From Business Application Execution to Design through Model-Based Reporting

Ta‘id Holmes
Software Engineering & Tools
SAP Research, Darmstadt, Germany
taid.holmes@sap.com

***Abstract*—Cross-disciplinary models constitute essential instruments to master complexity. Often it is easier to relate to high-level concepts than to deal with low-level technical details. In model-driven engineering (MDE) models are designated a pivotal role from which systems are generated. As such, MDE enables different stakeholders of business applications to participate in the engineering process. Until now however, MDE does not penetrate phases beyond generation and deployment such as monitoring, analysis, and reporting. To display information from runtime and analytics it would be interesting if reporting could utilize models from design time. Therefore, this paper presents model-based reporting (MbR). Bridging the gap between reporting and design, it enables stakeholders to intuitively specify the reporting through a domain-specific language (DSL) while accelerating development cycles. In non-model-driven settings, MbR can help to introduce models as a first step towards MDE.**

***Keywords*-model-based, reporting, end-to-end, business applications**

I. Introduction

The collection, preparation, and representation of runtime information from business applications and enterprise systems is of particular importance to stakeholders that need to review and analyze these data. While the use of models is a common practice for the design of computing systems (cf. [1]) the runtime usually is not aware of the models from which systems have been generated (cf. [2], [3]). This however is problematic when it comes to raise the information from the execution to higher levels of abstraction such as needed for stakeholder representation.

Ideally, stakeholders could relate to the same models during monitoring and reporting. That is, the models would be enriched with information from the runtime. With this, developers could directly work with the models in the course of a new development cycle, i.e., undertake changes in order to improve a business application. While reducing costs and risks, this shortens iterative and incremental development cycles.

Models are cross-disciplinary accepted and applied as a means to deal with complexity (cf. [4]). A view-based approach can support various stakeholders while realizing the separation of concerns (SoC) principle (cf. [5]). Furthermore, models can be specified and represented at different abstraction levels making it easier for stakeholders to relate to the concepts (cf. [6]). Finally, leveraging model-driven engineering (MDE) (cf. [7]) for the generation of executable artifacts, technical expertise can be captured in transformations. Resulting benefits among others are automation and portability. Traceability, however, of model elements from design artifacts over different phases of the lifecycle is often lost, e.g., after a generation step or due to impractical and disproportional extra efforts in technical implementations. Thus, model-driven systems usually have no relation to the models from which they have been generated. As a consequence, a synchronization with design models of model-driven enterprise systems and business applications becomes cumbersome (cf. [8]). Other systems such as legacy systems are neither at design nor during execution connected to conceptual models at all. In the few cases where design time models are also used for the reporting from execution the implementation is not easily extensible for considering new types of runtime information, nor does it permit application to arbitrary models. Being specific to a particular domain and product, they are often hard coded and neither applicable in different contexts nor portable to other platforms.

Ideally, it would be possible to always relate to models during the lifecycle of a business application. For reporting, it would be particularly interesting to utilize the models from design time. If information from the runtime could be directly displayed in these models, a gap between execution and design would be filled permitting for shorter, incremental development cycles. Annotated visual models ease the consumption of reporting data. A developer, for instance, can more easily relate to the information from the runtime and can, if required, directly operate on design models for adaptation and evolution scenarios.

This paper presents model-based reporting (MbR), a generic, conceptual approach to relate information from the runtime and business analytics with models for stakeholder presentation. Visualizing models enriched with reporting data, it is applicable to graphical domain-specific languages (DSLs). At the same time, MbR can be applied to different metamodels. Extensible in regard to the data, it is agnostic to the reporting domain. Customizable through a DSL, it permits stakeholders to intuitively specify the reporting. The application to process models for business process monitoring is demonstrated in a case study.

The remainder of this paper is structured as follows: In Section II MbR is motivated in the context of business applications followed by Section III that addresses the correlation of execution with models. The MbR approach is then presented in Section IV. Next, Section V demonstrates the application to a process model in a case study. After a discussion in Section VI, the contributions are related to existing work in Section VII. Finally, Section VIII concludes the paper.

II. Model-Based Reporting in the Context of Business Applications

In a MDE setting, business applications are specified in terms of models, generated through model-to-code transformations, and deployed to the destination platform. At runtime, information is provided to a business intelligence and stored in a data warehouse for reporting. Model-based reporting (MbR) (re)connects reporting data from systems with models and model elements that reflect these systems. For this, MbR visualizes reporting data from the runtime, monitoring, and analytics based on design time models.

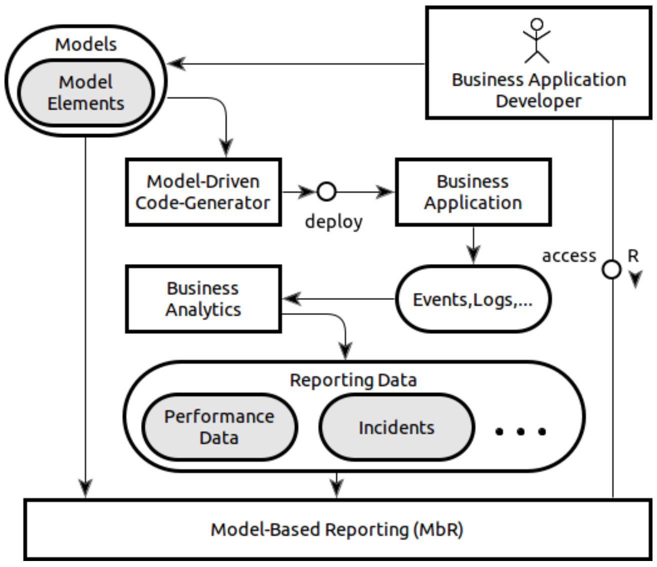

Figure 1. Model-based Reporting for Business Applications

An overview of MbR in the context of business applications is depicted in Figure 1. It displays a developer as a stakeholder of a business application storing models containing model elements in a model repository. The business application is generated and deployed by a model-driven code generator. During execution, it produces runtime information that is emitted via events or stored in log messages which are processed by business analytics. Reporting data, as the result from the analytics, constitutes the basis for reporting. It spans a variety of categories such as performance, usage, or incidents (i.e., exceptions from and violations of the business application).

Figure 2 displays a common architecture for business applications in regard to a business intelligence. The left part shows online monitoring through events and the right part offline analysis through log messages. The paths are not dependent on each other but serve independently to obtain results for a subsequent reporting. Results from business analysis through monitoring or mining and extract, transform, and load (ETL) data transformations are stored in a data warehouse that is utilized by reporting.

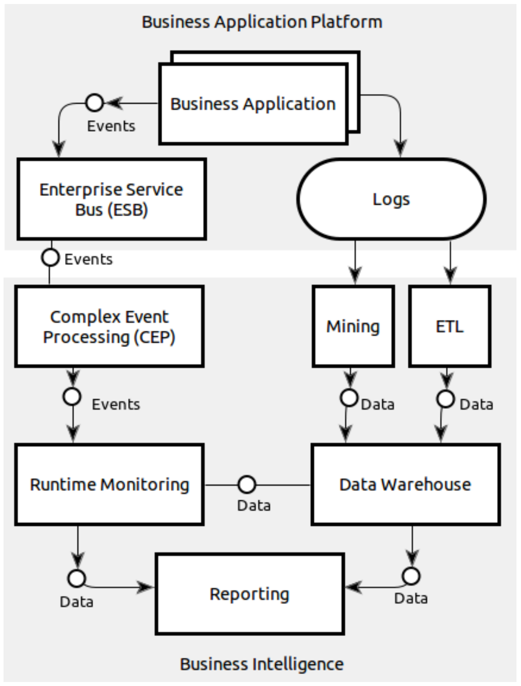

Figure 2. Architectural Overview for Business Applications

MbR can build on a business intelligence and fully benefit from its business analytics. In such cases stakeholders can be provided with highly relevant calculated data and metrics such as key performance indicators (KPIs). Nevertheless, the only prerequisite for MbR really is that information (coming from any source such as a business intelligence or from a system directly) can be correlated with model elements as discussed next.

III. Correlation of Execution with Models

An assumption for the MbR approach, as presented in this paper and explained in the next section, is a correlation of execution with models and model elements. For maximum flexibility such correlation should be established through dynamic resolution. That is, for a given model element (resp. point in execution) a set of execution points (resp. model elements) shall be returned by a correlation service. Table I lists operations of a correlation service for managing correlation entries in the data warehouse.

Table I
CORRELATION SERVICE

Return Type	Operation	Parameter
executionID[]	getEIDs	modelElementID
modelElementID[]	getMIDs	executionID
boolean	register	executionID, modelElementID
boolean	unregister	executionID, modelElementID

For reference, execution points within the business application shall uniquely be identifiable by an `executionID` [1]. Usually, the runtime environment internally provides traceability; yet, `executionIDs` need to be known outside of execution for the correlation with model elements. In order to track an execution instance an `instanceID` is required furthermore. Thus, a business application needs to disclose traceability information when emitting events and sending or writing log messages or database entries so that these identifiers can be determined and supplied to the reporting. Integrated into the business intelligence, the MbR service (see Section IV-C) can then relate the information from the runtime to the models using the correlation entries.

Although the unique identification might not take the form of a Universally Unique Identifier [9] (UUID), it is established in business applications with business relevant records and documents that need to be related to suppliers, vendors, and customers. To the degree built-in processes can be reconstructed, it is possible to establish correlation of execution points with models. Business applications that are model-driven can be instructed automatically with such identifiers and the granularity of instructions can be varied more easily. Also the correlation entries for the models and model elements can be generated. Finally, it shall be possible to "register" model elements. This is particularly useful when new models provide additional views on the system. These models can then be utilized for view-based model-based reporting.

IV. APPROACH & ARCHITECTURE

The proposed MbR approach comprises a metamodel for reporting data, a DSL for specifying the reporting, and a backend service. This service is invoked by the reporting frontend with a MbR DSL script. Relating to concepts of the reporting data metamodel, the script is processed by the MbR runtime. It retrieves reporting data and associates it with model elements. Such abstract annotations of model elements are finally transformed into concrete model annotations for the MbR frontend. An overview of the architecture with its components as explained in the following is depicted in Figure 3.

[1]This can be accomplished by using a model repository such as the Model-Aware Service Environment [3] (MORSE) repository that generates and registers unique identifiers for model elements.

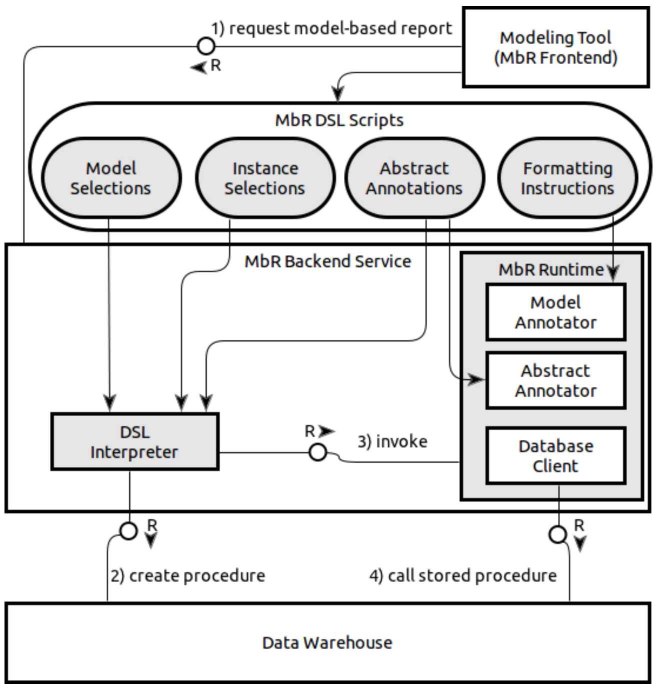

Figure 3. Model-Based Reporting — Architectural Overview

A. Reporting Data Metamodel

In selection and evaluation expressions, the MbR DSL relates to reporting data as residing in the data warehouse. In order to facilitate queries and for raising the abstraction to a technology and platform independent level, a reporting metamodel describes the data for the reporting. From this customizable and extensible metamodel the database for the MbR is generated in the data warehouse. The model-driven reporting database is a target of business analytics applications that populate it with reporting data (not the focus of this paper).

Figure 4 displays an excerpt of the metamodel comprising the concepts of `ExecutionPoint` and `ExecutionInstance` containing `executionID` and `instanceID` (see Section III) properties respectively. In the metamodel, runtime information is generically captured as `ContextData`.

Figure 5 depicts some dimensions such as temporal, locative, or user context data. In addition to such data, the reporting data metamodel comprises also the more prominent concepts of `Requirements` and `ComplianceResults` with `Violations`. Thus, `ExecutionPoints` can be associated with requirements from requirements engineering (RE) and compliance results such as violations that occur during runtime, as evaluated and identified by monitoring, are related to `ExecutionInstances` for model-based reporting. Addressing business application compliance (cf. [10]), MbR

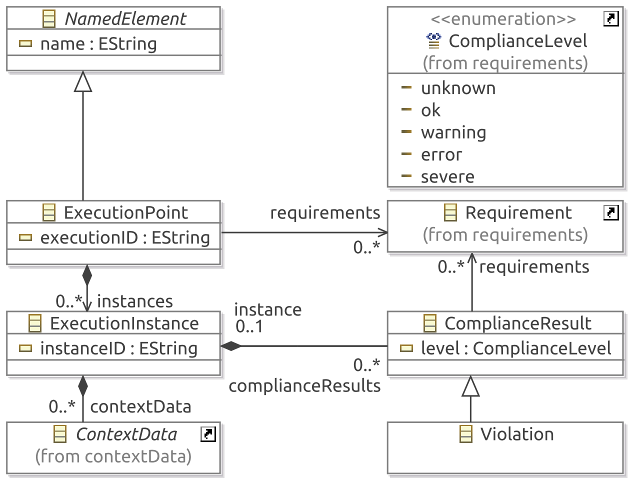

Figure 4. Reporting Data Metamodel

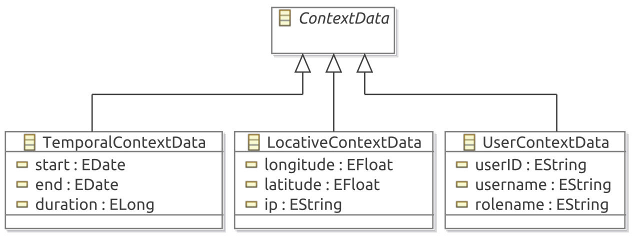

Figure 5. Runtime Context Data Metamodel

can utilize compliance relevant data as first-class entities and, e.g., highlight violations in reports.

The DSL relates to the reporting data using query expressions according to the metamodel. Thus, the metamodel can be considered for code completion and syntax checking. In addition to the names of classes and properties as found in the metamodel, alias names can be used for brevity in the DSL. The MbR service transforms the high-level and platform independent query expressions to low-level and technology specific Structured Query Language (SQL) code.

B. *A DSL for Specifying the Reporting*

The MbR DSL is at the heart of the approach. From a technical point of view it enriches models and model-elements with reporting data. From a conceptual point of view it enables stakeholders to specify the reporting in a tailored way. As such, the language comprises features to select runtime information from execution instances, combine it with the model for the model-based reporting, and specify its visualization. A MbR DSL script consists of four parts (explained next). The simplified grammar [2] is shown in Listing 1.

As a textual DSL, the scripts can be displayed, reviewed, and possibly stated by stakeholders in a comprehensive manner. Yet, the MbR frontend should provide a graphical DSL in addition so that end-users are not confronted with textual DSL scripts that operate in the background in between the frontend and the MbR service.

```
MbRscript =
   { ModelAssignment }
   { InstanceSetAssignment }
   { Annotation }
   { Formatting };
```

Listing 1. Syntax of a MbR Script

1) Model Selection: The first part (see Listing 2) merely serves to select a model for the model-based reporting and thus to define its context. For this a unique identifier such as a UUID of the model can be used. If models have been assigned names and if these names are unique across all models, also the name of the model can be used for the selection. This contributes to the readability of the script.

```
ModelAssignment =
   ModelVariable ':' Model;
```

Listing 2. Syntax of Model Selection

2) Instance Selection: Once the context is defined with a model for the reporting, instances of runtime executions are selected in the second part of the MbR DSL (see Listing 3). For this, sets of execution instances are specified using queries. A set containing a single instance can for example be specified using its `instanceID` (cf. Section III). The sets can then be used for the abstract annotations.

```
InstanceSetAssignment =
   InstanceSetVariable ':'
   InstanceSelectExpression;
```

Listing 3. Syntax of Instance Selection

3) Abstract Annotation: The third part of the DSL (see Listing 4) assigns reporting data to model elements. In this way, the model is enriched with data from the runtime and analytics. The abstract annotations are finally supplemented by the formatting instructions at the end of the script to concretize the visualization. For the annotation, model elements are selected, e.g., by name, type, or UUID. Via a key name the annotation is associated with results from an operation.

```
Annotation =
   ModelElementSetSelectExpression {
   '<<' AnnotationKey
   '=' EvaluationExpression };
```

Listing 4. Syntax of Abstract Annotation

[2]The Extended Backus–Naur Form [11] (EBNF) is used in this paper.

4) Formatting Instructions: Finally, the fourth and last part of the DSL (see Listing 5) specifies how the abstract annotations should be displayed graphically for visualization. For this, the annotation key is used and an annotation element for presentation is created. The element and the annotation content are then assigned formatting styles. Style properties are assigned values that can depend on content of the annotation. This is useful for case-based visual presentation of reporting data. For example, values can be compared against thresholds and depending on the result the data can be highlighted or not shown at all.

```
Formatting =
   AnnotationKeyName
   [ { ',' AnnotationKeyName } ] '>>'
   AnnotationElementCreateExpression '{'
      [ { ElementStyleExpression } ]
   '}{'
      [ { ContentStyleExpression } ]
   '}';

AnnotationKeyName =
   AnnotationKey [ '(' AnnotationName ')' ];

StyleExpression =
   StylePropertyName ':'
   StyleExpression ';';
```

Listing 5. Syntax of Formatting Instructions

C. MbR Service

The MbR service, invoked by the MbR frontend with a MbR DSL script, delivers a model-based report for visualization. Its components are described next.

1) DSL Interpreter: First, the DSL interpreter processes the MbR script. From the first three parts of the script (i.e., model selection, instance selection, and abstract annotation) a SQL procedure is generated that is called later by the MbR runtime. Finally, the procedure is created at the data warehouse and the MbR runtime is invoked. For the generation of the SQL procedure, the selected model is related to the execution points. That is, for each model element of the model reporting data the respective `executionIDs` are looked up (see Section III). Also the type of information (either used in queries of the second part or in expressions of the third part) is related to the reporting tables (see also Section IV-A).

2) MbR Runtime: After the SQL procedure has been created as a preliminary action, the MbR runtime, consisting of the components described next, realizes the model-based reporting.

Database Client: The first step involves the retrievement of reporting data for the model-based reporting from the data warehouse. For this, the stored procedure is called and its results transformed if necessary.

Abstract Annotator: In the next step, the model is annotated with the data according to the third part of the MbR script. Figure 6 displays a metamodel for abstract annotations. Thus, the abstract annotator produces a model conforming to this metamodel. Note that in the model the `Annotations` relate to `ExecutionPoints` not model elements. Thus, at this step the correlation of model elements with execution points is evaluated.

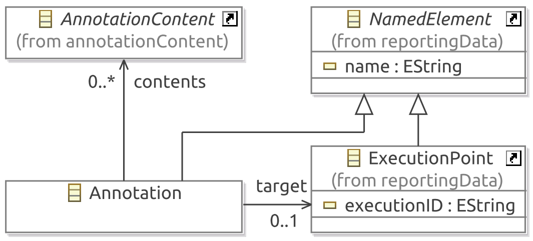

Figure 6. Abstract-Annotations Metamodel

Model Annotator: Via a model transformation the abstract annotations are transformed to the final format according to the fourth and final part of the MbR script (see Section IV-B4). For this, a model-to-model transformation is generated from the formatting instructions and executed using the abstract annotations as obtained previously.

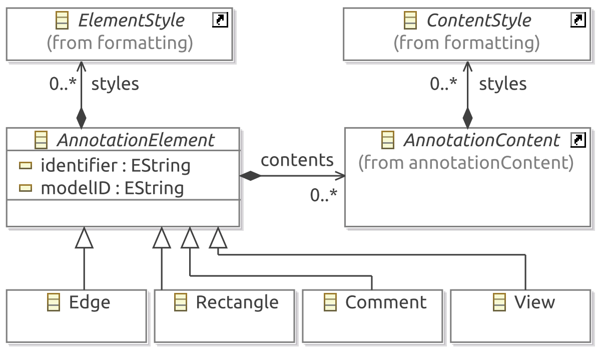

Figure 7. Metamodel of MbR Model Annotations

Figure 7 displays the target metamodel of the transformation. In this, the `AnnotationElement` constitutes a new graphical element for the model-based reporting by the MbR frontend. It comprises the annotation content with styles as well as styles for the element itself. These styles (i.e., `ElementStyle` and `ContentStyle`) are depicted in Figure 8. Although this formatting metamodel includes concepts as found, e.g., in Cascading Style Sheets [12] (CSS), please note that it is neither complete nor an elaborated model but serves rather as a proof of concept at this point. As is the case with the reporting data metamodel, also the formatting metamodel is customizable and extensible. The MbR frontend, however, needs to know how to consider and process the styles for the model-based reporting.

D. MbR Frontend

For model-based reporting, the MbR frontend issues a service call to the MbR service passing a MbR script and

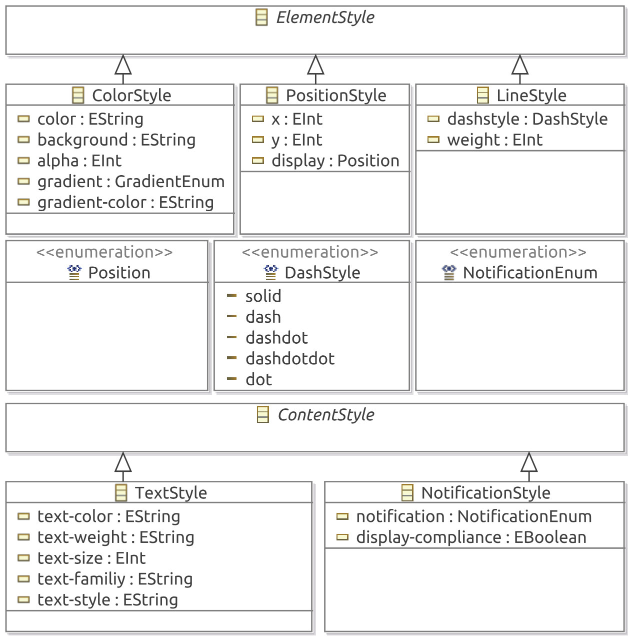

Figure 8. Formatting Metamodel

displays the reporting data in a graphical DSL. For this, the results from the service call in the form of a formatting model are processed and applied to the model. The MbR frontend serves as the reporting tool for stakeholders of business applications. In addition, it can be utilized as an end-to-end monitor and last but not least it can bridge the gap between business application execution and design if integrated into or realized within the integrated development environment (IDE) of developers.

V. Process Monitoring Powered by Model-Based Reporting

The presented MbR approach can be applied to different domains and models such as use case, architectural, and class diagrams. In this section the applicability of MbR for process monitoring is demonstrated as it is a well known and explored domain from business process management (BPM) (cf. [13]). For this, a *Quotation Creation* process from the sales and distribution domain is used and an example of a MbR script for specifying the reporting is explained. Finally, the MbR results are shown as data and within a monitoring prototype as MbR frontend.

A. Scenario and Setup

Having received a customer request for quotation, the process is initiated by a sales clerk when creating a sales quote. After submission, it is either approved or reviewed and revised. When the sales quote has been approved, a corresponding sales order is created and sent to the customer. For facilitating the approval process a *customer rating* service can recommend details of the sales quote such as discounts or prepayments. As the external business service for the customer rating might occasion expenditures per usage it is not invoked by default by the process.

The process is provided from an enterprise resource planning (ERP) system that – during execution – supplies business intelligence with traceability information. The resulting reporting data is stored in a data warehouse according to the metamodels for reporting and context data as shown in Figures 4 and 5.

In the scenario the MbR user (i.e., the stakeholder that specifies and consumes the model-based reporting) is interested (1) in how often the customer rating is advised and revisions take place, (2) in how long it takes for the ratings, the revisions, and the overall process to complete, and (3) if there are incidents in the *Edit Sales Quote* activity due to compliance violations of separation of duties (SoD) regulations (i.e., a sales clerk may not edit the sales quote).

B. Applying the Model-Based Reporting Approach

For answering the stakeholder's questions while expressing these interests the reporting is specified in the form of a MbR DSL instance. This is shown in textual form in Listing 6 that exemplifies the four parts of a MbR script [3]. First, the `Quote2Order` process model is chosen for the model-based reporting (Line 2). Second, a set of instances is selected given start and end timestamps (Lines 5-6). Next, some abstract annotations are defined such as an annotation for the process model with the average duration of the process instances (Line 9). Two model elements that correspond to sequence flows are identified by UUIDs and are annotated with the usage in percent (Lines 11-13). Note that instance sets in evaluation expressions of abstract annotations are reduced for each model element to actual corresponding execution points by default. For example, in the first annotation the average duration is calculated for the process. Thus, by convention a model element context is used in queries. Using an asterisk ($\star$) it is possible to use the entire instance set. In the second annotation, thus, the usage rate is calculated by dividing the visit counts by the total number of instances. In the third annotation (Lines 15-16) the same evaluation expression is used as in the first annotation, yet with the customer rating service selected it relates to a different model element. The last example (Lines 18-21) defines multiple annotations for the *Edit Sales Quote* activity.

Finally, the formatting instructions are declared for the abstract annotations. For this the annotation keys are used and it is specified which `AnnotationElement` from the model-annotation metamodel shown in Figure 7 shall be created. In the first (resp. second) block defined by curly brackets

[3]Please note that in the example the resolution to UUIDs for the selection of model elements, e.g., by names (see also Section IV-B3), as performed by the MbR service or frontend, has already taken place.

`ElementStyles` (resp. `ContentStyles`) can be specified according to the formatting metamodel shown in Figure 8. For simplicity, it is possible to create model-annotations (i.e., `AnnotationElements`) with equal styles for different annotations as shown in the last example (Lines 46-50). It also explicitly sets the name of the `durESQ` annotation key which name otherwise is used by default. Using a sigil (`$`) it is possible to make the value of style properties dependent on the annotation. For the `rate` annotation (Lines 31-37) the `color` and `dashstyle` properties are dependent on the value of the according usage. As multiple model elements were selected also multiple model-annotations are created. That is, each model element (of an abstract-annotation) is annotated.

```
 1  // Model Selection (1st part)
 2  m: Order2Cash
 3
 4  // Instance Selection (2nd part)
 5  b: m.start >= "2012-02-07 13:42:14"
 6     & m.end < "2012-02-14 13:42:14"
 7
 8  // Abstract Annotation (3rd part)
 9  m << duration = AVG(b.duration)
10
11  557dbb5d-7558-4f8e-acc6-d618f12487a6
12  b3b6e8e5-3b30-44a3-8dd5-29f354c60877
13     << rate = COUNT(b.visit)/COUNT(b*)
14
15  8ee48a10-01f0-49a3-b4f5-e13acb5829c5
16     << durRCR = AVG(b.duration)
17
18  f35837fa-7fcc-4fc4-9e2a-ac2ddbd696ed
19     << durESQ = AVG(b.duration)
20     << incidents = COUNT(b.violation)
21        + " / " + COUNT(b*)
22
23  // Formatting Instructions (4th part)
24  duration >> Rectangle {
25     display: absolute; x: 465; y: 60;
26     background: lightgray;
27     gradient: left2right;
28     gradient-color: white;
29     dashstyle: dash;
30  }{}
31  rate >> Edge {
32     display: relative; y: -15;
33  }{
34     color: $ < 50 ?
35        rgb(128,128,128) : rgb(255,255,255);
36     dashstyle: $ < 50 ? dot : dash;
37  }
38  durRCR("duration") >> Comment {
39     display: inline;
40     background: #b0c4de;
41     gradient: left2right;
42     gradient-color: white;
43  }{
44     display-compliance: true;
45  }
46  durESQ("duration"),incidents >> Comment {
47     display: inline;
48  }{
49     display-compliance: true;
50  }
```

Listing 6. Model-Based Reporting DSL Script Example

C. Utilizing Model-Based Reporting

Invoked with the MbR script, the MbR service returns annotations for model-based reports in the form of an annotation model (conforming to the metamodel shown in Figure 7). That is, the script is interpreted and the MbR runtime is executed while reporting data for the model elements is resolved using the correlation entries.

Listing 7 partially displays the results for the MbR script. The first annotation relates to the sequence flow invoking the customer rating service and the second annotation is for the *Edit Sales Quote* activity. For some annotations the `display-compliance` property was specified in the MbR script. The evaluated compliance level is thus indicated for the respective model elements. For the sequence flows the `color` and the `dashtype` were evaluated and set.

```
<ModelAnnotations>
 <annotationElements
  xsi:type="modelAnnotations:Comment"
  modelID="557dbb5d-7558-4f8e-acc6-d618f12487a6">
  <styles xsi:type="presentation:PositionStyle"
   display="relative" y="-15"/>
  <styles xsi:type="presentation:ColorStyle"
   color="#888888"/>
  <styles xsi:type="presentation:LineStyle"
   dashstyle="dot"/>
  <contents xsi:type="annotationContent:Metric"
   value="2" unit="Percent"/>
 </annotationElements>
 <annotationElements
  xsi:type="modelAnnotations:Comment"
  modelID="f35837fa-7fcc-4fc4-9e2a-ac2ddbd696ed">
  <styles xsi:type="presentation:PositionStyle"
   display="inline"/>
  <contents
   xsi:type="annotationContent:KeyValuePair"
   complianceLevel="warning" key="duration">
   <value xsi:type="annotationContent:Metric"
    value="06:18:31.569" unit="Time"/>
  </contents>
  <contents xsi:type="annotationContent:Text"
   complianceLevel="ok" value="0 / 132"/>
 </annotationElements>
  <!-- ... //-->
</ModelAnnotations>
```

Listing 7. Example of Model-Annotations for MbR Frontends

Finally, Extensible Markup Language [14] (XML) serialized code is processed by the MbR frontend that creates annotations for the `AnnotationElements` using the contained styles and contents. That is, the MbR results are woven into the process model and displayed accordingly. Figure 9 displays the model-based report with the resulting process diagram in a monitoring prototype [4]. Icons indicate and thus report the compliance level. For the *Edit Sales Quote* a warning is displayed for the duration as it is beyond a certain threshold as specified in a requirement (delineated in Section IV-A, please note that compliance checking is not the focus of this work).

VI. Discussion

Having motivated and presented the approach with the idea of model-based reporting as the main contribution of this paper while having provided some implementation details and having demonstrated an application of MbR for process

[4]The monitoring MbR frontend has been developed in C# using the Microsoft Visual Studio Visualization and Modeling SDK [15].

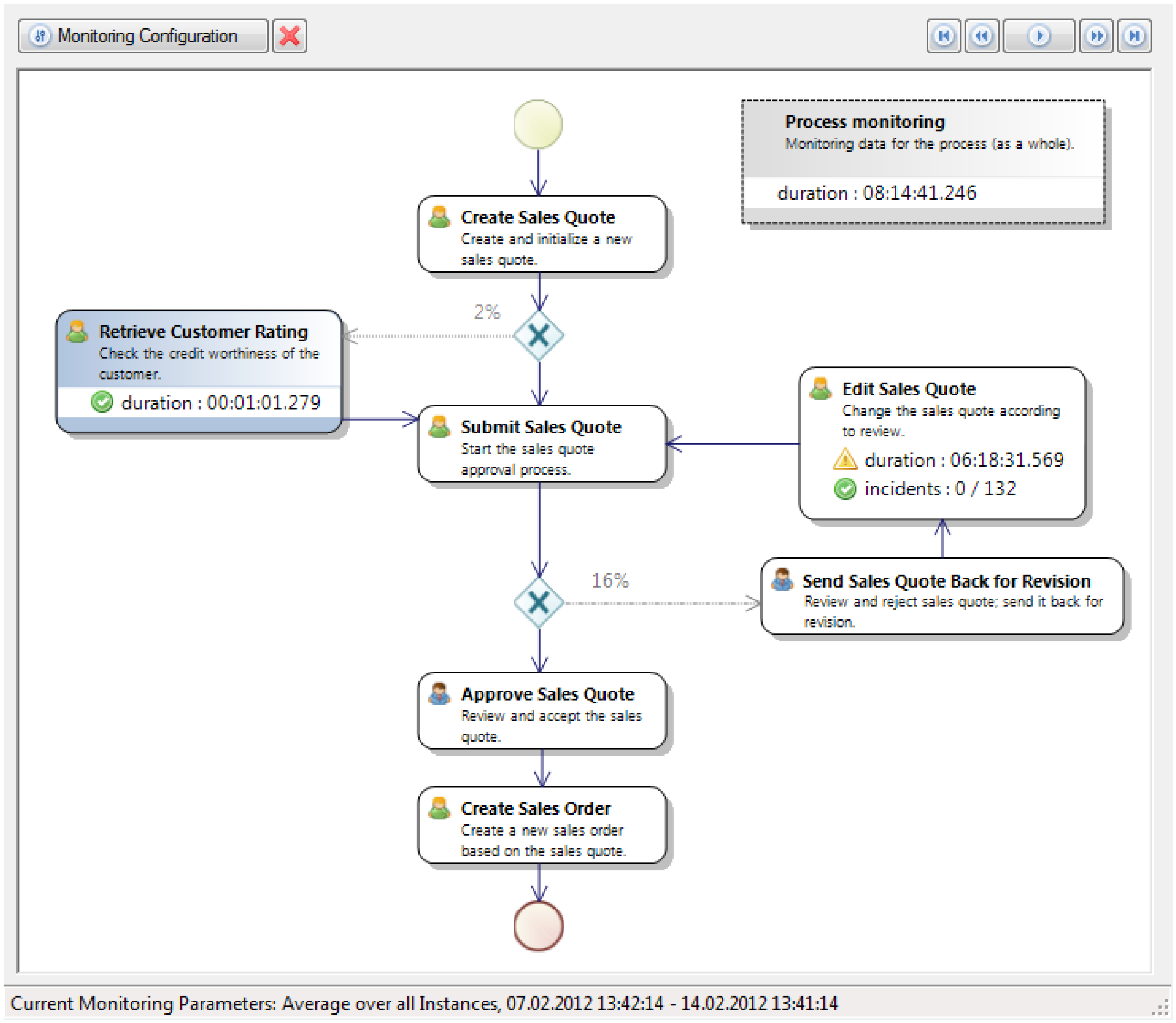
Monitoring Configuration
Process monitoring
Monitoring data for the process (as a whole).
duration : 08:14:41.246
Create Sales Quote
Create and initialize a new sales quote.
2%
Retrieve Customer Rating
Check the credit worthiness of the customer.
duration : 00:01:01.279
Submit Sales Quote
Start the sales quote approval process.
Edit Sales Quote
Change the sales quote according to review.
duration : 06:18:31.569
incidents : 0 / 132
16%
Send Sales Quote Back for Revision
Review and reject sales quote; send it back for revision.
Approve Sales Quote
Review and accept the sales quote.
Create Sales Order
Create a new sales order based on the sales quote.
Current Monitoring Parameters: Average over all Instances, 07.02.2012 13:42:14 - 14.02.2012 13:41:14

Figure 9. Monitoring Prototype using Model-Based Reporting displaying the Model-Annotations in the Process Model

monitoring, this section will now mention some benefits, consequences, and limitations of the approach and present lessons learned.

A. *From Business Application Execution to Design*

Relating information from runtime and analytics with models at reporting time, MbR bridges phases of the business application engineering lifecycle. It permits stakeholders to consume the reporting data while relating to models. In an MDE context, the models can be design time artifacts. Stakeholders of business applications can use model-based reports as a starting point for modifying and evolving model-driven business applications. As a result, development cycles are shortened.

1) A Unified Environment for Business Application Engineering: An IDE can be adopted as a frontend for different phases of a business application lifecycle, i.e., as a modeling or reporting tool. This provides for a consistent environment to developers and other stakeholders of business applications. In a MDE context, the MbR frontend can be integrated into the modeling tool. Models that are used for MbR can then be revised directly for a new, iterative development cycle.

2) Decoupling the Reporting: Due to the generic and conceptual approach, MbR can be adopted in different settings. For example, the MbR service can be used by diverse products. Also, MbR tools are decoupled from business intelligence and may potentially be utilized with different products, platforms, and technologies. For this, the metamodels for the reporting data and the model-annotations need to be agreed upon. In practice, there often exist dedicated, yet distinct, tools for similar purposes covering kindred functionalities.

For end-users of enterprise systems it is quite awkward to determine which tool to use in and for which situation. Often this relates to some technical conditions and details, e.g., compatibility of products. Decoupling the reporting clearly eases the interoperability of tools.

B. Applicability and Limitations of the Approach

While models are required for the reporting, the approach does not presume business applications to be model-driven. In fact, MbR can equally be applied to non-model-driven systems such as legacy systems. In such cases it suffices to define a graphical DSL reflecting the concepts in a stakeholder-tailored way. The underlying model (i.e., abstract DSL) can be used as a first step towards MDE.

As model-based reporting is not limited to a particular model it is applicable to different types of models such as process models (e.g., [16], [17]), architectural diagrams (e.g., [18]), graphical user interface (GUI) models, or data models. Having multiple models that reflect systems, it is possible to have different views (cf. [5]) on the system.

Besides the assumption that business applications disclose traceability information as discussed in Section III, it is further presumed that business analytic applications populate the reporting data in the data warehouse according to the reporting data metamodel. If the latter is not the case, reporting data needs to be transformed accordingly. A limitation of the presented approach is the ability for mashups, i.e., to combine data from various sources. Mashups need to be realized on top of MbR. Finally, online analytical processing (OLAP) functionality is not yet covered by the work presented in this paper; yet the MbR approach is not limited in supporting capabilities such as drill-down and navigability.

C. Design Decisions and Lessons Learned

Aiming at reconnecting data from business application execution with models the decision to opt for annotations was clear from the beginning. For specifying the reporting from an end-user perspective the need for a DSL was born. Thus, the establishment of a MbR service and the architecture was a natural result of the endeavor.

While the textual DSL was a first and valid approach for expressing stakeholder requests towards reporting, the MbR frontend should provide, nevertheless, graphical guidance in addition so that an end-user is not confronted with textual DSL scripts. Instead the frontend shall generate the MbR scripts and pass it to the MbR service transparently.

The development of the DSL underwent several iterations. For brevity the DSL and its design decisions are not presented in all detail in this paper as the focus of this work is rather the overall idea, approach, and architecture of MbR. To point out one particular feature of the DSL, note that in the second part of MbR scripts instance sets are selected. It is thus possible to compare different sets and visualize the results in the same model as chosen at the beginning of the script. In monitoring solutions (see Section VII) this is usually not possible and thus a distinction of this work.

A requirement in regard to the MbR frontend was that the results from the MbR service shall be easily consumable. Thus, after the abstract annotations, the MbR runtime calculates the layout information for the various model-annotations so that they can be processed directly by the MbR frontend. While this simplifies the implementation for enriching the models with the annotations, it comes with the tradeoff that the formatting is done at the server-side. That is, changes in the frontend imply the need to invoke the MbR service in order to obtain the new formatted annotations for the model. Clearly, this can be improved by permitting some client-side formatting. Also the amount of data can be varied. That is, the service may provide more annotations to the frontend than displayed initially but that may be visualized afterward.

VII. Related Work

Model-based reporting as a holistic and model-centered approach involves a number of issues as it effectively establishes and enables end-to-end solutions, e.g., for the monitoring of business applications. For this reason, the presented work does not claim to contribute new techniques or metamodels. Motivating the vision and presenting the novel notion of model-based reporting for positioning the work, it rather illustrates its feasibility and exemplifies the idea with an architecture, utilizing a modeling approach. By incorporating work from the state-of-the-art from various areas, the presented MbR approach can be refined for improving the maturity and quality of an implementation. Among these are the establishment and management of the correlation entries and the DSL for expressing the model selection, annotation, and formatting. Besides literature that addresses these problems also industrial products offer fine-grained support for distinct features. Yet, a conceptual, domain-agnostic, thus generic, and holistic approach of consolidating data with models is missing in both literature and implementations.

For the consolidation of data originating from execution and analytics with models, traceability is utilized. Thus, some work in this area is discussed next. Besides that, the topics of model annotation, business process monitoring, and reporting are related to this work in this section.

Traceability: A survey of traceability approaches in MDE is presented by Galvao et al. [19]. For establishing and utilizing traceability information Aizenbud-Reshef et al. [20] state that MDE provides for new opportunities such as trace generation and trace analysis. Among their suggestions are to establish a standard traceability metamodel and to uniquely identify artifacts across space and time. While the MbR approach is agnostic about a concrete traceability metamodel, it builds on the unique identification which is given when adopting UUIDs.

Traceability links can be established through trace generation (cf. [21]). For MbR these traces simply need to be registered too (see Section III). During runtime, traceability dependencies may be identified automatically through trace analysis. In this regard, Egyed and Grünbacher [8] present work in the context of RE focusing on requirements traceability and in [22] Egyed et al. report on the cost-quality trade-off for automated traceability. Independent from a specific domain (and model), trace analysis for model elements can be applied for creating and maintaining correlation entries in the data warehouse. Suppose a trace analysis would take the role of providing the correlation service in MbR, an end-user would be directly confronted with the results. In this case an advanced user could give feedback on false positives or negatives to the trace analyzer and even provide traceability dependencies manually. Thus, the MbR approach can be combined with research results from trace generation and analysis.

For mapping traces to architectural elements, Ubayashi and Kamei [2] introduce the notion of archpoints and program points in code. This translates to model elements and execution points as presented in Section III. Having considered a one-to-one mapping, the authors plan to support a one-to-many mapping in future. In contrast this work support relationships of type many-to-many between model-elements and execution points.

The MORSE traceability matrix [3, Listing 1] as presented in a previous work, directly maps execution points to model elements. Being agnostic as to the language, it has been applied to the Business Process Execution Language [23] (BPEL) where the traceability matrix is generated prior to deployment of business processes. During execution, events are raised that disclose the correlation with the model elements in form of traceability information. As the presented matrix is static the approach has a certain shortcoming: the relation of execution to models needs to be known at deployment time already. While this condition is fulfilled in the context of (static) model-driven BPEL processes, it may not hold when applied to a business application context. In a non-model driven setting, models do not exist a priori but are only introduced for MbR. In contrast, the approach as described in Section III supports dynamic resolution. Thus, new models that present new views on the system can be added gradually for MbR. Another distinction of this work is that the MbR approach is not limited to process models but can be applied to any metamodel. In this regard, the only presumption is the unique identification of model elements.

Model Annotations: Kolovos et al. [24] present a modeling language agnostic approach for annotating models with traceability links. The underlying model matching and merging could be applied for the abstract annotation from Section IV-B3 and within the MbR frontend. Beyond merging, the MbR DSL also comprises support for formatting. For comparing annotation approaches, performance evaluations need to be conducted.

Monitoring and Reporting: In the context of BPM there is a variety of industrial products for business process monitoring, often delivered as part of the BPM product of the companies portfolio. Among them are WebSphere [5], webMethods BPMS [6], and NetWeaver [7] to name a few. WebSphere Business Monitor allows end-users to select performance indicators, i.e., which values should be monitored, and define KPIs. Execution can be monitored graphically and based on the process models. Similarly, NetWeaver enables stakeholders of business processes to monitor a process instance using the process model. Multiple instances can be visualized with another tool for process performance management [8]. This serves as a starting point for process optimization.

In these solutions there is usually some support for customization in regard to the formatting. In the example presented in this paper, e.g., the color and the dashstyle of the annotated sequence flows are dependent on values of reporting data. The consuming user can specify these formatting instructions. A distinction of this work is the decoupling of the frontend that is not limited to process models. Thus, the approach can be applied for various purposes and in different contexts. As a result of the decoupling, the frontend becomes interoperable with different products. As a side benefit, models are fostered in the systems landscape. In contrast to monitoring products, reporting solutions such as Crystal [9] currently provide much more facilities for selecting data, calculating values, and specifying the formatting. Yet, they do not apply data from runtime and analytics to models as presented. Nonetheless the concepts can be adopted for MbR such as formatting instructions that are much more elaborate than presented here for exemplifying the feasibility of the approach.

VIII. CONCLUSION

Enriching models as central development artifacts in MDE with data from the runtime and from analytics, MbR closes the business application life-cycle from execution and analysis to development. It eases the creation of reports for stakeholders through a tailored DSL which can further be facilitated with a GUI that guides the user in specifying the model-based reporting. In addition, MbR strengthens the role of models in business applications and permits to introduce new conceptual models and views that are potentially valuable beyond the scope of reporting.

[5]http://ibm.com/software/integration/business-monitor
[6]http://softwareag.com/corporate/products/wm/bpm/bpms
[7]http://sap.com/netweaver
[8]see http://scn.sap.com/docs/DOC-8435
[9]https://sap.com/solutions/sap-crystal-solutions

Acknowledgments

The author would like to thank Heiko Witteborg and Wei Wei (危巍) for realizing and contributing to the monitoring MbR frontend prototype, Andreas Roth and Anis Charfi for fruitful discussions, Dirk Mayer for reviewing the state-of-the-art, and reviewers for valuable comments. The work presented in this paper was performed in the context of the Software-Cluster project Emergent [10]. It was partially funded by the German Federal Ministry of Education and Research under grant no. `01IC10S01`.

References

[1] M. Völter and T. Stahl, *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.

[2] N. Ubayashi and Y. Kamei, "Architectural point mapping for design traceability," in *FOAL*, S. Katz, G. T. Leavens, and H. Masuhara, Eds. ACM, 2012, pp. 39–44.

[3] T. Holmes, U. Zdun, and S. Dustdar, "MORSE: A Model-Aware Service Environment," in *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC)*, M. Kirchberg, P. C. K. Hung, B. Carminati, C.-H. Chi, R. Kanagasabai, E. D. Valle, K.-C. Lan, and L.-J. Chen, Eds. IEEE, Dec. 2009, pp. 470–477.

[4] R. Frigg and S. Hartmann, "Models in science," in *The Stanford Encyclopedia of Philosophy*, spring 2009 ed., E. N. Zalta, Ed., 2009, [accessed in June 2012]. [Online]. Available: http://plato.stanford.edu/archives/spr2009/entries/models-science/

[5] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42–50, 1995.

[6] H. Tran, U. Zdun, and S. Dustdar, "View-based and model-driven approach for reducing the development complexity in process-driven SOA," in *BPSC*, ser. LNI, W. Abramowicz and L. A. Maciaszek, Eds., vol. 116. GI, 2007, pp. 105–124.

[7] J. Bézivin, "On the unification power of models," *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[8] A. Egyed and P. Grünbacher, "Automating requirements traceability: Beyond the record & replay paradigm," in *ASE*. IEEE Computer Society, 2002, pp. 163–171.

[9] International Telecommunication Union, "ISO/IEC 9834-8 information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 object identifier components," Sep. 2004, [accessed in June 2012]. [Online]. Available: http://itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf

[10] H. Tran, T. Holmes, E. Oberortner, E. Mulo, A. B. Cavalcante, J. Serafinski, M. Tluczek, A. Birukou, F. Daniel, P. Silveira, U. Zdun, and S. Dustdar, "An End-to-End Framework for Business Compliance in Process-Driven SOAs," in *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE Computer Society, Sep. 2010, pp. 407–414.

[10]http://www.software-cluster.org

[11] International Organization for Standardization, "ISO/IEC 14977:1996 Information technology – Syntactic metalanguage – Extended BNF," 1996, [accessed in June 2012]. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=26153

[12] "Cascading style sheets, level 2 CSS2 specification," May 1998, [accessed in June 2012]. [Online]. Available: http://w3.org/TR/xhtml1/

[13] O. Zimmermann, V. Doubrovski, J. Grundler, and K. Hogg, "Service-oriented architecture and business process choreography in an order management scenario: rationale, concepts, lessons learned," in *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM Press, 2005, pp. 301–312.

[14] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML) 1.1," Aug. 2006, [accessed in June 2012]. [Online]. Available: http://w3.org/TR/xml11/

[15] "Visual Studio Visualization and Modeling SDK," Microsoft Corp., [accessed in June 2012]. [Online]. Available: http://archive.msdn.microsoft.com/vsvmsdk

[16] Object Management Group, Inc., "Business Process Model and Notation (BPMN), Version 2.0," Jan. 2011, [accessed in June 2012]. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[17] "Modern business process management: Yawl and its support environment," YAWL Foundation, [accessed in March 2012]. [Online]. Available: http://yawlbook.com

[18] Object Management Group, Inc., "Unified Modeling Language (UML)," Mar. 2000, [accessed in June 2012]. [Online]. Available: http://omg.org/spec/UML

[19] I. Galvão and A. Goknil, "Survey of traceability approaches in model-driven engineering," in *EDOC*. IEEE Computer Society, 2007, pp. 313–326.

[20] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model traceability," *IBM Systems Journal*, vol. 45, no. 3, pp. 515–526, 2006.

[21] J. Richardson and J. Green, "Automating traceability for generated software artifacts," in *ASE*. IEEE Computer Society, 2004, pp. 24–33.

[22] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher, "Determining the cost-quality trade-off for automated software traceability," in *ASE*, D. F. Redmiles, T. Ellman, and A. Zisman, Eds. ACM, 2005, pp. 360–363.

[23] Organization for the Advancement of Structured Information Standards, "Web service business process execution language version 2.0," OASIS Web Services Business Process Execution Language (WSBPEL) TC, OASIS Standard, Jan. 2007, [accessed in June 2012]. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[24] D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "On-demand merging of traceability links with models," in *3rd ECMDA Traceability Workshop*, Jul. 2006, pp. 47–55.