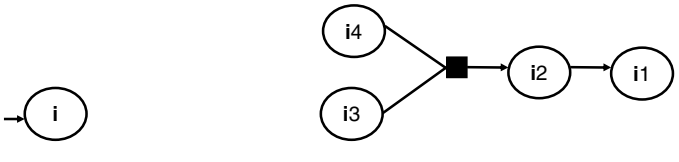

Fig. 6. a) **HC** external behavior represented by a single action; b) Interactions in decomposed **HC** system, implementing the **HC** external behavior

realize the requested production act, and if this requires another interaction's input, then in parallel with promising to realize the production act, the executor requests a production fact delivery, which actually is the triggering of another interaction. For more information on this, readers are referred to [11].

We therefore replace each interaction by its corresponding coordination acts (**r**, **p**, **s**, **a**) following the above mentioned 'mechanism'. In doing this, we group together coordination acts based on their relation to production acts (Figure 7).

We need nevertheless to model also the possible decline acts (see Section 2); we could model them (*decline-after-request* and *decline-after-state*) by special values of *information attributes* (e.g., Result r ! r = 'decline') of the *promise* and *accept* acts, respectively. Information attributes and related value constraints are not represented in the figure.

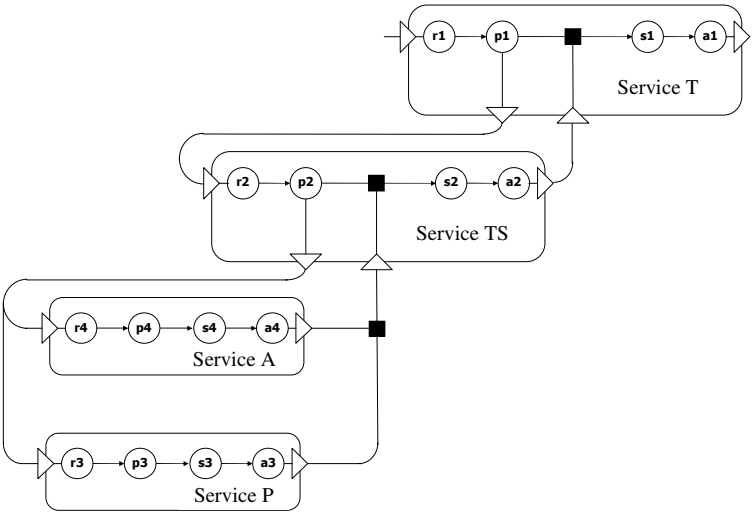

Fig. 7. Refined interactions in decomposed **HC** system, implementing the **HC**-service behavior

The model, presented in this way, *defines services* rooted in the GI pattern, consistently with our initial modeling output (Figure 5-a)).

5 Application Modeling

In achieving the second modeling milestone (as distinguished in Section 2) we come through the following 4 *sub-phases*.

The *Delimitation-requirements sub-phase* concerns the following decisions: (i) which part of the business model is addressed by the overall application service; (ii) what the user requirements are and how we are reflecting them in the application model. Decision (ii) is beyond the direct scope of this paper.

The *SOA decisions sub-phase* addresses SOA-related decisions that concern the further realization of the (distributed) application service. In particular, these are decisions concerned with the way in which re-usable services are to be addressed and

coordinated by application-specific component(s), in support of achieving the desirable application functionality.

The *Application design sub-phase* considers, on top of delimitation-requirements-related decisions and from a SOA perspective, the actual derivation of application models as proper refinements and extensions of the models from the business modeling phase.

The *Consistency analysis sub-phase* (not addressed in the current paper; addressed in [13]) envisions the desirable consistency between the original business models and the (derived) application models (such an analysis supports therefore the validation of the built application models).

5.1 Delimitation-Requirements Sub-phase

The scenario statement is not exhaustive, as the users' intended automation level or criteria are concerned, helping to make related choices (e.g., on non-functional aspects, such as cost/performance and ease-of-use). Getting the 'message' of the statement, we assume however that the whole business (**HC**) system should be automated. Thus, the **HC** business service is also the initial specification of the overall application service.

5.2 SOA Decisions Sub-phase

What is easiest-to-do is to map one-to-one between business modeling entities and application components. Such a mapping would be disadvantageous nevertheless, because the identified services (as according to the service model in the previous section) are *tightly coupled*. This means that there is a dependency of the service provided by one entity on services provided by other entities (see Figure 7). Avoiding this would be advantageous because then the flexible (re-)use of generic services would be stimulated. We claim that a solution would be to introduce 'in between' an additional overall-functionality-specific entity that has coordination tasks. We label it *'Orchestrator'*.

Hence the application component that is to realize such orchestration would be application-specific (as the coordination is application-specific). The (subordinate) services, however, which would be coordinated by this component, may be useful for many different types of applications. Their description may therefore be published through a public or corporate registry, such that they can be discovered, and selected for invocation. Related to its coordination tasks, the Orchestrator could sometimes

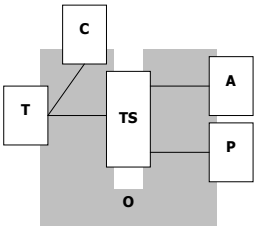

Fig. 8. Illustration of the role of the Orchestrator

supply to one service the result of another service, if this is necessary for the service to perform its task.

Figure 8 illustrates the Orchestrator's (**O**) role. It concerns the interactivities between the original entities as well as coordination. As it is seen from the figure, the Orchestrator mediates not only the interaction between the 'customer' (**C**) and the system but also all interactions between entities inside the system.

5.3 Application Design Sub-phase

In the application design, we firstly refine the Business entity model (Figure 5-a)), by reflecting there the Orchestrator entity (colored grey in Figure 9) that mediates interactions between entities.

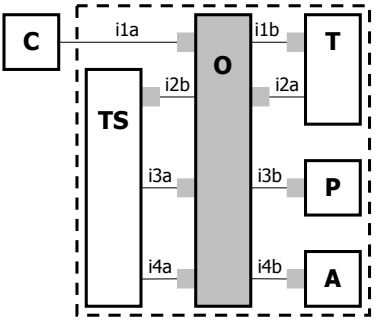

Fig. 9. The Application entity model

Then, analogously to what we did in Section 4, we can derive an *application behavior model* and a *service-oriented model*. We omit this for brevity.

6 Conclusions

This paper proposes *business-application-alignment*–related improvements relevant to the design of *context-aware applications*. A model-driven service-oriented approach has been introduced, which is essentially concerned with *consistency* as the target quality to ensure an adequate business-application alignment. We have shown how different business and application models that progressively capture more details, can be consistently derived from an initial business model. Moreover, the approach allows some useful design preparations in cases of desired adaptability of the application to possible context changes. In support of the proposed approach, is an explicit design decision - to specify applications according to the *Service-Oriented Architecture* – SOA. Such a SOA application model applies an *orchestration component* responsible for coordinating the use of subordinate services, such that the required external behavior is provided to the application's environment. The orchestration component in this model is typically application-specific, whereas the subordinate services are not: they could be discovered from a registry. The SOA application model is still at a high level of abstraction and does not depend on any

specific technology platform; in particular, the model uses integrated interactions. A further step in the design would be the distribution of such interactions, i.e. consider the exchange of information necessary for an interaction in a distributed environment, using a communication pattern supported by a commercially available middleware or data transport platform. Considering mappings onto particular technology platforms (such as Web services, CORBA, J2EE) is beyond this work's scope.

As for the modeling of real-life-level business requirements, we consider the theory of the *Language Action Perspective* – LAP, applying the LAP-inspired GI pattern that is particularly useful for modeling real-life communication/coordination.

We have also studied how the user *context states* could be taken into account (identifying them and studying their occurrence probabilities), in supporting the achievement of adaptability, as an important desirable quality of a CA application.

Finally, we have used a *case study* in order to better detail and also illustrate the different phases of the proposed approach.

Taking all this (above mentioned) into account, we claim that this paper makes useful contributions concerning (i) the possibility to *analyze user context* in support of the design of CA applications; (ii) the proposed use of LAP in business modeling, motivated by relevant strengths, namely possibilities for *capturing real-life aspects*; (iii) the *SOA focus* that facilitates an adequate business-application alignment.

To justify this claim, we have studied related work, identifying several approaches/ methods which usefully address the business-application alignment challenge, notably SDBC, Catalysis, Tropos [13].

SDBC supports the identification of re-usable business models that are soundly mappable to UML-driven software specification models. *Catalysis* provides a coherent set of techniques for business analysis and system development, and also well-defined consistency rules across models. *Tropos* facilitates application specification, supporting it with sound goal-driven requirements analysis.

A distinctive feature of our proposed approach (compared to the mentioned ones) however is the combination of: (i) LAP-based business-interaction identification and modeling; (ii) progressive and consistent further derivation of behavior models; (iii) sound business-application alignment; (iv) adequate consideration of user context states; (v) SOA focus.

Concerning SOA, the approach allows for an adequate consideration of relevant real-life aspects in consistency with which service models can be properly specified, guaranteeing that the developed services would appropriately function in their environment. These features distinguish the proposed approach also from currently popular SOA methods, such as *Crystal*, *XP* and *DSDM* [17].

To further this research, we plan to address some challenges concerning the *switching between alternative application behaviors* (as mentioned in Section 2), and we also plan to work on techniques that allow for *automated assessment of the consistency between business and application models*.

Acknowledgements. This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services, Concepts, Architectures and Applications. Springer, Heidelberg (2004)
2. Bunge, M.A.: A World of Systems, Treatise on Basic Philosophy, vol. 4. Reidel Publ. Company, Dordrecht (1979)
3. Caceres, P., Marcos, E., De Castro, V.: Integrating Agile and Model-Driven Practices in a Methodological Framework for the Web Information Systems Development. In: ICEIS 2004, 6th International Conference on Enterprise Information Systems (2004)
4. Levin, R.I., Rubin, D.S.: Statistics for Management. Prentice Hall, USA (1997)
5. Liu, K.: Semiotics in Information Systems Engineering. Cambridge University Press, Cambridge (2000)
6. Maamar, Z., Baina, K., Benslimane, D., Narendra, N.C., Chelbabi, M.: Towards a Contextual Model-Driven Development Approach for Web Services. In: ICEIS 2006, 8th International Conference on Enterprise Information Systems (2006)
7. Merriam-Webster, Inc.: Merriam-Webster, `http://m-w.com`
8. Newcomer, E.: Understanding Web Services, XML, WSDL, SOAP and UDDI. Addison-Wesley, Boston (2002)
9. Rational/ OMG MDA, Model-Driven Architecture, Object Management Group: `http://www.omg.org/mda`
10. Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. In: WMCSA 1994, Workshop on Mobile Computing Systems and Applications (1994)
11. Shishkov, B., Quartel, D.: Refinement of SDBC Business Process Models Using ISDL. In: ICEIS 2006, 8th International Conference on Enterprise Information Systems (2006)
12. Shishkov, B., Dietz, J.L.G., Liu, K.: Bridging the Language-Action Perspective and Organizational Semiotics in SDBC. In: ICEIS 2006, 8th International Conference on Enterprise Information Systems (2006)
13. Shishkov, B., Van Sinderen, M.J., Quartel, D.: SOA-Driven Business-Software Alignment. In: ICEBE 2006, IEEE International Conference on e-Business Engineering (2006)
14. Shishkov, B., Van Sinderen, M.J., Tekinerdogan, B.: Model-Driven Specification of Software Services. In: ICEBE 2007, IEEE International Conference on e-Business Engineering (2007)
15. Van Sinderen, M.J.: Architectural Styles in Service Oriented Design. In: ICSOFT 2006, International Conference on Software and Data Technologies (2006)
16. Van Sinderen, M.J., Van Halteren, A., Wegdam, M., Meeuwissen, E., Eertink, H.: Supporting Context-Aware Mobile Applications: An Infrastructure Approach. IEEE Communications Magazine
17. Wang, H., Zhang, H.: Enabling Enterprise Resources Reusability and Interoperability Through Web Services. In: ICEBE 2006, IEEE International Conference on e-Business Engineering (2006)