Model-Driven Integration and Management of Data Access Objects in Process-Driven SOAs

Christine Mayr, Uwe Zdun, and Schahram Dustdar

Distributed Systems Group
Information Systems Institute
Vienna University of Technology, Austria
christine.mayr@inode.at, {zdun,dustdar}@infosys.tuwien.ac.at

Abstract. In most process-driven and service oriented architectures (SOA), services need to access data stored in a database using database transactions. This is typically done using Data Access Objects (DAOs), but so far the integration of the business process, service, and DAO concepts is not well defined. As a consequence, when the number of services in a SOA grows, the number of DAOs can increase considerably and become hard to manage. In addition to this technical issue, business processes have to be highly adaptable to both functional and technical requirements. We propose a model-driven approach for integrating and managing DAOs in process-driven SOAs. We present a set of models providing different views tailored to the requirements of various stakeholders, such as business experts, database designers, database developers, etc. In process-driven SOAs, process activities running in a process-engine invoke services. We adapt these process flows to model a sequence of DAOs within a service. A DAO repository is used to manage DAOs more efficiently and to improve software reuse in database transaction development. The repository provides functionalities to create, update, retrieve, and delete database transactions. As our view-based models can flexibly be adapted to project requirements, our approach also aims at enhancing maintainability and increasing software development productivity.

1 Introduction

In a process-driven, service-oriented architecture, services are invoked from process activities running in a process engine [1]. In this paper we concentrate on an important part of process-driven SOAs: persisting the business objects (and other data) that is used and manipulated by the processes and services. Nowadays, this is often done by integrating Data Access Objects (DAOs) into services. DAOs are a special kind of objects providing access to data that is usually read or written from one or more database tables. Services invoke the DAOs to commit a database transaction to persistent storage. The goal of this design is to enhance software maintainability and strict separation of the layers providing business functionality and data access in a SOA. In addition, DAOs provide an interface that is independent of the underlying database technology. Common DAO implementations are provided by object-relational mapping (ORM) tools, such as Hibernate [2] or Ibatis [3]. ORM frameworks support developers in mapping data between object-oriented programming languages and relational database management systems (RDBMS).

P. Mähönen, K. Pohl, and T. Priol (Eds.): ServiceWave 2008, LNCS 5377, pp. 62–73, 2008.

As the number of DAOs, as well as the number of uses of a DAO, grows along with the number of services, maintaining and managing the DAOs becomes increasingly difficult when the SOA grows. That is, it becomes hard to deal with tasks such as finding out which services or processes require which DAO, deciding whether a DAO can be discarded because it is not used anymore, or whether a suitable DAO for a specific task has already been defined and can be reused. Different stakeholders involved in a process should be able to understand the SOA only from their perspective. For instance, data analysts require mainly information about which DAOs access which data, service developers require DAOs rather as interfaces to the data, and software architects require the big picture of service/DAO interconnection.

DAOs are an integral part of many SOAs, but unfortunately services and DAOs are not well integrated yet. A straightforward approach to solve this problem are cartridges, such as those provided by AndroMDA [4] or Fornax [5]. Cartridges support separation of concerns by providing mechanisms for accessing and manipulating data through DAOs. They are predefined components for model-driven generators that enable developers to integrate DAOs into services by generating either an instance of a DAO interface into the service code [4] or generating DAO instances into the service operation [5]. However, the relationships between DAO operations and service operations are not specified so far by cartridges. Even though the Fornax cartridge [5] connects DAOs to service operations, it lacks information about which DAO operations are invoked by which service operation. This information, however, is important for stakeholders, such as software architects and service developers, to gain a clear view about which database transactions are invoked by which service operation. To overcome this problem, we extend the cartridge approach with the integration of DAO operations into service operations and the introduction of service operation flows consisting of DAO operations.

In our earlier work, we introduced a view-based modeling framework (VbMF) [6] for separating different concerns in a business process into different views.The idea is to enable stakeholders to understand each view on its own, without having to look at other concerns, and thereby reduce the development complexity. VbMF is based on model-driven software development (MDSD) [7], i.e., executable code, such as BPEL/WSDL in the example, can be generated from the views.

In this paper, we tackle the problems of integrating DAOs into a process-driven SOA by extending the VbMF using different *views* for managing data objects more effectively and efficiently. One important aspect of our solution is the need for a specific view: the Data Access Object Repository View that particularly offers a fast and efficient retrieval and management of DAOs. Establishing a Data Access Object Repository increases both software development productivity and maintainability by enabling loose coupling between DAOs and services, but still supporting the management of DAOs.

This paper is organized as follows: First, we present our approach for managing and integrating DAOs into process-driven SOAs from the architectural integration point of view. Section 3 provides a specification of the view-based data modeling framework and discusses the model-driven aspects of our solution. We validated the applicability of our approach through a case study in the area of modeling jurisdictional provisions in the context of a district court, described in Section 4. Section 5 discusses related work and finally Section 6 concludes.

2 Architectural overview

In this section we propose an approach for integrating and managing DAOs in process-driven SOAs. We discuss the relevant components and the relationships between them from an architectural point of view. In the next section we explain how to support this architecture using our view-based models. The goal of our integration and management approach is to enable effective software development, extended analysis methods, and efficient management of data-related issues in a process-driven SOA. As shown in Figure 1 the architecture consists of four main components:

1. *Process Flow:* An executable Process Flow, such as the BPEL process flow in the example configuration in Figure 1, consists of process activities that each can invoke a service operation. Process activities act as service requesters that invoke a specific service from a service repository.
2. *Service Repository:* The Service Repository serves providers for publishing and requesters for retrieving and invoking services. When a requester discovers a suitable service, it connects to the specific service provider and invokes the service [8]. In VbMF, the service repository is modeled by the Collaboration View. It supports creating, updating, retrieving and deleting of services.
3. *Service Operation Flow:* When a service operation invokes one or more DAO operations, the Service Operation Flow orchestrates these DAO operation calls by a data flow of basic process elements such as sequences, flows, and switches.
4. *DAO Repository:* The Data Access Object Repository is used as central component for managing DAOs. It provides basic functionality for accessing DAOs and in particular aims at efficiently retrieving a suitable DAO operation.

These components are part of the runtime architecture generated from our models. In MDSD [7] models are used to abstract from programming languages and platform-specific details. In this paper, we use BPEL as an example for a specific process execution language. In the following sections, we focus on the Service Operation Flow and the DAO Repository that are novel contributions of the work presented in this paper.

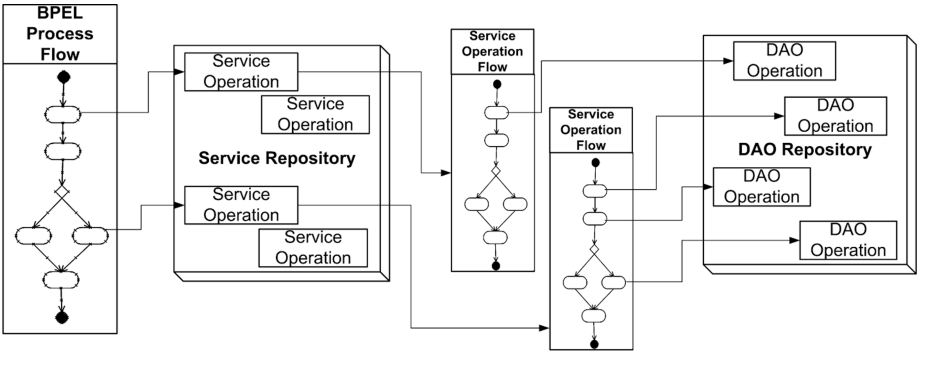

Fig. 1. DAO Service Integration Overview

2.1 DAO Repository

As the number of DAOs can grow considerably large as the number of services grows, retrieving a certain DAO operation, for instance for reusing the operation, can be complex and time consuming. Also, the potentially large number of DAOs must be managed. For instance, it must be decided which DAO is obsolete and which DAOs can be merged. To support these issues, in our integration and management architecture, we established a DAO Repository as a central component for managing DAOs efficiently. Our DAO repository interface provides the following basic functionality:

- *Insert:* Create a new DAO operation
- *Update:* Change operation name or parameter definitions of a DAO operation
- *Search:* Retrieve a DAO operation by certain search criteria
- *Delete:* Remove a DAO operation

The DAO Repository is the central component for publishing and discovering DAO operations – the functional parts of a DAO. Within the DAO repository each DAO operation belongs to one or more database tables and in the end to a database. This classification enables advanced searching capabilities. Listing 1.1 shows a few example queries for selecting DAO operations by different search criteria. The query `getDaoOperation` returns a list of DAO operations matching a pattern describing a DAO operation name. The second query operation `getDaoOperationByDAO` returns a list of DAO operations belonging to the given `daoObject` type. The third query `getDaoOperationByDbName` returns a list of DAO operations by a given pattern describing the name of a certain database instance. Finally, the query `getDaoOperationByTableName` returns a list of DAO operations accessing a certain database table. In Section 4 we apply the DAO repository using a concrete example.

```
DaoOpList[] getDaoOperation(String pattern)
DaoOpList[] getDaoOperationByDAO(DAOObject daoObject)
DaoOpList[] getDaoOperationByDbName(String pattern)
DaoOpList[] getDaoOperationByTableName(String dbPattern, String tablePattern)
```

Listing 1.1. DAO Repository: Query examples

2.2 Service Operation Flows

In this section, we describe the Service Operation Flows (see Figure 1) in more detail. A Service Operation Flow supports separation of concerns by enabling service developers to extract the database transactions from the service implementation. This way, service developers get a clearer understanding about the data flows within a service. As a result, development complexity decreases and higher-quality documentation can be generated.

To specify Service Operation Flows we have to integrate DAO operations into service operations. Cartridges, such as AndroMDA [4] and Fornax [5], relate DAOs to services rather than service operations to DAO operations. Establishing the relationships between service operations and DAO operations provides the basis to enable advanced analysis capabilities of database transactions in process-driven SOAs. For example, we can estimate the average number of calls of a specific DAO operation within a process

flow for tuning technical database parameters or database indexes. Database indexes can significantly improve the performance of database transactions, but it must be considered that update costs have a considerable influence, both in the context of the physical database design and in access path selection for RDBMS query optimization [9]. The specification of a database transaction provides the information about which database indexes are required for a specific application. However, database transactions that are never invoked must not be selected for index creation.

A DAO consists of a set of DAO operations, and a service consists of `0..n` service operations. When a service operation needs access to the database, it invokes a DAO operation from the DAO repository. However, often a service operation invokes more than one DAO operations that can be dependent on each other. Therefore we introduce Service Operation Flows consisting of database transactions (DAO operations). Our Service Operation Flows support at the moment sequence and switch elements, but can be extended with any other kind of data flow structure. We show a detailed example of a Service Operation Flow in Section 4.

3 Model-Driven Integration of DAOs into Process-Driven SOAs

In this section we present a model-driven solution for the proposed integration and management architecture of DAOs in process-driven SOAs presented in Section 2. For that purpose we introduce the View-based Data Modeling Framework (VbDMF) as an extension of the Viewbased Modeling Framework (VbMF) described in detail in [6]. The rectangles in Figure 2 display models of VbMF and the ellipsoidal boxes denote the additional models of VbDMF. In VbMF new architectural views can be *designed*, existing models can be *extended* by adding new features, views can be *integrated* in order to produce a richer view, and using *transformations* platform-specific code can be generated. As displayed by the dashed lines in Figure 2 the new view models of VbDMF extend basic VbMF views namely the Information View, the Collaboration View, and the Control-Flow View. The dotted lines in Figure 2 are used to display view integration, e.g., the Service Repository View integrates the Collaboration and Information View to produce a combined view.

We describe the resulting view models specifying the architectural components in Figure 1. Following VbMF's concepts, we distinguish low-level, technical views from

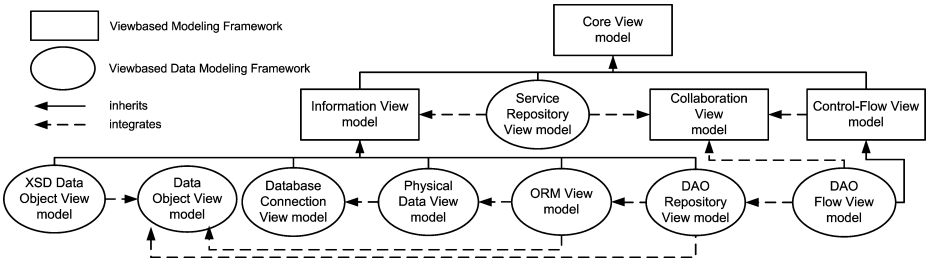

Fig. 2. VbDMF and VbMF – Overview

high-level, conceptual (i.e., often: business-oriented) views. In addition, our low-level technical view models support separating technology-specific from technology-independent views, both for presenting the information in the models to different stakeholders and for supporting platform-independence via model-driven software development.

3.1 Control-Flow View (BPEL Process) – High Level

The Control-Flow View is an essential part of the Viewbased Modeling Framework [6]. It extends the Core View model (see Figure 2) that defines basic elements of the framework. VbMF introduces this Core View model that is derived from the Ecore meta-model [10] to define and maintain relationships with other view models. The Control-Flow View describes the control-flow of a process, so we can for instance apply it for specifying the BPEL Process depicted in Figure 1. As shown in the figure, a BPEL Process activity can invoke a service operation from the Service Repository to realize the task of the activity. The relationship between a process-activity and a service operation is modeled by the Control-Flow View model by view integration with the Collaboration View model.

3.2 Service Repository View (Service Repository)– High Level

The Service Repository View integrates the Collaboration View and the Information View. Both view models belong to VbMF [6] and are derived from the Core View model (see Figure 2). The Collaboration View basically defines service operations required by a process activity. It defines the information needed to generate a Web Service Description Language document (WSDL). The Information View specifies the service operations in more detail by defining data types and messages. Technically speaking, the data types of the Information View are used to generate a schema definition (XSD). In distributed systems, data is passed from one system to another. Each system has its own underlying data structures. For this purpose we specify data type mappings to support data interoperability between diverse systems:

XSD Data Object View. The XSD Data Object View specifies conversions between Web Service Description Language (WSDL) Schema types and data types of the service providers' software system environment. For this purpose a class `XsdObjectMapping` associates each `XsdAttribute` with an `Attribute` of a locally defined Data Object.

3.3 DAO Flow View (Service Operation Flow)– High Level

The DAO Flow View model extends the Control-Flow View model in order to specify Service Operation Flows, as illustrated in Figure 1, and integrates the Collaboration View model to associate each flow with a specific service operation. The primary entity of the Control-Flow View is the `Activity` class that is the base class for other classes such as `Sequence`, `Flow`, and `Switch` [6]. We extend the class `Activity` to associate a service operation `Operation` of the Collaboration View with a flow of DAO operations. The Control-Flow View model uses the class `SimpleActivity` for

representing a concrete process activity [6]. By extending the `SimpleActivity` class of the Control-Flow model we can associate each activity `SimpleActivity` with a `DAOOperation`.

3.4 DAO Repository View (DAO Repository)– Low Level

The DAO Repository View is a combined view that integrates the Object Relational Mapping (ORM) View model and the Data Object View model. Since the main purpose of the ORM View is to map physical data to data objects, it consists of both the Physical Data View model, integrating the Database Connection View and the Data Object View. As a result of this view integration, a DAO Repository service can process complex queries for retrieving a specific DAO operation by joining the data from different views (see Section 2.1). DAOs provide an interface that is independent of the used specific ORM technology. That is, this view model specifies a conceptual view rather than a technical view. It consists of a list of `DAOOperation` elements that each holds `0..n InputParameter` parameters and a `ReturnParameter`.

Database Connection View. The Database Connection View comprises a list of arbitrary, user-defined connection properties and therefore is a conceptual rather than a technical view. We also support database driver dependent views through model extension, e.g., a JDBC Database Connection View.

Physical Data View. The Physical Data View contains a class `Database Connection Pool` specified by a list of the class `DBConnection`. We reference the class `DBConnection` of the Database Connection View using model integration mechanisms (see Figure 2). The Physical Data View contains two more basic classes: `Tables` and `ColumnTypes`. We support most common data types for current RDBMSs. As data types can differ among different RDBMSs, developers can create a technology-dependent view by extending this conceptual view model.

Object Relational Mapping View. The Object Relational Mapping View is a technology-dependent model that provides the basis for specifying object relational mapping mechanisms in VbDMF. The defined elements result from studying a range of ORM tools in particular Ibatis [3] and Hibernate [2]. In order to support ORM framework's special features, developers should specify a technology-dependent view by model extension. The basic view specifies a mapping between the two below-mentioned models – the Data Object View model and the Physical Data View model. The class `DataObjectToTableMapping` maps a data object (`DataObject`) to a database table (`Table`). The class `MemberVariableToColumnMapping` allows for a more specific mapping between `MemberVariable` and a table `Column`.

Data Object View. In object-oriented programming languages information is stored in the objects' member variables. We provide a conceptual, technology-independent model, that consists primarily of a list of data objects `DataObject` and types `MemberVariableTypes`. Again, to define additional data types developers can extend this view model to gain a technology-dependent view.

4 Case Study

In this section we present a case study in the area of modeling jurisdictional provisions in the context of a district court, which we have performed to validate the applicability of our approach. First of all, let us explain the Business Process flow and an exemplary Service Operation flow (see Figure 3) at the land registry court. We use UML extensions to model our process flows.

As shown in Figure 3, the process starts when a new application is received. The `ValidateApplication` activity invokes a service that checks the incoming jurisdictional application for correct syntax and semantic. Successfully validated applications are saved by the service triggered by the `SaveApplication` activity. The activity `DeliverDismissal` invokes a service that returns incomplete or inconsistent applications back to the applicant. Stored applications can be executed by the registrar within the human process activity `ExecuteApplication`. If the registrar approves the application, the service-based activity `AccountFees` is invoked. Finally, after accounting the fees, the next activity `DeliverApprovalAndFees` calls a service that delivers an approval and, if required, a pre-filled payment form to the applicant. In case of dismissal the activity `DeliverDismissal` invokes a service that informs the applicant about the dismissed application.

Let us now illustrate how to integrate DAO operations into service operations using the process activity `AccountFees` as an example to demonstrate the database transaction flow within an activity. In Figure 3 the dashed lines indicate the data flows between the DAO operations. The DAO operation `isExemptedFromFees` of `ApplicantDAO` checks whether the applicant is exempted from fees. If the DAO operation `isExemptedFromFees` returns `true`, no further operations are necessary

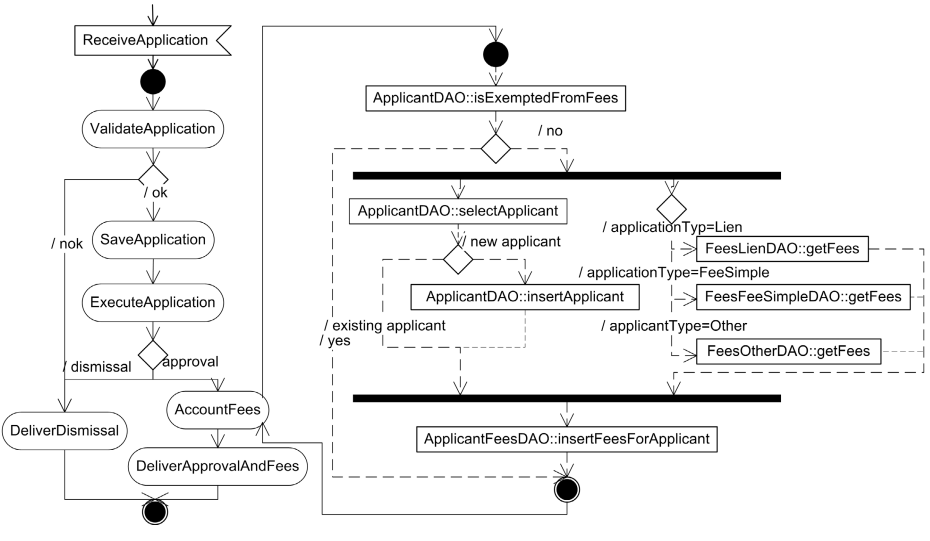

Fig. 3. Case Study: Process Flow and Service Operation Flow (Process Activity AccountFees)

Table 1. Case Study: DAO Repository View

DAO	DAO operation	Transaction Type	Database	Table
ApplicantDAO	isExemptedFromFees	select	DBTest1	Applicant
ApplicantFeesDAO	insertFeesForApplicant	insert	DBTest1	ApplicantFees
ApplicantDAO	isExemptedFromFees	select	DBProduction1	Applicant
ApplicantDAO	selectApplicant	select	DBProduction1	Applicant
FeesLienDAO	getFees	select	DBProduction2	FeesLien

to terminate the flow. Otherwise the DAO operations for accounting fees and selecting the applicant can run in parallel, because their data-flows are independent from each other. Fees are accounted by the fees department and are system-dependent on the type of application: Fees for applications of type 'Lien' are calculated by the DAO operation `getFeesForApplicationType` of DAO `FeesLienDAO`, the DAO operation `getFeesForApplicationType` of DAO `FeesSimpleFeeDAO` accounts fees for applications of type 'FeeSimple', and the DAO operation `getFeesForApplicationType` of DAO `FeesOtherDAO` calculates the costs for applications of type 'Other'. The DAO operation `insertFeesForApplicant` of DAO ApplicantFeesDAO requires a stored applicant as input parameter. For this purpose the `ApplicantDAO`'s DAO operation `selectApplicant` looks for an existing applicant. In case the applicant is not stored in the database yet, the DAO operation `insertApplicant` is invoked to return the new applicant required as an input parameter for the DAO operation `insertFeesForApplicant`. Let us now consider the main view models instances specifying the process flow illustrated before:

Control-Flow View instance. In our prototype implementation the Control-Flow View specifies a BPEL Process Flow. A graphical layout of the resulting BPEL model instance is depicted left in Figure 3. The BPEL source code that is not shown here represents another layout of this view instance.

Service Repository View instance. The Service Repository View models the services and data types of services often invoked by a process activity. In our concrete example, the Service Repository View instance specifies the services depicted left in Figure 3.

DAO Repository View instance. Table 1 shows an extract of the data stored in the DAO Repository after modeling our example process. Due to the view integration mechanisms (Database Connection View, Physical Data View, etc.) this DAO Repository View instance contains data of various categories (DAOs, database, tables, etc.). In our example we query all DAO operations that belong to a certain database 'DBTest1'. In another query we can ask for existing DAO operations depending on a specific table such as 'Applicant'.

DAO Flow View instance. The DAO Flow View instance depicted in Figure 3 illustrates the connection between the process activity 'AccountFees' and its Service Operation Flow. In general, the DAO Flow View is intended for software architects, data analysts, and service developers to get both a general understanding of the data flows within

service operations and to get the information which service operation depends on which DAO operations. Furthermore we can use this view to query all DAO operations that are needed by a specific process.

We use the oAW's Xpand language for source code generation. A BPEL definition for the process flow and a service description in WSDL are generated from an instance of the Control-Flow View, the Information View, and the Collaboration View respectively. The Apache Axis2 Web services engine [11] supports us in building Java proxies and skeletons for the services with WSDL descriptions. So we can generate a service implementation and use the DAO Flow View model instance to inject the flow of DAO operations into the relevant service operations. The DAOs themselves are generated from plenty of model instances, namely of the DAO Repository View, the ORM View, and the Data Object View. In contrast to generating DAOs, the DAO interfaces are automatically implemented simply from the DAO Repository View instance and the Data Object View instance.

5 Related Work

As mentioned before, our approach is related to other model-driven solutions for integrating DAOs into services, such as AndroMDA's EJB3 cartridge [4], generating a persistent tier by integrating DAOs into services, and the Fornax platform [5], aiming at a more specific integration by modeling the relationships between DAOs and service operations. In our solution we associate DAO operations with service operations and thus provide a more in-depth integration solution than these cartridges. Furthermore, in contrast to earlier model-driven approaches, in our approach a data flow of database transactions is modeled within a service operation that can be used to extract data dependencies from the whole business logic.

Our work aims at integrating DAOs into process-driven SOAs, so it is concerned with both processes that typically invoke services and with services that can access data. Ricken's top-down approach [12] addresses the same concern by adding service-orientation to process models to support IT developers in translating business needs into executable code. In [13] a set of architectural and software design models based on proven patterns is introduced for process-driven SOAs. Both approaches, however, do not separate different views or focus specifically on data-related views.

Akin to the approach by Wiederhold [14] our approach uses a mediator-based architecture for integrating data stored in heterogeneous database systems. As the DAO concept provides an abstract and technology-independent interface for accessing data, Wiederhold's mediators enable the homogeneous data access by integrating and selecting necessary data among different sources. In the proposal of Roth and Schwarz [15], a wrapper encapsulates the underlying data and acts as a mediator between the data source and the middleware. In contrast to these mediator approaches, we propose a more abstract, higher-level approach by using a DAO repository for managing DAO operations. Kurz et al. provide a schema-level data integration which specifies how to select, integrate, and transform data from heterogeneous data sources [16]. Like our solution, this modeling approach provides a specification in a user-friendly and platform-/language-independent way. In Section 2 we presented our architectural con-

cept for managing DAOs in process-driven SOAs. These architectural components are supported by technology-independent view models and their technology-specific extensions. As in our approach, Marcos at al. [17] support two different aspects. They distinguish between platform independent (PIM) and platform specific models (PSM) and separate models according to distinct aspects. Besides Marcos et al. [17], there are several approaches for including software architecture into MDA platform. For example, Alti et. al [18] integrate software architecture concepts into MDA platform by a UML profile for Component-Object based Software Architecture extending UML 2.0.

Our view models extend the VbMF [6] to integrate data-related views. Mendling et al. propose a similar approach for efficient view integration. They identified formal semantic associations between elements of the process view. Just like VbMF, our data-related views, in contrast, use a name-based matching algorithm to integrate views. For the establishment of the DAO repository we were inspired from current web service registry standards such as UDDI [19], ebXML [20] and WSIL [21]. EbXML Web Service Registries [20] have interfaces that enable submission, query, and retrieval of the contents of the registry. We adopted the fundamental interface abstractions, used in these approaches, to integrate the DAO repository into our process-driven architecture.

6 Conclusion and Future Work

In this paper we identified current problems in managing DAO operations in process-driven SOAs. In order to efficiently manage and integrate data into process-driven SOAs, we proposed an architecture, that consists of four main components namely the BPEL Process flow, the Service Repository, Service Operation Flows, and a DAO Repository. We further provide a model-driven solution to support this architecture by specifying a set of view models. As our view models are based on VbMF, each model represents a specific view tailored to the requirements of specific stakeholders. In particular, we introduced a view model for specifying database transaction flows to extract data flows from the whole business logic. Up to now, no standard retrieval or submission interface for DAO repositories has been defined. As the number of services grows, data development complexity increases with the number of data access object operations. Hence, retrieving a particular DAO operation can be complex and time-intensive. More powerful searching capabilities, such as those that can be provided on top of our approach, are hence desirable. However further work is necessary for specifying the requirements for a DAO Repository in detail. Further work also includes runtime statistics for measuring how often a DAO operation has been invoked, etc.

Acknowledgement. This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

References

[1] Zdun, U., Hentrich, C., van der Aalst, W.: A survey of patterns for service-oriented architectures. International Journal of Internet Protocol Technology 1(3), 132–143 (2006)
[2] Hibernate: Hibernate (2006), `http://www.hibernate.org`
[3] Ibatis: Ibatis (2006-2007), `http://www.ibatis.org`

[4] AndroMDA: AndroMDA EJB3 Cartridge (August 2007), http://web.aanet.com.au/persabi/andromda/
[5] Fornax-Platorm: Fornax-Platform Cartridges (August 2006), http://www.fornax-platform.org/cp/display/fornax/Cartridges
[6] Tran, H., Zdun, U., Dustdar, S.: View-based and model-driven approach for reducing the development complexity in process-driven SOA. In: Abramowicz, W., Maciaszek, L.A. (eds.) Business Process and Services Computing: 1st International Conference on Business Process and Services Computing (BPSC 2007), Leipzig, Germany, September 25-26, 2007. LNI, GI, vol. 116, pp. 105–124 (2007)
[7] Völter, M., Stahl, T.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)
[8] Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 43–58. Springer, Heidelberg (2003)
[9] Schkolnick, M., Tiberio, P.: Estimating the cost of updates in a relational database. ACM Trans. Database Syst. 10(2), 163–179 (1985)
[10] Eclipse: Eclipse Modeling Framework (2006), http://www.eclipse.org/emf/
[11] Apache Software Foundation: Axis2/Java (2004-2008), http://ws.apache.org/axis2/index.html
[12] Ricken, J.: Top-down modeling methodology for model-driven soa construction. In: OTM Workshops, vol. (1), pp. 323–332 (2007)
[13] Zdun, U., Dustdar, S.: Model-driven and pattern-based integration of process-driven soa models. Int. J. Business Process Integration and Management 2(2), 109–119 (2007)
[14] Wiederhold, G.: Mediators in the architecture of future information systems. Readings in agents, 185–196 (1998)
[15] Roth, M.T., Schwarz, P.M.: Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In: VLDB 1997: Proceedings of the 23rd International Conference on Very Large Data Bases, pp. 266–275. Morgan Kaufmann Publishers Inc., San Francisco (1997)
[16] Kurz, S., Guppenberger, M., Freitag, B.: A uml profile for modeling schema mappings. In: ER (Workshops), pp. 53–62 (2006)
[17] Marcos, E., Acuña, C.J., Cuesta, C.E.: Integrating software architecture into a mda framework. In: Gruhn, V., Oquendo, F. (eds.) EWSA 2006. LNCS, vol. 4344, pp. 127–143. Springer, Heidelberg (2006)
[18] Alti, A., Khammaci, T., Smeda, A.: Integrating software architecture concepts into the mda platform with uml profile. Journal of Computer Science 3(10), 793–802 (2007)
[19] Clement, L., Hately, A., von Riegen, C., Rogers, T.: UDDI Version 3.0.2, UDDI Spec Technical Committee Draft (October 2004), http://www.uddi.org/pubs/uddi_v3.htm
[20] OASIS/ebXML Registry Technical Committee: OASIS/ebXML Registry Services Specification v2.0. (December 2001), http://www.ebxml.org/specs/ebrs2.pdf
[21] Ballinger, K., Brittenham, P., Malhotra, A., Nagy, W.A., Pharies, S.: Web services inspection language (ws-inspection) 1.0 (November 2001), http://www.ibm.com/developerworks/library/specification/ws-wsilspec/