MODEL-DRIVEN DESIGN OF CONTEXT-AWARE APPLICATIONS

Boris Shishkov and Marten van Sinderen
Department of Computer Science, University of Twente, 5 Drienerlolaan, Enschede, The Netherlands

Keywords: Business modeling, Application design, Context, SOA, LAP.

Abstract: In many cases, in order to be effective, software applications need to allow sensitivity to context 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. Firstly, at the business modeling level, it is considered crucial that the different possible context states can be properly captured and modeled, states that correspond to certain desirable behaviors. Secondly, it must be known what are the dependencies between the two, namely between states and behaviors. And finally, what is valid for application design in general, business needs are to be aligned to application solutions. In this work, we address the mentioned challenges, by approaching the notion of context and extending from this perspective a previously proposed business-software alignment approach. We illustrate our achieved results by means of a small example. It is expected that this research contribution will be useful as an additional result concerning the alignment between business modeling and software design.

1 INTRODUCTION

In designing a software application, the engineer should take into account not only the user requirements but also the characteristics of the environment in which the application will be used (Shishkov et al., 2006*b*). This sometimes leads to the identification of different possible environmental states – referred to as *context states*, where by *context* is meant 'the interrelated conditions in which something exists' (Van Sinderen et al., 2006). Hence, sensitivity to context changes is sometimes essential for the effectiveness of applications, in this case labelled as *C̲ontext-A̲ware* (CA) *applications*. It should be decided therefore which of the relevant context states would be considered by the application designer. Further, the application should be capable of capturing context, interpreting it, and reacting on it; we call this quality *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 modeling level, it is considered crucial that the different possible context states can be properly captured and modeled, states that correspond to certain desirable behaviors; (ii) It must be known what are the dependencies between the two, namely between states and behaviors, as illustrated in Figure 1; (iii) Business needs are to be aligned to application solutions (this 'must' is valid for application design in general).

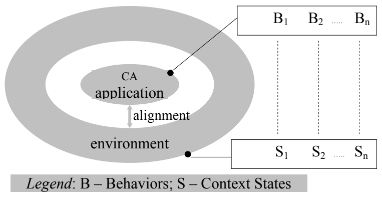

Figure 1: Essential issues in designing CA applications.

By *business modeling* we mean the modeling of business-level entities and their corresponding relations and behaviors. The desired application behaviors must (logically) be appropriate refinements of those business-level behaviors. This implies that, in addressing the business-application alignment, it might help approaching business modeling and corresponding application design separately: We could model firstly the *entities* and *behaviors* that concern the technology-independent 'view' on business processes and secondly, we could model the entities and behaviors that concern the functionality of the application (viewing this is inevitably technology-rooted). These modeling endeavors concern *different abstraction levels* – high-level business logic and technology-driven application functionality. Bridging this gap is partially considered in this paper and more thoroughly approached in previously reported work (Shishkov et al., 2006*b*).

The desired context sensitivity implies the necessity for adequate capturing of context and reaction on context changes as stated already. Although and application would react on context changes at real time, those changes should be foreseen at design time, so that proper desirable application behaviors are prescribed.

This context-driven *design preparation* is focused in the current paper. In particular, we further the development of a business-software alignment approach (Shishkov et al., 2006*b*), by extending it in the mentioned direction. The paper not only considers the notion of context but also focuses on *consistency*, as an issue claimed to be important in the business-application alignment (concerning especially CA applications). Consistency is a desired relationship between models that address separate concerns, for instance business and application concerns (Alonso, 2004). We illustrate our achieved results, by means of a small example.

In tackling this, we adopt *service-orientation* (Alonso, 2004; Newcomer, 2002) as a preferred architectural style (this decision is motivated and inspired by previously achieved results (Shishkov et al., 2006*b*)), meaning that at any design step we only consider the external behaviors of entities. In addition, composing services at high level (thus hiding the technological complexity concerned with service realization) is a way to speed up the development of business-aligned application models, and also to flexibly utilize advanced technological platforms for their implementation.

We acknowledge that the models of the application's (business) environment have to be faithful to the domain for which they are used, and also that they are inevitably driven by the subjective perception of the engineer who expresses through them either observed or desired business situations (Shishkov & Quartel, 2006). To be useful, such 'descriptions' must exhaustively disclose both statics (entities) and dynamics (behaviors), as mentioned already, and also corresponding governing norms (Liu, 2000). Reflecting these considerations in the design process, we take *additional constraints* into account; they concern the desired adequacy of the application's operation in its environment and the user requirements, and also of course the technology platforms to be used and the project-driven technical restrictions. In the current work however, we largely ignore these constraints because they do not immediately concern the derivation of (CA) application models from business models.

We expect that this work would be useful as an additional result related to the alignment between business modeling and software design.

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

2 MODELING APPROACH

A consideration of business/application models, concerns fundamentally the notions of business/software *system* and *environment* (Bunge, 1979). They both are composed of entities which could fulfil different *roles*. In doing this, entities perform behaviors (Shishkov & Quartel, 2006). A system integrates (complex) processes which comprise together its overall external behavior. It manifests the system's service provided to the system's environment.

As mentioned before, such a service provisioning needs to appear sometimes in different 'versions', driven by corresponding environmental states. Said otherwise, for one state of the environment, the system should deliver one type of external behavior while for another state, another behavior is to be delivered. Hence, context changes trigger changes in the system behavior (Maamar et al., 2006) including changes in the behaviors of the same entities or even changes in the statics (removed and/or added entities). Based on these basic considerations, concerning CA applications' design, we identify a number of challenges. Among them are:

- The application should be able to *sense* context changes;
- It should also be able to *interpret* those changes as triggers to alternative services;
- The application should be able to handle the *switching* between its alternative services;
- It should be able to deliver adequate and exhaustive domain-driven services covering all possible context states.

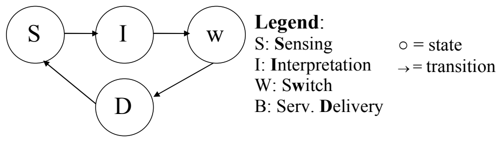

Figure 2: Model of a CA application's overall behavior.

As illustrated in Figure 2, the overall behavior of a CA application can be seen as a behavior cycling through the following states: S (sensing a context change), I (interpretation on what external behavior is required in the new context state), w (switching from one alternative service to another one, driven by a context state change), D (delivery of an alternative service).

In the following, we largely ignore Sensing (supported by sensors, for example) and Interpretation (supported by reasoning techniques and rules, for example), because they are addressed in a related work (Van Sinderen et al., 2006). Further, we pay little attention to Switching between alternative services of the application; this is positioned as future research.

Hence we focus here on the modeling of different alternative desired service behaviors (as needed by the user in corresponding context states) and their consequent realization by an application. We face thus the gap, mentioned in Section 1, between domain-driven requirements on the external application behavior, or its alternative services, on one hand, and technology-rooted application realization of this behavior, on the other hand. In properly addressing this, we need to consider different aspects of consistency:

- Correspondence between environmental (context) states and the business model (that concerns the desired external behaviors);
- Consistency between the application model and the business model;
- Consistency between dynamic aspects (behavior) and corresponding static (entity) aspects of business/application models.

Figure 3 illustrates these consistency aspects (designated by dotted lines).

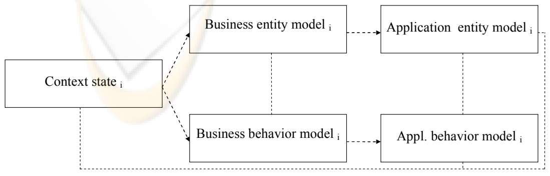
Context state $_i$
Business entity model $_i$
Application entity model $_i$
Business behavior model $_i$
Appl. behavior model $_i$

Figure 3: Consistency aspects in the appl. design process.

As shown in the figure, the models considered in the application design process take a context state as input (i.e., different states lead to alternative models). Two aspect models are considered, namely entity and behavior models. Models are refined in the design process, starting with abstract business level models and ending up with detailed application models, through gradually increasing consideration of computational and technology platform aspects. 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.

As for our business/application models, we envision them in consistency with the *Model Driven Architecture – MDA* (Rational, 2006; Caceras et al., 2004), by considering: (i) business modeling from a computational independent perspective (no decisions are made with respect to the (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 specific technological platform(s) on which the application components are implemented). The consideration of such specific technological platforms is left beyond the scope of this paper – these issues seem to be well studied (Newcomer, 2002).

Further, the mentioned adoption of service orientation, affects our modeling in a way that we are mainly interested in external behaviors (*services*) that are relevant to application's environment (Wang & Zhang, 2006). We hence could arrive at a service model from two directions: either by identifying (high-level) services from business models (which is certainly straightforward) or by abstracting from application models (discarding some technology-driven information). We claim that a business-requirements-driven service model would possess the right restrictions whose fulfilment (in application design) would guarantee that the application would be adequate to the external (context-driven) demands.

With respect to the modeling of real-life-level business requirements, we consider a theoretically-rooted approach, namely the *Language-Action Perspective – LAP* (Shishkov et al., 2006*a*) that possesses 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 fulfil (*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* of a real-life interaction (Bunge, 1979; Shishkov et al., 2006*b*). Complex interactions can in most cases be decomposed into such patterns.

According to the pattern, 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 him, by his delivering the desired production fact, through performing a corresponding production act. If the executor has declined the request, he and the initiator enter a negotiation whose negative result leads to interaction's failure. If they find a compromise however, the executor must take commitment of delivering the 'updated' desired result. The production act is responsibility of the executor. However, it 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). If unsuccessful, the negotiation 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 Sections 4 and 5) the different modeling phases, supported by a *health-care scenario* (outlined below), inspired by a broader case (Van Sinderen et al., 2006).

In the scenario, we consider patients who are suffering from conditions that are characterized by occasional occurrences of undesired effects. For this reason, these 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 treatment realized from distance, using advanced technology.

SITUATION 1. In approaching the traditional institutional-care situation, we identify the role of *Caregiver* (fulfilled by medical doctors or medical nurses) who provides help to patients. In this help provisioning, the caregiver receives support from medical workers who fulfil 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 analyst), 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**, 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 sugar and blood pressure). The Analyst delivers some conclusions resulting from the medical history of the patient. In **State 2**, it is desired, if possible, to minimize the work directed to doctors and to replace them (in some cases) by nurses. Then nurses take action in helping a patient 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 to the patient. Hence, the Advisor needs input from the Triager who in turn needs input similar to 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 fulfil the roles of Triager, Trend Synthesizer, Processor, Analyst, and Advisor, are not human beings but 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 will result in a CA model of Situation 1. Section 5 outlines, on this basis, the specification of an application that could run on (advanced) devices, adequately fulfilling the corresponding requirements.

4 BUSINESS MODELING

In achieving the first modeling milestone we come through the following 3 *sub-phases*:

The *Context analysis sub-phase*, approaching the possible context states and corresponding desired 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 systems 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 desired 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 (Shishkov et al., 2006*b*).

4.1 Context Analysis Sub-phase

Deciding about states, the engineer is sometimes inevitably driven by subjective judgements that are hardly supportable by rules: How a situation is perceived? What behaviors can be expected? Further, the engineer must often make pragmatic decisions – ignoring, for example, states that usually do not occur (although they may occur). Besides such subjective decisions, there are some steps which in general help (in our view) to adequately approach the context analysis challenge. These steps concern the consideration of *random variables*. Exploring their probabilities, allows us to apply *statistical analysis*, including *hypotheses testing* and *parameters estimation* (Levin & Rubin, 1997).

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* (Levin & Rubin, 1997) since it may take on only a countable number of distinct values (in our case 2). Provided the number of possible distinct values is exactly 2, 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).

A conducted experiment shows that on average, the 'busy' hours are 3 per a 24-hour period – 1 during daytime and 2, during night-time. We therefore conclude that the *a priori* probability of the first of the mentioned possible values is around 0.9. The *a priori* probability of the second alternative outcome is thus 0.1 (1-0.9).

Our context states represent the 'not too busy' and 'very busy' alternatives, with *a priori* probabilities 0.9 and 0.1, respectively (Figure 4).

Knowing the *occurrence probability* of each outcome helps in deciding of the 'default' desired external behavior and also what could be ignored.

In order to prescribe how to recognize each of these 2 states, 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.

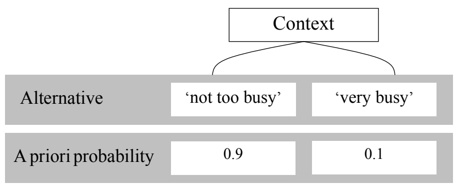

Figure 4: Two context-state alternatives.

For brevity, we exemplify with just two parameters, namely p_1 and p_2:

- p_1 is about the ratio between the number of patients and the number of doctors at a moment, and is with just 3 possible values: v_{11} (the number is less than 1), v_{12} (it is exactly 1), and v_{13} (it is more than 1);
- p_2 concerns the particular moment – *normal* or *not* ('not' would be during night-time, for example), and has just 2 possible values, respectively for 'normal' and 'not' (not normal), namely v_{21} and v_{22}.

Hence, there are 6 possible value (p_1,p_2) combinations, namely $v_{11}.v_{21}$, $v_{11}.v_{22}$, $v_{12}.v_{21}$, $v_{12}.v_{22}$, $v_{13}.v_{21}$ and $v_{13}.v_{22}$. Driven by some additional domain analysis, omitted here for brevity, we determine the last combination only as validly corresponding to the 0.1-*probability* alternative (the 'Second' alternative), and thus all the rest, corresponding to the 0.9-*probability* alternative (the 'First' alternative), as depicted in Figure 5.

	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	$v_{13}.v_{22}$

Figure 5: Context state's recognition.

Knowing the values of the 2 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 steps leading to the derivation of Business entity models concerning each of the two desired behaviors, namely the ones corresponding to the 'First alternative' and 'Second alternative' states, including steps concerning decisions on what are the relevant entities and how they are related to each other. We omit these steps not only because the *SDBC* approach is exhaustive regarding them, possessing capabilities to transform unstructured case information into a Business model (Shishkov et al., 2006*a*), but also because a consideration of such early business analysis problems would shift the focus from the business-software alignment issue.

Hence, we directly 'arrive' at the Business entity model for the *Health-Care* (HC) *case* (Figure 6); the model is expressed using a diagramming technique, inspired by *DEMO* (Shishkov & Quartel, 2006). 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 to achieve the objective of delivering a health-care service; with each such 'connection' we associate a single interaction, as follows: C(*CaregiverD*)-T (i1), T-TS (i2), TS-P (i3), and TS-A (i4). 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 i1, 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, as depicted in the left part of the figure, labelled 'a)'.

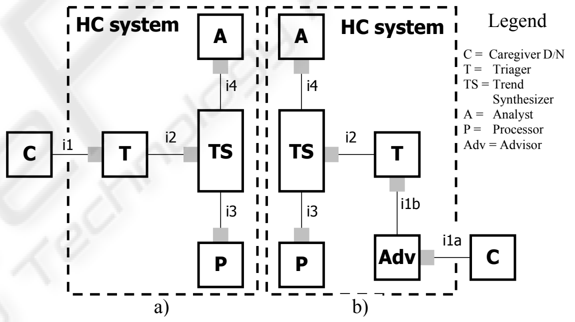

Figure 6: Business entity model for the HC case.

In the Second alternative model - 'b)' an Advisor (Adv) is envisioned 'between' C (*CaregiverN*) and T (interaction i1 is replaced by two interactions, namely i1a and i1b).

For brevity, we will consider further only the First-alternative model since it would allow us to discuss the business-software alignment almost sufficiently. As for modeling a transition from one state to another, this can be done using Semiotic norms (Liu, 2000).

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 which 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 7-a).

Regarding the internal behavior model, it should reflect the interactions between the entities of the system, as exhibited in Figure 7-b). This model shows how the interaction i1 (between the *CaregiverD* C and the *Triager* T) is made dependent on other interactions (i2, i3 and i4). The black box indicates that the results of both i3 and i4 are necessary for the triggering of i2. Such models can be extended further (e.g., with attributes) and interested readers could find more on this issue in (Shishkov et al., 2006*b*).

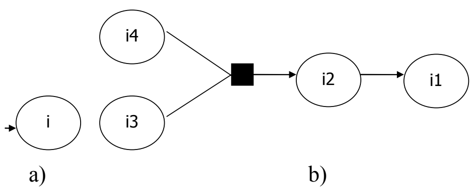

Figure 7: a) HC external behavior represented by a single action; b) Interactions in decomposed HC system, implementing the HC external behavior.

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 mentioned already, we will 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': if the initiator of one interaction requests something and if the executor promises to do the requested production act, and if this requires another interaction's output then in parallel with promising to realize the production act, the executor requests a result delivery, which actually is the triggering of another interaction. For more information on this, readers are referred to (Shishkov & Quartel, 2006).

We replace each interaction by its corresponding coordination acts – r, p, s, a, following the above mentioned 'mechanism'. We group together coordination acts based on their relation to production acts (Figure 8).

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

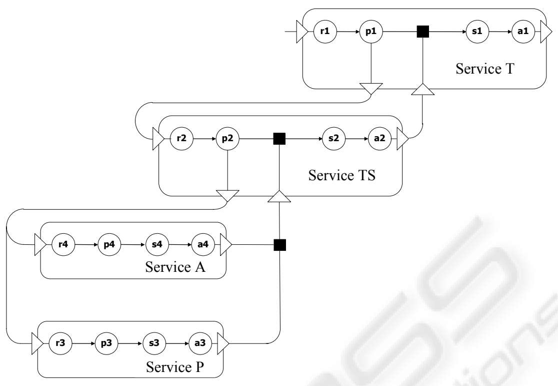

Figure 8: Refined interactions in decomposed HC system, implementing the HC-service behavior.

The model, presented in this way, *defines services* rooted in the GI pattern, consistently with our initial modeling output (Figure 6-a)).

5 APPLICATION MODELING

In achieving the second modeling milestone 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 are the user requirements and how are we reflecting them in the application model. Decision (ii) is beyond the direct scope of this paper.

The *SOA decisions sub-phase* addresses the SOA-related decisions on the desired realization of the (distributed) application service. In particular, these are decisions concerned with the way in which re-usable services are addressed and coordinated by application-specific component(s), in support of achieving the desired functionality of the application.

The *Application design sub-phase* is concerned with according refinement and extension of the models from the business modeling phase.

The *Consistency analysis sub-phase* (not addressed in the current paper; addressed in (Shishkov et al., 2006*b*)) envisions the 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 in connection to the (users') intended automation level or criteria 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 could assume nevertheless 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

The easiest-to-do decision is one-to-one mapping between business processes and application components. Such a mapping would be disadvantageous however, because the identified *services are tightly coupled*. This means that there is a dependency of the service provided by one entity on services provided by other entities (Figure 8). We claim that a solution would be to introduce 'in between' an additional application component that has the task of coordination. We label such a component as '*Orchestrator*'.

The Orchestrator is an application-specific component (as the coordination is application-specific). The (subordinate) services, however, which are coordinated by the Orchestrator, 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 by an orchestration component. Related to its coordination tasks, the Orchestrator could sometimes supply to one service the result of another service, if this is necessary for the service to perform its task.

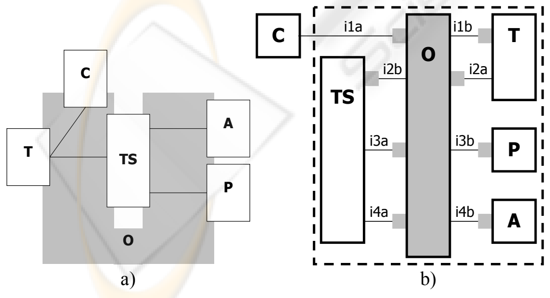

Figure 9: a) Illustration of the role of the Orchestrator; b) The Application entity model.

Figure 9-a illustrates the Orchestrator's (O) role. It concerns the interactivities between the original entities as well as coordination. 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 6-a)), by reflecting there the Orchestrator entity (colored grey in Figure 9-b)) that mediates interactions between entities.

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 improvements with respect to the *business-application alignment* in 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 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 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 that is supported by a commercially available middleware or data transport platform. The consideration of mappings onto particular technology platforms (such as Web services, CORBA or J2EE) is beyond the scope of this work.

We claim that this paper makes useful contributions concerning (i) the possibility to *analyze application's context* in support of the (application's) design; (ii) the proposed use of the *Language-Action Perspective* (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 our claim, we have studied related work. On the basis of the study, we have identified several approaches/methods which usefully address the business-software alignment challenge, notably SDBC, Catalysis, Tropos (Shishkov et al., 2006*b*).

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) is the combination of: (i) LAP-based business-capturing; (ii) behavior model consistency; (iii) SOA focus; (iv) CA-related strengths (presented). This allows for an adequate consideration of relevant real-life aspects in consistency with which we specify service models, guaranteeing in this way that the developed services would adequately function in their environment. These features distinguish the proposed approach also from currently popular SOA methods, such as *Crystal*, *XP* and *DSDM* (Wang & Zhang, 2006).

To further this research, we plan to work on *procedures for automated derivation of the orchestration component*. We are also interested in specifying techniques that allow for *automated assessment of the consistency between business and application models*.

ACKNOWLEDGEMENTS

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

REFERENCES

Alonso, G., F. Casati, H. Kuno, V. Machiraju, 2004. *Web services, concepts, architectures and applications*, Springer-Verlag. Berlin Heidelberg.

Bunge, M.A., 1979. *A world of systems, Treatise on basic philosophy, Vol. 4*, Reidel Publ. Company. Dordrecht.

Caceres, P., Marcos, E., De Castro, V., 2004. Integrating agile and model-driven practices in a methodological framework for the Web information systems development. In *ICEIS'04, 6th International Conference on Enterprise Information Systems*. INSTICC Press.

Levin, R.I and D.S. Rubin, 1997. *Statistics for management*, Prentice Hall. USA.

Liu, K., 2000. *Semiotics in information systems engineering*, Cambridge University Press. Cambridge.

Maamar, Z., Baina, K., Benslimane, D., Narendra, N.C., Chelbabi, M., 2006. Towards a contextual model-driven development approach for Web services. In *ICEIS'06, 8th International Conference on Enterprise Information Systems*. INSTICC Press.

Newcomer, E., 2002. *Understanding Web services, XML, WSDL, SOAP and UDDI*, Addison-Wesley. Boston.

Rational / OMG MDA, 2006. *SModel-Driven Architecture, Object Management Group*, http://www.omg.org/mda.

Shishkov, B., Quartel, D., 2006. Refinement of SDBC business process models using ISDL. In *ICEIS'06, 8th International Conference on Enterprise Information Systems*. INSTICC Press.

Shishkov, B., Dietz, J.L.G., Liu, K., 2006. Bridging the Language-Action Perspective and Organizational Semiotics in SDBC. In *ICEIS'06, 8th International Conference on Enterprise Information Systems*. INSTICC Press.

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 Press.

Van Sinderen, M., Van Halteren, A., Wegdam, M., Meeuwissen, E., Eertink, H., 2006. Supporting context-aware mobile applications: an infrastructure approach. IEEE Communications Magazine.

Wang, H., Zhang, H., 2006. Enabling enterprise resources reusability and interoperability through Web services. In *ICEBE'06, IEEE International Conference on e-Business Engineering*. IEEE Press.