Semantic Analysis of E-Business Operations

Mehmet Sayal, Akhil Sahai, Vijay Machiraju, Fabio Casati

Hewlett-Packard Laboratories

1501 Page Mill Road

Palo Alto, CA 94304

[mehmet_sayal, akhil_sahai, vijay_machiraju, fabio_casati@hp.com]

Abstract. An e-business infrastructure is comprised of a large number of business processes that are exposed through web services. To manage such an infrastructure, it is necessary for business managers to be able to observe their e-business operations in greater detail. In particular, a framework and a tool is required that allows business users to define qualitative and quantitative metrics, depending on their own (business or IT) goals. Once metrics have been defined, the tool should be able to measure them and support users in semantic analysis of the results, allowing them to quickly identify quality degradations as well as their causes.

Keywords: business processes, web services, analysis, management.

1 Introduction

An e-business infrastructure would comprise of large number of business processes [1]. Steps in a business process are handled by either humans (as is the case in work-flow management systems [2]), automated systems, or some times will be outsourced to external e-businesses. As the intention of an e-business is to undertake business on the web, they will also need mechanisms that enable customers to access their services through the Internet. *Web services* [3] are becoming a well-accepted way of putting e-businesses on the web so as to enable users (either humans or other web services) to use them. Web service refers to a service delivered using standard web technology, such as HTTP (Hypertext transfer protocol), XML (Extensible markup language) [4], and SOAP (Simple object access protocol) [5]. According to many

market research firms [6, 7], it is likely that, before the year 2005, many of the companies' offerings will be available as web services, and that large corporations will deploy tens or hundreds of web services. These web services have to be interfaced with the internal business processes to receive, fulfill and deliver orders. A complex infrastructure is usually a reality in an e-business. In order to manage the e-business infrastructure it is necessary to have frameworks and tools that allow business managers/analysts/users to define, compute, and analyze business and IT metrics.

In this paper, we present a platform called *Business Cockpit (BC)* that assists users in managing their processes and services. In particular, the cockpit supports users in defining, computing, and analyzing business and IT metrics on e-business data, in order to assess the quality of their operations, understand problems, and identify their root causes.

The business cockpit operates by tracking and logging message exchanges as well as the execution of internal business operations, by extracting logged data and transferring them into a data warehouse, and by then adding a semantic layer that enables the specification of user-defined, qualitative metrics on top of the detailed, low-level data extracted by the e-business infrastructure. For example, using the cockpit, users can define a metric *quote outcome* with values of *quote delivered on time, quote delivered late,* or *quote not delivered,* and define how it can be computed based on the available data. Once computed, measures can be analyzed under different perspectives and aggregated at different levels of abstraction.

One of our goals in designing the cockpit is its simplicity. In fact, users can deploy the cockpit (including the instrumentation, the data warehouse, and the semantic mapping layer) as well as define and analyze business metrics without writing code. This is in general a desirable feature that becomes a fundamental requirement when users are at the business level. Another important aspect of the solution is that users can look at *heterogeneous* collections of objects (e.g., different business processes or web services) in a *homogeneous* way, i.e., using a reduced set of metrics that can be used to analyze most or all of them. To achieve this goal, the cockpit separates the *definition* of the metric from its *computation logic*, depending on the specific object being measured. This independence enables the specification of different ways to compute the same metric for different objects. For example, the outcome of a supply chain process can be considered *successful* if the process ends its execution in a *purchased* state, while a driving direction service is *successful* if the directions are delivered within 3 seconds. In this way, analysts can assess at a

glance the behavior of their services, regardless of their type (e.g., they can see which services are successful or unsuccessful), easily identify the problematic ones, and then drill down to understand the source of the problem.

In the following, we first present a scenario that motivates our work and provide an example that will be used throughout the paper to demonstrate our solution. Then, we dive into the details of the cockpit, by presenting its three main components: the instrumentation, the data warehouse, and the semantic mapping layer. Finally, we show how the cockpit can be used to quickly and easily analyze e-business operations.

2 E-business Infrastructure

An e-business infrastructure will comprise of set of business processes that are exposed through web services to other e-businesses. This will necessitate definition of web services that interact with other web services. Analyzing e-business operations will necessitate analysis of business process and web service data.

We will use the following scenario for explaining our motivation, problems that we want to attack, and our solution. We assume that business partners provide web services or use third party web services in order to enable dynamic interactions with each other. Individual web services are implemented as business processes that are managed by workflow management systems. Figure 1 summarizes overall layout of our example scenario. The names of web services provided by participating businesses are shown in brackets.

The scenario includes the following participants:

- **Clients**: PCBuyer1 and PCBuyer2 are two example clients that purchase PCs from a vendor (PCMaker). We assume that some clients, such as PCBuyer1, are capable of using the web service provided by their vendors in order to submit their purchase orders; whereas, some other clients, such as PCBuyer2, prefer to submit their orders through fax or phone calls.
- **Providers**: PCMaker Sales Department provides the web service "PCSupply" in order to allow its clients to interact with itself electronically. This department also acts as client and uses web services of various service providers and product suppliers in order to satisfy its own clients' orders. In our scenario, PartSupplier, DeliveryProvider, and PaymentProvider are three external service providers, and PCMaker Assembly Department and PCMaker Manufacturing Department are internal service

providers. There may be many more providers, customers, and internal departments involved in such a scenario, but we list only few for simplicity.

The PCMaker.com e-business is shown in Figure 2. This infrastructure is set up by PCMaker.com that receives orders from companies/humans interested in buying PCs. It has internal business processes like user authentication, PC manufacturing, preparation of invoices etc. For some of the PC order parts, it needs to contact its supplier and similarly uses a shipping company to ship the PCs it makes. In order to do business on the web it needs to have a web service, that enables other users to access its e-business. The PCMaker.com web service has a web service that has operations, namely Login, Price_quote, Order_request, Send_invoice, and Send_shipment. It also has other operations, namely Order_parts and Ship_order.

Figure 3 shows the definition of a sample business process that runs at the back-end of Price_quote operation. The process is initiated when a Get_PriceQuote (PQ) request is received from a client. The process proceeds in three parallel branches to collect quote information from three business partners, then calculates the total cost, and finally sends the total cost to the client. Similarly, there could be many other business processes implemented to handle client requests at the back-end of a web service.

Behind all the logical business processes, web services and operation as described above, the e-business infrastructure needs hardware, software and humans that support the web service infrastructure. For example, it would require web sites, web server farms, applications servers and business processes execution platforms (e.g., Process Manager, MQSeries, Web Methods).

From a business manager's point of view, it is important to have overall information about the activities of a business and its interactions with business partners. For example, a manager at the Sales Department of a PC manufacturing company might want to know what percentage of the Purchase Orders (PO) with a particular client are satisfied within a reasonable amount of time. Such information may be critical for the manager in evaluating the success of the business. The manager may also want to see other summary information, such as total purchases made by a particular client, overall value of products sold by the business in the current month, or what percentage of clients that sent a Price Quote request also sent a Purchase Order within a week following the quote request. Moreover, the manager might like to define taxonomies over the data based on user-defined criteria and view the data using his own definitions. For

example, the manager might want to categorize the internal business process instances that handle POs into three categories: slow, acceptable, and fast. Then, she might want to see how many instances of POs fall into each category in order to estimate the satisfaction of the clients. Considering similar examples to those listed above, our goal is to provide high-level and flexible business view to managers so that they can observe and control both the activities inside their own businesses and the interactions with business partners.

3 Analyzing E-business operations

In order to manage the e-business infrastructure, it is thus necessary to obtain a complete picture and to collect raw measurement data from both the web service and business process infrastructure (Figure 4). The collection of measurement data is done through instrumentation as described in section 3.1. Once the data is collected, it is cleaned from inconsistencies and transferred to a data warehouse. The data model used in the warehouse is described in section 3.2. Once the data is collected and modeled, it can be used for semantic analysis by the business cockpit as described later in section 3.3.

3.1 E-Business Instrumentation

Instrumentation of e-business infrastructure consist of two stages:

- Instrumentation of web service interactions in order to keep track of message exchanges among business partners
- Instrumentation of business process management tools in order to keep track of internal business processes that automate internal activities

Before we go into details of each stage, we explain our basic assumptions. First of all, we assume that the web services communicate with each other using a standard protocol, such as SOAP. We also assume that the interfaces of web services and their interactions with each other are described using industry standards, such as Web Services Definition Language (WSDL) [8], Web Services Flow Language (WSFL) [9], and Universal Discovery, Description, and Integration (UDDI) [10] models.

We assume that most web services make use of a back-end software component, such as a Workflow Management System (WfMS), in order to carry out their complex tasks, which usually involve multiple

sub-tasks. We also assume that such WfMS either provide APIs in order to collect data about their execution, or log such data to a file or database table that is accessible by other software components.

We use the term Global Flow (GF) to describe the activities that are executed by the web services in order to satisfy a client request. A GF instance starts when a client submits a request, which is not triggered by the receipt of another message at the client side, to a web service. This initial request may cause the web service to submit sub-requests to other web services in order to carry out sub-tasks for satisfying the client's request. After getting back responses for the sub-requests, the web service responds to the client with one or more messages that explain the result of the initial request. The initial request message of the client, the sub-requests and responses that are exchanged among web services, and the result messages sent to the client describe the execution of one GF instance. We assume that web services use industry standards for communicating, exposing their interfaces, and describing the Global Flow of activities, so that it is possible to easily deploy a new solution on existing web services and start collecting and analyzing data about their interactions. Finally, we assume the existence of a unique identifier for each Global Flow (GF). We assume that each global flow instance can be assigned a Globally Unique Identifier (GUID), and this GUID will be forwarded among the web services and other back-end software components (e.g. WfMS) that are involved in the execution of the global flow instance.

3.1.1 Instrumentation of web service interactions

It is necessary to interfere with message exchanges among web services in order to collect information about the interactions with business partners. An acceptable solution should not impose any modifications or limitations on existing web services. Since SOAP is rapidly becoming the preferred standard for web service interactions, we assume SOAP messages are used among web services in order to submit request and response messages. We have implemented a small proxy component that tries to capture incoming and outgoing messages, and records data about the message exchanges, then forwards the captured messages to the actual recipients. We have considered various alternatives for easily attaching a proxy component to existing web services in order to listen to incoming and outgoing messages: port sniffing, server-side filters (Microsoft's Internet Server Application Programming Interface [11], or Netscape's Netscape Server Application Programming Interface [12]), API provided by web services themselves, and modification of SOAP toolkit. Port sniffing and server-side filters are usually not suitable because the message contents are

encrypted by SOAP toolkit. Most web services do not provide an API for controlling or querying about their activities due to security issues or simply because the web service developers did not feel any need for such interfaces. Consequently, we have chosen to keep track of message exchanges among web services by modifying SOAP toolkit. The most popular implementations of SOAP toolkits share common components, called routers. SOAP routers receive the messages from SOAP clients and submit them to the receivers. A proxy can be easily attached to SOAP toolkit routers with minor modifications to the toolkit. We used this approach for collecting data from SOAP message exchanges among web services, because it does not require any modifications to existing web services, and does not require re-compilation of existing SOAP toolkit installations.

In order to correlate individual message exchanges with each other, we use the notion of Global Flow (GF) as described within our assumptions above. The Globally Unique Identifier (GUID) is used for keeping track of a GF. Every time our proxy component catches a message that is exchanged between web services, it first checks whether a GUID exists. If a GUID does not exist in the message, the proxy inserts a GUID into SOAP header of the message. All web services and other software components propagate the GUID in their communications. Consequently, our proxy components that are attached to SOAP toolkits at business partner sites can easily figure out which SOAP message is sent in the context of which previous messages.

3.1.2 Instrumentation of business process

It is necessary to collect data from the software components at the back-end of web services in order to gather detailed information about internal activities of a business, and to correlate those internal activities with external message exchanges. As we indicated among our assumptions, most WfMS, such as HPPM, log data about internal business process executions into a raw log file or database. A proxy component can be configured in order to read logged data from proper database tables. Some WfMS provide interfaces or adapters for other software components to interfere with the execution of the business processes, or at least collect limited data about the progress of process execution. HPPM provides a Java API for process executions to be controlled by other software components. A proxy component uses this Java API to feed in the GUID into HPPM process instances and retrieve it when necessary. Since GUID is passed into WFMS as one of the parameters, the log files of WFMS contain the GUID, and therefore the internal

activities that are managed by WFMS can be easily correlated with external activities using the GUID. This approach applies to other internal software components as long as they provide an API to pass parameters and a means to retrieve the execution logs.

3.2 Modeling e-business infrastructure

Once the instrumentation data is collected it is necessary to model the data in the data-warehouse for semantic analysis by the business cockpit. The data model that we use is inspired by the web service and business process standards. These standards (e.g., WSDL and WSFL) propose how web services should be defined, how business processes should be defined, and the latter should be interfaced with the former.

3.2.1 Standards

In WSDL, a web service is implemented through a set of endpoints. An endpoint is in turn a set of operations. An operation is defined in terms of messages that they receive and send out:

- Message – an abstract definition of data being communicated. Messages are made up of message parts.
- Operation – an abstract definition of an action supported by the service. Operations are of the following type namely, one-way, request-response, solicit-response, notification.
- Port type – an abstract set of operations supported by one or more end points
- Binding – a concrete protocol and data format specification for a particular port type
- Port – a single end point defined as a combination of a binding and a network address
- Service – a collection of related end-points

WSFL introduces the notion of activities and flows – which are useful for describing both local business process flows and global flow of messages between multiple web services. WSFL models business processes as set of activities and links. An activity is a unit of useful work. The links could be control links where decisions are made to follow one activity or another, data links where data is fed into an activity from another. These activities could be exposed through one or more operations that are grouped through end-points (as defined in WSDL). A service is comprised of a set of end-points. A service provider can provide multiple services. Just like internal flows, global flows can be defined. Global flow consists of plug

links that link up operations of two service providers. This helps in creation of complex services that can be recursively defined.

3.2.2 Model

The data warehouse uses a model inspired by the web services and business processes. It models the e-business infrastructure in terms of web service constructs (namely operations, end points, services) and business process constructs (namely business processes, activities, plug-link, global flows).

A business owns a set of processes, each of which consists of a set of activities. Each of these activities could be implemented or exposed as an operation. An end-point is a construct that groups operations together. An end-point is associated with an address and a protocol (TCP/IP, http, etc). A service could be comprised of a set of end-points. Each operation is defined by messages that they exchange. A service-provider groups services. A plug-link links two operations from two different services. This is useful for modeling a B2B infrastructure that involves multiple suppliers and customers. A UML representation of the managed object model with these constructs is shown in Figure 5.

3.3 Business Cockpit Metric Model

The management infrastructure presented in the previous sections is in itself very beneficial to users that need to analyze business operations, in that it provides a repository of operational data collected from different, possibly heterogeneous platforms, cleaned from errors and inconsistencies always present in operational systems, and endowed with correlation information, so that analysts can identify the different data items that relate to the same Global Flow. However, this by itself is not sufficient to fulfill the analysis requirements described earlier in the paper. Indeed, the analysis of the data repository can provide users with reports on the average execution time for each node in the process, or on the total number of executions, but such reports are of limited usefulness to business analysts, who are interested in information that is at a higher level of abstraction, such as the number of successful quotes depending on the customers, suppliers, or items for which the quote has been requested. In addition, besides these “objective” reports, users may want to define subjective criteria for the analysis of operational data, to look at process and service execution under their own perspective. For example, they may want to define their own notion of quality, and be able to analyze which processes, services, customers, suppliers, or other entities are responsible for low quality execution of the business operations.

In order to enable this *semantic* analysis of operational data, the cockpit allows users to define business and IT *metrics* for measuring and analyzing e-business operations. Once metrics have been defined, the cockpit can compute measures and support users in analyzing results. Note that the purpose of the business cockpit is not that of providing a complete model or ontology so that all business metrics relevant for the business process and web service domain are predefined and available with the cockpit. Instead, our goal is to enable users to easily and quickly define the semantic concepts (i.e., metrics) that they consider relevant, as well as the computation logic for mapping low-level execution data into business metrics.

The metric framework is composed of three basic entities: the *metrics* to be computed, the *mappings*, that define how operational data can be mapped into qualitative and quantitative measures, and *meters*, that define which mapping should be used to compute which metric depending on the element being measured, thereby allowing the homogeneous analysis of heterogeneous objects, as described in the introduction (see Figure 6). We next discuss these three different components of the semantic analysis cockpit model.

3.3.1 Metrics

In the cockpit, a metric is characterized by:

A *name*, unique within the cockpit system.

A *data type*, that can be *Numeric*, *Boolean*, or *Taxonomy*. For taxonomies, the definition also includes a description of the categories that are part of the taxonomy. For example, a *quote outcome* taxonomical metric could be characterized by categories *quote delivered in time*, *quote delivered late*, *quote not delivered. Quote amount* is instead an example of numeric metric.

Metrics (like every other cockpit abstraction) can be defined in XML, internally transformed into insertions into the cockpit's (relational) database. Alternatively, we provide an user interface that can be used for both defining and analyzing metrics (see Figure 7).

3.3.2 Mapping templates

A mapping template is a parametric definition of a mapping from operational data into a numeric or Boolean measure. Examples of templates are:

A. *Did conversation end in state **S**?* (Boolean template)
B. *Number of invocations* (numeric)
C. *Percentage **P** of the value of numeric output variable **V*** (numeric)

D. *Did at least **P**% of the operations take less than **M** minutes to complete?* (Boolean).

Meters can then instantiate templates (i.e., reuse templates with specific values of the parameters) and apply them to specific elements in the data warehouse, to define how measures for a given metric are computed, depending on the elements being measured.

Templates are defined by XML documents. Each template includes a *specification* part and an *implementation* part. The specification part has three purposes: first, it provides a human readable description about the purpose of the template and about how to use it (i.e., the semantics of the template parameters). Second, it provides information that can be consumed by a (Java or otherwise) Graphical User Interface (GUI). This is important, because new templates can be dynamically added to the cockpit library: Since different templates may have different characteristics (e.g., different number and type of parameters), the GUI that assists users in selecting and instantiating templates also needs to be dynamic, and must be able to "understand" which are the valid options that can be given to users in managing templates. Third, it provides information that the cockpit engine can use to optimize measure computation.

The *implementation* part contains the (parametric) code executed by the cockpit to compute the metric. In the current prototype, the implementation is specified in SQL, although other languages may be used if the data sources are not relational (the cockpit passes the query for execution to the underlying DBMS). More in detail, the specification part of a template includes a name (unique in the cockpit template library), a description, and a set of attributes:

- *Composition type* defines whether the mapping is only based on service data (*direct*), or it is an aggregation (*aggregate*) or composition (*composite*) of other measures.
- The *target entity* specifies the kind of elements that are measured (e.g., process flows, services, etc).
- The *return type* defines the data type of the value returned by the template. This can be either *Numeric* or *Boolean*.
- For each template parameter, the definition includes the indication of the parameter name, data type, default value, and a textual description that can be used to convey the semantics of the parameter (for example, whether a numeric value is interpreted by the template as describing seconds, minutes, Dollars, Euros, etc). Data types include any SQL type.

Template definitions have other parameters used by the run time engine to compute measures in the most efficient way. The description of such parameters is omitted due to space limitations.

The implementation part of the template is represented by an SQL query that returns (i.e., *selects*) a tuple compatible with the cockpit tables where measures are stored. The tuple includes the measure plus attributes qualifying the measure, such as the identifier of the element being measured and the measurement time. In the condition part, the query must be able to capture, for all meters using the template to compute measures, both the template instantiation parameters and the set of elements that should be measured with this template, in order to compute measures appropriately.

As an example, we next show a simple template that determines whether the quoting process described above belongs to the category *quote not delivered*. The determination depends on which node is executed in the process: if node "notify inability to quote" is executed, then the mapping logic will determine that the process instance falls into the *quote not delivered* category of the *quote outcome* metric. We observe that the template definition is rather simple, and only requires basic XML and SQL knowledge. In addition, the cockpit assists users in preparing templates in that it automatically inserts the (SQL) code necessary to capture the template instantiation parameters and the set of elements that should be measured. In the example below, code that is automatically generated is shown in italics.

```
<MAPPING>
<NAME>Node executed</NAME>
<GROUP>Process-node</GROUP>
<DESCRIPTION>Determines whether a node has been executed at least once in a process instance
</DESCRIPTION>
<RETURN_TYPE>BOOLEAN</RETURN_TYPE>
<TARGET_ENTITY> PROCESS_FLOW </TARGET_ENTITY>
<CONTAINED>TRUE</CONTAINED>
<INVARIANT>TRUE</INVARIANT>
<COMPOSITE>FALSE</COMPOSITE>
<COMPLEMENT>Node not executed</COMPLEMENT>
<MAPPING_PARAMETERS>
<PAR>
        <PAR_NAME>Node name</PAR_NAME>
        <PAR_DESCRIPTION>
                Name of the node whose execution must be detected
        </PAR_DESCRIPTION>
        <PAR_TYPE>STRING</PAR_TYPE>
</PAR>
</MAPPING_PARAMETERS>
<IMPLEMENTATION>
<TYPE>SQL</TYPE>
<CODE>
INSERT INTO MEASURES
(
SELECT DISTINCT
MTE.METRIC_ID,
```

```
MTE.ID,
E.ID,
SI.PROC_DEF_ID,
SI.PROC_INST_ID,
NULL,
NULL,
MTE.CATEGORY_ID,
NULL,
SYSDATE,
NULL,
NULL
FROM ACTIVITY_INST SI, NODE_DEFS_D ND, PROC_FLOWS_D PD, MAPPINGS MP,
U_PROCESS_EXECUTIONS PE, ACTIVE_METERS MTE, CONTEXTS CTX, METER_PARS METP, ENTITIES E
WHERE MP.NAME="Node executed"
AND MP.GROUP_NAME="Process-node"
AND E.EXTENDED_NAME="PROCESS INSTANCE"
AND MTE.MAPPING_ID=MP.ID
AND METP.METER_ID=MTE.ID
AND CTX.METER_ID=MTE.ID
AND SI.NODE_DEF_ID=ND.ID
AND ND.NODE_NAME=METP.VALUE
AND SI.PROC_DEF_ID=PD.ID
AND PD.ID=PE.PROC_DEF_ID
AND ND.PROC_DEF_ID=PD.ID
AND (
        (PD.PROC_GROUP_NAME=CTX.PROC_GROUP OR CTX.PROC_GROUP IS NULL) AND
        (PD.PROC_NAME=CTX.PROC_NAME OR CTX.PROC_NAME IS NULL) AND
        (PD.PROC_VERSION=CTX.PROC_VERSION OR CTX.PROC_VERSION IS NULL)
        )
)
</CODE>
</IMPLEMENTATION>
</MAPPING>
```

The cockpit includes a large number of predefined, built-in templates, stored in the cockpit *template library*. Pre-defined templates range from simple ones, such as the one defined above, to more complex ones, computing measures on composite services, such as the number of basic services invoked within a composition, or the total operation cost based on the sum of the costs of the invoked operations. Users can define additional, more sophisticated templates that are aware of the extended service model, and possibly of other user-defined data structures.

3.3.3 Meters

Meters are the "instruments" used to compute metrics. In particular, meters define which *mapping* should be applied to compute a *metric* within a given *context*. For example, a meter can define which mapping should be used to compute the metric *quality* for *price quote* operations, while another meter can define a different mapping for computing the *quality* of *process order* operations. It is this capability of using

different meters for different contexts that allows users to map heterogeneous elements into a homogeneous metric, i.e., allows analysts to see all elements under the same lens.

Meters are specified by instantiating templates. To do so, each meter must specify which template it instantiates and which are the values to be given to instantiation parameters. For example, to compute the metric *quote outcome*, and specifically to compute whether an instance belongs to category *quote not delivered*, the analysts can reuse mapping "process node executed" (that detects whether a certain node was executed within a process instance), instantiated with parameter "notify inability to quote".

```
<METER>
<NAME>Compute quote rejected</NAME>
<DESCRIPTION>Computes the semantics of rejection </DESCRIPTION>
<VERSION>1</VERSION>
<METRIC>
        <METRIC_NAME>Quote outcome</METRIC_NAME>
        <METRIC_GROUP_NAME>Supply Chain</METRIC_GROUP_NAME>
        <CATEGORY_NAME>Quote not delivered </CATEGORY_NAME>
</METRIC>
<MAPPING>
        <MAPPING_NAME>Node executed</MAPPING_NAME>
        <MAPPING_GROUP_NAME>Process-node</MAPPING_GROUP_NAME>
</MAPPING>
<ENABLED>TRUE</ENABLED>
<MEASUREMENT_TIME>SYSTEM</MEASUREMENT_TIME>
<MAPPING_PARAMETERS>
<PAR>
         <PAR_NAME>Node name</PAR_NAME>
         <PAR_VALUE>notify inability to quote</PAR_VALUE>
</PAR>
</MAPPING_PARAMETERS>
<CONTEXTS>
        <CONTEXT>
                <ENTITY_NAME>PROCESS_FLOW</ENTITY_NAME>
                <ELEMENT_NAME>PRICE QUOTE</ELEMENT_NAME>
        </CONTEXT>
        </CONTEXTS>
</METER>
```

In a meter, the *context* defines the set of elements used as the base for the computation, and it includes a *space* and a *time* component. The space component restricts the set of elements being considered in the computation. For example, the above described meter should only be applied to *price quote* processes.

The time component, if present, limits the computation to elements (processes, operations, etc..) that have been started, completed, or both started and completed within a specified time window.

Finally, we observe that mappings can compute metrics not only based on operational data, but also on the measures computed for other metrics. In this way, the cockpit natively supports composite metrics.

3.3.4 Metric Computation

The previous sections have introduced the basics of the metric model, characterized by the notions of metrics, meters, and mappings. In the following we detail the behavior of the cockpit, specifying in particular how business metrics are computed out of low-level execution data, collected in the logs.

The computation proceeds as follows: periodically, data are collected from the execution logs and inserted into the data warehouse, as shown at a high level in Figure 4 and in more detail in Figure 8. The new data will typically include executions of process and operation instances. Upon loading the data into the warehouse, the *Template Execution Engine* executes all the mapping templates defined in the cockpit and that are used by at least one metric. Note that due to the set-oriented nature of SQL and to the way mappings are written, a single execution of a mapping template *mp* computes data for *all* the metrics that use *mp* as their computation logic (i.e., all the metrics *M* such that a meter linking *M* with *mp* has been defined). Results are stored into a "Precomputed_measures" table.

The above described technique is applicable if mappings are *invariant*, i.e., if the metric data they compute for a certain element (e.g., a process instance or a web service operation execution) do not change with time. For example, consider the computation of the cost for a process instance, based on some mapping logic defined in a mapping template. This information can be computed at the time process instance data are loaded into the warehouse, and does not change afterwards. However, consider instead the computation of a *ranking* metric that defines, for each process instance, its rank with respect to other instances, based on its duration. This metric cannot be computed as the process instance data is loaded into the warehouse, as it is likely to change whenever data from other (faster) process instances become available. In this case, instead of precomputing metric data by executing the mapping templates and inserting results into a persistent table, the system generates database views out of mapping definitions. These views will be accessed when the user request metric data, thereby providing for the on-the-fly computation of up to date metric data (although performances will be slower with respect to accessing precomputed metrics). The information about whether a mapping template is invariant must be provided by the user at mapping definition time.

4 E-business Operation Analysis

The previous sections have shown how metrics can be defined and computed in the cockpit model. In the following we focus instead on quality measure *analysis*. The cockpit metric database (shown in Figure 6) is structured according to data warehousing techniques. Measures, along with operation and conversation execution data, are represented as *facts*. These facts can be analyzed according to several warehouse *dimensions,* represented by the different entities in the warehouse data model (such as processes and activities) and by the metrics that need to be analyzed. Structuring measure information along a star schema such as the one shown in the figure enables and simplifies multi-dimensional analysis of quality measures, and allows the cockpit to take advantage of the star schema optimization techniques offered by most DBMSs.

Once metrics have been defined, the cockpit operates by defining *database views* that can be used to access measures. Users also have the option of pre-computing measures as soon as new data are loaded into the warehouse. To compute measures, the cockpit executes the templates associated to the meters that have been defined. Note that, due to the set-oriented nature of SQL, a single execution of a mapping template is sufficient to compute all the metrics that make use of that template.

One the metrics have been defined and operational data collected by the instrumentation starts flowing into the warehouse, users can begin analyzing their operational system according to the different perspectives they consider significant.

In particular, the cockpit allows users to view metric data from three different perspectives:

- By *elements*, that is, depending on the element on which the metric has been computed. For example, users can view how the distribution of a *quality* metric varies with the process flow being considered
- By time. For example, users can view how the value of an operation-level metric changes with the operation invocation time
- By metric. For example, users can view how the value of a metric "execution cost" changes with the value of metric *quality*, thereby enabling the identification of correlations among these two metrics.

We next provide an example of how the cockpit can be used to analyze business metrics. Assume that the metric *quality* (including categories good, fair, and poor) has been defined for several different processes.

By combining meters and mapping templates, users can define different ways to measure quality depending on the process being considered. The definition of the quality metric as well as of the meters is performed through a GUI. Then, users can analyze how the quality distribution varies with the process being considered, as shown in Figure 9. The figure displays data in the form of a stacked bar chart. The width of the bars is proportional to the number of elements used in the computation. This is helpful in that in not only gives an indication of low quality area, but also of the significance of the quality problem, i.e., how many process executions it affects.

The chart enables the identification, in one shot, of the low-quality areas, regardless of what the definition of "quality" is for the different elements being considered.

Users can then "zoom in" on problematic processes to further analyze the problem. For example, the chart shows that most processes have satisfactory quality. However, almost 75% of the processes belonging to group "Supply Chain" have either *fair* or *poor* quality. Users can then "zoom-in" into the problem, to further analyze it by drilling down to the details. In the present example, users can drill down by clicking on the "problematic" bar in the chart and requesting a detailed analysis of supply chain processes, for example by service provider, as shown in Figure 10, thereby revealing that many of the problems are caused by the invocation of *money transfer* services within supply chain processes.

Note that the polymorphic nature of the metric model enables analysts to not only examine the quality across different processes, but also switch from process quality to service quality. The metric being analyzed is always the same.

In this way, analysts can quickly and easily detect problems and identify or locate their causes. We observe that, unlike "traditional" analysis tools, that typically require the users to (1) be aware of the details of the data model, (2) write code to define new metrics, and (3) write code to get reports, by using the business cockpit analysts can easily add new metrics, as well as flexibly define their computation logic depending on the elements being considered.

Further analysis (not shown here) on the problem can be performed by accessing the meter used for the computation, to understand what exactly is the nature of the low quality detected by the tool.

5 Related Work and Conclusions

To the best of our knowledge, there are no platforms that enable the definition, computation, and analysis of quality metrics on web services. However, there is a huge body of research on metrics in general, in just about every field of science, ranging from physics to philosophy. In fields close to web services, a lot of work has been done in the domains of networks, distributed systems, and distributed objects running on top of interoperability platforms. Despite the fact that this earlier work had different scope and objectives, it inspired some of our design choices. In the following, we briefly present some existing contributions, and highlight differences with respect to the approach we propose.

Networks and distributed systems are probably the areas in which the most work on measuring and management issues has been done, resulting in the development of standards and even commercial management platforms, such as HP OpenView or IBM Tivoli. For example, ITU-T in the early nineties has defined recommendations for measuring distributed software applications, and for deriving statistics from collected measures [13, 14]. In particular, ITU-T defines a framework for measuring and monitoring distributed objects, to standardize the interface and protocols of managing and managed applications. The basic component of the framework is the *metric object*, that specifies how a certain application object should be monitored by defining how the application object's attributes should be transformed into measures (represented by *counters, gauges*, and *derivatives* of gauges), and which are the measure thresholds above which a management application should be notified. Metric objects can also define data collection details such as sampling frequency. Support for the definition of statistics on measures is also provided (mean managed class, or moving average managed class).

Although ITU-T recommendations, like other network and distributed systems standards and applications (such as the Simple Network Management Protocol and the Open Group Universal Management Architecture [15]) are concerned with metrics for distributed objects, they mostly focus on interfaces and protocols for management applications, and are mostly concerned with performance and monitoring issues. They do not deal with what we consider to be the innovative aspects of this paper, i.e., how to define IT and business-level metrics for web services, how to separate metric definition from computation, how to reuse metric computation logic, how to perform metric aggregation over user-defined web services models, how

to compute and analyze such metrics efficiently, and how to make the tool easy to use and accessible to business users.

More recently, several research contributions have appeared in the measurement of distributed objects running on top of middleware platforms.

The majority if these contributions focus on the definition of *Quality of Service* (QoS) criteria. This is relevant to the approach described in the paper, as QoS is one of the most common metric that business users want to analyze. For example, Adam [16] is a tool for the management of QoS requirements in middleware applications. It assumes that basic measures are available, and it then allows the definition of *composite measures* by processing basic metrics through arithmetic functions. QoS goals can then be specified constraints on composite measures.

An interesting approach is presented in [17]: the paper proposes a language, called QML (QoS Modeling Language), for specifying QoS criteria and associating QoS requirements to objects' interfaces. QoS requirements are defined by *contracts*, that define constraints on the value of a set of basic metrics (such as *average response time < 10 msec*). *Profiles* can then be used to associate contracts to interfaces. The framework also includes a programming language, called Activity Monitoring Language (AML), that declaratively specifies what should be measured, and provides abstractions to map end-user business metrics into a Service Level Objective (SLO). AML includes the notion of *events*, qualified by a name and a set of parameters. Objects that produce events are called *providers*. Metrics can be associated to providers, to specify ho events should be transformed into measures meaningful for the user.

The above approaches have been inspiring, in that they propose the notion of having high-level, user-defined metrics as well as a method for composing basic metrics into complex ones, which are two problems that are also addressed in this paper. However, the approach presented in this paper has many differences with respect to the above: it is focused on web services, enabling the definition of extensible service models and the automatic computation and aggregation of metrics on top of the user-defined model. It is more flexible in the metric definition and computation, going beyond QoS and allowing more generic forms of composition, and including support for measure *analysis*. Other important differences are in the provision for a measurement system that performs correlation among operations executed within the same

global flow, and in the development of a concrete implementation that has a strong emphasis on simplicity, being (also) targeted to business users.

Finally, we mention that contributions in the area of metrics for web services exist, but are focused on the collection of performance measures to support performance and availability, and capacity planning. They do not address the problems and therefore do not include any of the innovative features discussed in the introduction. The interested reader is referred to [18] for a comprehensive coverage of this topic.

We have used the approach to undertake automated SLA monitoring[19]. . We plan to extend the approach to automated SLA assurance in order to foresee SLA violations,react to such (predicted) violations, in order to avoid their occurrence or at least to reduce their impact.

References

1. Frank Leymann, Dieter Roller. Production Workflows. Prentice Hall, 2000.
2. The Workflow Management Coalition. The Workflow Handbook 2002. Future Strategies Inc. 2002
3. Ethan Cerami. Web Services Essentials. O'Reilly. 2002
4. Natanya Pitts and Cheryl Kirk. The XML Black Book. Coriolis Technology press, 2000.
5. Don Box, David Ehnebuske, Gopal Kakivaya. Simple Object Access Protocol (SOAP) 1.1. Available from http://www.w3.org/TR/SOAP/
6. Milind Govekar. Managing Total Business Integration. *Procs. of the Garner Symposium ITXPO*. Cannes, France. Nov. 2001
7. Kerstetter, J. The Web at your service. BusinessWeek e.biz cover story. March 18, 2002. http://www.businessweek.com/magazine/content/02_11/b3774601.htm
8. Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Available from http://www.w3.org/TR/wsdl
9. Frank Leymann. Web Services Flow Language (WSFL) 1.0. May 2001. Available from http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
10. The UDDI consortium. UDDI Technical White paper. September 2000. Available from uddi.org.

11. Internet Server API (ISAPI) Extensions. Available from http://msdn.microsoft.com/library/en-us/vccore98/html/_core_internet_server_api_.28.isapi.29_.extensions.asp
12. Netscape Server API (NSAPI) Programmer's Guide. Available from http://developer.netscape.com/docs/manuals/enterprise/nsapi/index.htm
13. International Telecommunication Union. Systems Management: Summarization function. Recommendation X.738. Nov 1993.
14. International Telecommunication Union. Systems Management: Metrics Objects and Attributes. Recommendation X.738. Nov 1993.
15. Open Group, UMA model. Available from http://www.opengroup.org
16. Joseph Martinka, Kave Eshghi, An Architecture for Adaptable Distributed Application Management, HP Technical Report HPL-96-30, March 1996.
17. Svend Frolund, Jari Koistinen. Quality of Service Specification in Distributed Object Systems Design. Distributed Systems Engineering Journal 5(4), Dec. 1998
18. Daniel Menasce', Virgilio Almeida. Capacity Planning for Web Services. Metrics, Models, and Methods. Prentice Hall. 2002.
19. Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad van Moorsel, Fabio Casati. Automated SLA Monitoring for Web Services. IEEE/IFIP DSOM 2002. Montreal, Canada, Oct. 2002.

Mehmet Sayal received his Ph.D. in Computer Science from Northwestern University in February 2000. His areas of interest include content delivery networks (CDN), data mining, e-business applications, B2B interactions, workflow management, and business intelligence solutions. He is currently a software researcher at HP Research Labs.

Akhil Sahai is a senior scientist at HP Laboratories, Palo alto. He is currently researching service management, solutions and architectures. He was one of the initial members of the e-speak team that shaped Hewlett Packards's pioneering web services technology. He has led research earlier at Multi-Media Systems group at Kent Ridge Digital Labs (earlier ISS) Singapore. He has about 7 years of industrial experience. He has a ph.D is computer science from INRIA-IRISA France and Master's degree in computer science from Indian Institute of Science Bangalore. He has published widely in distributed systems, network/system/service management and mobile computing.

Vijay Machiraju is a Research Manager at Hewlett-Packard Laboratories. He received a Masters degree in Computer Science from the University of Utah in 1997. Since then, he has been leading research projects at HP in the area of management of distributed systems and e-services. His research interests are in the areas of distributed systems, architectures and frameworks, application and service management systems, and agent technologies. Earlier, he has conducted research on parallel computers, multiprocessor task scheduling, and fault tolerance and has numerous publications in these areas.

Fabio Casati is a researcher at HP Labs, Palo Alto. He got his PhD from Politecnico di Milano (Italy) in 1999. His research interests include workflow management, web services, and business intelligence. He is author of more than 40 papers in international conferences and journals, and has served as organizer and program committee member in several conferences.

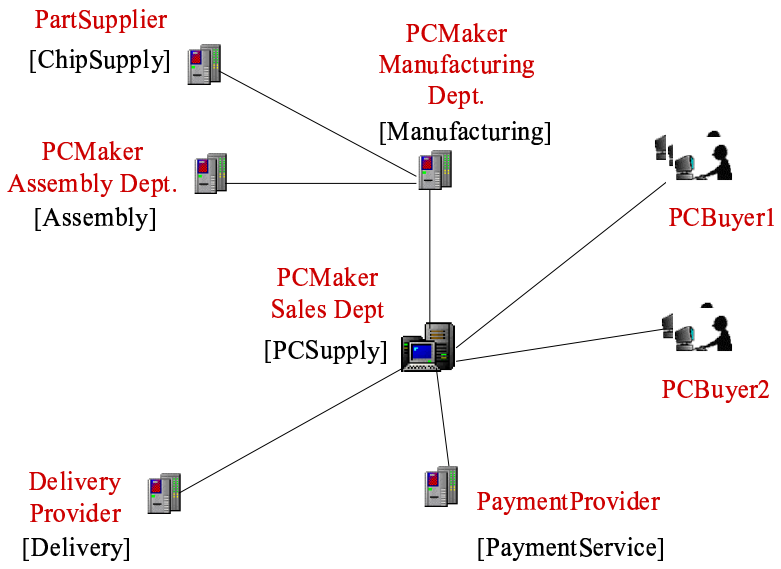

Figure 1: PC Purchase scenario

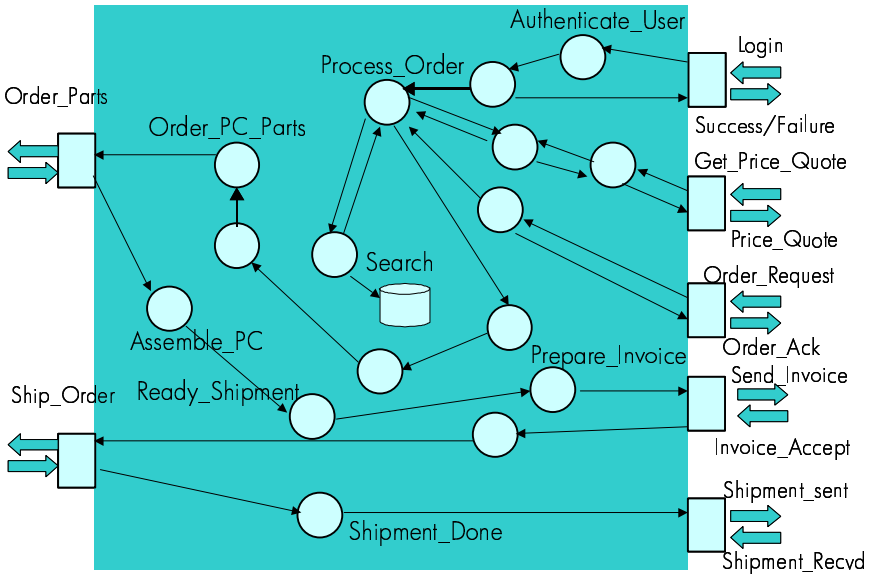

Figure 2: Business processes and web services in PCMaker

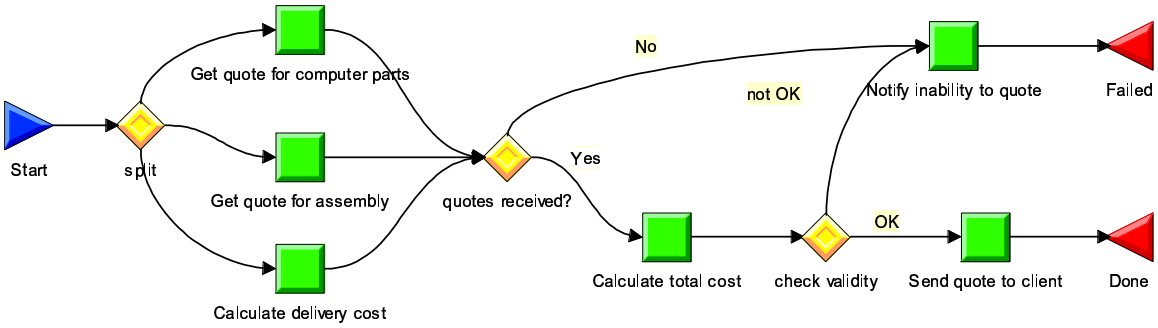

Figure 3: Business process that handles price quote requests

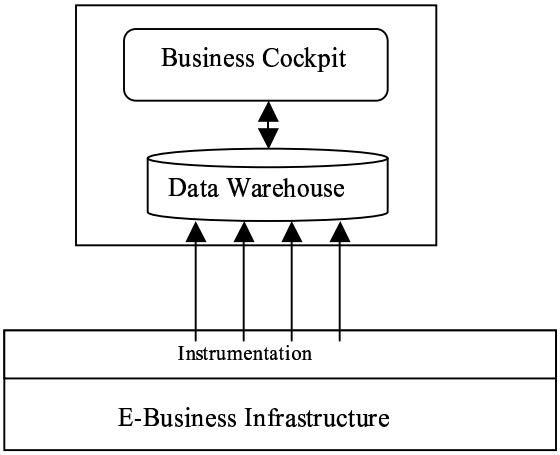

Figure 4: Architecture of business cockpit

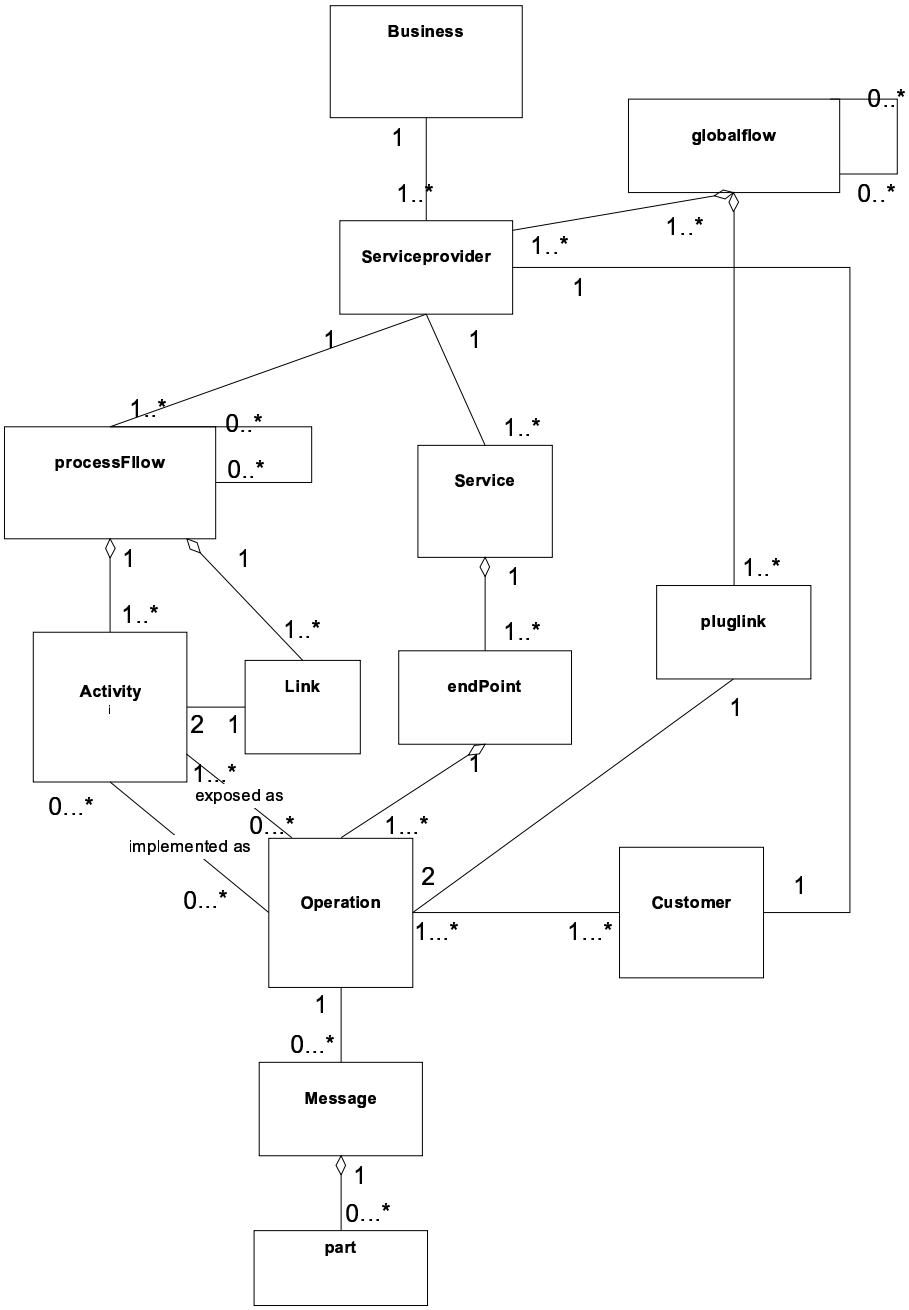

Figure 5: Managed object model in data warehouse

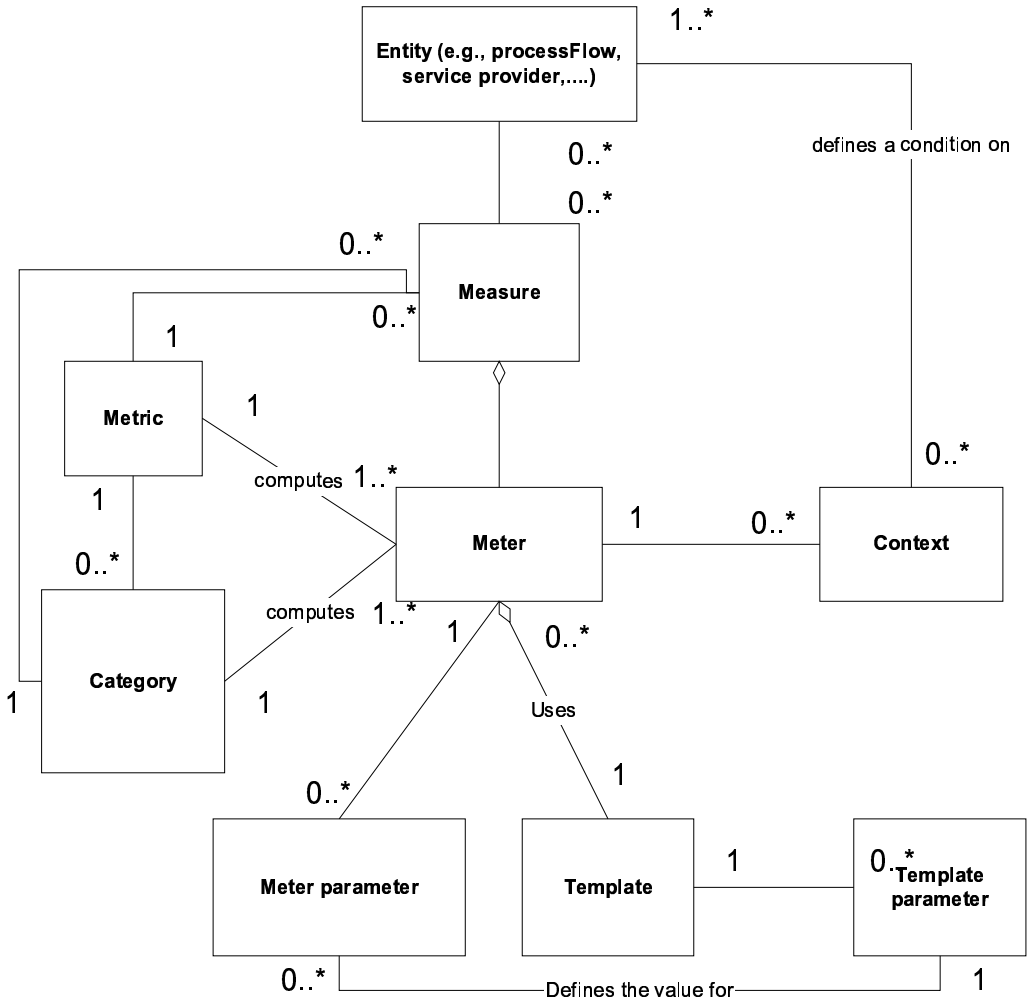

Figure 6 - Schema of the cockpit metric database

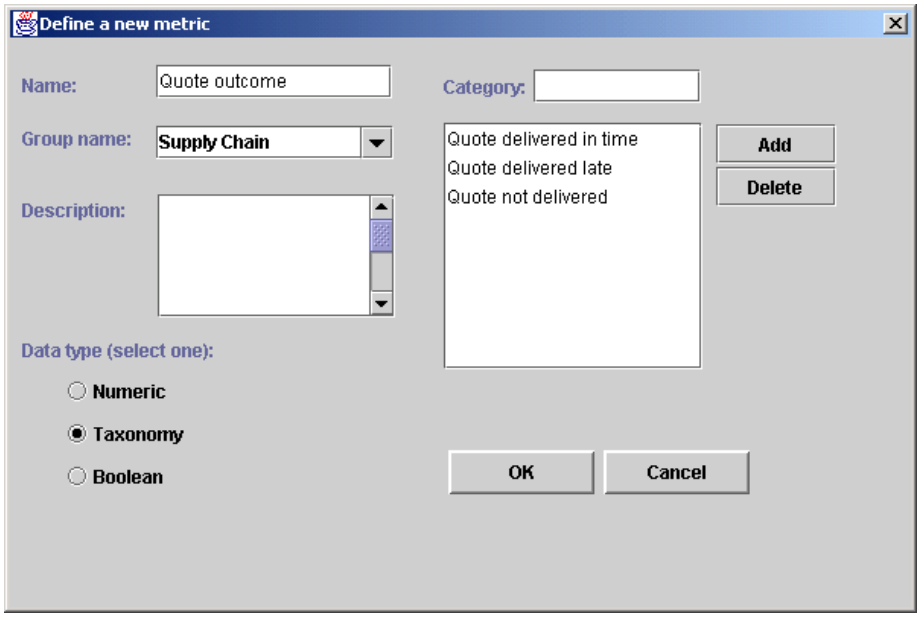

Figure 7 - metric definition

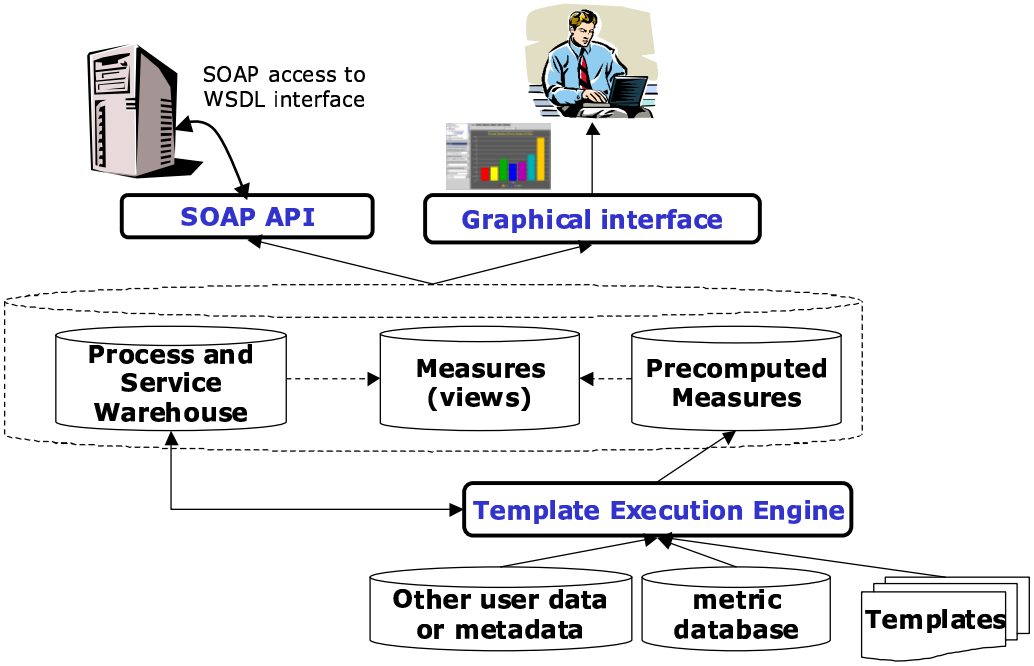

Figure 8: Business cockpit

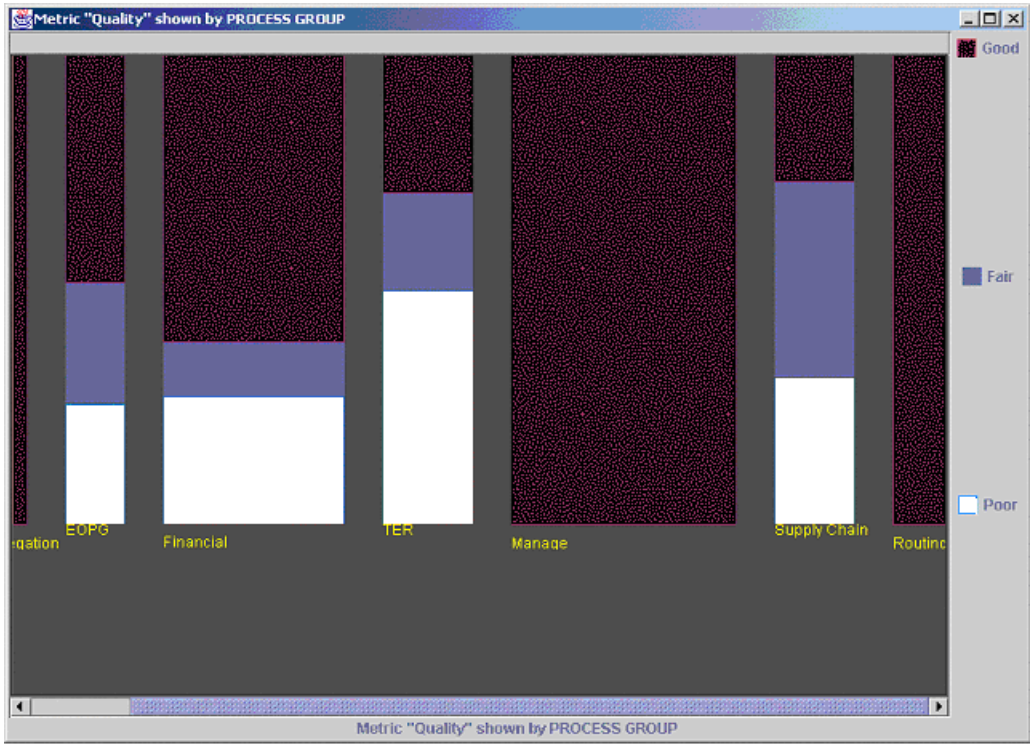

Figure 9 - Quality distribution analysis by process group

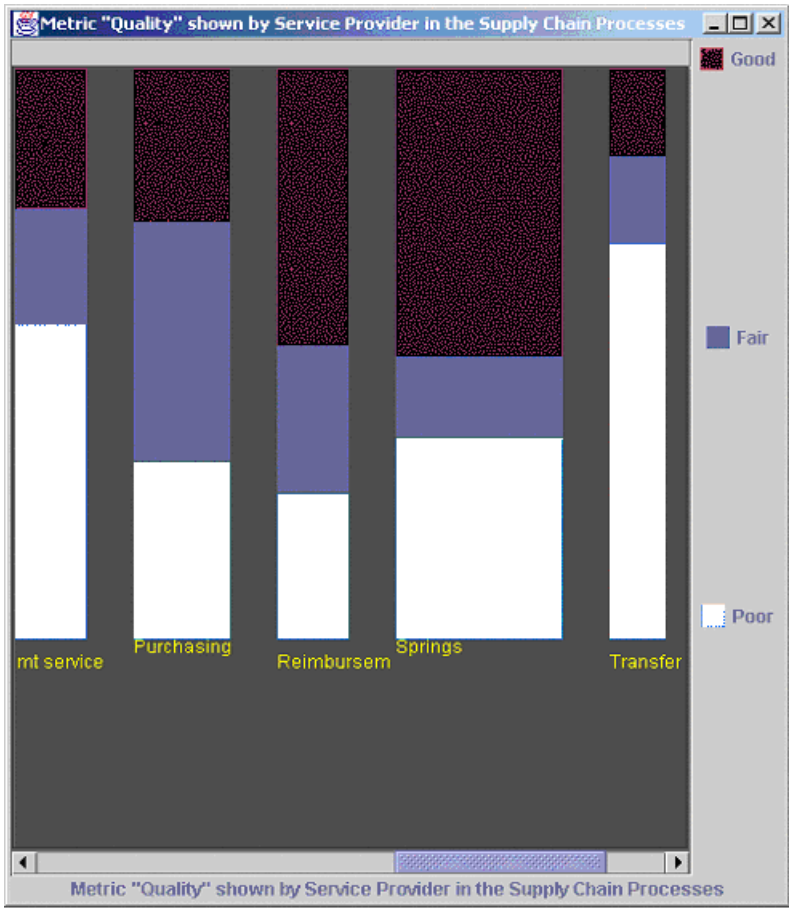

Figure 10 - Quality metric shown by service provider invoked in the execution of supply chain processes