SOA-Driven Business-Software Alignment

Boris Shishkov, Marten van Sinderen and Dick Quartel
Department of Computer Science, University of Twente, The Netherlands
{B.B.Shishkov, M.J.vanSinderen, D.A.C.Quartel}@ewi.utwente.nl

Abstract

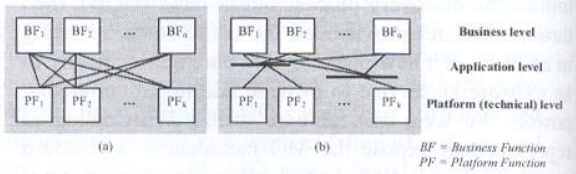
The alignment of business processes and their supporting application software is a major concern during the initial software design phases. This paper proposes a design approach addressing this problem of business-software alignment. The approach takes an initial business model as a basis in deriving refined models that target a service-oriented software implementation. The approach explicitly identifies a software modeling level at which software modules are represented as services in a technology-platform-independent way. This model-driven service-oriented approach has the following properties: (i) there is a forced alignment (consistency) between business processes and supporting applications; (ii) changes in the business environment can be traced to the application and vice versa, via model relationships; (iii) the software modules modeled as services have a high degree of autonomy; (iv) migration to new technology platforms can be supported through the platform independent software model.

1. Introduction

An important concern of application software projects is to avoid a mismatch between (user) requirements and (application) functionality [11]. We thus claim that there is a need for improving the current business-application alignment practices.

When designing application software, one inevitably faces the necessity of bridging different abstraction levels – a high-level business logic and a technology-driven application functionality. A business function (corresponding to a unit of business logic) is specific for a particular business and necessarily abstracts from technological solutions that can be used to support it. A technology platform offers a generic engineering abstraction (hence hides implementation details) which is nonetheless technology oriented. It is the role of the application designer to suggest software solutions that bridge this gap (Figure 1).

We thus argue that adequate business-application alignment can only be achieved if the initial business model (i) is a valid reflection of the relevant real-life aspects and (ii) is a suitable foundation for the generation of application models, preferably by using automated transformations. Nevertheless, the alignment cannot be accomplished only by prescribing how to define a business model. An additional demand should be that (iii) the 'architectural style' used for organizing the application modeling should facilitate the alignment; it cannot be obtained solely from top-down, but also requires a bottom-up 'preparation'.



(a) Bridging the gap between required support for business functions and offered support from platform functions using dedicated application solutions;
(b) idem, using two-step application mappings via intermediate abstractions, thus reducing the number of specific solutions.

Figure 1: Bridging the business-technology gap

In tackling this, we adopt service-orientation [1,4,14] as a preferred architectural style, meaning that at any design step we only consider the external behavior 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.

Further, we acknowledge that the derivation of a business model should be rooted in a (business) situation description reflecting either observed or desired situations [3]. To be useful, such a description must exhaustively disclose both structure and behavior

0-7695-2645-4/06 $20.00 © 2006 IEEE

as well as rules that govern the described entities and behaviors. Such a description would then be used as input to the design process, taking additional constraints into account, such as: (i) imposed by technology platforms to be used; (ii) motivated by project-driven technical restrictions; (iii) reflecting the demands of the future users of the application-to-be. In the current work, we largely ignore these constraints because they do not immediately concern the derivation of application models from business models.

This paper focuses thus on consistency and indirectly on traceability, as two important qualities of a design process, which help to address business-application alignment. Consistency is a desired relationship between models that address separate concerns, for example business and application concerns. Traceability allows appropriate reflection of changes in the business environment to the application and vice versa.

The outline of the paper is as follows: Section 2 further motivates our proposed design approach and also introduces the modeling concepts/theories and techniques that we use. Section 3 introduces a case study that is elaborated in the following sections to describe 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

The main concepts we consider, with respect to our approach, are: system, environment, entity and behavior. They will be further elaborated, together with some related concepts.

A *system* is a regularly interacting or interdependent group of *entities* forming a unified whole [10]. Thus, a business system consists of interdependent business processes, and an application software system consists of interdependent software components. We assume that a system is functioning in a (social or technical) context or *environment*, as in Bunge's categorization [2]. The functionality of a system as observed or experienced by its environment is often called *service*. This is the unified whole view, or external perspective, of the system. A system has also an internal perspective that reflects the composition of entities responsible for providing the system's service. The identification of entities of a system may be such that each entity can again be considered as a system, i.e., it has an environment consisting of other entities, it offers a service, and it has internal structure. Each system or entity has an associated behavior. A *behavior* is what a system or entity does, i.e. what activities it performs. For example, a service behavior (or service for short) is the external behavior of a system. We model a behavior as a set of related events, where 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* and *interaction*. An action is performed by a single entity. An interaction is performed by two or more entities, in cooperation. An interaction is expressed as two or more connected *interaction contributions* when representing the participation of the involved entities.

These concepts are to be put in our particular modeling perspective which concerns the derivation of service-oriented application models, for the support of business processes. We envision therefore two fundamental modeling *phases* and *milestones* in the mentioned perspective, namely *business modeling phase*, leading to a *business model* and *application modeling phase*, leading to an *application model*. An application modeling phase should be preceded by a corresponding business modeling phase. This should enable the alignment of services performed by the application with its corresponding business environment. These phases and milestones are concerned with different levels of abstraction. However, with respect to the modeling, we claim that no matter what our particular level of abstraction is, we need to consider the same types of (meta) models; otherwise the traceability between the abstraction levels would be hardly achievable.

We suggest two essential types of models in our approach, namely *structural aspect model* (envisioning the statics of a system) and *behavioral aspect model* (envisioning system's dynamics).

In our modeling approach, we are concerned with the current *de facto* standard: *MDA* – Model Driven Architecture [9], given particularly the levels of abstraction that we address. Firstly, we consider business modeling to be computational independent, i.e., no decisions are made with respect to the (partial) automation of business processes. This coincides with the CIM viewpoint of MDA. Secondly, we consider application modeling from a platform independent perspective, i.e., no decisions are made with respect to the specific technological platform(s) on which the application components are implemented. This coincides with the PIM viewpoint of MDA.

Furthermore, we adopt the *SOA* paradigm [1,4,7], to address service oriented application modeling: each application component cooperates with other application components only through the services of the latter. We also like to extend the SOA approach to the business level, meaning that we are only interested in the service of a business process, i.e. the external

0-7695-2645-4/06 $20.00 © 2006 IEEE

behavior that is relevant to the environment of the business process in achieving desired results. Only when a business process is decomposed into smaller processes, e.g. because some of these smaller processes can be used in other contexts, then internal behavior is considered, albeit only in terms of the services of the identified smaller processes.

With respect to (inter)actions and their capturing, specification and elaboration at the business level, we have been inspired by Habermas' Theory of Communicative Action [5] according to which communication among entities is oriented to achieving, sustaining and reviewing consensus. This has been acknowledged also by Winograd and Flores in the *Language-Action Perspective (LAP)*, and further reflected in the DEMO methodology and the SDBC approach [3,10]. For this reason, we follow LAP in our approach. LAP has been established as a theoretically based approach towards modeling and developing business processes. The approach recognizes that actions, as in promises or orders, are to represent the foundation of communities and organizations, and must be understood to create valid business models [13]. In LAP, the inter subject relationships among (human) entities, brought about and maintained in communication, constitute the real basis of an organization's existence. Business processes then become structures of commitments and the real important activity of entities (actors) in these processes is that they enter into and comply with commitments.

We support LAP's vision on interaction modeling, that two types of *acts* (activities performed by entities) contribute to an interaction between two entities: *production acts* and *coordination acts* (as such they can be seen as interaction contributions, mentioned above). By performing production acts, entities contribute to delivering the desired result to the system's environment. By performing coordination acts, entities enter into and comply with commitments and agreements towards each other regarding the performance of production acts. Hence, coordination acts should receive adequate attention, given the coordination-related real-life complexity. Coordination acts are driven by a proposition consisting of a fact and an associated time – they concern of course the (corresponding) production act. Furthermore, coordination acts concern particular intentions. The coordination act's intention represents the 'social attitude' taken by the performer with respect to the proposition. Examples of intentions in DEMO and SDBC are 'request', promise', 'state' and 'accept'. They correspond to the distinct *illocutions* in the terminology of Habermas; the 'decline' illocution is added to them, to allow the modeling of unsuccessful scenarios. Hence, we approach the modeling of coordination acts through the notions of *request*, *promise*, *state* (announce), *accept* and *decline*, and we argue that, based on such a view on coordination acts, a real world business invariance can be defined.

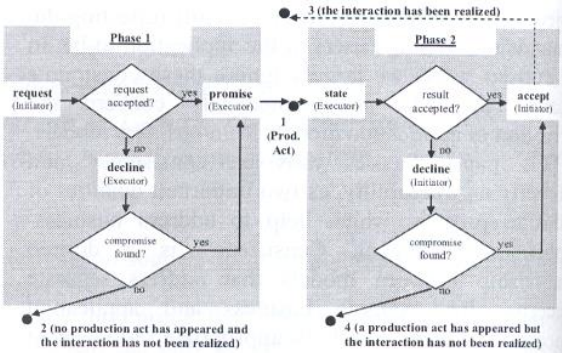


Figure 2: GIP interaction pattern

We consider hence a generic business interaction pattern where two entities are involved, driven by production and coordination acts. Following LAP, we determine particular roles for the involved entities, namely *Initiator* and *Executor*. The pattern suggests that the initiator requests something, giving two options to the executor – to respond either positively or negatively to the request. The first option means that the executor takes the responsibility to fulfill the request and thus promises to realize the requested job. The second option is that the executor does not take this responsibility. Then a negotiation may follow as a result of which the initiator and the executor could either reach another agreement or fail to reach any agreement. Once the executor has promised to fulfill a request, it is his obligation to realize it. The actual realization of course corresponds to a production act. However, the realization itself does not mean completion of the interaction because we do not have the guarantee that the executor has correctly understood the request (and has not delivered something else to the initiator) or that the executor has fulfilled the request with the adequate quality, and so on. Since we cannot have objective criteria about this, we usually acknowledge the right of the party accepting goods/services to decide whether they are adequate or not. We reflect this, by considering the state illocution (no matter if it is explicitly revealed or not) – this is the actual announcement of what has been done by the executor. The initiator must have certainly the options to accept it (modeled through the accept illocution) or not (modeled through the decline illocution). Therefore, something may be delivered but

still an interaction is not realized (e.g., a delivered pizza that is not accepted by the customer).

Figure 2 shows the discussed business interaction pattern, referred to as *Generic Interaction Pattern (GIP* or *GI pattern)*. As seen from the figure, only point 3 corresponds to a successful realization of an interaction – the executor must have promised to fulfill a request and then he must realize a production act (point 1), delivering goods/services; after his announcing the delivery, it is up to the initiator to accept them or not. If the initiator accepts them, then the interaction is complete. We see from the pattern also the two points of unsuccessful evolvement of an interaction, namely points 2 and 4.

We apply the GIP pattern in the early business modeling phase because of its real-life-related LAP-driven strengths, and also because it usefully allows for capturing failures at two stages: (i) when the requested (desired) result is irrelevant with respect to the service provider and (ii) when a realized result is not accepted by the requesting party.

As a means to express structural and behavioral aspects, we use the language ISDL - Interaction Systems Design Language [6,8] not only because it supports a GIP-driven business/application modeling, as it has been studied [12], but also because the concepts introduced at the beginning of the current section, are reflected in ISDL. Next to that, we benefit from the graphical notations of ISDL especially with respect to the modeling of behavioral aspects.

3. The FM Case

We use the Financial Mediator (*FM*) case as a basis for illustrating our design approach; FM is derived from the real and broader Icomp Case [12].

FM supports registered insurance companies in a number of ways. In this paper, we address only FM's advice provisioning service: a customer can receive from FM advice which of the insurance products (of the registered companies) best satisfies a need.

To receive advice from FM, the customer approaches FM's Advisor (an entity inside FM, which is responsible for handling the advice provisioning), specifying a request: type of insurance (e.g., health or property insurance), preferences (e.g., highest possible coverage), and so on. Based on this (and acting 'through' the Match-maker (introduced below)), FM's Request handler (an entity inside FM, which processes requests) generates a standardized request specification, appropriately synthesizing some of the information provided by the customer. This is then delivered to FM's Match-maker (an entity inside FM, which is responsible for finding a match between the standardized request and available insurance products). The Match-maker realizes a match that is driven by a particular criterion which is chosen by the customer (and represented in the standardized request), for instance: a preference for the cheapest or the most reliable product available. In order to realize such a criterion-driven match, the Match-maker applies relevant rules and procedures. However, the Match-maker needs input from FM's Data searcher (an entity inside FM, which is responsible for information searching). The Data searcher searches through the information concerning insurance products of registered companies, and applies procedures to it. This supports the identification of candidate matches relevant to the particular customer's request. The Match-maker applies its rules and procedures to realize a final match, passing this information to the Advisor.

4. Business Modeling

We use the following *sub-phases* to achieve our first modeling milestone:

1. The *Structural modeling sub-phase* includes the identification of: (i) the business system to be studied; (ii) the relevant entities belonging to the system/environment; (iii) the relations between entities (expressed as connections, representing the ability of the connected entities to interact; here we only consider interactions between just two entities); (iv) the entities' Initiator/Executor roles towards these interactions. All this builds up a *Business entity model* that covers the structural aspects.

2. The *Behavior modeling sub-phase* adds information on related behavior aspects, by modeling entities' integrated interaction behavior (abstracting from interaction contributions, and concerned with different levels of abstraction and elaboration), as follows: (i) the system's external behavior is firstly modeled (considering the system as 'black box'); (ii) the system's internal behavior is disclosed on this basis. This means that relevant interactions between entities are modeled as well as the way the interactions relate to each other, for example: the realization of interaction 'a' is necessary in order for interaction 'b' to occur; (iii) then, each interaction is replaced with a GI pattern, providing in this way adequate means to model real-life situations. The correctness of behavior refinement steps (i.e., the consistency of the resulting models) in this sub-phase, should be verified.

3. The *Service identification sub-phase* includes: (i) identifying units of (composite) behaviors, by grouping of interactions, e.g. by putting together the coordination acts based on their relations to production acts, such that each identified behavior can be

0-7695-2645-4/06 $20.00 © 2006 IEEE

considered as a (self-standing) business service; (ii) modeling the relations of these behaviors, arriving thus at a simplified representation of the detailed behavior model of the previous sub-phase.

4.1. Structural modeling sub-phase

We omit the steps leading to a Business entity model's derivation not only because the SDBC approach is exhaustive regarding this, possessing capabilities to transform unstructured case information into such a model [10] but also because a consideration of early business analysis problems would shift the focus from the business-software alignment issue.

For this reason, we arrive directly at the Business entity model for the FM case (Figure 3-a). The model is expressed using a diagramming technique inspired by DEMO [3]. The identified entities are presented in named boxes – these are Customer (C), Advisor (A), Match-maker (MM), Request handler (R), and Data searcher (D), while the small grey boxes, one at the end of each connection, indicate the executor role of the connected entities. The connections indicate the need for interactions between entities, in order to achieve the business objective of financial mediation; with each connection, we associate a single interaction (as will be seen in the next sub-phase, i1 – i4, as follows: C-A (i1), A-MM (i2), MM-R (i3) and MM-D (i4). As for the delimitation, C is positioned in the environment of the financial mediation system – FM, and A, MM, R and D together form the FM system. Through i1, FM is related to its environment (represented by C). Thus, from the perspective of C, there is no difference between FM and A.

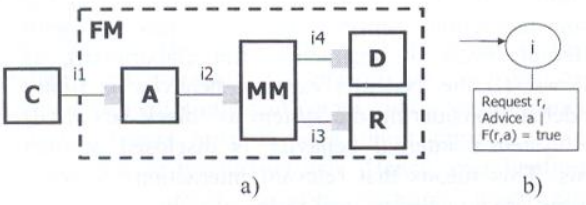


Figure 3: a) Business entity model for the FM case; b) FM service behavior represented by a single action

4.2. Behavioral modeling sub-phase

We firstly decide on the external behavior of FM, at a high level of abstraction, and then we move to the abstraction level which concerns the internal behavior of FM.

With respect to the *external behavior model*, it should envision the interaction between the customer (C) and the system (FM), and is represented by a single action (expressed by an oval) in Figure 3-b. The depicted action has also attributes (put in a box) elaborating the result of the action.

This single action i corresponds to the business objective of the FM system: to serve the request (r) of a customer, by giving advice (a) that satisfies certain criteria (F(r,a) = true).

Regarding the *internal behavior model*, it should reflect the interactions between the entities of the system, as exhibited in Figure 4. This model shows how the interaction i1 between the Customer C and the Advisor A is made dependent on other interactions (i2, i3 and i4) in the system. Each interaction between two entities (e.g., C and A) represents a request (e.g., from C to A, of type RequestC-A) and advice (e.g., from A to C, of type AdviceA-C), where the advice satisfies certain criteria (e.g., as expressed by the truth value of function FA).

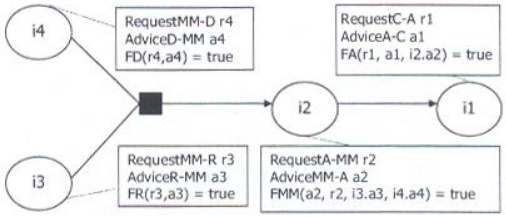


Figure 4: Interactions in decomposed FM system, implementing the FM service behavior

Assuming that the models of Figure 3-b and Figure 4 represent the same request from the customer (r = r1) and the same advice to the customer (a = a1), it follows that F(r,a) = true iff (FA(r1, a1, i2.a2) = true and FMM (a2, r2, i3.a3, i4.a4) = true and FR (r3,a3) = true and FD (r4,a4) = true).

We now need to further elaborate this model, in order to achieve a better link to relevant real-life (business) aspects. We claim that this would allow modeling of failure-scenarios (not only success-scenarios). Further, we acknowledge the essential role of real-life communication and coordination in a business system.

Expecting it to enrich our behavior model from the mentioned perspective, we are applying the GI pattern. We will firstly express the GI pattern using our notations, inspired by ISDL.

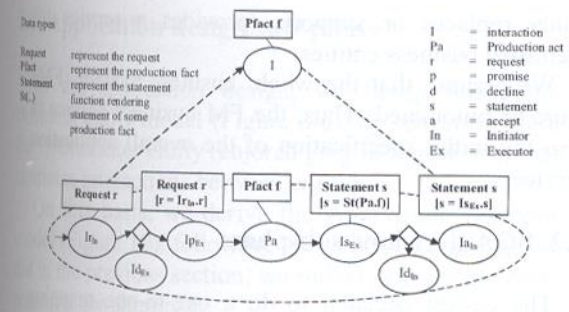


Figure 5: GIP in ISDL notation

Figure 5 exhibits the generic process of an interaction modeled at two different abstraction levels. At the highest level, the interaction is represented by a single action which models the production fact that is established. Characteristics of the production fact are modeled using the information attribute. At a lower abstraction level, the interaction's communication aspects are modeled conforming to the GI pattern. Separate actions are used to model the interaction's request, promise, state, accept and decline, and the production act. Observe that actions Id_{Ex} and Id_{In} correspond to the decline of an interaction followed by a unsuccessful negotiation; and actions Ip_{Ex} and Ia_{In} represent the promise and acceptance, respectively, which are followed by a successful negotiation.

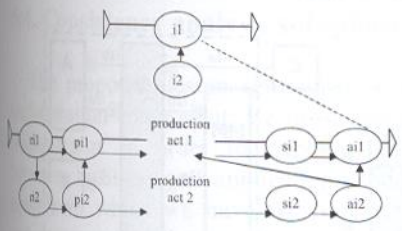


Figure 6: Illustration of the adequacy of replacing interactions with GI patterns

Thus, we replace each interaction from the behavior model (Figure 4) with a GI pattern; nevertheless, we must firstly prove that replacing any two interactions (having a simple enabling relation among each other) by two patterns would actually have the same effect.

According to the consistency criteria we follow, the results must be preserved, and also the relationships between the results. This is fulfilled, as seen from Figure 6, since: The top part of the figure shows an interaction i1 that, for example, could only appear on the basis of the realization of another interaction, namely i2. Said otherwise, the result of i1 depends on the result of i2. The bottom part of the figure shows a replacement of these interactions by GI patterns. The dashed line shows the result correspondence. As it is known from LAP, the occurrence of an 'acceptance' corresponds to the occurrence of the result of an interaction; thus, we have results consistency because ai1 (ai2) corresponds to i1 (i2). Further, we have also a consistency of the results' relationships because ai2 relates to ai1 in the same way in which i2 relates to i1. Hence, applying the GIP pattern does not change the interaction's result and we can make such a replacement as shown in Figure 7.

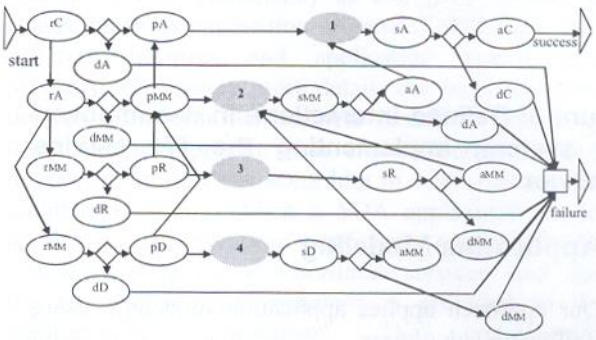


Figure 7: Detailed behavior aspect model of the FM

Observe that the number labels of production acts (grey ovals in the figure) correspond to the interactions i1 – i4 (Figure 4). Further, following one instance of the behavior, we have two possible outcomes, namely successful and failure outcomes.

4.3. Service identification sub-phase

Based on the detailed behavior model and through simplification, we arrive at a service-oriented model (Figure 8): we group together coordination acts based on their relations to production acts. Furthermore, we straightforwardly reflect (from the detailed behavior model) the information on how these groups relate to each other; we use an alternative way to model the decline acts: a decline-after-request act and a decline-after-state act are represented 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 on the figure. The model, presented in this way, defines services rooted in the GI pattern, consistently with the achieved modeling output.

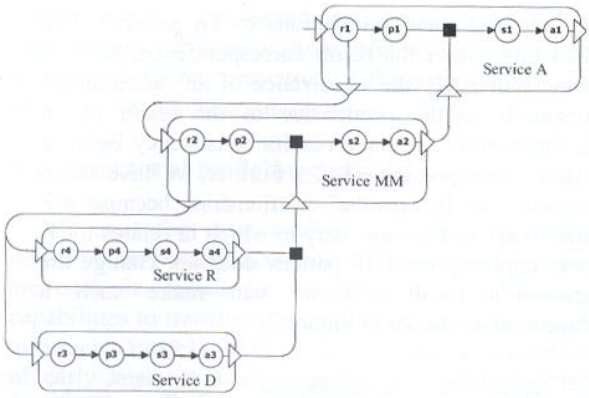


Figure 8: Refined interactions in decomposed FM system, implementing the FM service behavior

5. Application Modeling

Our approach applies application modeling, using the following *sub-phases*:

1. The *Delimitation-requirements sub-phase* concerns decisions as follows: (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.
2. 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 the achievement of the desired functionality of the application.
3. The *Application design sub-phase* is concerned with refinement and extension of the models from the business modeling phase; this is driven by the results of the previous mentioned sub-phases.
4. The *Consistency analysis sub-phase* envisions the consistency between the original business models and the proposed application models; this analysis would support the validation of the models derived.

5.1. Delimitation-refinement sub-phase

The case briefing does not provide information on the intended automation level or criteria helping to make related choices (e.g., on nonfunctional aspects such as cost/performance and ease-of-use). Hence, our decision is rather arbitrary; the refined business models should be such that the preferred application system either replaces or supports (provides a service to) identified business entities.

We assume that the whole business system (FM) must be automated. Thus, the FM business service is also the initial specification of the overall application service.

5.2. SOA decisions sub-phase

The easiest decision to do a one-to-one mapping between business processes and application components has the disadvantage that 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 (as seen from Figure 8). We argue that a solution would be to introduce an additional application component, called *Orchestrator*, that has the task of coordination.

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.

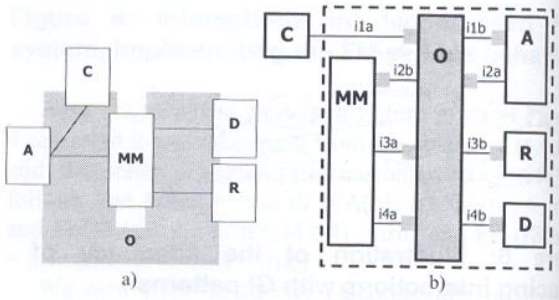


Figure 9: a) Illustrating the desired role of the Orchestrator; b) The application entity model

Figure 9-a depicts the Orchestrator's (O) desired 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 3-a), by reflecting there the Orchestrator entity (colored grey in Figure 9-b) that mediates interactions between entities.

On this basis, we derive the Application behavior model (Figure 10), following the same procedure as we did in the previous section; we reflect only the success-scenario, for simplicity.

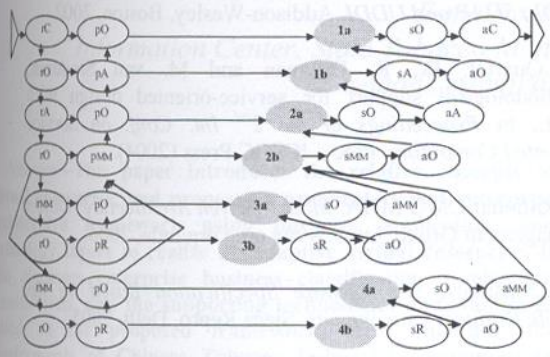


Figure 10: Detailed application behavior model

Then, analogously to what we did in Section 4, we could further derive (from the model above) a service-oriented model; we omit this for brevity.

5.4. Consistency analysis sub-phase

The proposed design refinement is driven by the orchestration component; we must hence focus on the Orchestrator in order to explain the consistencies between business and application models. It leads to models in which we have two interactions replacing one corresponding business-level interaction. Referring to LAP, we can replace an interaction with two other interactions, keeping the same effect on the interaction's environment, only if we have fulfilled the following: (i) the interaction's request corresponds to the one of the triggering interaction (from the two); (ii) the interaction's production act corresponds to the one of the triggered interaction; (iii) the interaction's accept corresponds to the one of the triggering interaction.

This is fulfilled in the way we replace an interaction from our business models with two Orchestrator-related (application-level) ones: the replacement, for example, of i1 with i1a and i1b (consider Figure 7 and Figure 10) shows that rC (i1) corresponds to rC (i1a) and also aC (i1) corresponds to aC (i1a). Further, the production act of i1 corresponds to the production act of i1b because this is actually, in both cases, the advice delivered by the Advisor.

Therefore, we have done a replacement which does not change the overall effect to the environment of FM.

6. Conclusions

This paper suggests improvements with respect to the business-application alignment in the design of application software. We propose a model-driven service-oriented approach which is essentially concerned with consistency as the target quality to ensure business-application alignment. We show how different business and application models that progressively capture more details can be consistently derived from an initial business model. In support of the presented approach, is an explicit design decision to specify applications according to a 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 public or corporate registry, or they can be designed such that they can later be made available for re-use through 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, and a next 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 however outside the scope of this work.

We further claim that this paper makes useful contributions concerning (i) the proposed use of the Language-Action Perspective (LAP) in business modeling, motivated by relevant strengths concerning possibilities of capturing real-life aspects; (ii) the SOA focus in our design approach for business-application alignment. In support of these claims, other related work has been studied. On the basis of this study, we identified several approaches/methods which adequately address the business-software alignment challenge, notably SDBC, Catalysis and Tropos.

SDBC supports the identification of re-usable business models that are soundly reflected in UML-

driven software specification models. Catalysis provides a coherent set of techniques for business analysis and system development as well as well-defined consistency rules across models. Tropos facilitates the specification of an application, by supporting it with sound goal-driven requirements analysis. A recent detailed analysis and comparison of these approaches/methods can be found in [11].

A distinctive feature of our proposed approach (compared to the mentioned ones) is the combination of a LAP-based business-capturing, behavior model consistency and SOA focus. This allows for an adequate capturing of relevant real-life aspects in consistency with which we specify our service models, guaranteeing in this way that the developed services would adequately function in their environment. This feature distinguishes the suggested approach also from currently popular SOA methods, such as Crystal, XP and DSDM [1].

To further this research, we plan to work on proposing procedures for an automatic derivation of the orchestration component. We acknowledge as well the need of techniques allowing for an automatic 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., F. Casati, H. Kuno, V. Machiraju, *Web Services, Concepts, Architectures and Applications*, Springer-Verlag, Berlin Heidelberg, 2004.

[2] Bunge, M.A., *A World of Systems, Treatise on Basic Philosophy, Vol. 4*, Reidel Publ. Company, Dordrecht, 1979.

[3] Dietz, J.L.G., Understanding and modeling business processes with DEMO, In *Proceedings of the 18th Int. Conf. on Conceptual Modeling (ER)*, Springer LNCS 1728 (1999).

[4] Dirgahayu, T., *Model-driven Engineering of Web Service Compositions: a Transformation from ISDL to BPEL*, University of Twente – UT Press, Enschede, 2005.

[5] Habermas, J., *The Theory of Communicative Action*, Cambridge, 1984.

[6] ISDL. *Interaction Systems Design Language*, http://isdl.ctit.utwente.nl, 2005.

[7] Newcomer, E., *Understanding Web Services, XML, WSDL, SOAP and UDDI*, Addison-Wesley, Boston, 2002.

[8] Quartel, D., R. Dijkman and M. van Sinderen, Methodological support for service-oriented design with ISDL, In *Proceedings of the 2nd Int. Conf. on Service Oriented Computing*, Proc – ICSOC Press (2004).

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

[10] Shishkov, B., *Software Specification Based on Re-usable Business Components*, Sieca Repro, Delft, 2005.

[11] Shishkov, B., J.L.G. Dietz and K. Liu, Bridging the Language-Action Perspective and Organizational Semiotics in SDBC, In *Proceedings of the 8th Int. Conf. on Enterprise Information Systems (ICEIS)*, Iceis Press (2006).

[12] Shishkov, B. and D. Quartel, Refinement of SDBC business process models using ISDL, In *Proceedings of the 8th Int. Conf. on Enterprise Information Systems (ICEIS)*, Iceis Press (2006).

[13] Winograd, T. and F. Flores, *Understanding Computers and Cognition: a Foundation for Design*, Ablex, Norwood, 1986.

[14] World Wide Web Consortium. *Web Services Description Language 1.1*, W3C Note, http://www.w3.org/TR/wsdl, 2005.