IBM Research Report

Model-Driven Dashboards for
Business Performance Reporting

**Pawan Chowdhary, Themis Palpanas, Florian Pinel,
Shyh-Kwei Chen, Frederick Y. Wu**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Model-Driven Dashboards for Business Performance Reporting

Pawan Chowdhary, Themis Palpanas, Florian Pinel, Shyh-Kwei Chen, Frederick Y. Wu
IBM T.J. Watson Research Center
{chowdhar, themis, pinel, skchen, fywu}@us.ibm.com

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 also demonstrate how all these models can be combined and automatically generate the final solution.

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.

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 [5][26] and business activity monitoring models [24][25]. Business Performance Management (BPM) [10] includes a suite of components that are used to monitor the health of the business. It delivers significant benefits to the businesses, by offering them the ability to react promptly to changes in their environment [2]. 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. Dashboards are essential elements in the day-to-day operation of modern enterprises, as they provide to the analysts the view of all the critical business metrics that reflect the performance of the business.

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 [3].

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:

- Management of the data to be displayed, involving the creation of relevant databases and access to them.
- 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 [4]. 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 5 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. Furthermore, making changes to the dashboard design is painless, since these changes only have to be made in the high-level models, and then the new code is automatically regenerated.

• 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. We describe the main part of our work in Section 4, where we present in detail the models and the process we employ in our framework. In Section 5, we present a case study, where we apply our technique to a real-world problem, and we conclude in Section 6.

2. Related Work

There is a growing trend in using model-driven methodologies [7][20] 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 [12][13], reverse engineering [14][15], and user interface design [3]. The benefits of adopting model-driven design include reduced software development time, enhanced code quality, and improved code maintenance [16][17]. There are also numerous related works about business processes. Business process management enables the management and analysis of operational business processes [18].

Business processes can be implemented using a workflow or a state machine model [19]. BPEL [11] 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) [5][6]. 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.

There is also much interest around the concept of dashboards, with several companies providing relevant solutions, such as IBM [21], Business Objects [22], and Hyperion [23]. 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 [5][6] 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) [8][9] 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 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 [10]. 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 Figure 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.
• 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.
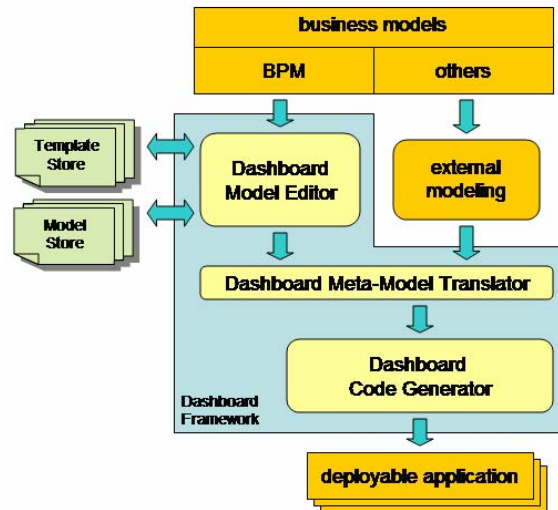

Figure 1: Model-driven Dashboard Framework.

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. Figure 2 shows part of the Dashboard Meta-Model XML Schema definition. 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 Figure 1 by the box labeled "external modeling".
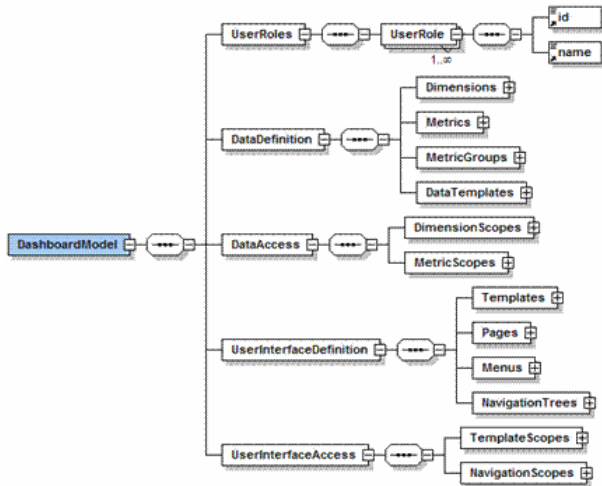
Figure 2: Dashboard meta-model (XML schema).

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 Figure 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 3 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 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.
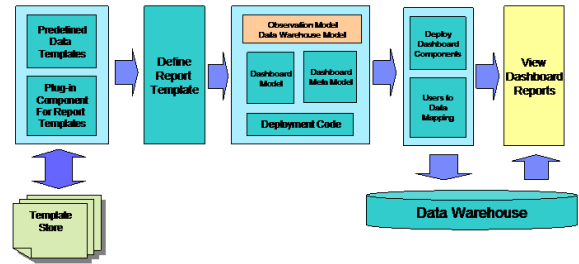

Figure 3: End-to-end dashboard component flow.

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.

4.1.1.1 Dashboard Data Model

In our framework, we assume that all the necessary data can be stored in a data warehouse, using a star schema [28]. Therefore, we use the data model shown in Figure 4, where each data element is marked as either a dimension, or a metric.

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 Figure 4 we are also introducing the Metric Group modeling element, which is used for grouping of relevant metrics. Such a grouping is useful when modeling relationships to other artifacts, where all the members of the Metric Group participate. Figure 4 depicts the Metric Group UML class that connects to the Metric class in an aggregation relationship.
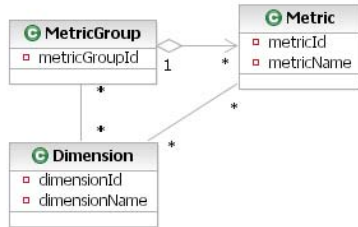

Figure 4: Dashboard Data model.

4.1.1.2 Dashboard Navigation Model

In Figure 5, 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.
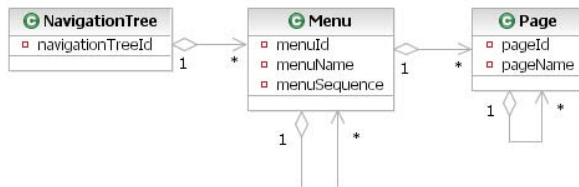

Figure 5: Dashboard Navigation Model

4.1.1.3 Dashboard Report Template Model

The Report Templates are used to define the information content of the individual pages. Figure 6 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.
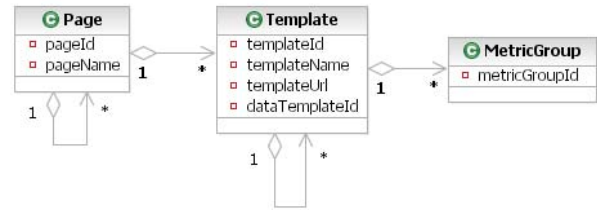

Figure 6: Dashboard Report Template Model

4.1.1.4 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 7 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.
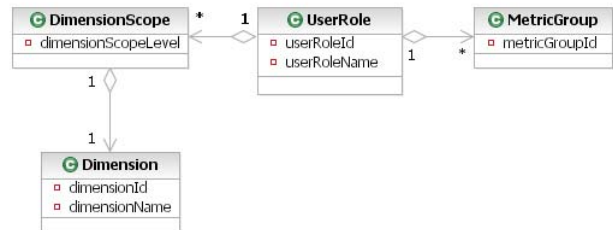

Figure 7: UserRole to Metric and Dimension model.

- *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.

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 8 shows the User Role to Report Template relationship.
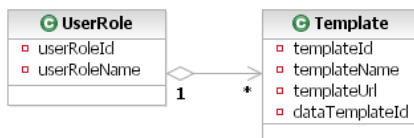

Figure 8: UserRole to report Template model.

Finally, our framework allows the business analyst to define access control based on the Navigation Trees (Figure 9). 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.
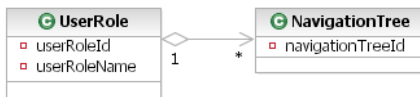

Figure 9: User Role to Navigation Tree model.

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.
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.

Figure 10 depicts the components that are relevant to this activity. 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.
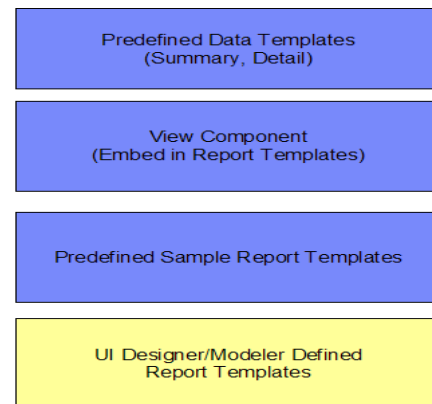

Figure 10: Pre-modeling activity.

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. As illustrated by the box at the bottom of Figure 10, 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. 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 the TeleSale Representatives (TSRs) of IBM that are responsible for the sales of the entire range of xSeries 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.

5.1 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 Figure 11, 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.)
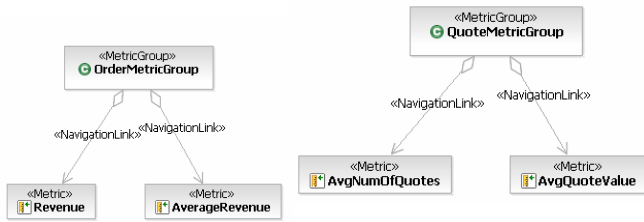
Figure 11: Metric group definition.

The relationships among metrics and dimensions are captured by the data model, shown in Figure 12. 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 13 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 Figure 14 shows, we can define several Navigation Trees, and each page may belong to more than one Navigation Tree.
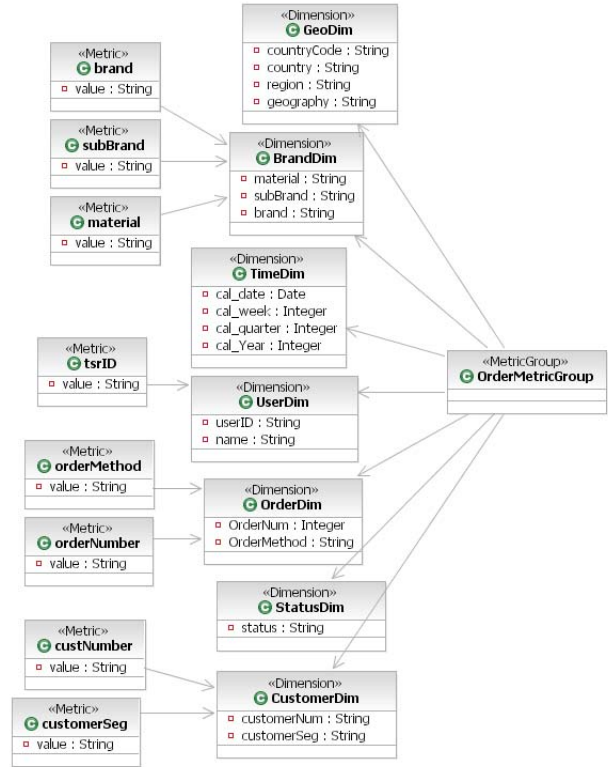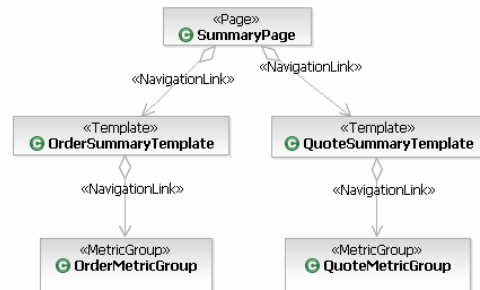

Figure 12: Data model.

Figure 13: Report template.

Subsequently, we define all the data access privileges for our dashboard. Figure 15 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 15 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.
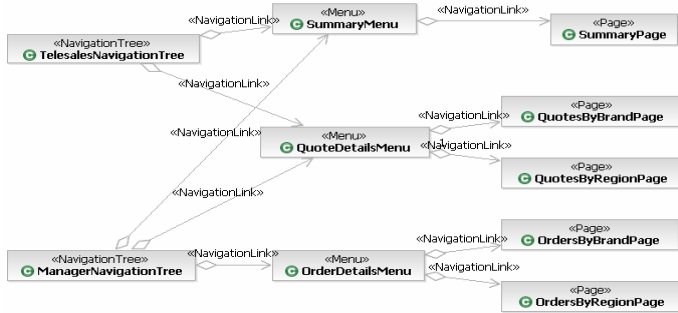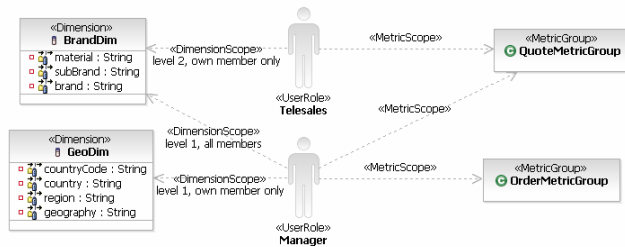

Figure 14: Navigation tree model.

Figure 15: Role to data access mapping.

Figure 16 and Figure 17 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.
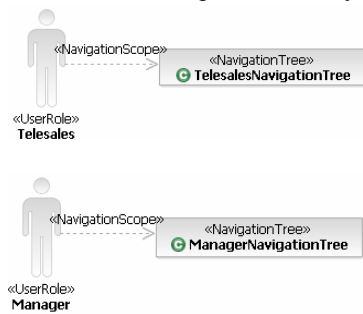

Figure 16: Role to navigation tree access mapping.

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

5.2 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 [21] 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 Figure 18, 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).
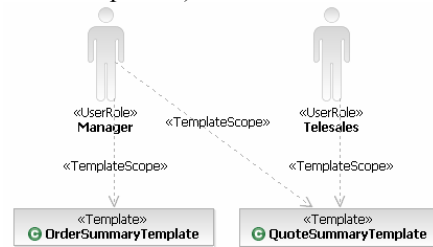
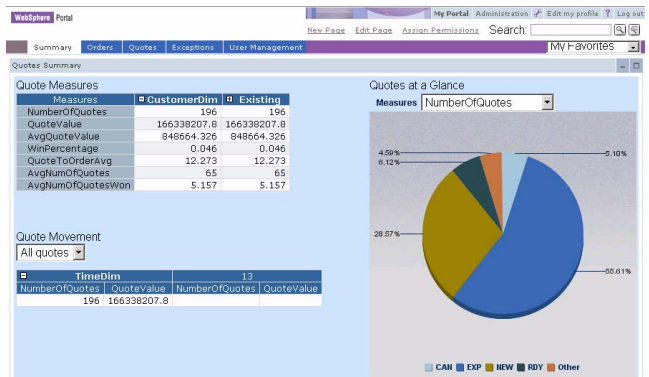

Figure 17: Role to report template access mapping.

Figure 18: Generated dashboard page.

5.3 Discussion

Our experience with the model-driven approach for dashboard development shows that we can achieve significant savings in terms of time and cost. A project that would normally require more than three months, we were able to complete it in just three weeks using the proposed framework. Moreover, the benefits of our approach extend to the future as well, since our

framework makes it very easy to introduce changes in the dashboard. Changes in metrics, dimensions, navigation paths, and access control are as simple as updating the corresponding models, making the maintenance of the dashboard an easy and manageable task.

In addition, the dashboard developers do not need to have any in-depth knowledge of databases and data warehouses, or access control mechanisms. All these aspects of the dashboard are completely hidden from the developer, and managed by the proposed framework.

6. 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

[1] Wei Zhao, Barrett R. Bryant, Fei Cao, Kamal Bhattacharya, Rainer Hauser: Transforming Business Process Models: Enabling Programming at a Higher Level. IEEE SCC 2005: 173-180.

[2] K. Bhattacharya, R. Guttman, K. Lyman, I. F. F. Heath, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A Model-Driven Approach to Industrializing Discovery Processes in Pharmaceutial Research. IBM Sys Journal 44(1), 2005.

[3] Noi Sukaviriya, Santhosh Kumaran, Prabir Nandi, Terry Heath. Integrate Model-driven UI with Business Transformations. *MDDAUI*, 2005.

[4] Myers, B.; Hudson, S.E., Pausch, R. Past, Present, and Future of User Interface Software Tools. ACM ToCHI 7, (2000).

[5] Kumaran, S. Model-Driven Enterprise. Proc of Global Enterprise Architecture Integration Summit. (2004).

[6] Kumaran, S., Nandi, P.; Adaptive Business Objects: A New Component Model for Business Integration. Proc of ICEIS (2005).

[7] Kleppe, A., Warmer, J., Bast, W. MDA Explained: The Model Driven Architecture – Practice and Promise. Addison-Wesley, Reading, MA (2003).

[8] S.-K. Chen, H. Lei, M.Wahler, H. Chang, K. Bhaskaran, and J. Frank. A model driven XML transformation framework for business performance management model creation. IJEB, 2006.

[9] P. Chowdhary, L. An, J.-J. Jeng, and S.-K. Chen, "Enterprise Integration and Monitoring Solution Using Active Shared Space," ICEBE 2005.

[10] L. Zeng, H. Lei, M. Dikun, H. Chang, K. Bhaskaran, and J. Frank. "Model-Driven Business Performance Management," ICEBE 2005.

[11] Business Process Execution Language for Web Services http://www-128.ibm.com/developerworks/library/ specification/ws-bpel

[12] W. B. Frakes and K. Kang, "Software Reuse Research: Status and Future," IEEE TSE 31, No. 7, 2005.

[13] J. Greenfield, K. Short, S. Cook, and S. Kent, "Software Factories Assembling Applications with Patterns, Models, Frameworks and Tools," 18th Annual ACM OOPSLA, 2003.

[14] S. Rugaber and K. Stirewalt, "Model-Driven Reverse Engineering," IEEE Software 21, No. 4, 2004.

[15] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, J.Cesar Sampaio do Prado Leite: Reverse Engineering Goal Models from Legacy Code. ICRE, 2005.

[16] A. Kleppe, J. Warmer, and W. Bast, MDA Explained, The Model Driven Architecture: Practice and Promise, Addison-Wesley, Boston, MA, 2003.

[17] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches," OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, CA, 2003.

[18] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, "Business Process Management: A Survey," BPM, Eindhoven, The Netherlands, 2003.

[19] J. Koehler, G. Tirenni, S. Kumaran: From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods, EDOC, 2002.

[20] Miller, J., Mukerji, J. (ed.): MDA Guide Version 1.0.1. Object Management Group (2003) http://www.omg.org/docs/omg/03-06-01.pdf.

[21] AlphaBlox. http://www.alphablox.com/

[22] Business Objects. http://www.businessobjects.com/

[23] Hyperion. http://www.hyperion.com/

[24] J.J. Jeng, S. Buckley, H. Chang, J.Y. Chung, S. Kapoor, J. Kearney, H. Li, and J. Schiefer, BAM: An Adaptive Platform for Managing Business Process Solutions. ICECR, 2002.

[25] J.J. Jeng, J. Schiefer, and H. Chang, An Agent-based Architecture for Analyzing Business Processes of Real-Time Enterprises. EDOC, 2003.

[26] C. McGowan, L. Bohmer, Model-based business process improvement, ICSE, 1993.

[27] Object Management Group (2005) 'XML metadata interchange (XMI) specification version 2.0.', http://www.omg.org/docs/formal/03-05-02.pdf.

[28] J. Gray, A. Bosworth, A. Layman, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. ICDE 1996.