Adaptation of a Cloud Service Provider’s Structural Model via BROS

Completed Research

Hendrik Schön
TU Dresden, Dresden, Germany
hendrik.schoen@tu-dresden.de

Raoul Hentschel
TU Dresden, Dresden, Germany
raoul.hentschel@tu-dresden.de

Katja Bley
TU Dresden, Dresden, Germany
katja.bley@tu-dresden.de

Abstract
Conceptual models of specific domains provide a general overview of a software system design. Structural models, as a technical version of conceptual models, serve as development blueprints that may have to be adapted to fit special enterprise-specific demands. However, a flexible and dynamic adaptation of the design is necessary to respond fast and efficiently to new or changing requirements for reducing time and costs in the later development. In this paper, we utilize the “Business Role-Object Specification” (BROS), a role-based modeling language, for dynamic and structured adaptation. We demonstrate our approach by adapting an existing structural model from the cloud service provider domain with regard to a partner program process. Further, by comparing the BROS adapted model with a traditional UML-based adaptation, we are able to evaluate both approaches and show the benefits of the new BROS adaptation method, e.g., extended expressiveness and flexibility towards changing requirements and features.

Keywords
Domain Model, Roles, Structure, Adaptation, Cloud Service Provider.

Introduction
The digitalization of the business world is accompanied by major structural changes. Due to constantly changing requirements on company structures and processes (e.g., legal issues, new features), intuitive and easy changes of existing software systems are needed, which can thereby respond to the required flexibility of the overlaying business domain. However, a constant trade-off needs to be tackled: domain modelers strive for uniform and standardized models (that is in-house or external template-like models called domain or reference models) in a structured way (e.g., UML class diagram). In contrast, companies want to stick to their own individual processes and specific requirements (often specified as procedural models, e.g., BPMN) and do not want to give up crucial requirements due to given model structures. This discrepancy should not be neglected as mastering it contributes significantly to the quality of the developed models. Nevertheless, new functionality and requirements to existing software systems are not introduced with the underlying software structure in the first place. Thus, the underlying structure of software has to be flexible enough to adapt easily towards new incoming business requirements, specified as behavior definitions or other abstract top-level descriptions. The approach of this paper addresses this issue by deploying a behavior-aware structural model that can be adapted towards new business requirements without invasive changes to the (already established) underlying software structure.

The goal of our paper is twofold: First, we propose the application of our modeling language “Business Role-Object Specification” (BROS), which was developed to specifically focus on constantly changing and increasing requirements in the development of flexible software systems growing correspondingly. BROS utilizes role-oriented modeling to overcome the limitations of the commonly used object-oriented modeling paradigm for software development via flexible structures and context-dependent behavior definition. The application of BROS leads to a behavior-aware structural model. Second, we evaluate our approach by applying BROS on a real use case. This is an existing structural (in-house reference) model of a cloud service
provider (CSP), where a new partner program business process is introduced as the element of changing requirements. Following the adaptation of the software system according to this new process, we compare the resulting BROS model with a model derived by adaptation via classical UML-based ad-hoc modeling in order to reveal the benefits and drawbacks of the new approach.

The remainder of the paper is structured as follows: In the second section, we present the conceptual background. Section three focuses on the chosen design science research approach. In the fourth section, we provide the starting point of the adaptation process: the domain model (acting as the in-house reference model that has to be adapted) and the use case of the new partner program acting as the targeted process. In section five we adapt the CSP’s model via BROS in conformance to the new process. After suggesting the alternative UML-based adaptation in the next section, we evaluate predefined properties of both resulting models. We conclude with a summary and an outlook on future work.

Conceptual Modeling Background

Conceptual Enterprise Modeling and Model Adaptation

The field of model-driven software development offers a variety of languages and model types for use in software design, development, validation, and others. The range, granularity, expressivity, and complexity vary with each single model type. On the one hand, analysis models may be used for a broad overview and to define the functional scope (e.g., use case diagrams, very coarse-grained class diagrams). On the other hand, design or implementation models are used to describe systems in detail including technical specification needed for implementation (e.g., class diagrams, petri nets, sequence diagrams). For complex software system modeling, conceptual models often serve as a bridge between the two. Conceptual modeling can be defined as "a description of the phenomena in a domain at some level of abstraction, which is expressed in a semi-formal or formal visual (...) language" (Krogstie 2012, p. 1). The often more abstract conceptual models describe a simplified overview of the system’s domain objects (called business objects in business domains), representing the targeted structure or processes with reduced complexity. The application areas of conceptual models are manifold (Fettke and Loos 2003). Often, there is an already established software structure, described as a conceptual model that must then be adapted towards new design decisions or business requirements. Such a basic standardized software structure can be used as a template, an in-house reference model. Reference models are templates that “formalize state-of-the-art or best-practice knowledge of a certain domain” (Becker et al. 2007, p. 2) aimed at being adapted and tailored, to obtain enterprise-specific models, called application models as opposed to the reference model they were derived from. This focus on adaptation while keeping a core model untapped is what makes the sub-area of reference model adaptation fundamental to our research. Although adaptation methods may differ significantly, Hofreiter et al. (2012) provide a systematic classification of approaches with our suggestion being of the type "specialization".

Role-based Modeling

To include the behavior-aware specification in structural models as well as adapting the structure non-invasively towards new business requirements, we utilize the role modeling paradigm. The object-oriented (OO) paradigm is an established concept in the software development area. Since the last decades, the role-oriented paradigm extended OO thinking. It focuses on the idea that objects, derived from types and classes, can play certain roles in different contexts. This serves a more accurate description of the domain’s entities with their context-dependent structure and behavior. The role as an intuitive modeling concept allows for the benefits of dynamism and context-dependency in formerly rather static OO modeling. Thus, the role concept is the subject of investigation on every layer of software engineering, ranging from role-based theories (e.g., (Steimann 2000)) and modeling languages (e.g., (Kühn et al. 2014)) up to role-based programming languages and runtime environments (e.g., (Herrmann 2005; Leuthäuser 2017)). The subject of role-based enterprise modeling also attracted interest in the information systems discipline (e.g., (Frank 2000)). The role’s abilities to represent an underlying object that is changing its behavior and characteristics in different contexts is advantageous for our idea of implementing new business requirements into structural specifications (Schön 2018). Technically, roles can be described with three main properties: (a) behavioral, (b) relational, and (c) context-dependent properties (Kühn et al. 2014; Steimann 2000). The behavioral property ensures that every object can play one or more roles to define its behavior. The roles adopt the identity of the object and can be taken on or dropped at any point in time. The relational property
states that roles emerge from their relation to other roles. This property is often already defined as names for association ends. However, name tags are not equal to a full-fledged role with its own behavior. Last, the context-dependency of roles describes the environment of roles. A role is always encapsulated in a context that defines the applicability and behavior of the role. A detailed description of role-based modeling concepts and the properties of roles can be found in Kühn et al. (2014).

**Business Role-Object Specification (BROS)**

We developed the role-based modeling language “Business Role-Object Specification” (BROS) that focuses on behavior-aware structural modeling (Schön et al. 2019). With BROS, it is possible to create or reuse structural models and adapt them towards specific system behavior and processes (what we call background process), often driven by new incoming business requirements. For that, the business objects of the underlying model (e.g., a class diagram or domain model) are adapted and further specified by roles to derive the enterprise-specific application model. BROS was initially developed for easy reference model adaptation, but can also be used for general software system construction or adaptation. Although it is a structural modeling approach, we call it behavior-aware because of two reasons: First, adaptation is usually motivated via changes in business processes or the emergence of new processes that require changes in the structural model. Although we do not focus on process modeling itself, we explicitly include events induced in the respective background processes. Second, via the inclusion of events, temporality, and context-dependent behavior via roles the modeling approach allows for behavioral modeling constructs within a mainly structural modeling language.

![Figure 1. The BROS model concepts (Schön et al. 2019)](image)

The main concepts of BROS are objects, roles, scenes, and events (Figure 1). An underlying structural domain model consists of given entities. In the final BROS model, the selected relevant entities are then called objects. The objects are the target of any adaptation done by using the remaining concepts. BROS utilizes roles as specific representations of objects in certain scenes (the role's context). In contrast, the enterprise-specific processes are the main drivers of the adaptation and are called background processes as mentioned before. They serve two kinds of information: (a) the scene as an encapsulation context of a use case or task, and (b) the events as certain points in time affecting the roles. The details of the language, its strengths and applicability as well as an example are described in Schön et al. (2019). BROS builds on CROM (Kühn et al. 2014). However, CROM is not designed for the purpose of behavior-aware structural model adaptation in the business context. Compared to other role-based modeling languages for conceptual modeling, BROS has the advantage of being suitable for the technical specification of the software system and of including the impact of procedural on structural models at the same time.

**BROS as Design Science Research Artifact**

BROS emerged from multiple, however, related scientific disciplines. On the one hand, the design and development of BROS as a structural modeling language is located in software engineering and the computer sciences. On the other hand, the intended application of BROS is a business environment and has an impact on the interpretation/understanding of domain models and business logic as well as their interaction. According to Design Science Research (DSR) methodologists, who emphasize the applicability of design science artifacts, the aim of design science research is a contribution to efficiency and effectiveness in organizational information systems (Peffers et al. 2007). Often motivated by problems from business
practice, the focus is on concrete solution artifacts. BROS thus focuses on both the technical aspect of a computer system and the social and user interaction aspects in a business environment, that is why it can be understood as DSR artifact in the information system discipline (Gregor and Jones 2007). While computer scientists have a stronger focus on the development/building process of an IT artifact as well as on its technological rigor, IS scientists focus on the evaluation of the impact of an IT artifact on an organizational level (March and Storey 2008; Sonnenberg and vom Brocke 2012). In our understanding, within BROS both aspects, building and evaluating, are equally important to its successful implementation. That is why we intend to do justice to both disciplines by choosing to lean on the DSR approach, as initially postulated by Hevner et al. (2004) and later on adapted by Peffers et al. (2007). By developing a novel modeling language (build process) called BROS, we intend to improve the development of flexible software systems that can easily adapt to changing business contexts. In this paper, we apply BROS in the context of a CSP who has to cope with changing requirements due to a new business process. We do so in order to discuss the strengths and weaknesses of BROS as well as improvement potential (evaluate process).

An often mentioned aspect of criticism in the build-evaluate process of Peffers et al. (2007) is the rather strict separation of both processes as well as the ex post evaluation of the IT artifact. With respect to BROS, the build-evaluate process should be regarded as a dynamic and iterative process in order to improve the artifact continuously. That is why our evaluation approach builds on the suggestion of Sonnenberg and vom Brocke (2012). As Figure 2 shows, the DSR cycle is an ongoing process with several evaluation activities. For each generic phase of the model, Figure 2 provides the relevant BROS research activity: To the best of our knowledge, there is no structural modeling language or method that, on the one hand, is able to specify technical depth of pre-defined or static software systems, and on the other hand, is able to adapt to changes driven by business requirements ('Identify Problem'). In the next step ('Evaluation 1'), we conducted various discussions with domain experts and searched the literature for comparable concepts to ensure that our method contains a new solution approach. The first draft of the role-based business object modeling approach was presented in an idea paper at a conference and received additional valuable feedback from modeling experts. The overall feedback was then used and implemented ('Design') resulting in the first version of BROS. Afterward, BROS was demonstrated and extended using a fictional pizza ordering process. The purpose of this step was to show that it is feasible to adapt a structural business domain model with respect to a targeted business process via role-oriented modeling. The focus of the paper was on feasibility, comprehensibility, pertinence, and simplicity as suggested by Sonnenberg and vom Brocke (2012) for ‘Evaluation 2’. After completing the ex-ante evaluation activities, we proceeded by refining the BROS language in its dependencies, relations, and semantics. By conducting ‘Evaluation 3’ in this paper, we applied the improved and enhanced BROS language on a real business process from a CSP. In doing so, we test the applicability of BROS on an organizational level. As Sonnenberg and vom Brocke (2012, p. 77,78) postulate “Since this activity ['Evaluation 3'] links ex ante as well as ex post evaluations of an artifact it is central for reflecting an artifact design and thus to initiate and inform subsequent iterations of the artifact design activity”.

In this evaluation activity, we focus on one of the main design criteria, which is functionality, as postulated by March and Smith (1995) for the evaluation of method artifacts. By doing so, we are able to further improve our approach for the implementation in a naturalistic business environment and use by modeling practitioners ('Use'), which can, later on, be evaluated for generality ('Evaluation 4'). However, these last two steps will not be part of this paper but future work.
Case Study: Cloud Service Provider Systems and Processes

Cloud Computing (CC) provides companies of all sizes advantages by consuming computing resources (e.g., networks, servers, storage, applications, and services) with low/minimal entry costs, pay-as-you-go, great flexibility, and scalability (Hentschel et al. 2018). CC services are typically classified by the type of service differentiated by, e.g., application (SaaS), platform (PaaS) and infrastructure (IaaS) level. However, in contrast to this rather technology-oriented classification, cloud services can also be classified in a more business-oriented manner, by market actors, that offer a certain class of services in a cloud value network (Böhm et al. 2010). In this paper, we use the “partner program” process of the CSP, which allows customers to resell purchased cloud services to their own customers, as the background process. We specified the market actors as follows: Partner as a synonym for value added resellers who either aggregate modular services to create value-adding, complex solutions for specific requirements (Aggregator) or integrate cloud services into the existing IT landscape (Integrator) and customer as consumers who receive these services.

Process Model (BPMN) as Background Process

In order to adapt the CSP’s structural model (in-house reference model) via BROS, the CSP’s partner management process (Figure 3), serves as the background process that motivates the structural model’s adaptation. A customer can subscribe as a partner to manage their own customers. For this purpose, an invitation is sent to the customer, who investigates the invitation and then either rejects or accepts it. If this invitation is not accepted within 24 hours, it expires automatically. If the customer accepts the invitation by sending the filled invitation form, the partner can operate the customer’s services. This includes the administration of the customer’s products and access to the administration account. The management of the services remains active until the customer cancels it or until the business relationship ends.

Figure 3. The BPMN model for the CSP’s partner program process

Domain Model (UML) as Reference Model

The CSP, a small German company, does not use an external reference model. Instead, the CSP established their own standard in-house model, requiring every employee involved in development to adhere to this domain model in order to prevent non-transparent interfaces and to ensure fast and inexpensive software system evolution. The relatively small size of this structural model is advantageous for our research as it provides a sufficiently sized but still comprehensible overview of the whole system and consists of about 150 business objects. Figure 4 presents a small extract of the structural model for the use case in this paper. It includes the main business objects that are relevant to the partner program process. We omitted the attributes and operations for the moment, as the more detailed representation is shown in Figure 5 and 6. The business properties are related as follows: In Figure 4, the Customer is the main business object. A HostserverAccount enables the customer to access the Hostserver, which contains either HostingDeals or ServerDeals that are grouped together as Products (together with Domain). HostingDeals are hosting plans with shared resources on a server (e.g., storage, databases, email addresses). ServerDeals are virtual servers with allocated resources on a dedicated server (e.g., CPU, memory, bandwidth), and Domains are domain names used for identification within the internet. The customer can cancel products by...
creating a CancellationOrder that includes the items to be canceled. The elements Cancellation and GenericEmail are not associated directly with the other business objects in this subset.

Figure 4. A (reduced) excerpt of the overall CSP’s structural (in-house reference) model

Application of BROS for Model Adaptation

The application of BROS for adaptation benefits from the given background process as the modeler’s main source of business knowledge for conducting the adaptation. There is no detailed step-by-step guide available yet for using BROS for adaptation purposes. Together with the CSP, we therefore followed the main steps that are conducted in the pizza ordering case example in Schön et al. (2019).

We encapsulated the adaptation part of the final BROS model in a package called Partner Program as a static context (see Figure 5 for the adapted model). In the BROS definition, a scene is defined as “an instantiable temporal collaboration context of roles and events, related to the same business logic part” (Schön et al. 2019, p. 9). Thus, we created two major scenes: Request and Customer Management. The former deals with the invitation, the latter with the association of the manager, the managed customer, and
the management account as well as the related products. However, subscribing and unsubscribing the partner is (in our adaptation variant) not an own scene but two single events attached to the role. We selected a particular set of business objects from the complete structural model (as shown previously in Figure 4), which serve as players for the roles. Further, we identified various roles within the adaptation use case, like the Partner role (the Customer plays that role as soon as he is subscribed) or the Invitation role (when the GenericEmail acts as an acceptance request in the Request scene). We enriched the roles with attributes that are needed for the use case, modeled as delta (prefixed “+”/“-”). However, due to space limitations and complexity reduction, we did not model all available delta attributes and operations for every role, but only some for illustration purposes. Regarding operations, we only modeled a subset for the Customer object and its fulfilled roles as delta examples. As a next step, we concentrated on the targeted process flow of the background process and added events for the different decisions and actions that “determine the object fulfillment of roles” (Schön et al. 2019, p. 8). For example, in accordance with the background process, we modeled the send request event as invoker event for the Request scene.

As soon as the Customer subscribes himself as a Partner, he may manage several other Customers as Manager. For this, he has to invite a Customer, who has to approve this request within 24 hours. If the Customer does so, he is a registered Managed Customer of the Manager who invited him. This leads to the fact that the Manager can administer the Managed Customers’ products with their accounts. Nevertheless, since we omitted most of the operations due to readability, the “complete” adaptation via BROS would be much more complex. The whole adaptation process is done in a straightforward manner. As there is no detailed adaptation process description yet, the adaptation is rather done in an ad-hoc manner taking the CSP’s recommendations and expectations into account. The primary goal was the creation of a valid BROS model as a structural adaptation, not the evaluation of procedural adaptation steps.

Comparison and Discussion

UML-based Application Model

In order to compare the BROS-based model adaptation with the UML-based adaptation, we adapted the initial CSP’s structural model with the traditional OO approach.

Figure 6. A possible UML-based adaptation of the structural model for the partner process

The UML-based adaptation can be executed in two different ways: one may intend to change the underlying model (“invasive” changes of the template) or utilize inheritance (and similar additive OO mechanisms) as
a “non-invasive” adaptation approach. Often, companies want to keep the core domain model stable which is a typical request in business environments, which is also the case with our CSP. Therefore, the adaption should be as separate as possible from the underlying structural model (non-invasive adaptation). As non-invasiveness was one of the design goals of BROS, it is more appropriate to compare it to the second (rather non-invasive) UML inheritance approach. Figure 6 shows the UML-based application model as a possible solution. The darkly shaded classes are added to the original model by inheritance. For comparison, we modeled the use case of the CSP’s partner program in a similar way as we modeled it with BROS. We utilize subclasses of Customer (Partner, ConcreteCustomer, and Manager) for an equivalent and non-invasive adaptation. Further, the GenericEmail gets an inherited subclass for the Invitation object. The Cancellation, however, is used as is. Again, we omitted all operations but the customer’s ones.

Comparison of BROS and UML-based Adaptation

Together with modeling experts and the CSP company itself, we compared both outcomes to identify the strongest benefits and drawbacks of modeling structural adaptation with BROS. Obviously, there are more elements available in BROS for structural and behavioral adaptation than those provided by standard UML. On the one hand, more elements often provide more expressiveness. On the other hand, a large amount of model elements frequently results in hard to read and more complex models. Regarding the conceptual expressiveness, we can observe that BROS has various benefits when it comes to modeling the system design (despite the language’s higher complexity). We state that, regarding the structural adaptation, in BROS the targeted structure can be expressed in a more suitable and intuitive way. The encapsulation of roles (representing the available objects) within scenes as a collaboration of participants makes adaptations explicit and maintainable. Instead of introducing new subclasses with fixed type and identity, the roles can be played or dropped by an object if a change in the object’s representation in a certain context is needed. For example, in BROS, it is rather easy to realize the fact that the partner program makes a distinction between Person and Company. The two new roles, fulfilled by the Customer object, allow the modeler to express the new structure on a given one. In contrast, UML-based adaptation can only use inheritance. Creating Person and Company subclasses has, however, the disadvantage that we always have to decide between either using, e.g., a Company object or a Partner object, as we do not have “combined” subclasses like PartnerCompany (except with multi-inheritance, which will lead to new problems, though). However, even if we would use a mechanism for combined subclasses for different property dimensions (e.g., the multi-inheritance), this would lead to huge “inheritance trees,” resulting in possible objects like PendingPartnerCompany that are far from being “real” business objects and are more like states. The static nature of subclasses in UML also leads to a loss of expressiveness as, for example, an object may not be partner and a customer of another partner at the same time. The identity is fixed with its subclass type (that is either partner or customer identity). Further, switching identities at runtime is also a known issue with sub-types. The modeling of context-dependent delta structure (e.g., the removal and addition of attributes of an object in certain contexts) is easier with roles than, e.g., using super-classes. BROS solves these problems by attaching roles to the original object, representing the object in a new context, change its characteristics, and leave its identity untouched. Regarding the system’s behavioral adaptation, BROS has the huge advantage of a temporal modeling element, called events. With this feature, BROS can model statements like, e.g., “after 24 hours without reply the invitation becomes invalid”. In UML-based adaptation, such temporal expressions are often not possible at all (at the most by comments or complex UML extensions). Nevertheless, BROS events are not always suitable as they cannot map a complete process into the structural model. Although this is not intended, the extent is probably an aspect that needs further attention. Besides, the inclusions of these elements within the model can lead to more difficulty in understanding. The focus on using roles as context-dependent adaptations in process collaborations can be a disadvantage if only simple (process-independent) adaptations are to be made, though.

Reusability is given in both, the BROS and the UML-based approach. However, BROS seems to be in advance as it enables strong separation of concerns via scenes and roles and multiple adaptations of the template. This also supports the flexibility of the adaptation, where an already BROS adapted (role-based) application model can easily be adapted further. In contrast, changes and extensions in a UML-based application model can be rather complicated because of the possibly high number of subclasses that have to be managed. Further, with the BROS language, one is able to adapt in a non-invasive way, leaving the underlying model (mostly) unchanged. The role-based abstraction from the objects helps to decouple the adaptation from the template. However, some minor changes within the code may be necessary, like
changed method calls or triggers of invoker elements. Theoretically, the UML-based method also supports non-invasive adaptation. However, in practice, some drawbacks must be taken into account (e.g., many overhead classes and reduced readability). The integrity of the application models (regarding the original structural model) is quite different. We argue that the UML-based approach results in models with higher integrity (compliance) with respect to the original model as the possibilities of adaptations are limited and strongly standardized. This leads to application models that are tightly connected to the template. In BROS, a role may change an object severely. Applying a role may theoretically lead to a complete redefinition of an object, e.g., removing everything and adding new operations. While this is in the responsibility of the modeler, the integrity towards the original model is hardly ensured in BROS. The temporal specification that comes with the event feature, which abstracts the time affecting the structural elements (despite the drawbacks) is a fully new feature the UML-based approach does not support at all. However, its specification is not comparable with a fully-featured procedural model. Table 1 summarizes our assessment of the comparison between BROS and UML-based modeling regarding structural model adaptation, derived by the feedback of experts and the CSP. It cannot replace experiments or extensive expert feedback (which could be future work though). However, it gives a first summary of our own experience while using BROS and UML for the use case we investigated.

<table>
<thead>
<tr>
<th></th>
<th>Adaptability</th>
<th>Reusability</th>
<th>Flexibility</th>
<th>Non-invasiveness</th>
<th>Integrity</th>
<th>Temporal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison of modeling concepts between BROS and UML-based adaptation
(☐ = barely supported, ☐ = semi-supported, ■ = well supported)

In sum, we consider BROS as a more advanced conceptual modeling language due to more expressiveness especially with respect to business logic and in adaptations targeting the technical perspective of software systems. Nevertheless, we also see disadvantages or challenges that still need to be tackled: First, we do not think that technical feasibility is fully developed yet as some concepts are unspecified with respect to their technical implementation. Further, some model constructs are logically questionable, like object instances that participate with its roles in multiple related scenes (e.g., ensuring that the Pending Customer role uses the same Customer instance as the consequential Managed Customer role). In contrast, UML is clearly defined, and direct technical implementation is feasible. Second, BROS usability seems to be less than the UML-based approach. While most modelers are confident with UML, BROS is rather new, and a modeler needs to familiarize with the BROS language concepts. The resulting models are quite complex and harder to read, which may also foster errors when modeling or reading BROS models.

**Conclusion**

In this paper, we investigated the use of BROS for the purpose of a CSP’s structural model adaptation. We compared the BROS adapted model with an UML-based adapted model and were able to show that the use of BROS is effective for the purpose of adapting a structural model (preferably a reference model) concerning a given exemplary background process and outlined the benefits of this approach in comparison to an ad-hoc UML-based adaptation. With BROS, we were able to model the application model in much more detail and expressiveness, which is not always possible with classical UML-based adaptation. The abstraction of the classic business object (roles for different contexts paired with temporal behavior specification via events) turns BROS into a strong and likely preferable alternative when the adaptation consists of non-trivial adaptations and can be supported by a background process. However, we identified limitations concerning the usage of BROS. First, specific structural models might not be suitable for an adaptation via BROS, as too coarse-grained or fine-grained reference models may cause problems when deciding on possible objects as players for roles. Second, one needs to reconsider when non-invasiveness is really needed or turns out to be a real drawback. In some cases, a non-invasive adaptation would not be advantageous and the direct change of a structural model easier than adaptation.

As future work, BROS has to be reviewed and extended for the inclusion of other conceptual constructs (e.g., the further usage of role constraints) and for its implementation in practice. For the latter, a
Structural Model Adaptation of a Cloud Service Provider

prescriptive step-by-step model adaptation guide will be needed as well as tool support. Both are currently developed and will be included in an additional evaluation step in a more naturalistic business environment (e.g., larger models, modeling practitioners, etc.). A more sophisticated study about, e.g., efficiency, usability, and operability would also be of advantage for the propagation of BROS.

Acknowledgments

This work is funded by the German Research Foundation (DFG) within the Research Training Group „Role-based Software Infrastructures for continuous-context-sensitive Systems“ (GRK 1907).

REFERENCES