Inf Syst Front
DOI 10.1007/s10796-007-9032-9

Integrated model-driven dashboard development

Themis Palpanas · Pawan Chowdhary · Florian Pinel · George Mihaila

Abstract Business performance modeling and model-driven business transformation are two research directions that are attracting much attention lately. In this study, we propose an approach for dashboard development that is model-driven and can be integrated with the business performance models. We adopt the business performance modeling framework, and we extend it in order to capture the reporting aspect of the business operation. We describe models that can effectively represent all the elements necessary for the business performance reporting process, and the interactions among them. We demonstrate how all these models can be combined and automatically generate the final solution. We further extend the proposed framework with mechanisms that can detect changes in the models and incrementally update the deployed solutions. Finally, we discuss our experience from the application of our technique in a real-world scenario. This case study shows that our technique can be efficiently applied to and handle changes in the underlying business models, delivering significant benefits in terms of both development time and flexibility.

Keywords Model-driven development · Dashboard · Business performance management

Work done while Palpanas was a member of the IBM T.J. Watson research Center.

T. Palpanas (✉)
University of Trento,
Trento, Italy
e-mail: Themis@dit.unitn.it

P. Chowdhary · F. Pinel
IBM T.J. Watson Research Center,
Yorktown, NY, USA

P. Chowdhary
e-mail: Chowdhar@us.ibm.com

F. Pinel
e-mail: pinel@us.ibm.com

G. Mihaila
IBM T.J. Watson Research Center,
Hawthorne, NY, USA
e-mail: Mihaila@us.ibm.com

1 Introduction

Enterprises are leveraging information technology solutions in order to increase their productivity and their business value in the marketplace by describing and monitoring their business operations. Nowadays, many vendors provide sophisticated tools to represent business process models (Kumaran 2004; McGowan and Bohmer 1993) and business activity monitoring models (Jeng et al. 2002, 2003). *Business Performance Management (BPM)* (Zeng et al. 2005) includes a suite of components that are used to monitor the health of the business, and offers them the ability to react promptly to changes in their environment (Bhattacharya et al. 2005).

The integration of various systems in the business allows for continuous monitoring of business performance, using carefully selected metrics, also known as Key Performance Indicators (KPIs). The KPIs are displayed to the analyst through a dashboard, that is, a user interface that organizes and presents information in a way that is easy to read and interpret.

In contrast to the usefulness and ease of use that dashboards represent, the amount of effort that is required for their development can sometimes be daunting. User interface development in general, and dashboard development too, requires a considerable investment of time, and can often take as much as 65–80% of the overall development time in a model-driven business transformation project (Sukaviriya et al. 2005).

In this paper, we claim that dashboard development can be fast and easy, while maintaining flexibility in the design, and without sacrificing versatility or performance. We propose a framework for dashboard design that is model-driven. This framework includes a number of user-customizable models that can effectively capture the functionality of a dashboard. We provide different models for modeling the data, the users and their data access privileges, and the navigation among the various data views.

Once the user has designed the dashboard with the desired functionality using the provided models, our framework is able to automatically generate code for the deployment of the dashboard, leaving only minor customization issues for the developer. The generated code covers all the aspects of the dashboard, such as the following.

- Management of the data to be displayed, involving the creation of relevant databases.
- Design of different views of the data, and of the navigation among those views.
- Assignment of access privileges to the users of the dashboard, so that each user can only access the data and views that are relevant to her.

Our approach allows the developer to focus on the dashboard functionality, and relieves her from the burden of the user interface development experience (Myers et al. 2000). The benefits of model-driven dashboard development include the graphical representation and easy manipulation of the solution, the error free code-generation, and the ability to capture the changes in business reporting processes quickly and cost effectively. To the best of our knowledge, this is the first comprehensive approach for model-driven dashboard design, and in Section 6 we demonstrate its application to a real-world problem.

The contributions of this paper can be summarized as follows.

- We describe a framework for model-driven dashboard design. The models we employ cover the many facets of this process, that is, the data to be displayed, the users of the system, the roles and access privileges of each user, the content of each dashboard page view, and the navigation among those views.
- The method we propose is complementary to business process and business performance modeling, and extends such models to provide a seamless experience.
- Our framework enables the automated generation of all the code necessary for the deployment of the dashboard. Therefore, it removes the burden of tedious programming, and it significantly reduces the time required for delivering the solution.
- We explicitly handle updates within the proposed framework, therefore, making changes to the dashboard design painless. Changes only have to be made in the high-level models, and then the new code is automatically regenerated. Furthermore, our mechanisms support incremental maintenance of the deployed solutions, thus, changes can be integrated in an efficient manner.
- Finally, we validate our approach using a real-world scenario. We discuss our experiences from applying the proposed method to a real problem, and demonstrate the benefits of our technique with regards to development time and flexibility of the solution.

The rest of this paper is organized as follows. We review the relevant literature in Section 2, and we discuss some necessary background material in Section 3. In Section 4, we present in detail the models and the process we employ in our framework, and in Section 5 we describe the mechanisms we have put in place for handling changes in the models. that In Section 6, we present a case study, where we apply our technique to a real-world problem, and we conclude in Section 7.

2 Related work

There is a growing trend in using model-driven methodologies (Kleppe et al. 2003; Miller and Mukerji 2003) for developing large system software, due to their high level abstraction and code re-use (or regeneration). They have been widely applied in related areas, such as software reuse (Frakes and Kang 2005; Greenfield et al. 2003), reverse engineering (Rugaber and Stirewalt 2004; Yu et al. 2005), and user interface design (Sukaviriya et al. 2005). The benefits of adopting model-driven design include reduced software development time, enhanced code quality, and improved code maintenance (Kleppe et al. 2003; Czarnecki and Helsen 2003). There are also numerous related works about business processes. Business process management enables the management and analysis of operational business processes (van der Aalst et al. 2003).

Business processes can be implemented using a workflow or a state machine model (Koehler et al. 2002). BPEL (Business Process Execution Language for Web Services, http://www-128.ibm.com/developerworks/library/ specification/ws-bpel) defines a program understandable language to represent business processes for web service environments. Yet, BPEL can only orchestrate the flow execution; business data are still not synchronized, correlated, or linked together for the auditing and analysis purposes.

An approach that tries to overcome the above shortcomings is the Model-Driven Business Transformation (MDBT) (Kumaran 2004, Kumaran and Nandi 2005). MDBT models business operations from the point of view of a business analyst, without regard to existing or planned information technology solutions. In other words, an MDBT operation model is a truly computation independent model.

Change management has been studied in various different contexts (Mumick et al. 1997; Palpanas et al. 2002; Marian et al. 2001). In all the cases, the main goal is to reduce the time and effort needed for incorporating the changes to a minimum. This is achieved by identifying which specific parts of the system are affected by the changes, and only updating those ones. In this study, we follow the same general principles, but apply them in a new environment.

There is also much interest around the concept of dashboards, with several companies providing relevant solutions, such as IBM (AlphaBlox, http://www.alphablox.com/), Business Objects (http://www.businessobjects.com/), and Hyperion (http://www.hyperion.com/). Nevertheless, these approaches do not integrate with the business process and business performance models, requiring much effort to develop and maintain. In contrast, we propose a method for dashboard design that is model-driven. The high-level models we define integrate seamlessly with the business performance models, leveraging the common parts of the design, and enabling an end-to-end design process.

3 Background

In addition to espousing a business artifact-centric approach to operation modeling, MDBT (Kumaran 2004; Kumaran and Nandi 2005) offers a model-driven development toolkit and technique. The tools automatically transform an operation model into a platform-independent solution composition model in UML2. In this stage of modeling, the solution architect fills in much of the IT detail that is outside the domain of the business analyst. These details include integration with external services as well as role-players. Following the completion of the solution composition model, MDBT code generation tools automatically create J2EE components that manage the process and provide a simple user interface by which users can interact with the solution. The automated transformations and code generation enable rapid prototyping, accelerate the development cycle, and allow for a fast turnaround iterative development regimen.

The solution composition model also provides the platform on which an observation model can be constructed. The elements of the observation model (e.g., events) are linked to those of the solution composition model (e.g., states and transitions) so as to define how the performance metrics will be gathered.

Business Performance Management (BPM) (Chen et al. 2006; Chowdhary et al. 2005) is an effective means of monitoring business processes. Model-based BPM normally includes an observation model that conforms to a pre-defined meta-model, such as the one provided by MDBT, which we discussed above. Entities such as input events, metrics, outbound events, situation detectors, and actions can be composed, monitored, and scheduled through the observation model. Using BPM, we can detect bottlenecks of business operations in real-time, and identify anomalies by correlating event sequences. Based on the observation model, actions triggered by the above situations may involve generating alerts or displaying statistics and aggregated information onto a dashboard.

In previous work, we implemented a BPM solution based on the model driven development methodology (Zeng et al. 2005). There are two approaches that we adopted for representing a BPM solution. The first approach utilizes the Unified Modeling Language (UML) with UML2 profile extension. The second approach utilizes XML schemas for defining BPM entities and the relationships between the entities. Both approaches are implemented as plug-ins on IBM Rational Software Architecture (RSA).

Although the work we describe in this paper fits under the general framework of MDBT and BPM, and we discuss it within this context, we stress that our approach is *not* tied to this framework in any way. As we explain in more detail in the next section, we have defined an XML interface that allows our method to operate with any other business process modeling framework.

4 Model-driven dashboard framework

In Fig. 1 we depict the high level architecture of the proposed dashboard framework. As mentioned earlier, the framework extends the existing BPM model in order to support the dashboard reporting needs. Specifically, we extend the BPM Observation Model (OM), one of the UML Models of MDBT Toolkit that captures the monitoring and alerting requirements of an enterprise. In order to visually represent these requirements as models, the OM makes use of the UML2 profiles to extend the base UML elements. The Dashboard Model employs similar techniques to represent its modeling elements, so that the solution designer gets to work with consistent models for the entire, end-to-end solution design. The models capture the following aspects of the BPM Dashboard.

- Definition of metrics and related context information to be displayed on the dashboard.

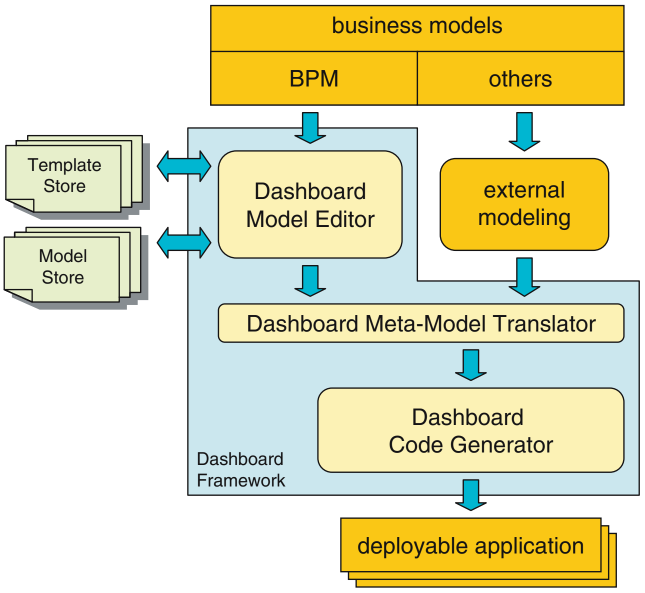


Fig. 1 Model-driven dashboard framework

- Organization of information into pages, and definition of navigation paths among these pages.
- Assignment of access control privileges to the dashboard information, depending on the user roles.

In order to capture the UML representation of Dashboard Models, we used the Rational Software Architect (RSA) tool. Note, however, that RSA can be interchanged with any other editor supporting the UML2 notations.

Even though we use UML for all the modeling requirements in our framework, we also provide an equivalent XML representation, which serves as our meta-model. In fact, the representation that the proposed approach uses internally is the XML representation. The transformation between the UML and the XML representations is lossless, in the sense that all the modeling elements and the relationships among them are preserved.

The decision to have the Dashboard XML Meta-Model as an additional level of abstraction allows us to decouple the dashboard modeling process from the modeling of the rest of the business processes. Therefore, changes in the OM will not affect the Dashboard Framework. Moreover, we may replace the OM with any other business modeling approach, without affecting the dashboard model. This option is represented in Fig. 1 by the box labeled "external modeling."

When the Dashboard Model has been transformed into the Dashboard Meta-Model representation, we feed this representation to the Code Generator, which subsequently produces the deployable dashboard application (refer to lower part of Fig. 1). The generated application consists of the Dashboard Application, which is the set of files that contain the actual code for the application, and the Dashboard DDL, which is the set of files that generate the auxiliary structures needed by the application, such as database tables. These tables are required to be created in the BPM Data warehouse.

The Dashboard Application can be readily deployed on a J2EE application server. The particular choice of the application server is orthogonal to our solution, and the Code Generator can be modified to generate deployable components for any application server.

Figure 2 shows the overview of the end-to-end dashboard-design process flow. We start by defining custom reports to be used by the dashboard, or by simply selecting some of the predefined reports from the template data store. As we will discuss later, the role of these report templates is to retrieve the

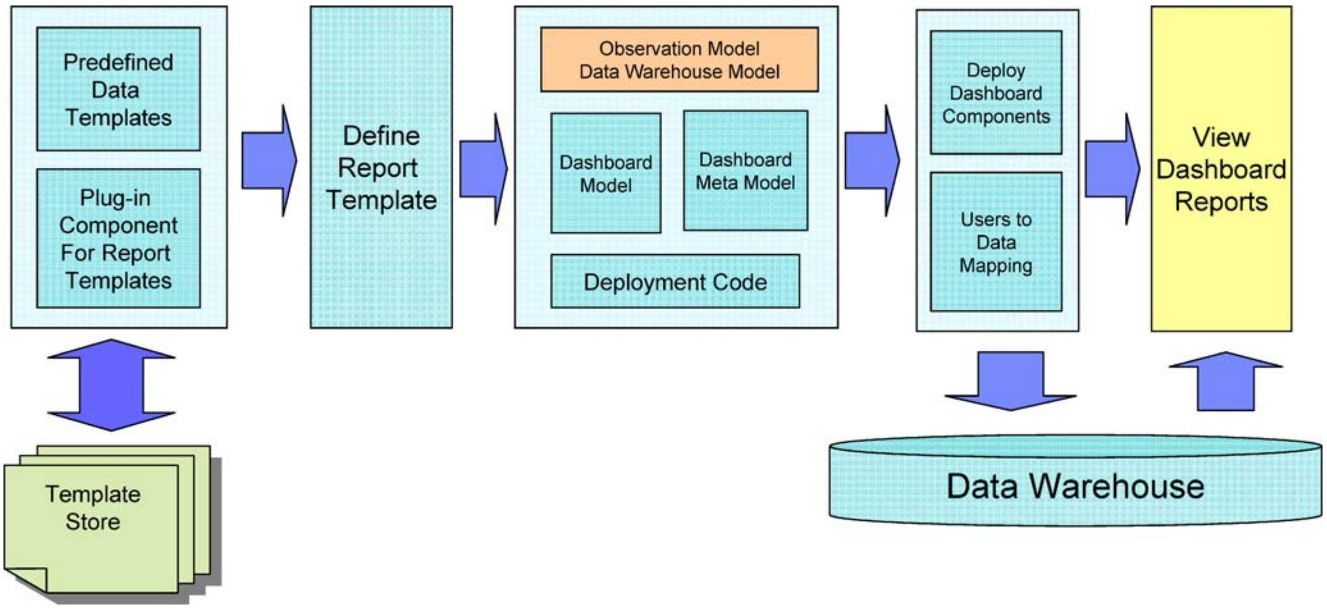


Fig. 2 End-to-end dashboard component flow

appropriate data and handle the presentation of these data on the screen. Then, the solution designer models the dashboard elements using the Model Editor, transform the result into the Dashboard Meta-Model representation, and invoke the Code Generator to generate the deployable software components. Once deployed, the Dashboard can be accessed using a web browser. The details of the different Dashboard Model elements are discussed in the subsequent sections.

4.1 Dashboard model artifacts

The dashboard model artifacts used in our approach can be classified into three categories. The first category is related to modeling the data that are necessary for the dashboard. It includes the data and the metric models. The second category corresponds to an abstract presentation layer, including navigation and the report template models. Finally, the third category is related to user roles and data access privileges. It includes models that define the dashboard access control, by relating user roles to data elements, as well as elements in the presentation layer.

In the following paragraphs, we elaborate on the different model artifacts.

4.1.1 Dashboard model definition

As discussed earlier in the paper, we chose to use UML for the entire dashboard modeling requirements as it is widely accepted in the industry, and also because it provides to the solution developer a consistent platform to work with, across the various MDBT models. In order to accommodate our needs for the Dashboard Framework, we have extended the UML meta-classes and relationships by introducing new stereotypes using UML2 profiles to model the dashboard elements.

Dashboard data model In our framework, we assume that all the necessary data can be stored in a data warehouse, using a star schema (Gray et al. 1996). Therefore, we use the data model shown in Fig. 3, where each data element is marked as either a dimension, or a metric. In Section 5, where we discuss issues related to management of changes in our framework, we elaborate on the data model and its implementation.

Even though the data model we support is simple, its semantics are rich enough to be able to model many real-life scenarios. This is because it is usual for real-world data-modeling problems (especially the ones that we are targeting) to have a natural star-like representation. An example scenario is product sale information, where the metrics include number of units sold and revenue, and the dimensions include geographies and time.

In Fig. 3 we are also introducing the Metric Group modeling element, which is used for grouping relevant metrics. Such a grouping is useful when modeling relationships to other artifacts, where all the members of the Metric Group participate. Figure 3 depicts the Metric Group UML class that connects to the Metric class in an aggregation relationship.

Dashboard navigation model In Fig. 4, we illustrate the GUI modeling Elements (stereotypes), that is, the Navigation Tree, Page, and Menu classes. These three classes form the Dashboard Navigation Model. In a typical scenario, the analyst starts by defining some pages, and she then associates these pages with menus. In the last step, she introduces a Navigation Tree element, in order to capture the navigation paths among the pages, which eventually form the Dashboard reports.

Dashboard report template model The Report Templates are used to define the information content of the individual pages. Figure 5 shows that a Report Template can be associated with a page, and may refer to several Metric Groups. When the page is displayed on the dashboard, the information about all the metrics corresponding to the templates is rendered on the screen. Note that each page can be associated with one or more Report Templates.

The Report Templates also define the details for the visual presentation of the data they contain. By creating a report template, the user can choose to display a set of metric data as a table, as a chart, or using both display modes.

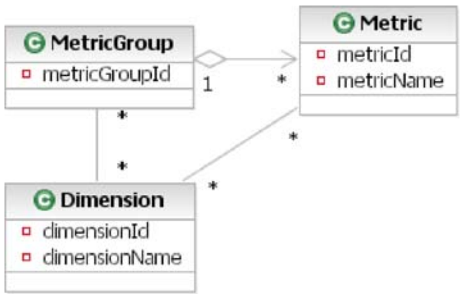


Fig. 3 Dashboard data model

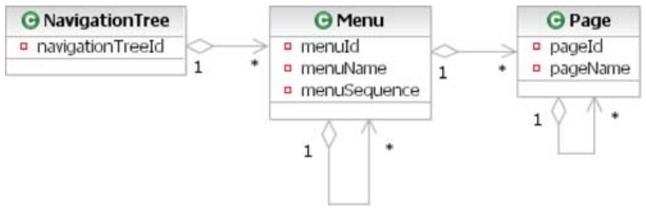


Fig. 4 Dashboard navigation model

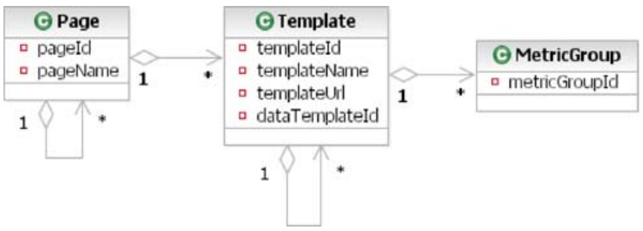


Fig. 5 Dashboard report template model

Dashboard access control model This model defines all the access control properties relevant to the dashboard. Using the various modeling elements, we can specify for each user role the access privileges to different parts of the data, as well to different pages of the dashboard. Thus, the dashboard users, according to their assigned roles, only have access to a subset of the dashboard reports.

Figure 6 illustrates how we model the above requirements in our framework. The business analyst can model the access privileges to the reporting data according to User Role (such as manager, data administrator, etc.), and by Metric Group and Dimension. We now explain in detail the relationships between user roles and metrics, and user roles and dimensions.

- *UserRole-MetricGroup*: This relationship specifies the access privileges of User Role to Metric Group. When the analyst creates an aggregation link between the above two modeling elements, all the users assigned to User Role gain access to all the metrics described by Metric Group. This lets the model capture the role based access to metrics. At runtime, based on this model, the system can determine what metrics to show on the dashboard based on the User Role (i.e., only those metrics for which the user has access are displayed on the dashboard).
- *UserRole-DimensionScope:* This relationship defines the User Role access privileges to various dimensions, as well as to the dimension levels in each dimension. This lets the business analyst define fine grained access control at the metric context.

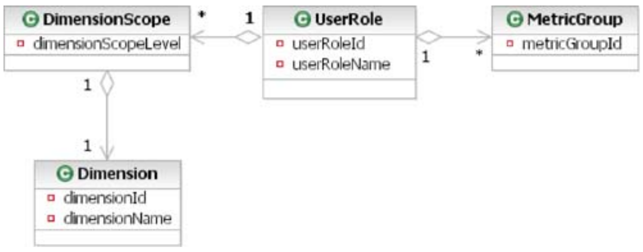


Fig. 6 User role to metric and dimension model

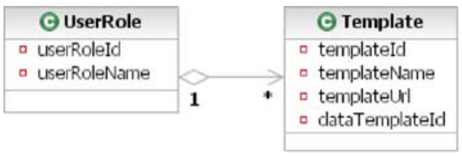


Fig. 7 User role to report template model

When the dashboard has been deployed and is ready for use, the administrator has the ability to further refine the data access control by the specific data values. The details of such access privileges are defined later in the paper (see Section 4.2.3).

Access by Report Template is another aspect of dashboard-report access-control modeling. A User Role may have access to one or more Report Templates, and the business analyst may select a set of (already defined) templates and associate them to the User Role elements. This lets the dashboard framework filter the templates that are shown to the user of the dashboard. Figure 7 shows the User Role to Report Template relationship.

Finally, our framework allows the business analyst to define access control based on the Navigation Trees (Fig. 8). We expect that a single Dashboard Model will involve several Navigation Trees. In this case, the business analyst may wish to provide different access privileges to each one of the navigation trees, according to User Role.

All the access control models discussed in the previous paragraphs comprise a powerful and flexible toolset. Not only do they provide coarse- and fine-grain access control to the data, but they also allow the business analyst to design a small set of pages, which at run-time will display different information, according to the access privileges of the user accessing the dashboard.

4.2 Dashboard model solution methodology

We now turn our attention to the solution methodology we have in place for our Dashboard Framework, and describe the required steps for developing a solution. Even though the model-driven approach brings efficiency to BPM solutions development, there is a need to understand and follow a specific methodology that can lead to a successful and efficient solution.

The Dashboard modeling methodology can be divided in the following three main activities.

1. Pre-modeling activity.
2. Modeling activity.

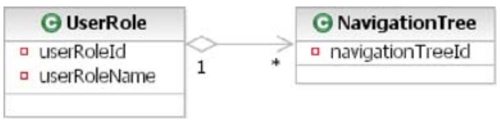


Fig. 8 User role to navigation tree model

3. Post-modeling activity.

In the next paragraphs, we discuss in detail each one of these activities.

4.2.1 Pre-modeling activity

Before starting to create models in order to capture the dashboard requirements, the business analyst is required to understand the predefined components and templates that are included in the Dashboard Framework tool. These components can aid in quickly and efficiently designing the solution.

One of the important parts of this framework is the predefined *data templates* (data structures). Since the data model is only comprised of a well-defined, limited set of data elements (that is, metrics and dimensions), the framework publishes predefined sets of data structures as part of the tool. Then, each report template may choose the data structures that are suitable for its reporting purposes.

The framework provides another software component, the *view component*, which is responsible for connecting the data layer with the presentation layer of the dashboard. The view component uses the data structure and User Role elements to connect to the data sources, and to generate an instance of the data structure, which during runtime is passed to the Report Template instance (discussed below) that renders the visual widgets. In order to achieve seamless integration, the view components need to be embedded in the Report Templates. In our implementation, they are included as JSP tag libraries.

Finally, we also provide a set of predefined Report Templates. In the current version of our tool, we offer a table and a chart component. Our framework can also support user-defined Report Templates. The only restriction is that the new template has to support the data templates in its input.

4.2.2 Modeling activity

After the custom Report Templates have been defined, the next step is to model the reporting requirements. During this step, the user may need to perform the following tasks.

- Identify the metrics that will become part of the dashboard views, and create Metric Groups by grouping together similar metrics.
- Create report templates for all the different types of information that are to be displayed on the dashboard.
- Create page elements, and associate them to one or more of the report templates defined earlier.
- Create the menu elements for the dashboard portal, and link the menu items with the corresponding pages. Finally, introduce navigation tree elements in order to define the navigation flow of the portal.
- Define the different user roles that need access to the dashboard portal. Individual users are assigned a role by the portal administrator during the portal configuration time.
- Associate each user role with Metric Groups, Dimensions, Report Templates, and Navigation-Trees, so as to specify the access control privileges.

Once the Dashboard Model is ready, it is automatically transformed into our intermediate XML representation, which is independent of the tool used to build the Dashboard Model. Subsequently, this model is processed by the Code Generator that produces all the required deployable software components.

4.2.3 Post-modeling activity

We now discuss the artifacts related to the post-modeling phase. The Code Generator produces two deployable software components, namely, the Dashboard DDL and the Dashboard Application. The Dashboard DDL contains the definitions for all the tables that need to be created in the BPM Data Warehouse. It also contains the necessary SQL scripts for reading data from and inserting data in those tables.

The Dashboard Application is a J2EE application that needs to be deployed on a J2EE Application Server. It contains the web module that consists of the chosen report templates along with other supporting software components provided by the framework.

As the final step in the dashboard deployment procedure, the user has the ability to define fine-grain data access control, according to specific data values of the warehouse. When we discussed access control in the Dashboard Model (see Section 4.1.1.4), we described how the model allows to define access privileges based on the data dimensions. For example, we may allow a particular user role to roll-up and drill-down on the geography dimension. Even though the above kind of access control is very useful, in some cases it may not be enough. Consider the situation where two different managers are responsible for the Europe and America geographies. In this case we may want to restrict the access of each manager to the geography for which she is responsible.

In order to achieve this fine-level access control, we augment the User Role to Dimension model with special annotations that specify the levels of each dimension that can be accessed by the User Role. Note that we cannot perform this step of access control during the modeling phase, because it depends on the specific data of the application, which are only available in the warehouse after the application has been deployed.

5 Change management

We now turn our attention to the problem of change management within the proposed framework. In this case, when we use the word "change," we refer to insertions, updates, or deletions that may occur to any of the Dashboard Models we described in the previous section.

Note that a model-driven development environment offers a de facto advantage in handling changes over the non model-driven approach. This is because changes happen at the model level, which is much easier to manipulate. Therefore, when any of the models change, we can simply re-generate and re-deploy the entire application. This procedure is to a large extent automated, and consequently relatively fast.

Nevertheless, re-generating and re-deploying an entire application can often times be overkill. This is especially true when the changes that trigger this procedure are rather insignificant, or pertain to a small part of the application. In such cases, we would like to be able to propagate only the necessary changes to the deployed applications, thus, minimizing the resulting downtime.

In the following paragraphs, we describe the methods we employ in order to support incremental maintenance within the proposed framework. These methods aim at identifying the portions of the models that have changed, and subsequently, at applying the required changes only to the affected parts of the deployed applications. We focus our discussion in the incremental maintenance of the data model and the data warehouse, since this forms the basis for the entire Dashboard Model.

5.1 Data warehouse level

As mentioned earlier, the *Observation Model* (OM) covers the business performance management aspects of the business process modeling, and includes business performance monitoring (observation) and control. OMs are typically constructed top–down starting from the business measures or *Key Performance Indicators* (KPI), and are assigned to process specific events (Zeng et al. 2005). For each relevant incoming event, a monitoring context will typically compute one or more measures, and store them in a data warehouse for subsequent analysis. In the context of the data warehouse, some of these measures are treated as dimensions (e.g., customer type, time, and location) and others as metrics (e.g. revenue, cost, and profit).

As part of the model-driven approach to design, the data model, and consequently the database schema for the data warehouse are automatically generated from the OM. Then, when the OM changes, the data warehouse schema needs to be updated. This would normally require the migration of the already collected data to a new data warehouse schema associated with the new OM. Such migration of data would cause unnecessary downtime for the data warehouse and the dashboard application.

What we propose instead is a method for incremental maintenance of the data model, which can efficiently keep the data warehouse up to date with the changes in the OM. This is achieved by extending our models with special annotations that are used to track the changes triggered by the users of the system, and propagate those changes to the data model.

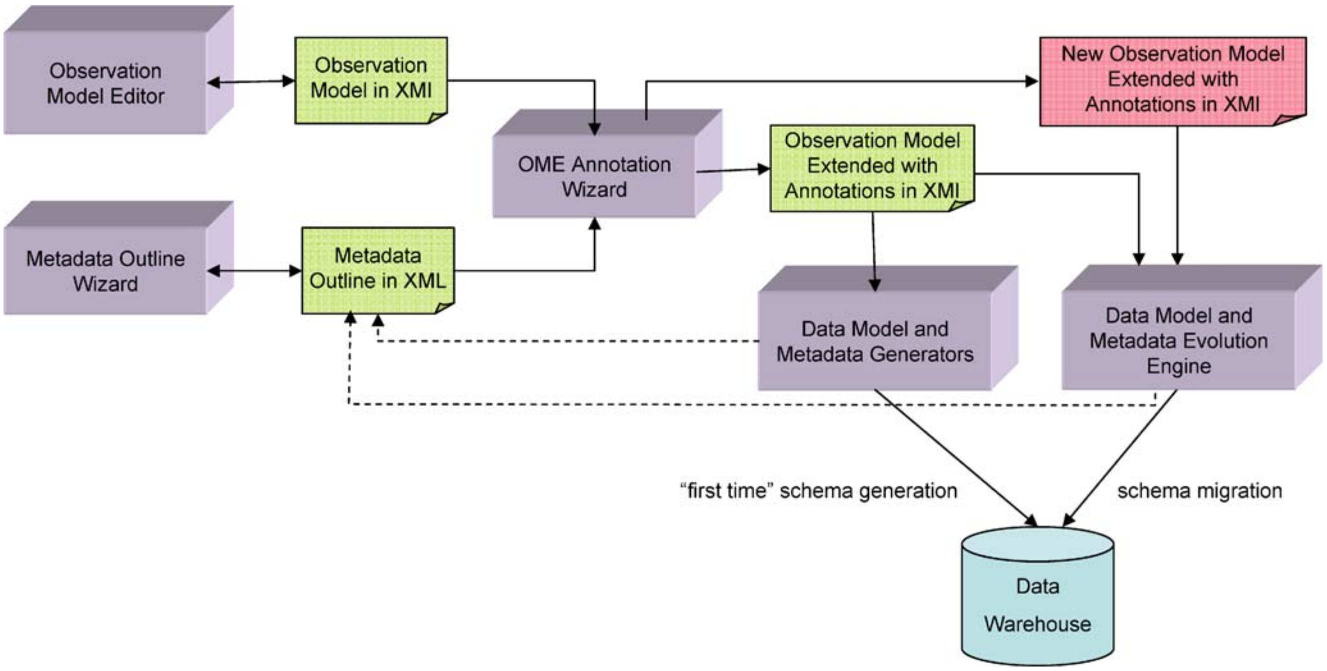

Observation Model Editor
Observation Model in XMI
OME Annotation Wizard
New Observation Model Extended with Annotations in XMI
Observation Model Extended with Annotations in XMI
Metadata Outline Wizard
Metadata Outline in XML
Data Model and Metadata Generators
Data Model and Metadata Evolution Engine
"first time" schema generation
schema migration
Data Warehouse

Fig. 9 The incremental maintenance process

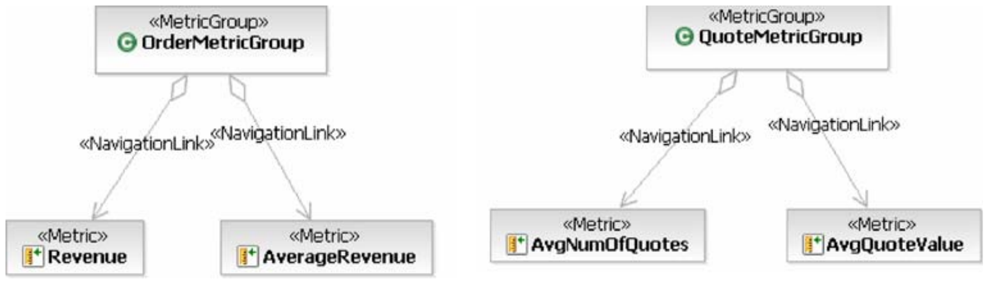


Fig. 10 Metric group definition

In Fig. 9 we depict the process we have incorporated in our framework in order to support the incremental maintenance of the data model. The business analyst starts by specifying the events of interest and the associated metrics in the *Observation Model Editor* (OME). The OME Annotation Wizard is used to annotate each of the measures identified in OME either as a dimension or a metric. (This information is necessary for building the data model.) The OME Annotation Wizard also takes as input an existing data model, if such a model already exists for the data warehouse. In the next step, the data model is produced, and the associated database operations generate the appropriate schema (i.e., the physical model) in the data warehouse. At the same time, a set of metadata that describe the generated schema are produced, for use by the OME Annotation Wizard and the Metadata Outline Wizard. The role of the latter is to provide a graphical user interface for displaying detailed information about the data model, and also for managing different versions of the model.

In the case where the business analyst changes an existing OM, the OME Annotation Wizard recognizes all the changes and tags them with information related to the nature of the changes that occurred. Subsequently, the Data Model and Metadata Evolution Engine parses the set of changes, and translates them to updates in the data model. In order to do this, the engine uses special rules for transforming the data model into a data warehouse schema. The above rules ensure that the resulting data model can accommodate updates, without the need to be re-generated from scratch. In the interest of space, we will not get into the details of these rules. We present though some key ideas that make this functionality possible.

First, the metrics in the data model are organized in multiple tables (also know as fact tables), according to the dimension measures associated with them. This decision makes it possible to have several different types of business events that are monitored, under the same OM. Furthermore, event types can be added or dropped from the OM and the Data Model, without affecting the rest of the objects in the models.

Second, in contrast to traditional data models for data warehouses, there is an implicit dimension associated with the metrics themselves. This way, we are able to decouple the functionality of the metric (i.e., keeping track of some real-world quantity) from its characteristics as a member of the model (i.e., name, associated event type, etc.). As a result, we can easily handle changes in metrics. Name change and addition of a new metric are straightforward operations. When a metric is removed, we mark it as inactive. Therefore, we can still access, manipulate, and compute aggregations on all the past values for that metric, all of which are desirable properties.

Last, when there are changes in dimensions, the situation is less involved. In these cases, we just have to add a new dimension and connect it to the relevant fact tables, or update the name of an existing dimension. If a dimension is removed, then we insert a dummy value (null) in its place in the fact table, so as to maintain all the past associations between the removed dimension and the related metrics.

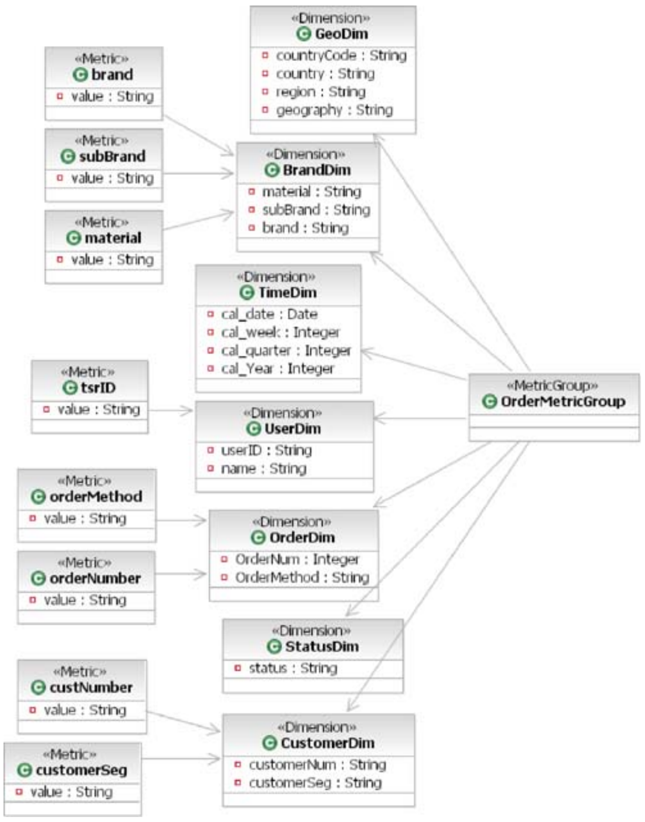


Fig. 11 Data model

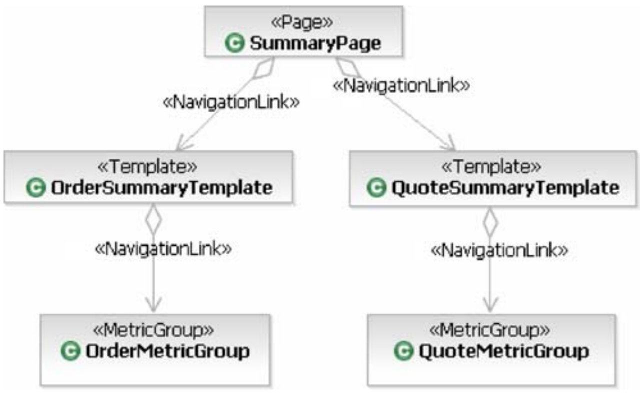


Fig. 12 Report template

5.2 Dashboard application level

In the previous paragraphs we described how changes in the environment of the dashboard (that is, in the OM) affect the data model, and what mechanisms we have in place in order to support efficient incremental maintenance. This represents the most important part of change management within the proposed framework. We now turn our attention to the rest of the Dashboard Model, and briefly discuss the effect that changes have in the other components of the model.

In the Data Model, changes in the Dimension and Metric classes reflect the corresponding changes in the OM, as discussed earlier. The business analyst in this case may only change the Metric Groups by reorganizing dimensions and metrics. All the above changes cascade seamlessly to the rest of the Dashboard Model, and reveal themselves when the dashboard application renders the various pages on screen.

The same is true for several of the other changes that the business analyst can apply to the Dashboard Model. In other cases, the changes manifest themselves within the Dashboard Model. For example, when the Navigation Tree class is changed in the Navigation Model, these changes are automatically reflected in the User Role to Navigation Tree Model.

Finally, there is the special case of the Dimension Scope class in the User Role to Metric and Dimension Model. Changes to this class pertaining to specific dimension-level values may only be applied at deployment time (see Section 4.2.3). These changes do not affect any of the other models, but make a difference in the way data are presented to the user. This kind of changes is also efficiently handled at run-time, through the use of database tables that store the necessary information.

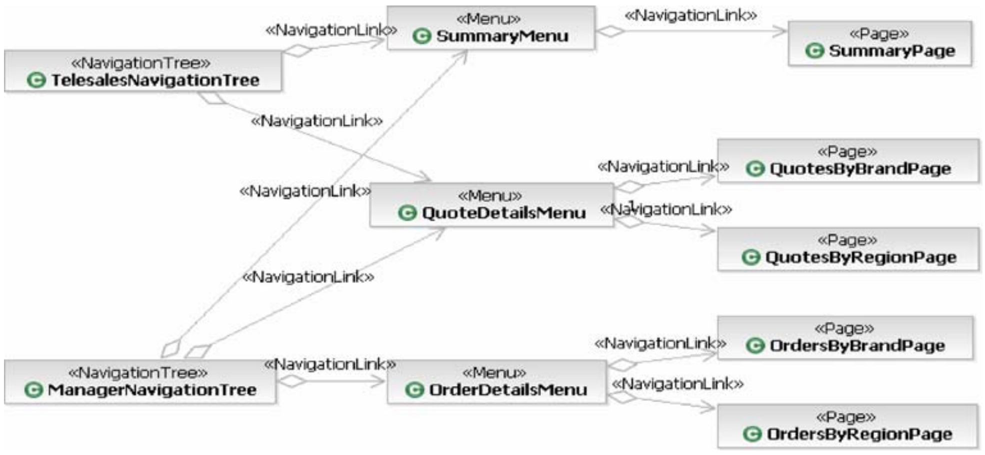
6 Case study

In order to assess the feasibility and effectiveness of the proposed approach, we applied it to a real-world problem. In this case, the objective was to develop a dashboard to support the business operation of TeleSale Representatives (TSRs) that are responsible for the sales of an entire range of products across the globe. The TSRs are responsible for the entire life-cycle of a sale. Initially, a customer expresses an interest to buy, to which the TSR responds with a quote. If the customer decides to close the deal, then the quote is turned into an order.

In their day-to-day operations, the TSRs need to have a concise view of their business, so as to plan their actions accordingly. The dashboard has to display information on both, the quotes and the orders, capturing various metrics related to these activities, such as number of quotes and orders, revenue, and others. These metrics may be organized according to several dimensions, such as time, geography, product type, customer type, and others. Furthermore, access restrictions should be in place, limiting the views of the data offered to the TSRs and the region managers.

We now describe the steps we went through during the solution development process, using the Dashboard Framework.

Fig. 13 Navigation tree model



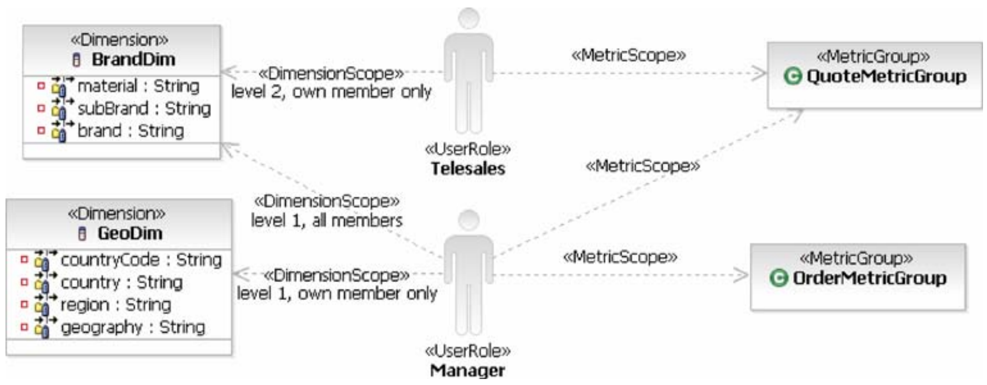


Fig. 14 Role to data access mapping

6.1 Dashboard solution model

We start by presenting the models we created for the dashboard. Note that for brevity, in all the following diagrams, we only depict part of the models that form the complete solution.

As mentioned in Section 4.2, we first identify the Report Templates that are needed. If the existing, predefined templates are not suitable, then we define custom Report Templates. For this case study, we are using pre-defined summary templates (e.g., OrderSummaryTemplate), as well as some custom-made templates (e.g., OrderDetailTemplate).

Subsequently, we identify similar metrics and group them together as MetricGroups. As shown in Fig. 10, Revenue and average revenue for orders are grouped into OrderMetricGroup, while average number of quotes and average quote value are grouped into QuoteMetricGroup. (Section 4.1.1 discusses the benefits of such groupings.)

The relationships among metrics and dimensions are captured by the data model, shown in Fig. 11. This diagram contains relationships that connect dimensions to metrics, as well as metric groups. The latter case is translated as a relationship between the dimension and each one of the metrics under the Metric Group. A link between a metric and a dimension means that the metric can be aggregated along this dimension.

In order to organize the information into different views (or pages), we use the Report Template model. Figure 12 shows this model for a summary view we have defined, which will display data relevant to orders and quotes. More specifically, this summary page will contain data for orders revenue and average revenue (represented by OrderMetricGroup), and average number and value of quotes (represented by QuoteMetricGroup).

Once we have defined all the pages and menus that we are going to use in our dashboard, we proceed to model the Navigation Trees. The Navigation Trees represent the paths that the dashboard user can follow when navigating from page to page. As Fig. 13 shows, we can define several Navigation Trees, and each page may belong to more than one Navigation Tree.

Subsequently, we define all the data access privileges for our dashboard. Figure 14 depicts the assigned privileges for the Telesales and Manager user roles, with respect to metrics and dimensions. The model we created allows Telesales users to access quote metrics and aggregate them along the brand dimension. In addition to the above, Manager users can also access order metrics and aggregate these metrics along the geography dimension.

Figure 14 also illustrates how we model fine-grain data access control using the dimension levels. In this example, we limit the access on the Brand and Geography data. A Telesales user will only be able to aggregate data up to the sub brand level (i.e., level 2) in the Brand dimension hierarchy. (The “own member” annotation only instructs the tool that fine-grain access control is required to be applied (Fig. 15).

Figure 16 show the User Role access privileges in terms of Navigation Trees and Report Templates, respectively. For our dashboard, we specify that Telesales and Manager users access different Navigation Trees, which translates to a different experience, both visually and content-wise. We also specify that Manager users can access the summary templates for the orders and the quotes, while Telesales users only have access to the quote summary template.

When we complete the modeling phase, we initiate the deployment of the different software components, described in the following section.

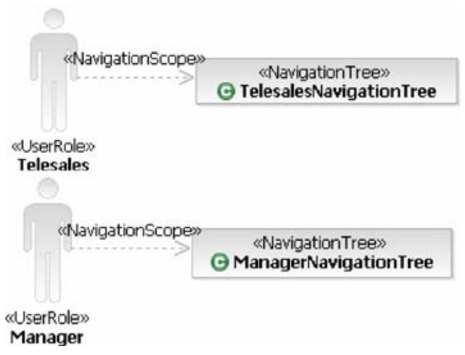


Fig. 15 Role to navigation tree access mapping

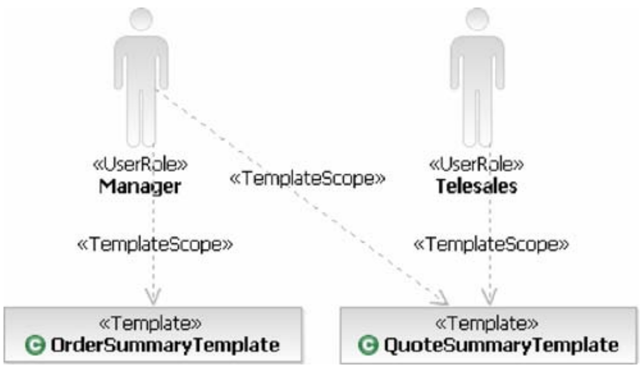


Fig. 16 Role to report template access mapping

6.2 Dashboard deployment

There are two deployable components generated as a result of the modeling activity. The Dashboard DDL component is the data warehouse schema script that supports the dashboard functionality. This schema stores and manages all the information relating to metrics, and maintains the fine grained access control to this information by user role.

The Dashboard Application component is an Enterprise Application that must be deployed on a J2EE application server, and can subsequently be accessed using a web browser. In our implementation, the generated application is deployed on WebSphere Portal Server, and uses Alphablox (http://www.alphablox.com/) for rendering the reports (the framework provides a tag-library that allows the report template to connect to Alphablox; we provide similar tag-libraries for other commercial data visualization tools, as well).

In Fig. 17, we show a screen-capture from the deployed dashboard application. This particular example illustrates a page that uses tables to display two different types of data regarding quotes (left side of the picture), and a graph to visualize these data (right side of the picture)

6.3 Discussion

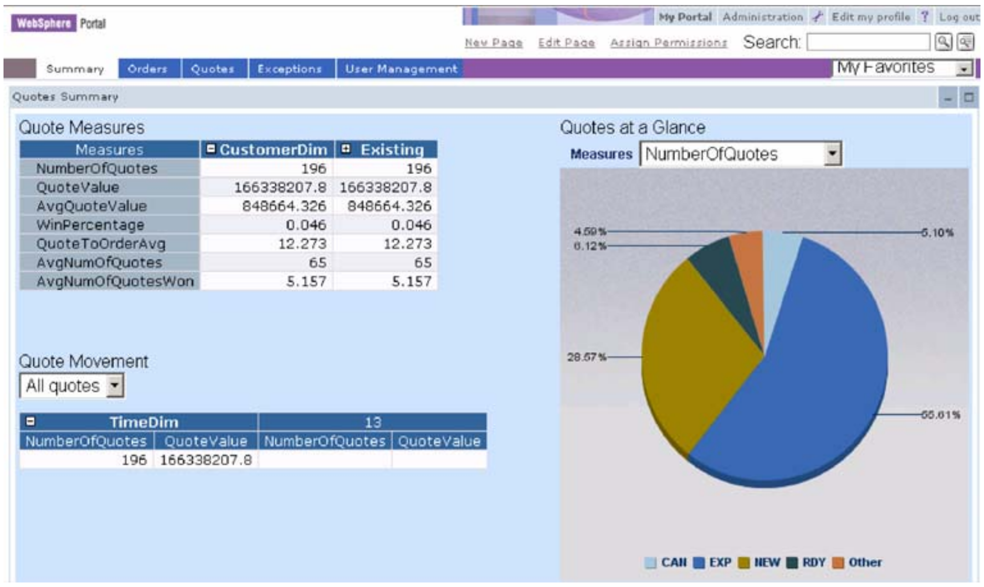
Our experience with the model-driven approach for dashboard development shows that we can achieve significant savings in terms of time and cost. Using the proposed framework, we were able to complete a project that would normally require more than 3 months, in just a fraction of the time (i.e., less than 1 month in our case).

Moreover, the benefits of our approach extend to the future as well, since our framework makes it very easy to maintain the dashboard in the presence of changes. When there are changes in metrics or dimensions, we can efficiently handle them by incrementally maintaining the corresponding data model, and thus, we avoid the need for data migration or service disruption. Changes in other elements of the dashboard model, such as in navigation paths, and access control are as simple as updating the corresponding models, making the maintenance of the dashboard an easy and manageable task.

Our experience with the real case study proved the importance of the above functionality. Through the course of modeling and deploying the dashboard, and even after deployment, changes in requirements forced us to modify the original design. Nevertheless, in all cases the changes were seamlessly handled by our framework, allowing smooth transitions from one state to the next, and avoiding altogether downtimes and complete re-builts of the dashboard application.

We should also note that the dashboard developers do not need to have any in-depth knowledge of databases and data warehouses, or access control mechanisms. All these

Fig. 17 Generated dashboard page



Quote Measures

Measures	CustomerDim	Existing
NumberOfQuotes	196	196
QuoteValue	166338207.8	166338207.8
AvgQuoteValue	848664.326	848664.326
WinPercentage	0.046	0.046
QuoteToOrderAvg	12.273	12.273
AvgNumOfQuotes	65	65
AvgNumOfQuotesWon	5.157	5.157

Quote Movement

All quotes

TimeDim		13	
NumberOfQuotes	QuoteValue	NumberOfQuotes	QuoteValue
196	166338207.8		

aspects of the dashboard are completely hidden from the developer, and managed by the proposed framework.

7 Conclusion

In this study, we propose an efficient and effective model-driven dashboard design technique. We extend the business performance modeling framework by providing a number of new models that enable the process of dashboard design. Our model-driven approach renders the dashboard design and deployment process less time-consuming and less cumbersome. It leads to automated code generation, and allows fast and easy integration of design changes in the final solution.

We applied the proposed technique for designing and deploying a dashboard for a real-world business, and the results of this experiment demonstrate the feasibility and effectiveness of our approach. We observed a significant reduction in terms of required development time when compared to a more traditional dashboard deployment process.

References

Bhattacharya, K., Guttman, R., Lyman, K., Heath, I. F. F., Kumaran, S., & Nandi, P., et al. (2005). A model-driven approach to industrializing discovery processes in pharmaceutial research. *IBM Systems Journal, 44*(1), 145–162.

Chen, S.-K., Lei, H., Wahler, M., Chang, H., Bhaskaran, K., & Frank, J. (2006). A model driven XML transformation framework for business performance management model creation. *IJEB, 4*, 281–307.

Chowdhary, P., An, L., Jeng, J.-J., & Chen, S.-K. (2005). Enterprise integration and monitoring solution using active shared space. In *ICEBE* (pp. 295–304).

Czarnecki, K., & Helsen, S. (2003). Classification of model transformation approaches. In *OOPSLA workshop on generative techniques in the context of model-driven architecture, Anaheim, CA*.

Frakes, W. B., & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering, 31*(7), 529–536.

Gray, J., Bosworth, A., Layman, A., & Pirahesh, H. (1996). Data cube: A relational aggregation operator generalizing group-by, Cross-Tab, and Sub-Total. *ICDE*.

Greenfield, J., Short, K., Cook, S., & Kent, S. (2003). Software factories assembling applications with patterns, models, frameworks and tools. In *18th annual ACM OOPSLA*.

Jeng, J. J., Buckley, S., Chang, H., Chung, J. Y., Kapoor, S., & Kearney, J., et al. (2002). BAM: An adaptive platform for managing business process solutions. *ICECR*.

Jeng, J. J., Schiefer, J., & Chang, H. (2003). An agent-based architecture for analyzing business processes of real-time enterprises. In *EDOC* (pp. 86–97). Washington, DC: IEEE Computer Society.

Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA explained, the model driven architecture: Practice and promise*. Reading, MA: Addison-Wesley.

Koehler, J., Tirenni, G., & Kumaran, S. (2002). From business process model to consistent implementation: A case for formal verification methods. In *EDOC* (pp. 96–106). Washington, DC: IEEE Computer Society.

Kumaran, S. (2004). Model-driven enterprise. In *Proceedings of global enterprise architecture integration summit* (pp. 166–180).

Kumaran, S., & Nandi, P. (2005). Adaptive business objects: A new component model for business integration. In *Proceedings of ICEIS* (pp. 48–58).

Marian, A., Abiteboul, S., Cobena, G., & Mignet, L. (2001). Change-centric management of versions in an XML warehouse. In *VLDB* (pp. 581–590). San Francisco, CA: Morgan Kaufmann.

McGowan, C., & Bohmer, L. (1993). Model-based business process improvement. *ICSE*.

Miller, J., & Mukerji, J. (Eds.) (2003). MDA guide version 1.0.1. Object management group. Retrieved from http://www.omg.org/docs/omg/03-06-01.pdf.

Mumick, I. S., Quass, D., & Mumick, B. S. (1997). Maintenance of data cubes and summary tables in a warehouse. *SIGMOD*.

Myers, B., Hudson, S. E., & Pausch, R. (2000). Past, present, and future of user interface software tools. In *ACM ToCHI, vol. 7* (pp. 3–28). New York: ACM Press.

Palpanas, T., Sidle, R., Cochrane, R., & Pirahesh, H. (2002). Incremental maintenance for non-distributive aggregate functions. In *VLDB* (pp. 802–813).

Rugaber, S., & Stirewalt, K. (2004). Model-driven reverse engineering. *IEEE Software, 21*(4), 45–53.

Sukaviriya, N., Kumaran, S., Nandi, P., & Heath, T. (2005). Integrate model-driven UI with business transformations. *MDDAUI*.

van der Aalst, W. M. P., ter Hofstede, A. H. M., & Weske, M. (2003). Business process management: A survey. Eindhoven, The Netherlands: BPM.

Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A., & Sampaio do Prado Leite, J. C. (2005). Reverse engineering goal models from legacy code. In *ICRE* (pp. 363–372). Los Alamitos, CA: IEEE Computer Society Press.

Zeng, L., Lei, H., Dikun, M., Chang, H., Bhaskaran, K., & Frank, J. (2005). Model-driven business performance management. In *ICEBE* (pp. 295–304).

Themis Palpanas is a faculty member in the Department of Information and Communication Technology, at the University of Trento, Italy. He received his Bachelor of Science degree from the National Technical University of Athens, Greece, and his M.Sc. and Ph.D. from the University of Toronto, Canada. Before joining the University of Trento, Prof. Palpanas worked at the IBM T.J. Watson Research Center. He has also worked for the University of California at Riverside, and visited Microsoft Research, and the IBM Almaden Research Center. Prof. Palpanas is serving in the program committees of several top database and data mining conferences, is a member of ACM and the Technical Chamber of Greece, and has been a member of the IBM Academy of Technology Study on Event Processing. His interests include data management, data analysis, streaming algorithms, outlier detection, incremental view maintenance, caching, and prefetching. He has applied his research solutions to real world industry problems, and is the author of five US patents.

Pawan Chowdhary has been associated with various divisions of IBM during his career span of 10 years, architecting, designing and implementing complex, high performance and scalable distributed object-oriented applications. In 2004, he joined IBM Research as an Advisory Software Engineer in the Analytic Models & Architecture Department and has been working on the Sense and Respond and Business Performance Management (BPM) technologies. Lately he is actively involved in the Model Driven Software Development technology research. He has received several projected related awards and writes extensively in the area of BPM and Model Driven Techniques. He received his Bachelors degree in Electronics Engineering from Nagpur University, India in 1996.

Florian Pinel is an Advisory Software Engineer at IBM's T.J. Watson Research Center. He received his Master's Degree in Computer Science in 1999 from Ecole Centrale Paris, France. Since joining IBM in 1999, he designed and implemented complex e-commerce solutions. He currently works on Model-Driven Business Transformation in the Business Informatics department, focusing on Business Performance Management. He is actively publishing his work, and holds several patents in this field.

George Mihaila is a Research Staff Member at IBM Watson Research Center. He has a Bachelor of Science degree from the University of Bucharest and M.S. and Ph.D. degrees from the University of Toronto, all in Computer Science. He also holds an adjunct faculty appointment at Columbia University. Dr. Mihaila's research interests include Web-based information discovery, data integration, data warehousing, event processing, and XML storage and processing. His research was published in high quality journals and conferences including the Journal of Digital Libraries, ACM Principles of Database Systems (PODS), Extending Database Technology (EDBT), IEEE International Conference on Data Engineering (ICDE) and W3C World Wide Web conferences. His research was supported by grants from the Canadian Natural Sciences and Engineering Research Council (NSERC) and the United States Defense Advanced Research Projects Agency (DARPA).