RC23820 (W0512-007) December 1, 2005
Computer Science

IBM Research Report

Proceedings of the First International Workshop on Service-Oriented Business Processes Integration (SOBPI'05)

Amsterdam, The Netherlands, December 12, 2005

Held in conjuction with the 3rd International Conference on Service Oriented Computing (ICSOC '05)

[1]Stéphane Gagnon, [2]Heiko Ludwig, [3]Marco Pistore, [4]Wasim Sadiq (Eds.)

[1]New Jersey Instiute of Technology

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[3]University of Trento, Italy

[4]SAP Research
Brisbane, Australia

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

Preface

A widely recognized trend in business and markets is the push towards the idea of companies and enterprises as networked organizations, which can gain profit from collaborations in the net, improving their flexibility, and reducing operational costs. This transformation requires the adoption of more collaborative working practices based on the integration of business processes within a wide community of business partners, suppliers, vendors, and public bodies.

In parallel to this trend, service oriented technologies are transforming the web from an infrastructure for sharing information to a place where the networked organizations can meet to integrate their business interests. Service Oriented Computing (SOC) is currently the most promising technology supporting the development and execution of business processes that are distributed among the most disparate entities, both within an organization and across organizational borders.

However, a practical industry-wide adoption of web services, to support the collaboration among networked organizations and the integration of their business process, is still an open challenge. This challenge can be addressed only by a tight integration of service-oriented development within new business models and new ways of management that are able to exploit new technological solutions. Moreover, web service technology is still emerging, and therefore their industry-wide adoption requires to inject research results within real industry practices, but, even more important, to drive research activities and roadmaps according to industrial needs.

The aim of this workshop is to bring together researchers, practitioners, and experts in business models and management and to promote cross-fertilization among their competencies, focusing on the usage of service oriented technologies to support the Business Processes Integration (BPI). The goal is to build bridges and convergences among the complementary views of experts in IT and service oriented technologies and experts in management, and the different approaches of academy and industry, to the problem of business processes integration. The emphasis of the workshop will be on the demonstration of the practical application of research results, and on the description of industrial case studies.

December 2005

Stéphane Gagnon
Heiko Ludwig
Marco Pistore
Wasim Sadiq

SOBPI'05 chairs

Organization

Workshop Chairs

Stéphane Gagnon (New Jersey Institute of Technology, USA)
Heiko Ludwig (IBM Watson Research Center, USA)
Marco Pistore (Univeristy of Trento, Italy)
Wasim Sadiq (SAP Research Brisbane, Australia)

Program Committee

Morad Benyoucef (University of Ottawa, Canada)
Antonio Brogi (University of Pisa, Italy)
Jen-Yao Chung (IBM Watson Research Center, USA)
Marlon Dumas (Queensland University of Technology, Australia)
Schahram Dustdar (Vienna University of Technology, Austria)
Elena Ferrari (University of Insubria at Como, Italy)
Oleg Gusikhin (Ford Research & Advanced Engineering, USA)
Patrick Hung (University of Ontario Institute of Technology, Canada)
Anup Kumar (University of Louisville, USA)
Miguel Vargas Martin (University of Ontario Institute of Technology, Canada)
Angel Ortiz (Polytechnic University of Valencia, Spain)
Katia Passerini (New Jersey Institute of Technology, USA)
Pascal Pecquet (Université de Montpellier I, France)
Manfred Reichert (University of Twente, The Netherlands)
Colette Rolland (Université Paris 1 Panthéon Sorbonne, France)
Richard Soley (Object Management Group, USA)
Cheickna Sylla (New Jersey Institute of Technology, USA)
Yazhe Tang (University of Western Ontario, Canada)
Kerry Taylor (CSIRO, Australia)
Vladimir Tosic (Lakehead University, Canada)
Hannes Werthner (University of Innsbruck & EC3 Vienna, Austria)
Andreas Wombacher (University of Twente, The Netherlands)
Dan Zhang (University of Ontario Institute of Technology, Canada)
Jia Zhang (Northern Illinois University, USA)
Aoying Zhou (Fudan University, China)

Table of Contents

Technical papers

Information Exchange Among Collaborating Enterprises

David Levermore[1], Gilbert Babin[2], and Cheng Hsu[1]

[1] Rensselaer Polytechnic Institute,
Decision Sciences and Engineering Systems, Troy, NY, USA
leverd@rpi.edu, hsuc@rpi.edu
[2] HEC Montréal, Information Technologies, Montréal, Québec, Canada
Gilbert.Babin@hec.ca

Abstract. On-demand business/service and other emerging models for service enterprise integration all involve on-demand information exchange. The best practices in the field tend to rely on homogeneous semantics, which is difficult to achieve for independent databases owned by independent enterprises. Otherwise, the exchange tends to be limited at the level of file transfers. To overcome these limits and achieve on-demand pull of information resources at the global database query level, we develop a new information exchange model to extend previous global query results and cover independent databases. The new model provides a four-schema architecture to allow enterprises to offer the information that they wish to share with others (query for users), as well as request what they want (query for data). A central element is information matching; which has to unify the representation and processing of both offering and requesting queries, and integrate them with traditional global database query results. We develop such a new information matching model in this paper. It employs a unique "query database" approach, an "export database" design, a "four-schema" architecture, and new matching algorithms that promise computing efficiency to achieve the purpose. The new method also allows for inclusion of constraints (rules) in the matching for information exchange.

1 On-Demand Enterprise Collaboration

A challenging problem in service enterprise integration is to drill through supply chains to coordinate schedules on a real time and online basis. To do this, trading partners need to go beyond the traditional "fixed" cooperation and develop on-demand information exchange for flexible collaboration. An example of fixed cooperation between partners is the effort that a retailer (e.g., Wal-Mart) links its demand forecasting databases with its manufacturer's production scheduling systems (e.g., Warner-Lambert) to shorten the replenishment cycle for certain products. This example is not new, but the collaboration still faces daunting technical limits. First, the information exchange was hard-wired rather than being on-demand; and second, the inter-operation mechanism was

not easy to extend to other likely participants in the supply chain. In principle, these two problems can be facilitated by global database query results such as federated databases; however, these results run into problems, too, because they require the participating companies to open up their databases for direct inter-operation. Furthermore, the trading partners have a many-to-many relationship among them; therefore, even if the prime (e.g., Wal-Mart) could impose a single authority on the entire supply chain, the members still have to reconcile this particular Wal-Mart standard with their other primes or buyers (i.e., other federations).

The above example shows that supply chain integration is ***on-demand collaboration*** in nature, characterized by independent databases that control when and what information to offer and to whom, as well as issue queries for information. These databases may also project different images/personalities onto different global models in their inter-operation with other systems. There are other examples of independent databases, such as the participants in Homeland Security and industrial exchanges. The traditional global query assumptions do not cover them.

In this context, business process integration must rely on flexible information exchange mechanisms between systems. Then again, these exchange mechanisms must be able to evolve as local processes change and collaboration requirements increase. In this paper, we propose the Two-Stage Collaboration Model (TSCM) as a framework that support such flexible and evolvable information exchange mechanisms. The TSCM architecture and methods enable an enterprise to "offer" information it is willing to share with partners or to "request" information it needs to perform its internal processes. Information matching comes naturally when we realize that the unique characteristics of independent databases are actually their ability to make information offering bids to many potential users, as well as issue information requesting bids (with or without a pricing mechanism). As such, the TSCM constitutes an information service infrastructure, as it provides a matching service between offers of and requests for data. On-demand information exchange at the database level, as described herein, is an extension of the database query (SQL) capabilities that companies have come to expect of all enterprise databases. Queries provide ad hoc information for decision support, and hence complement the application level collaboration, and enhance/support service level collaboration.

The TSCM integrates market-style information matching with research results on global query processing. In a general sense, research has shown that market-style self-allocation of users/providers [4] is a promising solution approach that supports the collaborative paradigm of global database query. However, previous results of artificial markets (e.g., Covisint and CommerceOne [5, 6]) that support collaboration do not include global database query; and market style global query [12] has not afforded on-demand offering. More to the point of this paper, previous matching methods are focused on matching bids based on price and product definition, rather than matching database queries and views based on information semantics. While the matching on price could be academi-

cally intriguing [11], the basic matching on product is straightforward in practice since product definition can be standardized for particular commercial practice. Information semantics (including rules), on the other hand, is inherently more difficult to match. To solve the matching problem, one needs a common representation method to define the requests and offerings, to lodge the active information bids, and to match them according to their rules and data semantics. In this work, we employ a metadata representation method developed for multiple databases integration [9] as the basis for developing these new results. The basic logic is that both information requests and offerings are formulated as global queries (metadata) against the collaborating community, using the method as the query language. These queries are stored in a database (the query database) whose structure is also based on the metadata representation method, and hence is consistent with the language. The query database manifests the universe of community collaboration at any given time. Each request or offer is both a runtime database query against the query database and a persistent object to be saved (updated) in the query database. Therefore, the matching is performed directly when newly created queries are processed as ordinary database operations against the query database. Search algorithms determine the cases for complete matches and various partial matches, and take into account the participants' conditions.

The core of the paper is the new matching method: the metadata language, the query database, and the search algorithms. However, to properly present these results, we briefly review in Section 2 the context of the new matching method: a complete solution approach to the global query problem under the collaborative paradigm. The new results are briefly presented in Section 3. The last section, Section 4, concludes the paper with a summary of the status of this ongoing research.

2 The Information Architecture for the Collaboration

The overall information architecture of on-demand enterprise collaboration features a **four-schema** concept shown in Figure 1. which supports the following **Two-Stage Collaboration Model**. Local schemata (first schema) represent enterprise data as stored in their local databases. Export databases are used to store data that the enterprise wants to make available to partners. These export databases are described using a publication query, which acts as a schema of the data made available (second schema). Subscriptions describe data that is required by partners (third schema). All these are represented in the Blackboard which extends the Metadatabase (fourth schema). The Two-Stage Collaboration Model includes matching for collaborators and fetching of the required information, for on-demand enterprise collaboration. The technical problem involved is referred to as the participatory database query problem in [9]. The first stage matches information offerings with requests in a market style, and establishes optimal participation of databases for a search task. The second stage, then, executes the task in a traditional distributed database query manner. The model

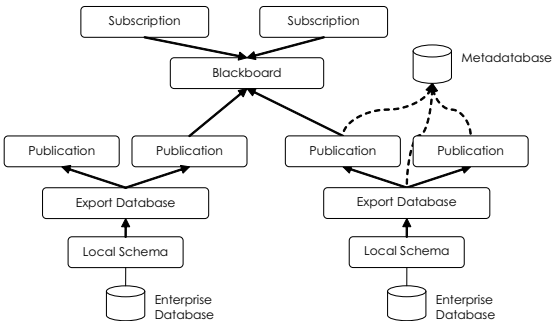


Fig. 1. Four-schema Architecture of the Two-Stage Collaboration Model

uses a global blackboard, but also distributes the blackboard to local sites as mini-blackboards to allow for peer-to-peer implementation. Requirements to the new model include that, at the first stage, it reconciles the different data semantics at the participating databases to allow determination of the "right" matches, and provides massively concurrent bidding, matching, and, if price is involved, auction/negotiation that lead to self-allocation of databases for particular tasks. The second stage needs to perform concurrent global queries for each task, according to its conditions and constraints, and, if necessary, join partial results from various sources.

The query database approach developed for matching integrates the usual market functions of bidding, matching, and auction/negotiation with global query processing. In this approach, the usually custom-designed **blackboard** of a market that conducts these market functions becomes an ordinary database management system, and can be implemented by using off-the-shelve DBMS technology. To implement this concept, the **query database** requires an integrated representation of data (information contents) and rules (task conditions and constraints) since the queries involve both. This requirement goes beyond traditional query processing. As a response, this research develops the dedicated databases collaboration language that we call exMQL (extended Metadatabase Query Language) to formulate the queries (and execute them). The semantics and syntax of exMQL are rooted in the Metadatabase model [8], which readily provides the schema for the query database — i.e., the queries are represented in and stored as semantic constructs of the Metadatabase. The exMQL is a metadata language based on SQL; thus, exMQL expressions are processed by the DBMS of the query database using the PL/SQL facility. The blackboard algorithms are also constructed in PL/SQL. The use of the PL/SQL facility affords the flexibility of adopting matching-negotiation-auction results from elsewhere. Comparing to concepts adopted in the literature, the query serves the purpose of a software agent on the market, and the query database satisfies the goals of the previous agent community.

Alongside with the query database, a **Metadatabase** (a repository of metadata structured as a database) is created to include and integrate local data

models (for the export databases, or the views that participants publish) into metadata tuples and tables, with the help of a registration process. As shown in the literature [1–3, 8], metadata entries (data models) are added, deleted, updated, and retrieved as ordinary database operations against the Metadatabase without interrupting the continuous operation of the market. Since the Metadatabase is also implemented using a standard DBMS, it allows for high intensity and concurrent updates. The Metadatabase facilitates the reconciliation of data semantics for query processing (matching) at the query database. A formal registration process with varying degrees of central coordination is required of all participants, through a CASE (computer-aided software engineering) tool. In the maximum registration regime, the Metadatabase integrates all published local views; while in a minimum regime, the Metadatabase contains only global equivalence information. The local sites are responsible for maintaining a global version of their respective database views in the minimum regime. In any case, the participants could opt to register a large, stable data model within which they publish their smaller, *ad hoc* offerings as often as they wish; or, they could frequently register different, small local data models that represent their published views and change them on the fly. The most demanding part of the registration is the establishment of a list of global data items across local sites, and the maintenance of the data equivalence among them. This is not an easy task and could easily become the bottleneck to any real-time, peer-to-peer collaboration. We stress, however, that this is a common problem in the field and the databases collaboration model does not add to the burden; if anything, the new model could ease the problem because the participants are now committing their proxies rather than production databases to a standard medium. We also wish to point out that the Metadatabase method affords a list that is automatically derived from all registrations, and can reveal any peer-to-peer correspondence of local data items. In this sense, the Metadatabase serves as an open common schema for the community.

The second stage, global query processing, is essentially an extension of the traditional database query against the export databases. The differences reside mainly in the *ad hoc* conditions and constraints under which the queries are processed; since the information requests and offerings may include additional requirements. The Two-Stage Collaboration model addresses this issue by extending established results from the Metadatabase research, including GQS (global query systems) [3], ROPE (rule-based programming environment) [1], and rule-base processing methods [2]. The new result becomes an integrated query processing and management system for the blackboard. The exMQL-based queries map into SQL expressions to be processed at the proxy servers of the local sites, which may inter-operate with the local enterprise databases depending on the local policies. Results are assembled at the blackboard according to the particular cases of joint implied by the original tasks.

The local sites are connected to the global site and to each other through a **proxy server** which is constructed and maintained according to a global design but is otherwise administered by the local site at which it resides. In ad-

dition to the export databases, the proxy server also replicates the blackboard, along with the capacity to maintain a query database and Metadatabase, into its functions for any local site that requires peer-to-peer information exchange capabilities. The proxy server, then, has the ability to initiate a virtual circle of matching among peers and serve as the "global" site of the circle during the life of the match. In this way, the global blackboard co-exists with many concurrent, distributed local blackboards and their virtual sub-communities of information exchange. This design promises to reduce the possible computing bottlenecks in the community, enhance overall system reliability (against system or network failures), and support maximum flexibility of the collaboration regime. The proxy server may reduce its requirement on a full Metadatabase when peer-to-peer metadata interfacing or interchanging is included. Similar to the minimum regime of registration, a minimum set of metadata at proxy servers includes only the list of global data equivalents. A partial Metadatabase ranges between the minimum and maximum contents. A continuing challenge facing this model is the maintenance of the distributed copies of global equivalents. It represents an upper bound to the real-time performance of peer-to-peer collaboration.

3 The Information Matching Method

We briefly present below the new query database method developed in this paper for the first stage, to provide the matching for information exchange. The metadata language that defines queries for the matching also defines the queries for final global query processing at the local sites. In a similar way, the query processing algorithms that perform the matching also extend to include functions pertaining to the second stage, the fetching of information for users.

3.1 The Query Language: exMQL

The Extended Metadatabase Query Language (exMQL) is designed to provide a uniform query format for the various query operations that are required in the Two-Stage Collaboration. The structure is derived from the original MQL specification in [3], which in turn is based on the TSER representation method [10] and the GIRD metadata model [8]. However, it extends significantly from the original MQL to support information publication and the inclusion of constraints/conditions in the queries. The full specification of exMQL is provided in Figure 2. Each query operation is performed via the extended Metadatabase Global Query System (exMGQS) and so the query specification described below is provided for illustrative purposes only.

3.2 The Query Database Schema

The conceptual structure of the Blackboard in general is based on a derivative of the GIRD, the integrated representation of the Metadatabase [8]. The requirements of the GIRD are relaxed for the purposes of the Blackboard, and so a

```
<QUERY> ::= <COMMAND> <ITEMS> ['DO' <ACTIONS>]* ['FOR' <CONDITIONS>]* ;
<ITEMS> ::= ITEM [, ITEM ... ] ;
<COMMAND> ::= 'GET' | 'PUT' ;
<CONDITIONS> ::= <CONDITION> <conjoin> <CONDITION> ;
<conjoin> ::= 'AND' | 'OR' ;
<CONDITION> ::= <SC> | <JC> | <NC> ;
<SC> ::= ITEM <bound> VALUE ;
<JC> ::= ITEM <bound> ITEM ;
<NC> ::= ATTRIBUTE <bound> VALUE ;
<bound> ::= '<>' | '=' | '<' | '>' | '< =' | '>=' ;
<ACTIONS> ::= action [, action ... ] ;
<DELETE_QUERY> ::= 'DELETE' query_name ['CASCADE'] ;
<DELETE_RULE> ::= 'DELETE' rule_name [, rule_name ...] 'IN' query_name ;
<DELETE_CONDITION> ::= 'DELETE' condition_name [, condition_name ...] 'IN' query_name ;
<UPDATE_QUERY> ::= 'UPDATE' <ITEMS> ['DO' <ACTIONS>]* ['FOR' <CONDITIONS>]* 'IN' query_name ;
```

Fig. 2. The Syntax of exMQL

number of the elements of the GIRD are unused but are not removed in the information model. We will highlight the changed features in any discussion of the Blackboard. The intent of the Blackboard is no different from the Metadatabase, in that it is used for the management of metadata, specifically the schema of the queries. The new structural model of the Blackboard is illustrated in Figure 3 using the TSER modeling methodology [10], and we discuss the conceptual structure of the query database and rulebase in the sub-sections that follow.

The Structural Model of the Query Database. The addition of the SYSTEM, QUERY, and VIEW meta-entities, which replace the APPLICATION, SUBJECT and ENTREL meta-entities, are the primary differences between the structural model of the Blackboard and the GIRD. The remaining meta-entities, and meta-relationships: Functional Relationships (FR), Mandatory Relationships (MR) and Plural Relationships (PR) that connect them, retain their original definitions as outlined in [2, 8]. We briefly illustrate the most relevant elements of the query database in the paragraphs below.

The SYSTEM meta-entity identifies the proxy database servers that are currently participating in global query, and so represents a dynamic view of the Two-Stage Collaboration System. Each proxy database server is defined by a unique identifier, which is determined at runtime. However, this is not provided to the Blackboard unless a query has been submitted by the proxy database server.

The QUERY meta-entity identifies the queries submitted by proxy database servers. Every query is unique, and each proxy database server can submit more than one query at any time. A timestamp attribute is generated automatically by the Blackboard and associated with the query, to facilitate the measures utilized during the query assignment process. The **components** meta-MR identifies the queries that belong to each SYSTEM.

The VIEW meta-entity provides a mechanism to associate data items with multiple queries, in various arrangements.

The ITEM meta-entity remains unchanged from its original definition, and represents the data items specified in each query. The **belongto** meta-PR rep-

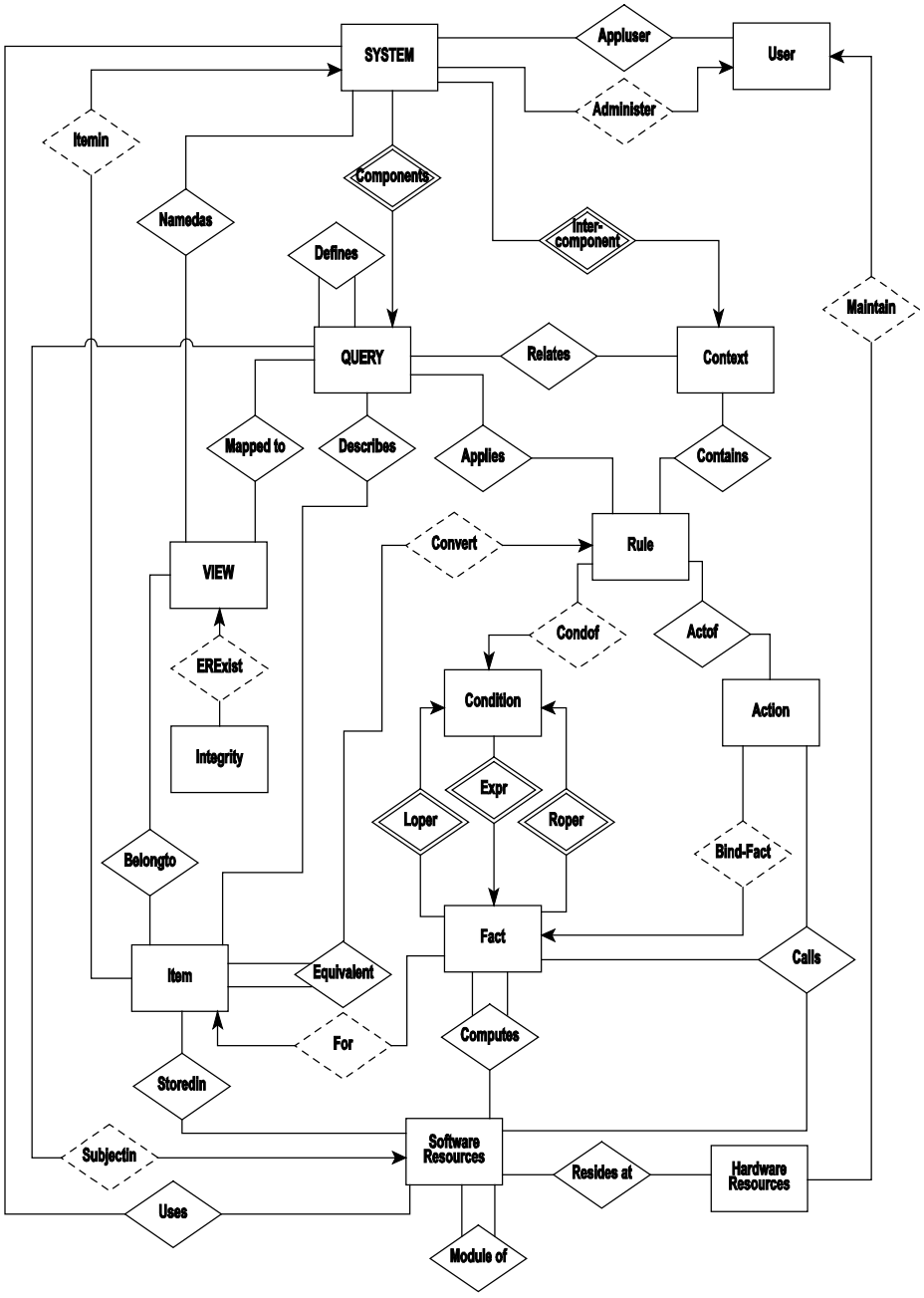


Fig. 3. Conceptual Structure of the Blackboard

resents the assignments of items to a specific VIEW. The **describes** meta-PR specifies the data items that are aggregated into each QUERY.

The Structural Model of the Rulebase. The rulebase inherits the rule modeling paradigm of the GIRD as illustrated in Figure 3. The RULE, CONDITION, and ACTION meta-entities, and their associated meta-PR's, meta-FR's and meta-MR's. retain their original functionality, however they now must provide for the management of data objects as opposed to metadata objects. Specifically, rulebase facilitates the modeling of all constraints (conditions) and associated actions.

3.3 Query Processing

Following the formulation of a query, the participant submits the query for processing. This processing involves three distinct phases: (1) match, (2) assign, and (3) execute.

Search for Matching Queries. In the match phase we identify queries (which we denote as target queries) that contain any number of data items specified in the submitted query (which we will denote as the source query). Note that either type of query (source or target) can represent a information request or offer. The search process for a given query (S) can produce four classes of results; (1) an *exact* match, (2) a *closest subset* match, (3) a *subset* match, and (4) a *closest* match. Exact match is not illustrated.

Definition 1. *In an* exact match*, the number of data items, the syntax, and the semantics of the data items (in other words, the items that match) in each target query are equivalent to those in the source query.*

Definition 2. *A* closest subset match *indicates that the number of items matched is less than that specified in the source query but equivalent to those defined in a target query.*

Definition 3. *A* subset match *indicates that the number of items matched is equivalent to the number of items in the source query, but less than that the number of items specified in the target query.*

Definition 4. *A* closest match *is the case where queries contain common items. That is the number of items matched is less than the number of items specified in both the source query and the target query.*

As a first step, we take a source query as input and compare it with a set of target queries. For each target query that has common items with the source query, we count the number of data items that intersect and classify these according to the aforementioned definitions. The result of this step is the set of queries that match the source query.

Permute Queries to Identify a Feasible Solution. If we are unable to identify target queries that contain all the data items in the source query, then we enter into an additional step of processing. Here, we permute the results found (most likely closest subset and closest) and evaluate each of the resulting permuted queries to determine if they contain the data items found in the source query. The resulting permuted queries are classified as, (1) *permuted exact* match, (2) *permuted closest subset* match, (2) *permuted subset* match, and (4) *permuted closest* match, where each, respectively, is the analogue of the four classes posited above.

A permuted exact or permuted subset match does not indicate that a solution for the source query has been found; rather, only that the disparate combined queries contain data items common to the source query. For the permuted query to be a feasible solution, the disparate queries that constitute the permuted query must be logically connected. We illustrate this problem with the following example:

Example 1

$S = \{item_1, item_2, item_3, item_4\}$
$T_A = \{item_1, item_2, item_n, item_{n+1}\}$
$T_B = \{item_2, item_3\}$
$T_C = \{item_4, item_m, item_{m+1}\}$

Let S, T_A, T_B, and T_C correspond to the source query and target queries illustrated in Example 1. Obviously, T_A, T_B, and T_C match S on particular data items. The union of T_A, T_B, and T_C, which we denote as T_{ABC} is the permuted query that contains all data items in the source query; however this is not necessarily a feasible solution. If T_{ABC} is to be a feasible solution for S then there must be logical relationships among T_A, T_B, and T_C. The Metadatabase is employed to evaluate the permuted queries. In this case, the Metadatabase will reveal that T_A contains the primary key $item_1$. Accordingly, we find that there exists a transitive dependency between T_A and T_B, as demonstrated below, but this will only be true if $item_2$ is the primary key for T_B. Since, $T_A : item_1 \rightarrow item_2, item_n, item_{n+1}$, and assuming $T_B : item_2 \rightarrow item_3$ then, by Armstrong's inference rules (transitivity), $item_1 \rightarrow item_3$. Therefore, T_A and T_B are logically connected. If $item_2$ was not the primary key for T_B then we could not postulate this result.

By consulting the Metadatabase we find that the attributes of T_C are dependent on the primary key $item_0$. Even if a logical relationship exists between both T_A and T_B, we cannot determine T_{ABC} since $item_0$ is not included in T_C. This leads us to the following definition that each permuted query must respect to be considered a feasible solution for a source query.

Definition 5. *To be considered a* feasible solution*, a permuted query must be logically connected, and all logically connected data items must exist in the target queries.*

Example 2 illustrates the permutations of the three target queries, T_A, T_B, and T_C. It is safe to ignore Round 1, since these correspond to the singular target queries.

Example 2

Round	Permutations
1	T_A, T_B, T_C
2	T_{AB}, T_{AC}, T_{BC}
3	T_{ABC}

The feasibility of a permuted query is determined by the Shortest Path algorithm [3], which determines if the entities and relationships (which in the Metadatabase are referred to as oe/pr, respectively) in a query are connected. However, since permuted queries do not contain oe/pr's, then these must be determined as well. The result of this second step is the permuted query (or queries) that contain the greatest number of data items common to a source query.

Evaluation of Constraints. A successful match process is also contingent on the agreement of preferences that have been associated with queries. Each query can be optionally qualified with constraints that restrict or limit a query to specific values, or restrict or limit the query in general to particular system-defined variables. These constraints are manifested as rules, adhering to the Event-Condition-Action paradigm [1] where each rule is processed subsequent to an event occurring, which will execute an associated action if the evaluation of the related condition returns true. In general, all events correspond to a successful query match.

As a last step, we evaluate the rules associated with the query from the steps above. If multiple target queries are determined, we retrieve all rules and combine them into a single fact table. We then iterate through each condition in the source query, and tests the associated facts with those of the target query (or queries) in the fact table. If we identify a matching attribute in the fact table, we evaluate the condition, i.e., comparing the values in the input condition with the output condition. If a match is **not** found, we then mark the specified conditions for additional processing. We repeat this process until all conditions in the source query are exhausted and return the set of conditions that do not match.

Assign Queries. After a match (or matches) has been found, then we enter the assignment phase where the source query is assigned for processing. The basic task is to determine the proper space of query processing, i.e., what subscribers and what publishers are to participate in this particular global query processing. It is trivial if there is only one match (exact or joint) to be assigned for processing; but it becomes complicated when multiple matches must be considered together. There are a number of criteria by which these ties may be broken if the "competing" queries are equivalent in every respect, i.e., they are of the same class of match, and the conditions that match the source query are equivalent.

A basic, and default, criterion we consider in this research is to permit multiple subscriptions of a single publication, and the single subscription of the multiple publications. That is, we would execute all equivalent multiple matches and present all the results for the users to interpret. We call this the **maximum collaboration** principle, which makes sense for information exchange (but not for merchandise trading) since information can be shared without loss by multiple users and fused freely from multiple sources. However, if this principle is not true for whatever reasons, then either a round of auction/negotiation could be conducted or the system could apply some automated decision rules. The selection of these criteria is essentially a matter of system design.

The list below provides a few basic functions that facilitate the choice of automatic tie breaking among multiple matches. The users could specify their preference of any of these choices during query construction as actions of a query. Note that multiple actions can be specified for a single query.

Simple heuristics – uses the system-defined timestamp of each query to select a winner. The query with the oldest/latest timestamp, i.e., the query that has been registered with the Blackboard for the greatest/least amount of time is selected.

Network Performance – considers the geographical location of the associated proxy database server and the load of the server, assuming that further distances and loads contribute to performance latency, and chooses the match that is in closer proximity to its location.

Participant History – the proxy database server that has reliably provided solutions for many participants and received good ratings will be chosen over a proxy database server that has not been similarly prolific.

User Preference – the winner will be chosen from a list of attributes of the proxy database servers that have been specified during query construction, such as the locality, organization, and other indicators of the site that the users prefer.

Execute. The execution of the query on the publication databases constitutes the final phase of query processing. It is necessary however, to perform some data conversion and translation on the queries before they can be processed on the publication databases. Briefly, all data processing that exists beyond the boundaries of the publication databases are defined in global terms, and so queries must be converted to local values before they can be processed on a proxy database server. Conversely, when the publication query is created the parameters are converted to global values. In general, the subscription query is always created in global values, since the exMGQS provide the interface of the global information model for the construction of information requests.

The conversion algorithm employs the global equivalence functionality of the Metadatabase to (1) identify equivalent data items in the query, and (2) convert to the data items to the local/global value depending on the perspective. In that event, a unit conversion is required, which the Metadatabase allows for; the algorithm in concert with the Metadatabase will address this as necessary.

4 Beyond the new matching method

We might submit that the global database query capabilities are a key to achieving service enterprise integration under virtually all conditions. The problem is only that how much results the field affords. We develop a new approach to the problem and thereby extend the previous results to support independent databases, which are a basic characteristic of on-demand enterprise collaboration. A new enterprise integration architecture is developed for the approach, which includes the two-stage collaboration model and the four-schema design, as well as the new information matching method.

The query language and processing algorithms span both the matching (the first) stage and the global query processing (the second) stage of databases collaboration, and hence provide the integration of the two for the final system. The algorithms constitute a major part of the Blackboard. The new matching method is currently being tested in a laboratory setting for the purpose of performance evaluation as well as for the verification of the Two-Stage Collaboration model. Beyond the matching method itself, and also beyond the technical verification of the complete solution, the ongoing research is seeking to improve the registration methods and the maintenance of global equivalents, as a means to enhance its efficiency in practice. In this direction, the research also investigates novel ways of defining and managing global equivalents. The goal is to eventually allow proxy servers to discover data equivalents from direct data or metadata interfacing with peers (that certain data items from peers are in fact equivalent to their own), on an on-the-fly basis. This way, local sites would be able to add data items without having to update the global list — i.e., without much delay. This field remains challenging and widely open to new ideas, despite the new results obtained [7].

References

1. G. Babin and C. Hsu. Decomposition of knowledge for concurrent processing. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):758–772, 1996.
2. M. Bouziane and C. Hsu. A rulebase management system using conceptual modeling. *J. Artificial Intelligence Tools*, 6(1):37–61, March 1997.
3. W. Cheung and C. Hsu. The model-assisted global query system for multiple databases in distributed enterprises. *ACM Transactions on Information Systems*, 14(4):421–470, 1996.
4. S.H. Clearwater, editor. *Market-Based Control : A Paradigm for Distributed Resource Allocation*. World Scientific Publishing, River Edge, N.J., 1996.
5. Covisint. http//www.covisint.com.
6. R. Glushko, J. Tenenbaum, and B. Meltzer. An XML framework for agent-based e-commerce. *Communications of the ACM*, 42(3):106–114, 1999.
7. A.Y. Halevy, Z.G. Ives, J. Madhavan, and P. Mork. The piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, July 2004.
8. C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information resources management in heterogeneous distributed environments: A metadatabase approach. *IEEE Transactions on Software Engineering*, 17(6):604–625, 1991.

9. C. Hsu, C.D. Carothers, and D.M. Levermore. A market mechanism for participatory database query: A first step of enterprise resources self-allocation. *Information Technology and Management*, forthcoming.
10. C. Hsu, Y. Tao, M. Bouziane, and G. Babin. Paradigm translation in manufacturing information using a meta-model: The TSER approach. *Ingénierie des systèmes d'information*, 1(3):325–352, January 1993.
11. T. Sandholm. Making markets and democracy work: A story of incentives and computing. In *IJCAI-03*, 2003. Computers and Thought Award Talk Abstract.
12. M. Stonebraker, P. Aoki, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide area distributed database system. *International Journal on Very Large Databases*, 5(1):48–63, 1996.

Service-Oriented Modelling and Analysis: A Practical Case

Maria-Eugenia Iacob, Henk Jonkers, Marc M. Lankhorst, Maarten W.A. Steen

Telematica Instituut, PO Box 589, 7500 AN Enschede, The Netherlands
marc.lankhorst@telin.nl

Abstract. In order to validate the concepts and techniques for service-oriented enterprise architecture modelling, developed in the ArchiMate project (2005), we have conducted a number of case studies. This paper describes one of these case studies, conducted at the Dutch Tax and Customs Administration. It shows how business processes, applications, and technical infrastructure can be modelled and related using the 'service' concept as a bridge, and visualised by applying the ArchiMate viewpoints. It also shows the usage of quantitative analysis techniques to compute the application and infrastructural workloads imposed by the characteristics of the business processes.

1 Introduction

The emergence of the service-oriented computing (SOC) paradigm and Web services technology, in particular, has aroused enormous interest in service-oriented architecture (SOA). Probably because such hype has been created around it, there are a lot of misconceptions about what SOA really is. Numerous Web services evangelists make us believe that if you could divide the world into service requestors, service providers and a service registry, you would have an SOA (e.g., Ferris and Farrell 2003). Others emphasise that SOA is a way to achieve interoperability between distributed and heterogeneous software components, a platform for distributed computing (e.g., Stevens 2002).

Services are a much broader concept, however. In modern enterprise modelling, for example, the service concept also plays a central role. We define a service as a unit of functionality that some entity (e.g., a system, organisation, or department) makes available to its environment, and which has some value for certain entities in the environment (typically the 'service users'). This general service concept is widely applicable, in both business and IT contexts. Service orientation supports current trends ranging from the service-based network economy to ICT integration with Web services. These examples already show that services of a very different nature and granularity can be discerned: they can be provided by organisations to their customers, by applications to business processes, or by technological facili-

ties (e.g., communication networks) to applications.

Service orientation typically leads to a layered view of enterprise architecture models, where the service concept is one of the main linking pins between the different layers. *Service layers* with services made available to higher layers are interleaved with *implementation layers* that realise the services. Within a layer, there may also be *internal* services, e.g., services of supporting applications that are used by the end-user applications. This leads to a stack of service layers and implementation layers.

Although, at an abstract level, the concepts that are used within each layer are similar, we define more concrete concepts that are specific for a certain layer. In this context, we distinguish three main layers:

1. The *business layer* offers products and services to external customers, which are realised in the organisation by business processes (performed by business actors or roles).
2. The *application layer* supports the business layer with application services which are realised by (software) application components.
3. The *technology layer* offers infrastructural services (e.g., processing, storage, and communication services) needed to run applications, realised by computer and communication devices and system software.

In this paper we explain how the integration of the above-mentioned layers of an enterprise architecture can be achieved trough service-oriented modelling. Furthermore, we show that these models can form the basis for performance analysis. These ideas are demonstrated in the context of a practical case we have conducted at the Dutch Tax and Customs Administration, in which we have used the results of the ArchiMate project (2005).

The paper is organised as follows. The following section sets the background for our practical case by giving a brief description of the ArchiMate language and its modelling concepts. The remainder of the paper is reserved for a description of the main results produced during our case study. We provide an overview of how the different architecture layers have been modelled and integrated and we give an account on the application of our analysis method. Finally, we conclude by summarising our results.

2 The ArchiMate language

Within many of the different domains of expertise that are present in an enterprise, some sort of architectural practice exists, with varying degrees of maturity. All kinds of frameworks try to map these domains, such as the well-known Zachman framework (Zachman 1987), The Open Group Architecture Framework (TOGAF) (The Open Group, 2003), and many more. However, due to the heterogeneity of the methods and techniques used to document the architectures, it is very difficult to determine how the different domains are interrelated. Still, it is clear that there are strong dependencies between the domains. For example: the goal of the (pri-

mary) business processes of an organisation is to realise their products; software applications support business processes, while the technical infrastructure is needed to run the applications; information is used in the business processes and processed by the applications. For optimal communication between domain architects, needed to align designs in the different domains, a clear picture of the domain interdependencies is indispensable.

With these observations in mind, we conclude that a language for modelling *enterprise architectures* should focus on inter-domain relations. With such a language, we should be able to model:

- The global structure *within* each domain, showing the main elements and their dependencies, in a way that is easy to understand for non-experts.
- The relations *between* the domains.

To this end, we have defined the ArchiMate language (Lankhorst et al., 2005), an enterprise architecture modelling language that is gaining rapid acceptance in the Netherlands and abroad. The core concepts of the language are given in Fig. 1. A common abstract metamodel lies behind the concrete modelling concepts at all three layers, comprising (abstract) concepts for structure, behaviour, and information. The most important concrete modelling concepts are explained below. For a more detailed description please refer to (Lankhorst et al., 2005).

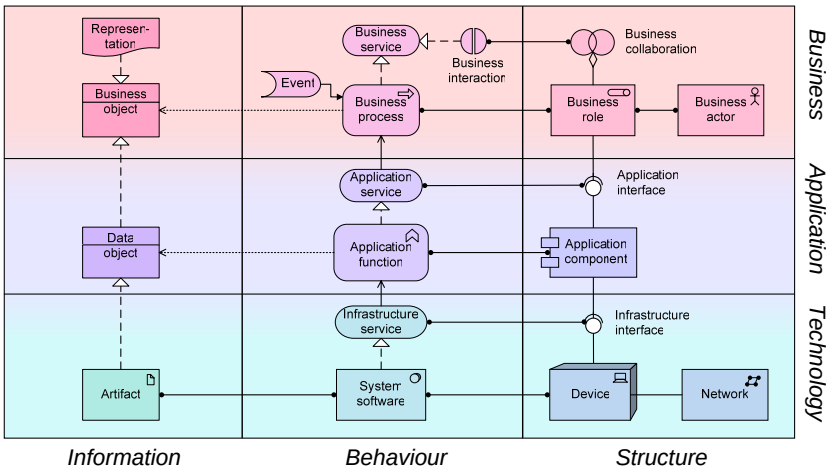


Fig. 1. Main concepts of the ArchiMate language.

The main structural concept at the business layer is the *business actor*, an entity that performs behaviour such as business processes or functions. Business actors may be individual persons (e.g. customers or employees), but also groups of people and resources that have a permanent (or at least long-term) status within the organisations. To each actor *business roles* can be assigned, which in turn signify responsibility for one or more *business processes*, which may manipulate *business*

objects. The externally visible behaviour of a business process is modelled by the concept *organisational service*, which represents a unit of functionality that is meaningful from the point of view of the environment. Services are grouped to form (financial or information) *products*, together with a *contract* that specifies the characteristics, rights and requirements associated with the product.

The main structural concept for the application layer is the *application component*. This concept is used to model any structural entity in the application layer: not just (reusable) software components that can be part of one or more applications, but also complete software applications or information systems. *Data objects* are used in the same way as data objects (or object types) in well-known data modelling approaches, most notably the 'class' concept in UML class diagrams. In the purely structural sense, an *application interface* is the (logical) location where the services of a component can be accessed. In a broader sense (as used in, among others, the UML definition), an application interface also has some behavioural characteristics: it defines the set of operations and events that are provided by the component, or those that are required from the environment. Behaviour in the application layer can be described in a way that is very similar to business layer behaviour. We make a distinction between the externally visible behaviour of application components in terms of *application services*, and the internal behaviour of these components to realise these services.

The main structural concept for the technology layer is the *node*. This concept is used to model structural entities in the technology layer. Nodes come in two flavours: *device* and *system software*, both inspired by UML 2.0 (the latter is called *execution environment* in UML). A device models a physical computational resource, on which artifacts may be deployed for execution. System software represents the software environment for specific types of components and data objects. An *infrastructure interface* is the (logical) location where the infrastructural services offered by a node can be accessed by other nodes or by application components from the application layer. An *artifact* is a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system. A *network* models a physical communication medium between two or more devices. In the technology layer, the central behavioural concept is the *infrastructure service*. We do not model the internal behaviour of infrastructure components such as routers or database servers; that would add a level of detail that is not useful at the enterprise level of abstraction.

3 Case Description

The Dutch Tax and Customs Administration (abbreviated TCA in the sequel) has a long history of continuously improving its organisation of process and ICT development. As early as the beginning of the 1980s, the ICT department started working with architecture. In the TCA architecture plays a prominent role, which is also exemplified by a total staff of over 100 architects. The importance of architecture has also increased the need for an enterprise architecture language to con-

nect different architecture domains.

In recent years, the organisation of social security in the Netherlands has changed dramatically. The goal is to arrive at a situation with a central contact point for organisations and citizens, and with unique 'authentic' data sources. Within this context, the collection of employees' social security premiums is transferred from UWV (the central social security organisation) to the TCA. This joint project of TCA and UWV is called SUB ('Samenwerking UWV–Belastingdienst').

A major challenge in this project is to handle enormous flows of data within and among the different organisations. This concerns more than 600,000 payroll tax returns each month, a large proportion of which arrive within a peak period of a couple of days. Moreover, it is expected that a substantial proportion of these tax returns need to be sent back for correction. Such requirements need to be addressed early on in the project.

These aspects of this case study made it an ideal testing ground for our service-oriented and viewpoint-based approach to enterprise modelling and visualisation (Lankhorst, 2005), and for the performance analysis techniques described in (Iacob & Jonkers, 2005). In the following, we will show how the different aspects of the business processes, applications, and infrastructure were modelled in a coherent and consistent way, and also show how the quantitative analysis techniques were used in the capacity planning of the infrastructure.

4 Architectural Views

By means of a number of different *views* (IEEE Computer Society 2000), the SUB information system architecture is presented from the perspective of the TCA. We have chosen not to show a model of SUB as a whole; instead, we start with a broad perspective and go into detail for a number of specific processes.

Subsequently, models are presented that describe the SUB business processes (viewpoint Process cooperation), the SUB application support for these processes (viewpoint Application usage), and the infrastructure support for the applications (viewpoint Infrastructure support).[1]

4.1 Process Cooperation: Client-to-Client Processes

The process architecture, depicted in Fig. 2, shows the most important client-to-client processes within the scope of SUB. Each process is initiated by a *trigger*. These triggers fall in one the following categories:

- *time* triggers, indicating that a process is executed periodically;
- *message* triggers, indicating that an incoming message initiates a process;
- *signal* triggers, indicating that an incoming signal initiates a process.

[1] The actual design of SUB further evolved after completion of the case study.

For each trigger, a *frequency* is specified, expressed in terms of the average number of 'firings' per month. Furthermore, the process architecture shows the most important messages that flow between the processes.

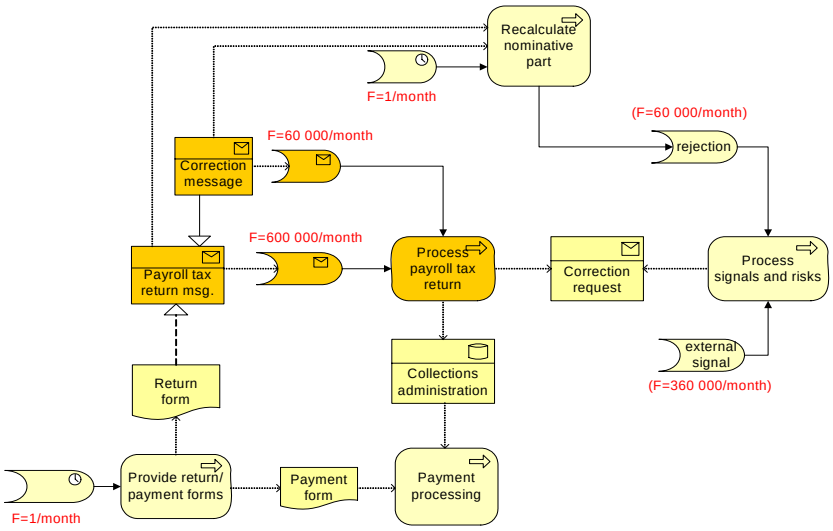


Fig. 2. Overview of the SUB Client-to-client processes.

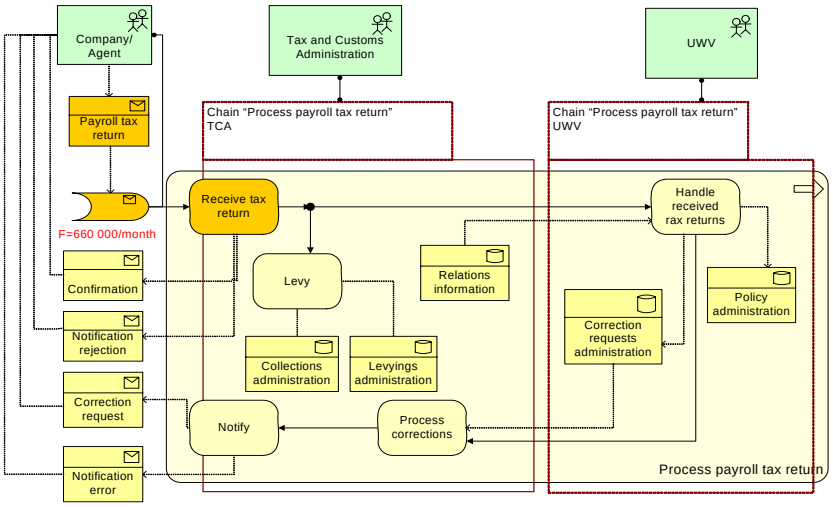


Fig. 3. Client-to-client process 'Payroll tax return'.

Obviously, each of the above-mentioned client-to-client processes can be described in more detail by further specifying the subprocesses of which they consist, the actors that are involved, the incoming and outgoing messages, and the databases that are being used. Next, we present a more detailed decomposition of the process 'Payroll tax return' (Fig. 3) from the overall SUB process architecture. The model shows, among others, which part of the process is executed by the TCA and which by UWV.

4.2 Application Usage

Going one level of detail deeper, we now zoom in on the 'Receive tax return' subprocess. The model of this subprocess and the corresponding application support are shown in Fig. 4.

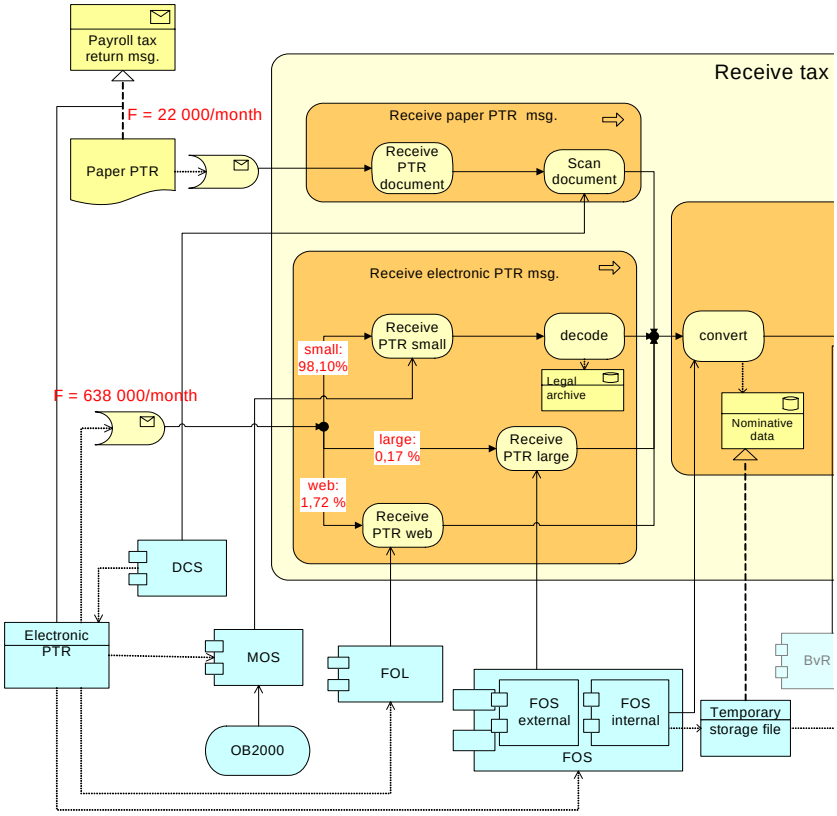


Fig. 4. Application and business process architecture for 'Receive tax return' (partial view).

A payroll tax return (PTR) can be submitted in two main formats: on paper or electronically. The electronic tax returns have three possible formats: Web-based messages, small messages sent via SMTP, and large messages sent via FTP. The model shows the expected distribution of the total number of messages over these different formats.

The first part of the 'Receive tax return' process transforms these formats into a common, medium-independent format. We will refer to this phase of the process as 'Medium-specific processing'. The second phase of the 'Receive tax return' process, 'Medium-independent processing', processes all the payroll tax returns in the same way, irrespective of their original format.

First, we detail the 'Medium-independent processing' phase. In the application architecture, the behaviour of each application component is partitioned into one or more *application functions* (denoting units of functionality used within the business processes), which may deliver *application services* to their environment, and *application interactions* to model communication between components, as well as the data stores involved. Part of the resulting model is shown in Fig. 5.

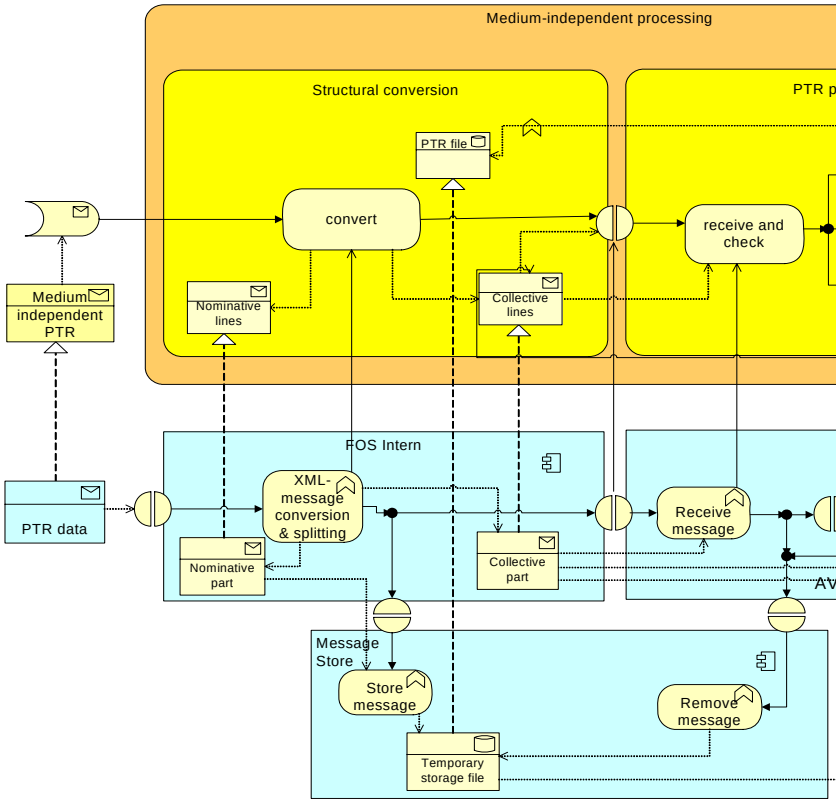


Fig. 5. Application support for 'Medium-independent processing' (partial view).

4.3 Infrastructure Usage

The next step is to take a closer look at the infrastructure support for the application architecture. We first illustrate the modelling approach for the 'Medium-independent processing'. A layer of infrastructure services supports the various application functions. We distinguish three types of infrastructural services:

– data storage and access services;
– processing services;
– communication services.

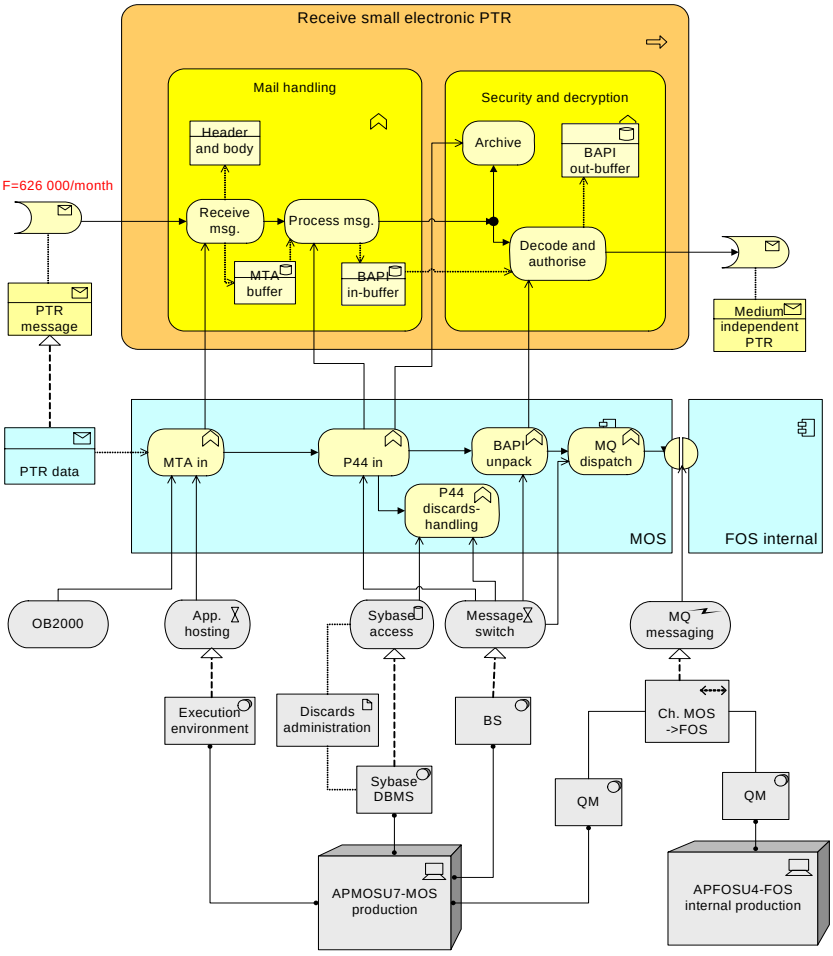


Fig. 6. 'Receiving small electronic payroll tax returns' architecture.

Data storage and access services are realised by, for example, a database management system. Processing services are typically realised by an execution environment or application server. Communication services support messaging between applications which is realised by, for instance, message queuing software. In this case, WebSphere MQ technology is used, where message brokers and message switches make use of functionality provided by queue managers. In MQ, communication services are realised by so-called *channels*. A channel between two devices is modelled as a communication path that represents a collaboration of two QM system software components, one for the sender and one for the receiver.

As mentioned above, the first part of the 'Return tax returns' process, 'Medium-specific processing', receives payroll tax returns from four information sources. Following the same modelling guidelines as in the case of the 'Medium-specific processing' part, we present in Fig. 6 the whole layered architecture (i.e., business process, application, and infrastructure architecture) of 'Receiving small electronic payroll tax returns'. The models for the other three sources of tax returns will not be shown here, but they can be constructed in a similar way.

5 Performance Analysis

This section illustrates the quantitative analysis of the model presented in the views in the previous sections, using the analysis approach described in (Iacob & Jonkers 2005). The results can be used to get an indication of the capacity that is required for the different resources in the infrastructure layer.

5.1 Viewpoints on performance

Architectures can be described from different viewpoints, which result in different views on architectural models (IEEE, 2000). For the performance aspects of a system, we can likewise discern a number of viewpoints, resulting in different (but related) performance measures:

- **User/customer view** (stakeholders: customer; user of an application/system): *response time*, the time between issuing a request and receiving the result; the response time is the sum of the processing time and waiting times (synchronisation losses).
- **Process view** (stakeholders: process owner; operational manager): *completion time*, the time required to complete one instance of a process (possibly involving multiple customers, orders, products etc., as opposed to the response time, which is defined as the time to complete one request).
- **Product view** (stakeholders: product manager; operational manager): *processing time*, the amount of time that actual work is performed on the realisation of a certain product or result: the response time without waiting times. This can be an order of magnitude lower than the response time.

– **System view** (stakeholders: system owner; system manager): *throughput*, the number of transactions/requests that are completed per time unit.
– **Resource view** (stakeholder: resource manager; capacity planner): *utilisation*, the percentage of the operational time that a resource is busy. On the one hand, the utilisation is a measure for the effectiveness with which a resource is used. On the other hand, a high utilisation can be an indication of the fact that the resource is a potential bottleneck.

This is a refinement of the views mentioned in, e.g., Herzog (2001), which only discerns a user view and a system view.

5.2 Analysis Approach

For the given type of analysis, the following input data is required:

– For each trigger the *arrival frequency* (average and possibly also peaks).
– For each process, function, or service the average *service time*.
– For each actor, component, or device the *capacity*.

Given these inputs, we can estimate the following performance measures:

– For each concept in the model (service, process, function, and resource) the *throughput*: the number of inputs/outputs that is to be processed per time unit. This is the *workload* that is imposed by the processes.
– For each actor, component, and device its *utilisation*: the percentage of time that it is active.
– For each process, function, and service the average *processing time* and *response time*.
– For each client-to-client process the average *completion time*.

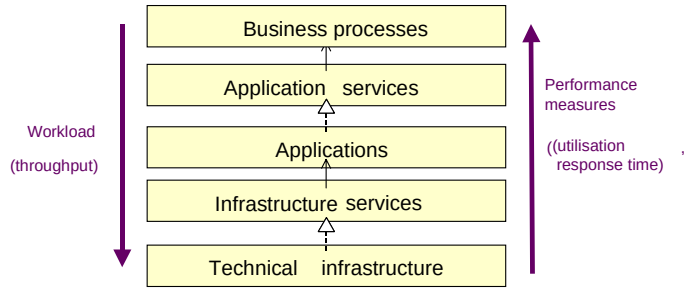


Fig. 7. Overview of the analysis approach.

The analysis approach is portrayed in Fig. 7. Starting with the arrival frequencies on the business process level, the workload (throughput) for all model elements in the layers below is calculated (top-down analysis). Together with the given service time of the infrastructure services, the utilisation of the resources,

and the processing and response times of the processes, functions, and services are calculated (bottom-up analysis). In (Iacob & Jonkers 2005) there is a detailed description of the analysis algorithms.

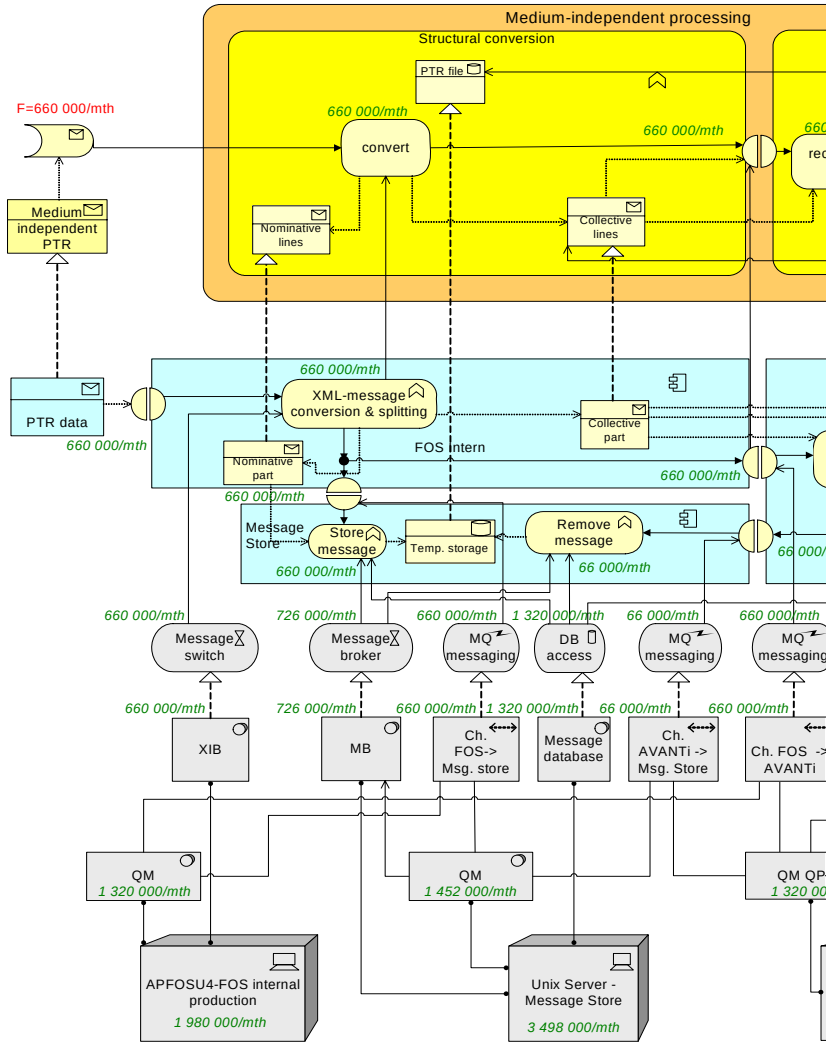


Fig. 8. Throughputs for the subprocess ‘Medium-independent processing’.

5.3 Workload Calculations (Top-Down)

Some of the results of the workload calculations are shown in Fig. 8 (in italics). These figures reflect the workload of applications and infrastructure imposed by the subprocess *Medium independent processing*, given an average monthly supply of 660,000 payroll tax returns. This workload is the basis for further performance analysis.

To obtain estimates of the total required infrastructure capacities, the same calculations also have to be made for the different *Medium-specific processing* parts of the *Receive tax return* process. The sum of the workloads from all the subprocesses results in a total workload for the SUB infrastructure, part of which is shown in Fig. 9. Similar calculations could be carried out for peak situations.

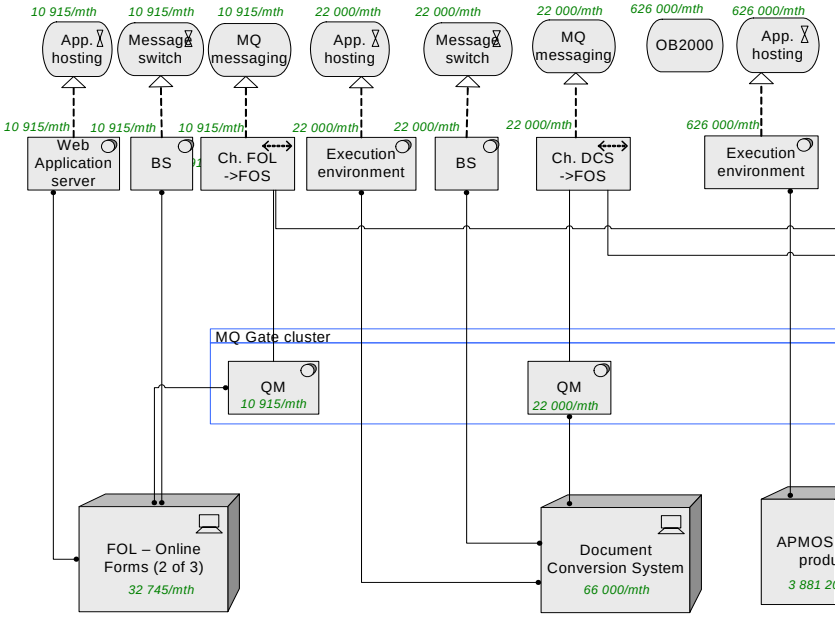


Fig. 9. Total workload of the SUB infrastructure (partial view).

5.4 Performance Measure Calculations (Bottom-Up)

To calculate performance measures such as response times and utilisation, service times are also needed as input data. These figures are often difficult to establish, especially in a design phase of a project when systems are not yet operational. Nevertheless, based on technical documentation and available historical informa-

tion (e.g., performance tests) of existing system components, and together with experts on the matter, reasonable estimates of these numbers could be made.

The numerical results of the bottom-up analysis of the process 'Receiving small electronic payroll tax returns' are given in Fig. 10. According to these figures the utilisation of the resources for an average workload is already quite high; this means that at peak loads the resources will almost certainly be overloaded. A solution to this problem may be to add additional resources or to increase the capacity of the resources. Further analysis can help to determine by how much the capacity needs to be increased.

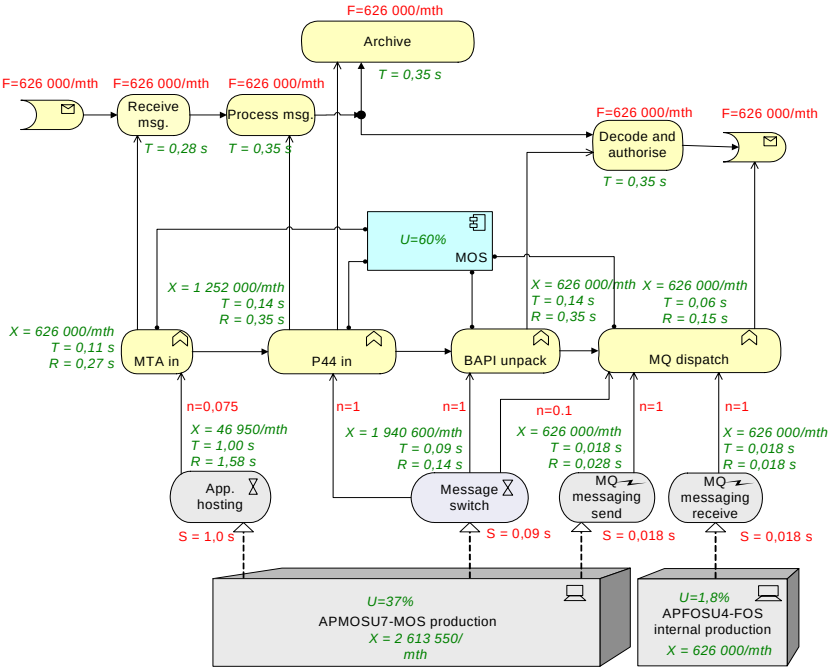


Fig. 10. Utilisation and response times for 'Receive tax returns (small)'.

6 Conclusions

This case study shows that the ArchiMate language is suitable for modelling the relevant aspects of the technical and application architectures, as well as relating them to each other and the business processes they support. Quantitative analysis offered a clear view of how activities at the business process level impose a workload on the application and infrastructure levels, thus providing a basis for capacity planning of the infrastructure. Performing these quantitative analyses at an

early stage considerably helps the realisation of the desired performance characteristics of the target system.

The case study clearly shows the feasibility and practical value of these results in various real-life settings. Both the service-oriented modelling language and the visualisation and analysis techniques have shown their merit in providing more insight into complex, wide-ranging enterprise architectures.

Acknowledgment

This paper results from the ArchiMate project (http://archimate.telin.nl/), a research initiative that provides concepts and techniques to support enterprise architects in the visualisation, communication and analysis of integrated architectures. The ArchiMate consortium consists of ABN AMRO, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Instituut, Centrum voor Wiskunde en Informatica, Katholieke Universiteit Nijmegen, and the Leiden Institute of Advanced Computer Science.

References

ArchiMate project (2005), http://www.archimate.com/, accessed: 30-9-2005.

Ferris C, Farrell J (2003), What are Web Services? Communications of the ACM, 46(6):31.

Iacob M-E, Jonkers H (2005), Quantitative Analysis of Enterprise Architectures, In: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005), Geneva, Switzerland, February 21 - 25.

Herzog, U (2001). Formal methods for performance evaluation, In Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science (LNCS 2090), pp. 1–37. Springer, Berlin.

IEEE Computer Society (2000), IEEE Std 1471-2000: IEEE Recommended Practice for Architecture description of Software-Intensive Systems. IEEE, New York.

Lankhorst, MM (2005), Enterprise Architecture at Work – Modelling, Communication, and Analysis. Springer, 2005.

Stevens M (2002), Service-Oriented Architecture Introduction, Part 1. www.developer.com/design/article.php/1010451, April.

The Open Group (2003), The Open Group Architectural Framework (TOGAF) Version 8.1 'Enterprise Edition'. The Open Group, Reading, UK. http://www.opengroup.org/togaf/.

Zachman JA (1987), A Framework for Information Systems Architecture, IBM Systems Journal, 26(3):276–292.

Relational to Object-Oriented Database Wrapper Solution in the Data Grid Architecture with Query Optimization Issues

Jacek Wislicki[1], Kamil Kuliberda[1], Radoslaw Adamus[1], Kazimierz Subieta[1,2,3]

[1] Computer Engineering Department, Technical University of Lodz, Lodz, Poland
[2]Institute of Computer Science PAS, Warsaw, Poland
[3]Polish-Japanese Institute of Information Technology, Warsaw, Poland
[jacenty, kkulibe, radamus]@kis.p.lodz.pl, subieta@pjwstk.edu.pl

Abstract: The paper introduces a solution to the problem of integrating relational databases with the data grid architecture and presenting their contents as a purely object-oriented business model. Authors describe a dedicated wrapper constructed with the stack-based approach (SBA) and updatable views concepts. The proposed architecture sustains grid's transparency and allows a grid user to operate in an object-oriented environment with SBQL – the stack-based query language. The strength of the described wrapper is a possibility of employment of native SQL optimizers. A query entering the front-end of the wrapper (object-oriented business model) can be rewritten according to the powerful SBQL optimization rules (within the wrapper) and then evaluated in the relational resource environment where appropriate SQL optimizers apply.
The paper contains a description of the wrapper and its place in the grid architecture with a query optimization procedure and an example of such a process.

1 Introduction

The grid is a novel technology contemporarily being widely researched by many academic and industrial institutions. Early grid concepts used to concern only computational networks. However, the rapid evolution of the Internet, Internet-related communities and the increase of a worldwide business information exchange have issued further expectations and, simultaneously, opened new opportunities for data and service integration. The new technologies are conceptually similar to the computational grid experiences and solutions. A significant branch of the grid researches is devoted to a concept of a distributed parallel database content processing, where various data and service resources residing in separate locations can be virtually available through their global representation. The technology is recently referred to as a “data-intensive grid” or a ”data grid”. Such a global representation (view) should abstracts its users from all the technical aspects of the process of integration (location, heterogeneity, fragmentation, replication, redundancy, etc.), which is referred to as the grid transparency. The effect of transparency requires enveloping (wrapping) the grid resources with dedicated programmatic structures – wrappers.

In this paper we deal with object-to-relational wrappers for distributed, heterogeneous and redundant data resources that are to be virtually integrated into a centralized, homogeneous and non-redundant whole. Our data grid term concerns higher forms of distribution transparency plus some common infrastructures build on top of the grid, including the trust infrastructure (security, privacy, licensing, payments, etc.), web services, distributed transactions, workflow management, etc [2].

Integration of dozens or hundreds servers participating in a grid requires different design processes in comparison to the situation when, e.g. a single object-oriented application has to be connected to a relational database. The common (canonical) database schema is the result of many negotiations and tradeoffs between business partners having incompatible (heterogeneous) data and services. The processes should take into account data models of the resources, but first of all the global canonical schema is influenced by the business model required by global applications (operating on top of a grid).

This makes development of an object-relational wrapper much more constrained than in a non-grid case. On one hand, the wrapper should deliver the data and services according to the predefined object-oriented canonical schema. On the other hand, its back-end should work on a given (preferably arbitrary) relational database.

The major problem with this architecture concerns how to utilize a native SQL optimizer. In all known RDBMS-s the optimizer and its particular structures (e.g. indices) are transparent to the SQL users. A naive implementation of a wrapper causes generation of primitive SQL queries such as *select * from R*, and then, processing the results of such queries by SQL cursors. Hence, the SQL optimizer has no chances to work. Furthermore, amounts of data returned from a relational resource by such form of queries are too huge to be effectively processed by a wrapper. Our experience has shown that direct translation of object-oriented queries into SQL is unfeasible for a sufficiently general case.

The solution to this problem presented in this paper is based on the object-oriented query language SBQL, virtual object-oriented views defined in SBQL, query modification methods [13], and an architecture that will be able to detect in a query syntactic tree some patterns that can be directly translated to optimizable SQL queries. The patterns match typical optimization methods that are used by the SQL query optimizer, in particular, indices and fast joins. The partial query results returned from a relational database are then evaluated in the whole query context (resulting from the already SBQL-optimized query form).

The concept of the two-stage optimization (object-oriented and relational) is original and innovatory and the whole presented approach is currently being implemented within our object-oriented platform ODRA.

The rest of the paper is organized as follows. In Section 2 we present our previous experiences, studies and researches concerning object-oriented wrappers built on top of relational databases, including a brief state of art and the fundamental concepts of the described solution: the stack-based approach with SBQL and updatable object-oriented views. The section presents only basic ideas – the approach has already resulted in extensive literature (e.g. [12]) and several implementations. Section 3 presents the data grid architecture. Section 4 discusses an object-relational wrapper and presents a simple example showing how it works. Section 5 concludes.

2 The State of Art and the Fundamentals of the Solution

The art of object-oriented wrappers build on top of relational database systems has been developed for years - first papers on the topic are dated to late 80-ties and were devoted to federated databases. The motivation for the wrappers is reducing the technical and cultural difference between traditional relational databases and novel technologies based on object-oriented paradigms, including analysis and design methodologies (e.g. based on UML), object-oriented programming languages (C++, Java, C#, etc.), object-oriented middleware (e.g. based on CORBA), object-relational databases and pure object-oriented databases. Recently, Web technologies based on XML/RDF also require similar wrappers. Despite the big pressure on object-oriented and XML-oriented technologies, people are accustomed and quite satisfied with relational databases and there is a little probability that the market will massively change soon to other data store paradigms.

Mapping between a relational database and a target global object-oriented database should not involve materialization of objects on the global side, i.e. objects delivered by such a wrapper should be virtual. Materialization is simple, but leads to many problems, such as storage capacity, network traffic overhead, synchronization of global objects after updates on local servers, and (for some applications) synchronization of local servers after updates of global objects. Materialization can also be forbidden by security and privacy regulations.

If global objects have to be virtual, they are to be processed by a query language and the wrapper has to be generic, we are coming to concept of virtual object-oriented database views that do the mapping from tables into objects. Till now, however, sufficiently powerful object-oriented views are still a dream, despite a lot of papers and some implementations. The ODMG standard does not even mention views[1]. The SQL-99 standard deals a lot with views, but currently it is perceived as a huge set of loose recommendations rather than as entirely implementable artifact. In our opinion, the Stack-Based Approach and its query language SBQL offer the first and universal solution to the problem of updateable object-oriented database views. In this paper we show that the query language and its view capability can be efficiently used to build optimized object-oriented wrappers on top of relational databases.

Current common development of exiting systems (languages and technologies) includes a large combination of object-oriented options with relational systems and applications where fundamental differences are based on models' particular artifacts. Focusing on wrappers, they can have different properties, in particular, can be proprietary to applications or generic, can deal with updates or be read-only, can materialize objects on the wrapper side or deliver purely virtual objects, can deal with object-oriented query language or provide some iterative "one-object-in-a-time" API, etc [1]. This causes an extremely huge number of various ideas and technologies. For instance, Google reports more than 100 000 Web pages as a response to the query "object relational wrapper".

[1] The *define* clause of OQL is claimed to be a view, but this is misunderstanding: it is a macro-definition (a textual shorthand) on the client-side, while views are server-side entities.

2.2 Our Experiences and the Resulting Proposals

Our first experience with this kind of a wrapper was in 1993, when we built a gateway from the DBPL system to Ingres and Oracle. The idea was that relational tables have to be transparently and virtually mapped as DBPL persistent collections, and DBPL queries (based on nested relational calculus) are to be automatically mapped to SQL queries. In this way a DBPL programmer might forget that he/she works with a relational database and used regular DBPL queries and programs instead of SQL. On the back-end the DBPL queries, mapped to SQL, were optimized by the SQL optimizer, thus the performance was quite good. Even such a simple wrapper presented a conceptual and implementation challenge, see [8] for details.

Our second experience with this kind of wrappers was connected with the European project ICONS (Intelligent COntent maNagement System), IST-2001-32429, devoted to advanced Web applications. ICONS assumes a simplified object-oriented model having virtual non-nested objects with repeated attributes and UML-like association links used for navigation among objects. Both repeated attributes and links were derived from the primary-foreign key dependencies in the relational database by a parameterization utility. The ICONS repository was available as an API to Java. However, because Java has no query capabilities, all the programming has to be done through sequential scanning of collections of objects. Obviously, this gives no chances to the SQL optimizer, hence initially the performance was extremely bad. It was improved by some extensions, for instance, by special methods with conditions as parameters that were mapped into SQL *where* clauses, but in general the experience was disappointing. (We think this may concern all Java-oriented persistent layers built on top of relational systems and using an own API instead of JDBC.)

Currently we are implementing (under .NET) an object-oriented platform named ODRA for Web and grid applications, thus the problem of a wrapper on top of relational databases comes again into the play. After previous experience we have made the following conclusions:

- the system will be based on our own object-oriented query language SBQL, which has many advantages over OQL, XQuery, SQL-99 and other languages. In particular, it is much more powerful than OQL and XQuery and has a precise formal semantics, which is a prerequisite for developing any automatic transformations of queries into semantically equivalent forms. SBQL is already implemented, including its typechecker and a query rewriting optimizer,
- the system will be equipped with a powerful mechanism of object-oriented virtual updateable views based on SBQL. Our views have the power of algorithmic programming languages, hence are much more powerful than, e.g. SQL views. There are three basic applications of the views: (1) as integrators (mediators) making up a global virtual data and service store on top of distributed, heterogeneous and redundant resources; (2) as wrappers on top of particular local resources; (3) as customization and security facility on top of the global virtual store. A partial implementation of SBQL views is ready too [7].

The architecture assumes that a relational database will be seen as a simple object-oriented database, where each tuple of a relation is mapped virtually to a primitive object. Then, on such a database we define object-oriented views that convert such primitive virtual objects into complex, hierarchical virtual objects conforming to the global canonical schema, perhaps with complex repeated attributes and virtual links

among the objects. Because SBQL views are algorithmically complete, we are sure that every such a mapping can be expressed. Moreover, because SBQL views can possess a state, have side effects and be connected to classes, one would be able to implement a behavior related to the objects on the SBQL side.

The major problem concerns how to utilize the SQL optimizer. After our previous experience we have concluded that static (compile time) mapping of SBQL queries into SQL is unfeasible. On the other hand, a naive implementation of the wrapper, as presented above, leaves no chances to the SQL optimizer. Hence we must use optimizable SQL queries on the back-end of the wrapper.

The solution of this problem is presented in this paper. It combines SBQL query engine with the SQL query engine. There are a lot of various methods used by an SQL optimizer, but we can focus on three major ones:

- rewriting, for instance, pushing selections before joins,
- indices, i.e. internal auxiliary structures for a fast access,
- fast joins, e.g. hash joins.

Concerning rewriting, our methods are perhaps as good as SQL ones, thus this kind of optimization will be done on the SBQL side. Two next optimizations cannot be done on the SBQL side. The idea is that an SBQL syntactic query tree is first modified by views [13], thus we obtain a much larger tree, but addressing a primitive object database that is 1:1 mapping of the corresponding relational databases. Then, in the resulting tree we are looking for some patterns that can be mapped to SQL and which enforce SQL to use its optimization method. For instance, if we know that the relational database has an index for Names of Persons, we are looking in the tree the sub-trees representing the SBQL query such as:

```
Person where Name = "Doe"
```

After finding such a pattern we substitute it by the dynamic SQL statement:

```
exec_immediately(select * from Person where Name = "Doe")
```

enforcing SQL to use the index. The result returned by the statement is converted to the SBQL format. Similarly for other optimization cases. In effect, we do not require that the entire SBQL query syntactic is to be translated to SQL. We interpret the tree as usual by the SBQL engine, with except of some places, where instead of some subtrees we issue SQL *execute immediately* statements.

2.2 The Stack-Based Approach, SBQL and Updatable Object Views

In the stack-based approach (SBA) a query language is considered a special kind of a programming language. Thus, the semantics of queries is based on mechanisms well known from programming languages like the environment stack (ES). SBA extends this concept for the case of query operators, such as selection, projection/navigation, join, quantifiers and others. Using SBA one is able to determine precisely the operational semantics (abstract implementation) of query languages, including relationships with object-oriented concepts, embedding queries into imperative constructs, and embedding queries into programming abstractions: procedures, functional procedures, views, methods, modules, etc.

SBA is defined for a general object store model. Because various object models introduce a lot of incompatible notions, SBA assumes some family of object store models which are enumerated M0, M1, M2 and M3. The simplest is M0, which

covers relational, nested-relational and XML-oriented databases. M0 assumes hierarchical objects with no limitations concerning nesting of objects and collections. M0 covers also binary links (relationships) between objects. Higher-level store models introduce classes and static inheritance (M1), object roles and dynamic inheritance (M2), and encapsulation (M3). For these models there have been defined and implemented the query language SBQL, which is much more powerful than ODMG OQL [10] and XML-oriented query languages such as XQuery [14]. SBQL, together with imperative extensions and abstractions, has the computational power of programming languages, similarly to Oracle PL/SQL or SQL-99.

SBA assumes the object relativism principle that makes no conceptual distinction between objects of different kinds or stored on different object hierarchy levels. Everything (e.g. a Person object, a salary attribute, a procedure returning the age of a person, a view returning well-paid employees, etc.) is an object. SBQL respects the naming-scoping-binding principle: each name occurring in a query is bound to the appropriate run-time entity (an object, an attribute, a method, a parameter, etc.) according to the scope of its name. The principle is supported by means of the environment stack (ENVS). The concept of the stack is extended to cover database collections and all typical query operators occurring, e.g. in SQL and OQL.

Due to the stack-based semantics, the full orthogonality and compositionality of query operators have been achieved. The stack also supports recursion and parameters: all functions, procedures, methods and views defined in SBQL can be recursive by definition. Rigorous formal semantics implied by SBA creates a very high potential for the query optimization. Several optimization methods have been developed and implemented, in particular methods based on query rewriting, indices, removing dead queries, and others [11].

SBQL is based on the principle of compositionality, i.e. semantics of a complex query is recursively built from semantics of its components. In SBQL each binary operator is either algebraic or non-algebraic. Examples of algebraic operators are numerical and string operators and comparisons, aggregate functions, union, etc. Examples of non-algebraic operators are selection (**where**), projection/navigation (the dot), join, quantifiers ($\exists$, $\forall$), and transitive closures. The semantics of non-algebraic operators is based on a classical environmental stack, thus the name of the approach.

The idea of SBQL updatable views relies in augmenting the definition of a view with the information on user intentions with respect to updating operations. The first part of the definition of a view is the function, which maps stored objects onto virtual objects (similarly to SQL), while the second part contains redefinitions of generic operations on virtual objects. The definition of a view usually contains definitions of subviews, which are defined by the same principle [4].

The first part of the definition of a view has the form of a functional procedure. It returns entities called *seeds* that unambiguously identify virtual objects (usually seeds are OIDs of stored objects). Seeds are then (implicitly) passed as parameters of procedures that overload operations on virtual objects. These operations are determined in the second part of the definition of the view. There are distinguished several generic operations that can be performed on virtual objects:

- *delete* removes the given virtual object,
- *retrieve* (dereference) returns the value of the given virtual object,
- *navigate* navigates according to the given virtual pointer,
- *update* modifies the value of the given virtual object according to a parameter,

– etc.

All procedures, including the function supplying seeds of virtual objects are defined in SBQL and can be arbitrarily complex [4].

3 Architecture of the Data Grid

Figure 1 shows the architecture of a data grid. The proposed solution provides a simplification of the access to the distributed, heterogeneous and redundant data, constituting an interface to the distributed data residing in any local resource provider participating in a grid. The goals of the approach are to design a platform where all clients and providers are able to access multiple distributed resources without any complications concerning data maintenance and to build a global schema for the accessible data and services. The main difficulty of the described concept is that neither data nor services must be copied, replicated and maintained on the global applications side (in the global schema), as they are supplied, stored, processed and maintained on their autonomous sites [5, 6].

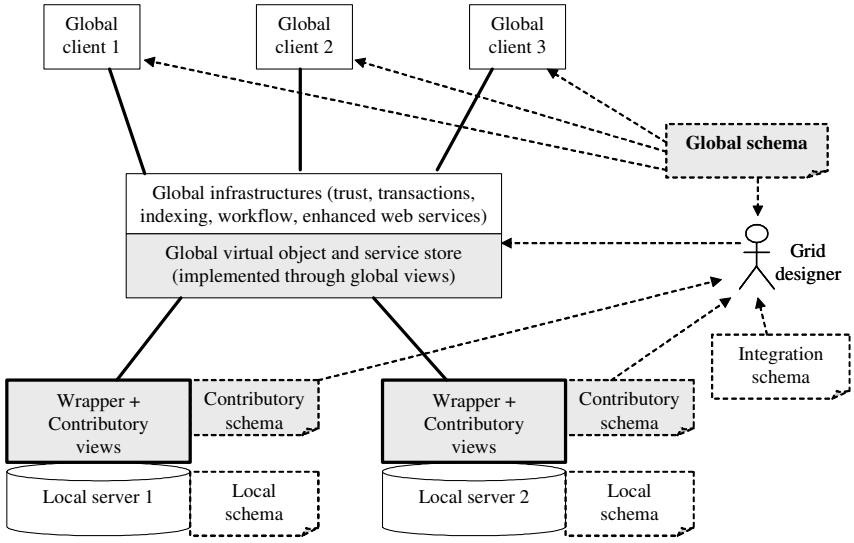


Fig. 1. The data grid architecture

We should describe the functionality range of the individual grid's architecture elements [5] shown in the above picture. The central part of a grid is a *global virtual store* containing virtual objects and services. Its role is to store addresses of local servers and to process queries sent from *global client* applications, as well as to enable accessing the grid according to the trust infrastructure (including security, privacy, licensing and non-repudiation issues). The global virtual store presents business objects and services according to the global schema, which has to be defined

and agreed upon the organization that creates a grid. This principal mechanism envelopes all the local resources into one global data structure. Physically, it is a composition of the contribution schemas which can participate in the grid. The *global schema* is responsible for managing grid contents through access permitions, discovering data and resources, controlling location of resources, indexing whole grid attributes. The global schema is also used by programmers to create global client applications.

As a grid integrates services and objects physically stored in the *local servers*, administrator of local servers must define *contributory schemata* and corresponding *contributory views* [3, 5], mapping local data and services to the global schema demands. A contribution schema is created by the *grid designer* and represents schema of the main data formalization rules for any local resource. Basing on this, local resource providers create their own contribution schemata adapted to an unique data structure present at their local sites. Resource formalization represented in such a form becomes a part of the global schema. *Contribution view* is the query language definition of mapping the local schema to contribution schema. A well defined contribution view can become a part of the *global view*. The mapping process consists of enclosing into the global schema particular contribution schemas residing in local sites, created earlier by local participants – resource providers [3]. The *integration schemata* contain additional information about dependencies between local servers (replications, redundancies, etc.) [3, 6], showing a method for integration of fragmented data into the grid, e.g. how to merge fragmented relational data structures where some parts of them are placed in separated local servers (in other words, fragmented). A grid designer must be aware of the structure fragmentation, which knowledge is unnecessary for a local sites administrator. A *grid developer* represents a person, a software development team or consortium which determine a primary dependencies and attributes of a grid. Initially, they create a global schema which later can be contributed during participation of clients and providers. Next, they define a metabase structure with contribution and integration schemata, according to the grid data structure intention. A *local schema* describes the data model of a local resource and can be presented in original data structure (i.e. relational, XML, HTML) and therefore it cannot be directly included in the global virtual store.

The crucial element of the architecture is a *wrapper* which enables importing and exporting data between two different data models, e.g. our object-oriented grid solution at one side and a relational data structure on the other side. The detailed description of wrapping mechanism is placed in next chapter of this paper.

The realization challenge is a method (in fact a core of the wrapper mechanism) of combining and enabling free bidirectional processing of contents of local clients and resource providers participating in the global virtual store as parts of the global schema. The presented architecture of a data grid is fully scalable, as growing and reducing the grid contents is dependent on the state of global schema and views.

4 Architecture of the Object-Relational Wrapper and Examples

Figure 2 presents the architecture of the wrapper. The general assumptions are the following:

- externally the data are designed according to the object-oriented model and the business intention of the *global schema*. This part constitutes the *front-end* of the wrapper and relies on SBQL,
- internally the relational structures are presented in the M0 model (excluding pointers and nesting levels above 2) [12]. This part constitutes the *back-end* of the wrapper and it also relies on SBQL,
- the mappings between *front-end* and *back-end* are defined with updatable object views. They role is to map *back-end* onto *front-end* for querying and *front-end* onto *back-end* for updating (virtual objects),
- for global queries, if some not very strict conditions are satisfied, the mapping form front-end into back-end query trees is done through query modification, i.e macro-substituting every view invocations in a query by the view body.

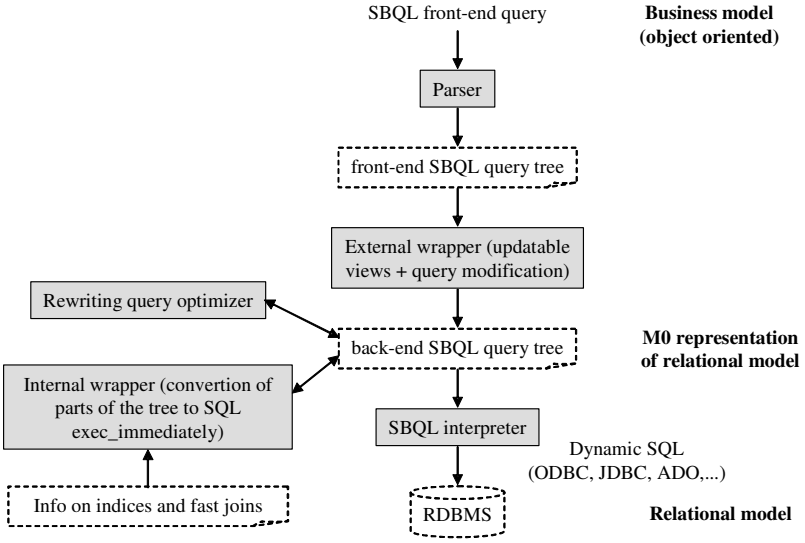


Fig. 2. The architecture of a generic wrapper for relational databases

4.1 Updates Through the Wrapper

The presented architecture assumes retrieval operations only, because the query modification technique assumed in this architecture does not work for updates. However, the situation is not hopeless (although more challenging). Because in SBQL updates are parameterized by queries, the major optimizations concern just these parameters, with the use of the query modification technique as well. Then, after the optimization, we can develop algorithms that would recognize in the back-end query tree all updating operations and then, would attempt to change them to dynamic SQL *update*, *delete* and *insert* statements. There are technical problems with identification of relational tuple within the SBQL engine (and further in SQL). Not all relational systems support *tuple identifiers* (tids). If tids are not supported, the developers of a wrappers must relay on combination (*relation_name, primary_key_value(s)*), which is

much more complicated in implementation. Tids (supported by SQL) simply and completely solve the problem of any kind of updates.

In Figure 2 we have assumed that the internal wrapper utilizes information on indices and fast joins (primary-foreign key dependencies) available in the given RDBMS application. In cases of some RDBMS (e.g. MS SQL Server) this information cannot be derived from the catalogs. If automatic derivation is impossible, the developers are forced to write a special utility allowing the wrapper designer to introduce this information manually.

4.2 Description of the Optimization Procedure

The query optimization procedure (looking from wrapper's front-end to back-end) for the proposed solution can be divided into several steps:

1. Query modification is applied to all view invocations in a query. All the invocations are macro-substituted by seed definitions of the views. If an invocation is preceded by the dereference operator, instead of the seed definition we have to use the corresponding *on_retrieve* function. (There are some modifications for virtual pointers.) The effect is a monster SBQL query referring to the M0 version of the relational model available at the back-end.
2. The query is rewritten according to static optimization methods defined for SBQL [11] such as removing dead sub-queries, factoring out independent sub-queries, pushing expensive operators (e.g. joins) down in the syntax tree, etc. The resulting query is SBQL-optimized, but still no SQL optimization is applied.
3. According to the available information about the SQL optimizer, the back-end wrapper's mechanisms analyze the SBQL query in order to recognize patterns representing SQL-optimizable queries. Then, *exec_immediately* clauses are issued.
4. The results returned by *exec_immediately* are pushed onto the SBQL result stack as collections of structures, which are then used for regular SBQL query evaluation.

4.3 Optimization Example

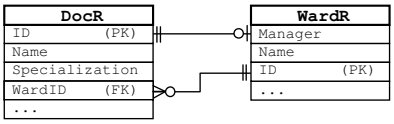


Fig. 3. The example of a relational schema

As the optimization example consider a simple two-table relational database containing information about doctors `DocR` and wards `WardR`, "R" stands for "relational" to increase the clearness (fig. 3).

The relational schema is wrapped into an object schema shown in figure 4 according to the following view definitions. The `DocR-WardR` relationship is realized with `worksIn` and `manager` virtual pointers:

```
create view DocDef {
   virtual_objects Doc {return DocR as d;}
```

```
    virtual_objects Doc(DocId) {return (DocR where ID == DocId) as d;}
    create view nameDef {
       virtual_objects name{return d.name as n;}
       on_retrieve {return n;}
    }
    create view specDef {
       virtual_objects spec {return d.specialization as s;}
       on_retrieve {return s;}
    }
    create view worksInDef {
       virtual_pointers worksIn {return d.wardID as wi;}
       on_navigate {return Ward(wi) as Ward;}
    }
}
create view WardDef {
    virtual_objects Ward {return WardR as w;}
    virtual_objects Ward(WardId) {return (WardR where ID == WardId) as
w;}
    create view nameDef {
       virtual_objects name {return w.name as n;}
       on_retrieve {return n;}
    }
    create view managerDef {
       virtual_pointers manager {return w.menagerID as b;}
       on_navigate {return Doc(b) as Doc;}
    }
}
```

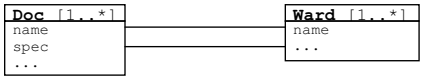


Fig. 4. Object schema used in the optimization example (wrapper's front-end)

Consider a query appearing at the front-end (visible as a business database schema) that aims to *retrieve names of the doctors working in the "cardiac surgery" ward having the specialization the same as Smith's specialization*. The query can be formulated as follows (we assume that there is only one employee with that name in the store):

```
((Doc where worksIn.Ward.name = "cardiac surgery") where
                    spec = (Doc where name = "Smith").spec).name;
```

The information about the local schema (relational model) available to the wrapper that can be used during the query optimization is that `name` column is uniquely indexed in either relation and there is a primary-foreign key integrity between `WardId` column (`DocR` table) and `ID` column (`WardR` table).

The optimization procedure is performed in the following steps:

- Introduce implicit `deref` function

```
((Doc where worksIn.Ward.deref(name) = "cardiac surgery") where
deref(spec) = (Doc where deref(name) =
"Smith").deref(spec)).deref(name);
```

- Substitute `deref` with the invocation of `on_retrieve` function for virtual objects and `on_navigate` for virtual pointers

```
((Doc where worksIn.(Ward(wi) as w).Ward.(name.n) = "cardiac
surgery") where (spec.s) = (Doc where (name.n) =
"Smith").(spec.s)).(name.n);
```

- Substitute all view invocations with the queries from *sack* definitions

```
(((DocR as d) where ((d.wardID as wi).(((WardR where ID == wi) as w)
```

```
as Ward)).Ward.((w.name as n).n) = "cardiac surgery") where ((d.spec
as s).s) = ((DocR as d) where ((d.name as n).n) = "Smith").((d.spec
as s).s)).((d.name as n).n);
```

– Remove auxiliary names `s` and `n`

```
(((DocR as d) where ((d.wardID as wi).(((WardR where ID = wi) as w)
as Ward)).Ward.(w.name) = "cardiac surgery") where (d.spec) = ((DocR
as d) where (d.name) = "Smith").(d.spec)).(d.name);
```

– Remove auxiliary names `d` and `w`

```
((DocR where ((wardID as wi).((WardR where ID = wi) as
Ward)).Ward.name = "cardiac surgery") where spec = (DocR where name =
"Smith").spec).name;
```

– Remove auxiliary names `wi` and `Ward`

```
((DocR where (WardR where ID = wardID).name = "cardiac surgery")
where spec = (DocR where name = "Smith").spec).name;
```

– Now take common part before loop to prevent multiple evaluation of a query calculating salary value for the doctor named *Smith*

```
((((DocR where name = "Smith").spec) group as s).(DocR where ((WardR
where ID == wardID).name = "cardiac surgery")) where spec = s).name;
```

– Connect where and navigation clause into one where connected with and operator

```
((((DocR where name = "Smith").spec) group as s).(DocR where (WardR
where (ID = wardID and name = "cardiac surgery")) where spec =
s).name;
```

– Because `name` column is uniquely indexed (in `DocR`), the sub-query `(DocR where name = "Smith")` can be substituted with *exec_ immediately* clause

```
(((exec_immediately("SELECT specialization FROM DocR WHERE name =
'Smith'")) group as s).(DocR where (WardR where (ID = wardID and name
= "cardiac surgery")) where spec = s).name;
```

– Because the integrity constraint with `DocR.WardId` column and `WardR.ID` column is available to the wrapper (together with information about the index on `WardR.Name`), the pattern is detected and another *exec_immediately* substitution is performed:

```
(((exec_immediately("SELECT specialization FROM DocR WHERE name =
'Smith'")) group as s).(exec_immediately("SELECT * FROM DocR d, WardR
w WHERE d.wardID = w.ID AND w.name = 'cardiac surgery'") where spec =
s).name;
```

Either of the SQL queries invoked by *exec_immediately* clause is executed in the local relational resource and pends native optimization procedures (with application of indices and fast join, respectively).

5. Conclusions

The presented approach to data grid concerning wrapping relational databases to object-oriented business model with application of the stack-based approach and updatable views is clear and implementable. As presented by the example, a front-end SBQL query can be modified and optimized with application of SBA rules and methods within the wrapper (updatable views) and then the native relational optimizers for SQL language can be employed. Due to including in SQL queries (invoked by *execute immediately* clause) appropriate conditions forcing a reduction of retrieval only to the matching records, the amounts of data subsequently processed by

the wrapper are satisfactorily small. The same conditions allow native SQL optimizers to act.

The described wrapper architecture enables building generic solutions allowing presentation of data stored in various relational resources as object-oriented models visible at the top level of the grid and accessing the data with object query language. Furthermore, assuming possibility of appropriate rearranging the wrapper's back-end (SBQL to a local query language mapping), the solution should be also applicable to other types of resources, especially object-relational, XML and other purely object-oriented DBMSs, with the local optimization mechanisms employment.

The described optimization process assumes correct relational-to-object model transformation (with no loss of database logic) and accessibility of the relational model optimization information such as indices and/or primary-foreign key relations (which can be read directly from the relational metadata or manually entered in the wrapper's schema if not available directly). The SQL optimization is out of the scope of the wrapper action and is assumed to be efficient and reliable.

The method is currently being implemented as a part of our new project ODRA devoted to Web and grid applications.

References

1. Bergamaschi, S., Garuti, A., Sartori, C., Venuta, A.: Object Wrapper: An Object-Oriented Interface for Relational Databases. EUROMICRO 1997, pp.41-46
2. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Global Grid Forum, June 22, 2002
3. Kaczmarski, K., Habela, P., Subieta, K.: Metadata in a Data Grid Construction. Proc. of 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), Italy, June, 2004
4. Kozakiewicz, H., Leszczylowski, J., Plodzien, J., Subieta, K.: Updateable Object Views. ICS PAS Reports 950, October 2002
5. Kozankiewicz, H., Stencel, K., Subieta, K.: Implementation of Federated Databases through Updateable Views. Proc. EGC 2005 - European Grid Conference, Springer LNCS, 2005, to appear
6. Kozankiewicz, H., Stencel, K., Subieta, K.: Integration of Heterogeneous Resources through Updatable Views. Workshop on Emerging Technologies for Next generation GRID (ETNGRID-2004), 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), University of Modena and Reggio Emilia, Italy, June 14-16, 2004, Proceedings published by IEEE
7. Kozankiewicz, H., Subieta, K.: SBQL Views - Prototype of Updateable Views. ADBIS (Local Proceedings) 2004
8. Matthes, F., Rudloff A., Schmidt, J.W., Subieta, K.: A Gateway from DBPL to Ingres. Proc. of Intl. Conf. on Applications of Databases, Vadstena, Sweden, Springer LNCS 819, pp.365-380, 1994
9. Moore, R., Merzky, A.: Persistent Archive Concepts. Global Grid Forum GFD-I.026. December-2003
10. Object Data Management Group: The Object Database Standard ODMG, Release 3.0. R.G.G.Cattel, D.K.Barry, Ed., Morgan Kaufmann, 2000
11. Plodzien, J.: Optimization Methods in Object Query Languages, PhD Thesis. IPIPAN, Warsaw 2000

12. Subieta, K.: Theory and Construction of Object-Oriented Query Languages. Editors of the Polish-Japanese Institute of Information Technology, 2004, 522 pages, currently only in Polish
13. Subieta, K., Plodzien, J.: Object Views and Query Modification, (in) Databases and Information Systems (eds. J. Barzdins, A. Caplinskas), Kluwer Academic Publishers, pp. 3-14, 2001
14. W3C: XQuery 1.0: An XML Query Language. W3C Working Draft 12, November 2003, http://www.w3.org/TR/xquery/

Service Adaptation through Trace Inspection

Antonio Brogi and Razvan Popescu

Computer Science Department, University of Pisa, Italy

Abstract. Service-oriented computing highly supports the development of future distributed business applications through the use of (Web) services. Due to the heterogeneous and evolving nature of business processes, *service adaptation* is necessary in order to overcome mismatches between the interacting parties. Our long term objective is to develop a general methodology for service adaptation capable of suitably overcoming semantic and behaviour mismatches in view of business process integration within and across organisational boundaries. In this paper we show how to adapt a service in order to fulfil a client query requesting a service with certain inputs and outputs. The proposed technique relies on inspecting service execution traces and it generates a service contract tailored to the client needs. Service contracts include a description of the service behaviour (expressed by a YAWL workflow) as well as an (ontology-annotated) signature.

1 Introduction

Service-oriented computing [19] is emerging as a new promising computing paradigm that centres on the notion of *service* as the fundamental element for developing future business applications. The platform-neutral nature of services creates the opportunity for building composite business processes by integrating existing elementary or complex services, possibly offered by different service providers [31]. In this scenario, two prominent issues involved in the development of next generation heterogeneous distributed software applications can be roughly synthesised as discovering available services that fulfil a given request, and suitably adapting and aggregating such services to build a needed business application.

Currently, WSDL [28] and UDDI [8] are the universally adopted standards for Web service description and discovery, respectively. Providers publish (purely syntactic) WSDL advertisements to UDDI registries (constructed in the style of yellow pages) which in turn provide clients with keyword- or taxonomy-based service discovery capabilities. On the one hand, WSDL descriptions do not include any *semantic information* and hence they are not "self-described" in a machine-interpretable way. This severely limits the quality of the discovery results as the matched services may not necessarily offer the requested functionality, and hence fully-automated service discovery becomes unfeasible. On the other hand, WSDL descriptions lack *behaviour information*. A direct consequence of this is that service compositions may lock during execution. Stated differently, without

any protocol information (e.g., order of messages sent/received) no guarantee on the behaviour of service compositions can be ensured.

Various proposals have been put forward in order to enhance service descriptions. WSDL-S [2], OWL-S [17], SWSO [21], WSMO [29], or METEOR-S [20] annotate services with semantic information. BPEL [5], WSCDL [27], METEOR-S [1], OWL-S [17], SWSO [21], or recently YAWL [24] add protocol information to service descriptions. All the above proposals can be in principle exploited for improving the accuracy of service matching, for extending the properties of service compositions, as well as for automatising both processes.

On the other hand, adding semantics and/or behaviour information to service descriptions leads to distinguishing *more* service descriptions, and hence to matching *less* services or service compositions with a given query. In this perspective, due to the heterogeneous and evolving nature of business processes, *service adaptation* is hence needed to overcome the (unavoidable) mismatches between the interacting parties.

Our long-term objective is to develop a general methodology for service adaptation capable of suitably overcoming both semantic and behaviour mismatches. In this paper we tackle the problem of discovering services that can be adapted to fulfil a client request. We consider a registry R of *service contracts* where each contract includes a description of the service behaviour (expressed by a YAWL workflow [24]), as well as an (ontology-annotated) signature. We also consider a query Q requesting a service with certain inputs and outputs.

The adaptation process roughly consists of:

1. matching a candidate service S in R by checking the compatibility of the execution traces of S with respect to the given query Q, and
2. generating the contract of the service S adapted so as to fulfil the query Q.

The rest of the paper is organised as follows. We first introduce service contracts (Subsection 2.1) and give an overview of the phases of the adaptation methodology (Subsection 2.2), followed by a motivating example (Subsection 2.3). We then describe the service matching phase (Subsection 2.4) and then the contract generation phase (Subsection 2.5). In Section 3 we briefly overview related work. Finally, Section 4 presents some concluding remarks.

2 Service Adaptation

2.1 Service Contracts

We consider services that are described by *contracts* [16], and we argue that contracts should in general include different types of information: (a) *Ontology-annotated signatures*, (b) *Behaviour*, and (c) *Extra-functional properties*. Following [17], we argue that WSDL signatures should be enriched with ontological information (e.g., expressed with OWL [15] or WSDL-S [2]) to describe the semantics of services, necessary to automate the process of overcoming signature mismatches as well as service discovery, adaptation, and composition. Still, the

information provided by ontology-annotated signatures is necessary but *not* sufficient to ensure a correct inter-operation of services (e.g., absence of locks). Following [16], we argue that contracts should also expose a (possibly partial) description of the interaction protocols of services. We argue that YAWL [24] (see below) is a good candidate to express service behaviour as it has a well-defined formal semantics and it supports a number of workflow patterns. Finally, we argue that service contracts should expose, besides annotated signatures and behaviour, also so-called extra-functional properties, such as performance, reliability, or security. (We will not however consider these properties in this work, and leave their inclusion into the adaptation framework to future work.)

We intend to build an adaptation framework capable of translating the behaviour of a service described using existing process/workflow modelling languages (e.g., BPEL [5], OWL-S [17], etc.) into equivalent descriptions expressed through an abstract language with a well-defined formal semantics, and vice-versa. Two immediate advantages of using such an abstract language are the possibility of developing formal analyses and transformations, independently of the different languages used by providers to describe the behaviour of their services. We consider that YAWL [24] is a promising candidate to be used as an abstract workflow language for describing service behaviour. YAWL is a new proposal of a workflow/business processing system, which supports a concise and powerful workflow language and handles complex data, transformations and Web service integration. YAWL defines twenty most used workflow patterns gathered by a thorough analysis of a number of languages supported by workflow management systems. These workflow patterns are divided in six groups (basic control-flow, advanced branching and synchronisation, structural, multiple instances, state-based, and cancellation).[1] YAWL extends Petri Nets by introducing some workflow patterns (for multiple instances, complex synchronisations, and cancellation) that are not easy to express using (high-level) Petri Nets. Being built on Petri Nets, YAWL is an easy to understand and to use formalism. With respect to process algebras, YAWL features an intuitive (graphical) representation of services through workflow patterns. Furthermore, as illustrated in [23], it is likely that a simple workflow which is troublesome to model for instance in π-calculus may be instead straightforwardly modelled with YAWL. A thorough comparison of workflow modelling with Petri Nets vs. π-calculus may be found in [23]. With respect to the other workflow languages (mainly proposed by industry), YAWL relies on a well-defined formal semantics. Moreover, not being a commercial language, YAWL supporting tools (editor, engine) are freely available.

2.2 Bird's-eye View of the Adaptation Methodology

The service adaptation methodology we propose consists of four main phases:

[1] Space limitations do not allow us to illustrate these patterns. A detailed description of them may be found in [25].

0. **Service Translation**. This preliminary phase deals with translating real-world service descriptions (e.g., BPEL + semantics, or OWL-S, etc.) into equivalent service contracts using YAWL as an abstract workflow language for expressing behaviour, and OWL for example for expressing semantic information. One may note that such a translation may be done off-line and hence it is not a burden for the adaptation process. (A thorough analysis of how to transform BPEL specifications into workflow patterns can be found in [26].)
1. **Service Matching**. The first step of this phase – to be done off-line – derives from each service workflow a *trace table* which associates each service process with preconditions, inputs and outputs. The compatibility between the trace table of the query and the trace table of each service is then checked, in order to determine the services (if any) that can be adapted to fulfil the query. It is worth noting that the trace compatibility check is ontology-aware in that it copes with exact/plug-in/subsumes semantic matches [18].
2. **Contract Generation**. The workflow of a candidate service found by the previous phase (if any) is then modified in order to generate the contract of the service adapted to fulfil the client query. Informally, the service workflow is modified so as to specify a refined behaviour of the initial service that enforces the needed adaptation.
3. **Service Deployment**. Finally, the generated contract can be deployed as a real-world Web service (i.e., described using OWL-S, or BPEL + semantics, etc.). The client will hence view the requested functionality as a new Web service that can now be discovered and further adapted to other requests. This operation is intuitively the inverse of the operation done during the Service Translation phase.

As mentioned in the Introduction, we will focus on phases (1) and (2) in the rest of the paper.

2.3 Motivating Example

Due to the particular usage of YAWL to model Web services/business processes, we shall use the term "process" to denote YAWL tasks as well as "service" to denote a workflow specification.

Consider the example in Figure 1 which will be used as a basis for describing the proposed methodology. As one may note, a service consists of processes and control-flow links among them. There are two specific processes (which we call *start* and *end*) corresponding to the YAWL *input* and *output condition*, respectively. A process Q is to be executed after another process P if there is a directed link from P to Q. Processes employ one *join* and one *split* construct. A join or split control construct may be one of the following: AND, OR, XOR, or EMPTY. Intuitively, the join specifies "how many" processes *before* P are to be terminated in order to execute P, while the split construct specifies "how many" processes *after* P are to be executed. The EMPTY-join (split) is used when *only one* process execution precedes (follows, respectively) the execution of P. We

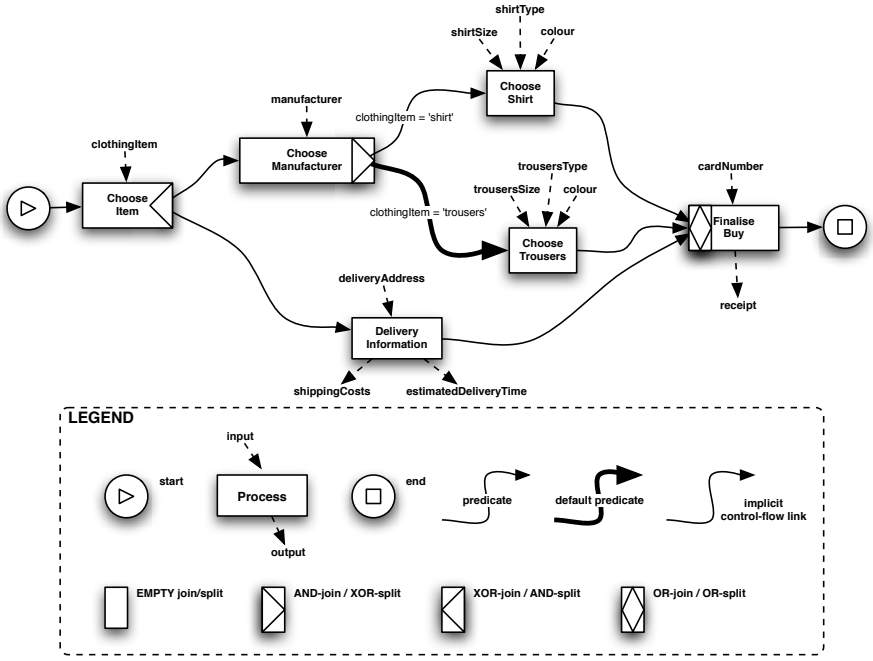


Fig. 1. *Clothing Shop* service selling shirts and trousers.

graphically represent parameters as well. YAWL uses predicates in the form of logical expressions to decide the control-flow in the case of XOR- and OR-splits.

Clothing Shop is a service that sells shirts and trousers. When invoked, the service firstly executes the *Choose Item* process, for which the client has to provide the item she is interested in (either "shirt" or "trousers"). The execution continues with both *Choose Manufacturer* and *Delivery Information* the AND-split of *Choose Item*. *Choose Manufacturer* inputs the desired designer and, based on the item to be bought, it enables *only one* of the following two processes: *Choose Shirt* if the requested item is a shirt, or *Choose Trousers* otherwise. Please note its XOR-split as well as the predicates annotating the respective control-flow links. Both processes input the type (e.g., "T-shirt" or "jeans"), size (e.g., "M" or "33") and colour (e.g., "black") of the requested item. *Delivery Information* is a process which inputs the address intended for delivery. Such information may be submitted by the client at any moment of the purchase (after choosing the item yet prior to the payment). *Delivery Information* outputs the shipping costs as well as the estimated delivery time. Last but not least, *Finalise Buy* employs an OR-join in order to wait for an item to be chosen as well as for the delivery information to be available. It inputs a credit card number and it generates the purchase receipt. Please note that the example is not supposed to present a software masterpiece and in order to keep it manageable we omit

scenarios in which, for example, a desired item is not available, payment issues, and so on.

Let us assume now that a client desires a service which sells trousers only. She might search one by issuing the following query:

- *requested inputs:* {*jeans, designer, address, size, dark-blue, cardNumber*},
- *requested outputs:* {*shippingCosts, receipt*}.

For simplicity we shall assume that both query and services use the same parameter ontology and that there is an exact/plug-in/subsumes match [18] between the following parameter pairs[2]: *jeans* and *clothingItem*, *jeans* and *trousersType*, *designer* and *manufacturer*, *address* and *deliveryAddress*, *size* and *trousersSize*, *dark-blue* and *colour*, *cardNumber* and *cardNumber*, *shippingCosts* and *shippingCosts*, and finally *receipt* and *receipt*.

It is easy to see that the service does not match the given query, since the service requires more inputs than those specified in the query (viz., *shirtSize* and *shirtType*). However, the *Clothing Shop* service could be in principle adapted so as to inhibit its capability of selling shirts and convert it into a "trousers shop" service.

In the rest of the paper we show how the proposed adaptation methodology performs an ontology-aware matching at the level of sub-services (viz., processes) and succeeds in adapting the initial service so as to fully satisfy the client query.

2.4 Service Matching

The objective of this phase is to determine whether there are services in the registry that can be adapted to fully match the given query.

Deriving the Trace Table. Firstly, each service workflow is analysed in order to generate a *Trace Table* (TT) which associates each service process with preconditions, inputs and outputs. More precisely, each process P is associated with a set of triples of the form <*Preconditions, Needed Inputs, Generated Outputs*>, where *Preconditions* represents the set of data and control constraints that must be satisfied to be able to execute P and the other processes executed so far (in that execution trace). *Needed Inputs* is the set of inputs that are requested by P together with the inputs of the other processes that have been executed so far. Similarly, *Generated Outputs* is the set of outputs generated by P together with the other processes executed so far.

We generate the TT for a service from its *Reachability Graph* (RG). The RG of a service can be derived from its YAWL workflow augmented with explicit conditions. An example of such workflow may be seen in Figure 2. It is worth noting that YAWL conditions can be represented as (Petri net) places holding tokens [24]. On the one hand, tokens are placed into places by firing processes depending on their split constructs and on the YAWL predicates (if present).

[2] The first parameter in a pair belongs to the query while the second to the service.

For processes with EMPTY- (AND-) splits, YAWL considers implicit conditions and a token is generated for (all) the output place(s). In the case of XOR- or OR-splits, YAWL uses predicates to determine which output places will receive tokens. All predicates of such a split are ordered (by the workflow designer) and one is chosen as default (with lowest priority). For a XOR-split, a token is sent on the link corresponding to the predicate with the lowest order that evaluates to true. For an OR-split, a token is sent along all links whose predicates evaluate to true. For both splits, if all predicates are false then a token is sent along the default link only. On the other hand, places are used to enable processes for execution. If the process has an EMPTY-join then its input place has to contain a token. For an AND-join, all input places have to contain tokens. In the case of a XOR-join at least one input place has to have a token. Finally, according to [24], if the process has an OR-join, it is enabled only when at least one of its input places contains a token and no other tokens can be placed in its remaining (empty) input places. For the example in Figure 2, *Finalise Buy* is enabled if and only if C_7 and *either* C_5 *or* C_6 contain tokens. Hence, it can not be enabled when only C_7 contains a token before a token will arrive at C_5 or C_6. One should note that tokens can not be added to both C_5 and C_6 during a workflow instance due to the XOR-split of the *Choose Manufacturer* process.

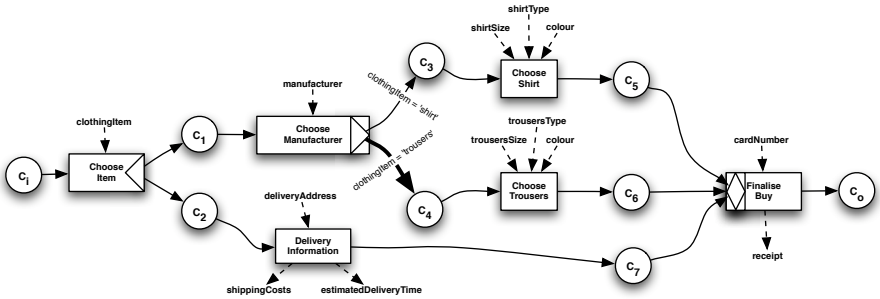


Fig. 2. *Clothing Shop* workflow with places/conditions.

Following [30] we derive a RG having *markings* as nodes and *labelled arrows* as edges. A marking M consists of the set of all places containing tokens and it is denoted as $C_i + ... + C_j$. An arrow states that the workflow execution state evolves from a marking M into a marking M' and it is labelled with the process that fires and – in the case of OR- and XOR-splits – also with the places that receive tokens. Figure 3 shows the RG corresponding to the *Clothing Shop* workflow in Figure 2. For example, the arrow from the initial marking C_i to the marking $C_1 + C_2$ is labelled as *Choose Item*. This is to be read as: "From the initial marking containing a token in C_i only, the *Choose Item* process is enabled by consuming the token in C_i and by firing it produces a token in C_1 and another in C_2". As one should note, in the markings circled in Figure 3 the *Finalise Buy*

process cannot fire as it expects one more token to be placed in one of its empty input places.

RG is incrementally built by starting from the initial marking (that contains C_i only) and by looking for processes which can be enabled. Labelled arrows and new markings are successively added to the graph. One should note that checking whether a process having an OR-join can be enabled is done using the algorithm given in [30].

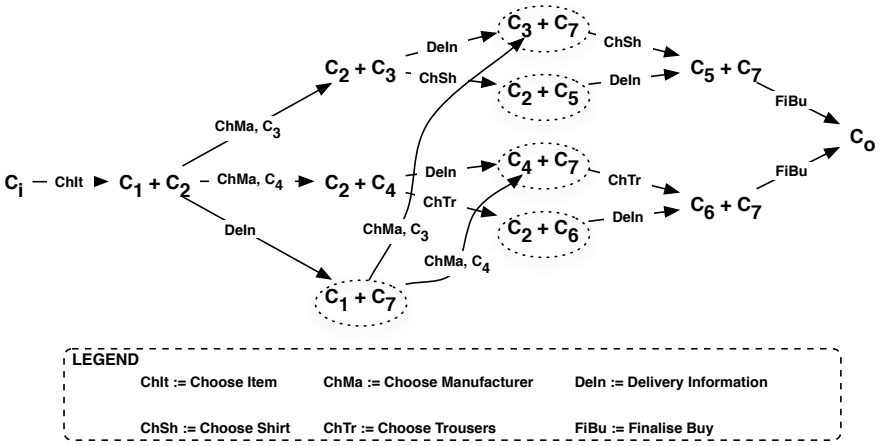


Fig. 3. RG for the *Clothing Shop* service.

For example, by considering the marking C_i, we generate and add to RG the marking $C_1 + C_2$ and we label the arrow as *Choose Item*. Or, by assuming that the markings $C_3 + C_7$ and $C_4 + C_7$ have already been generated, and by considering the marking $C_1 + C_7$, we have that *Choose Manufacturer* can fire and it can place a token *either* in C_3 *or* in C_4. Hence, we just add arrows from the $C_1 + C_7$ to $C_3 + C_7$ and to $C_4 + C_7$ and we label them as "*Choose Manufacturer,* C_3" and "*Choose Manufacturer,* C_4" respectively.

The process of generating the TT for a process P looks in the RG for all paths (i.e., traces) p originating in the initial marking and ending with a marking having an outgoing arrow labelled by P. The preconditions for the p execution of P are expressed as the set of all conditions (viz., places) in the markings of p. The set of needed inputs is obtained by taking the inputs of all processes labelling arcs of path p, together with the inputs of P. Similarly, the set of generated outputs consists of the outputs of all processes labelling arcs of path p, together with the outputs of P. For example, if we consider the arrow labelled as "*Choose Manufacturer,* C_3" that originates in the marking $C_1 + C_2$, we have to add to *TT(Choose Manufacturer)* the following entry: $<\{C_i, C_1, C_2\}$, {*clothingItem, manufacturer*}, $\oslash>$. If we consider the arrow labelled as *Choose Shirt* that orig-

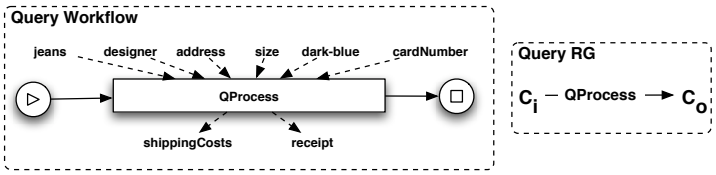
Process Name: $<$*Preconditions, Needed Inputs, Generated Outputs*$>$.
C_i*:* $<\oslash, \oslash, \oslash>$.
Choose Item: $<\{C_i\}$, {*clothingItem*}, $\oslash>$.
Choose Manufacturer: $<\{C_i, C_1, C_2\}$, {*clothingItem, manufacturer*}, $\oslash>$; $<\{C_i, C_1, C_2, C_7\}$, {*clothingItem, manufacturer, deliveryAddress*}, {*shippingCosts, estimatedDeliveryTime*}$>$.
Delivery Information: $<\{C_i, C_1, C_2\}$, {*clothingItem, deliveryAddress*}, {*shippingCosts, estimatedDeliveryTime*}$>$; $<\{C_i, C_1, C_2, C_3\}$, {*clothingItem, manufacturer, deliveryAddress*}, {*shippingCosts, estimatedDeliveryTime*}$>$; $<\{C_i, C_1, C_2, C_3, C_5\}$, {*clothingItem, manufacturer, deliveryAddress, shirtSize, shirtType, colour*}, {*shippingCosts, estimatedDeliveryTime*}$>$; $<\{C_i, C_1, C_2, C_4\}$, {*clothingItem, manufacturer, deliveryAddress*}, {*shippingCosts, estimatedDeliveryTime*}$>$; $<\{C_i, C_1, C_2, C_4, C_6\}$, {*clothingItem, manufacturer, deliveryAddress, trousersSize, trousersType, colour*}, {*shippingCosts, estimatedDeliveryTime*}$>$.
Choose Shirt: $<\{C_i, C_1, C_2, C_3\}$, {*clothingItem, manufacturer, shirtSize, shirtType, colour*}, $\oslash>$; $<\{C_i, C_1, C_2, C_3, C_7\}$, {*clothingItem, manufacturer, deliveryAddress, shirtSize, shirtType, colour*}, {*shippingCosts, estimatedDeliveryTime*}$>$.
Choose Trousers: $<\{C_i, C_1, C_2, C_4\}$, {*clothingItem, manufacturer, trousersSize, trousersType, colour*}, $\oslash>$; $<\{C_i, C_1, C_2, C_4, C_7\}$, {*clothingItem, manufacturer, deliveryAddress, trousersSize, trousersType, colour*}, {*shippingCosts, estimatedDeliveryTime*}$>$.
Finalise Buy, C_o*:* $<\{C_i, C_1, C_2, C_3, C_5, C_7\}$, {*clothingItem, manufacturer, deliveryAddress, shirtSize, shirtType, colour, cardNumber*}, {*shippingCosts, estimatedDeliveryTime, receipt*}$>$; $<\{C_i, C_1, C_2, C_4, C_6, C_7\}$, {*clothingItem, manufacturer, deliveryAddress, trousersSize, trousersType, colour, cardNumber*}, {*shippingCosts, estimatedDeliveryTime, receipt*}$>$.

Table 1. *Trace Table* of the *Clothing Shop* service.

inates in the marking $C_3 + C_7$ then we have to add to *TT(Choose Shirt)* the following entry: $<\{C_i, C_1, C_2, C_3, C_7\}$, {*clothingItem, manufacturer, deliveryAddress, shirtSize, shirtType, colour*}, {*shippingCosts, estimatedDeliveryTime*}$>$ which actually synthesises the two different traces leading to the marking $C_3 + C_7$. The final TT obtained for our example is illustrated in Table 1.

Trace Table Compatibility Check. In order to match the requested query with the TT of a given service, we firstly express the query as a simple service

contract whose workflow contains one process only (together with *start* and *end*). The inputs and the outputs of the process are as requested by the query. Figure 4 presents the workflow, the RG and the TT for the request described in Subsection 2.3.



C_i: $<\emptyset, \emptyset, \emptyset>$.
$QProcess$, C_o: $<\{C_i\}$, {*jeans, designer, address, size, dark-blue, cardNumber*}, {*shippingCosts, receipt*}$>$.

Fig. 4. Query workflow, RG and TT.

The compatibility check between the trace table of the query Q and the trace table of a service S consists of looking for a process P of S such that there exists a trace T in $TT(P)$ for which: (1) the set of inputs needed by T is included in the set of inputs of Q and, dually, (2) the set of outputs of Q is included in the set of outputs generated by T. In other words, Q has to provide all inputs needed by T, and T has to generate all outputs desired by Q. One should note that the inclusion relation is ontology-aware. The query of our example matches the service in Figure 1 as for the second trace (top to bottom) in *TT(Finalise Buy)* (for example) we have: (1) { *clothingItem, manufacturer, deliveryAddress, trousersSize, trousersType, colour, cardNumber* } $\supseteq$ { *jeans, designer, address, size, dark-blue, cardNumber* }, and (2) { *shippingCosts, receipt* } $\supseteq$ { *shippingCosts, estimatedDeliveryTime, receipt* }. One should note that the preconditions set constraining T into fulfilling Q is $\{C_i, C_1, C_2, C_4, C_6, C_7\}$.

In order to provide a more user-friendly answer to the query we construct a logical expression from the set of preconditions of a trace. We can achieve this by firstly assigning a logical expression to each place of the workflow and then by taking the conjunction of all the conditions in the preconditions set of a trace. For instance, the above preconditions set constraining T into fulfilling Q might be expressed as "*(clothingItem = 'trousers') or (not clothingItem = 'shirt')*".

Being inspired by the usage of YAWL predicates to enable processes [24], we enhance the expressiveness of YAWL conditions by assigning them a logical expression as follows. For C_i, C_o or for an output place of a process having an EMPTY- or an AND-split we consider an always "true" condition (e.g., C_1, C_2, C_7, and so on). In the case of a XOR-split, we consider an output condition to be true provided "*either* the YAWL predicate for the corresponding link is

true as well as the other lower-order predicates are false, *or* the corresponding predicate is the default one and all other predicates of the respective process are false" [24]. For example, for C_3 we consider the following expression "(*clothingItem = 'shirt'* and not *clothingItem = 'trousers'*) or (*clothingItem = 'shirt'* is default and not *clothingItem = 'trousers'*)", or simply "(*clothingItem = 'shirt'* and not *clothingItem = 'trousers'*)". Similarly, for C_4 we have "(*clothingItem = 'trousers'*) or (not *clothingItem = 'shirt'*)". Hence, a token is placed into C_4 even if a client requests "shoes" as item. Last but not least, for a process having an OR-split, we consider an output condition to be true if and only if "its corresponding predicate is true, *or* the respective predicate is the default one and all other predicates of the considered process are false" [24].

2.5 Contract Generation

Assume that the client wishes to have a deployment of a service that strictly satisfies queries of the type she has issued. In other words, she wants a service, say *Trousers Boutique*, that only sells dark-blue jeans made by a certain designer. This phase of the methodology achieves that by modifying the contract of a service so as to fulfil the query. This is done in two steps:

1. First, we choose one trace T (if any) from $TT(C_o)$ of a service that is compatible with the request. This is to ensure that we will not remove processes (see below) needed for the successful termination of the service's execution. Then, we individuate the processes of the trace T as service processes having as input places the preconditions set of T. For instance, the preconditions set $\{C_i, C_1, C_2, C_4, C_6, C_7\}$ corresponds to the set of processes {*Choose Item*, *Choose Manufacturer*, *Delivery Information*, *Choose Trousers*, *Finalise Buy* }. We call *redundant* all other service processes (but *start* and *end*).
2. Then we copy the contract of the original service and modify it by cancelling redundant processes (if any) and suitably redirecting the workflow links. Workflow redirection is necessary to ensure that the workflow of the new service is consistent with the trace satisfying the query. This is achieved by adding to the workflow a process *Absorb Tokens*, and by directly connecting it as output of the *start* process. Then, all initial control-flow links that point at a redundant process are redirected to the XOR-join of *Absorb Tokens*. Then all redundant processes together with their outgoing links are removed. Finally, service outputs that are not requested by the query are hidden (see below).

For instance, the adaptation scenario for our example yields the workflow presented in Figure 5.

Notice that a YAWL service is a workflow specification which consists of one or more workflow nets [24] – one of which is the starting net. Variables are defined at both workflow and task (i.e., process) levels and the data-flow is specified by binding parameters of the workflow net and of its processes. In Figure 5 one may see that the *Trousers Boutique* service is made of one

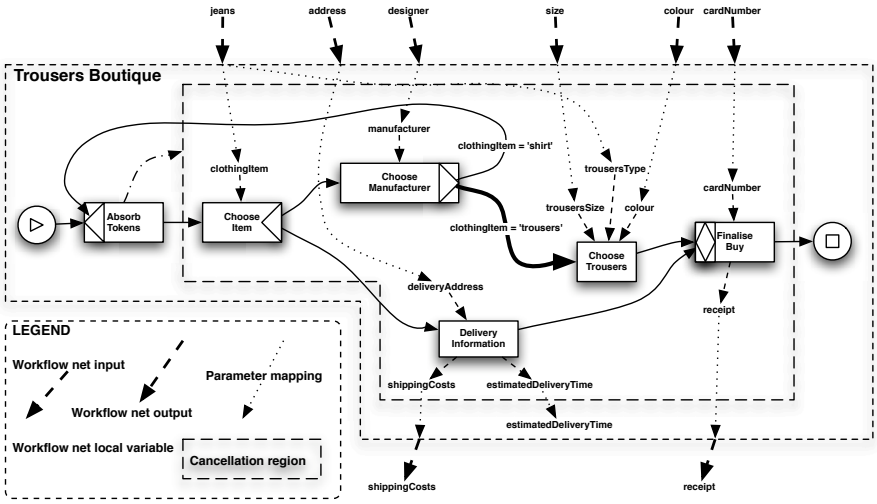


Fig. 5. Workflow of the *Trousers Boutique* service.

workflow net which contains the original processes of the *Clothing Shop* service except the redundant *Choose Shirt* process. As previously indicated, the process *Absorb Tokens* has been added at the beginning of the workflow and the original control-flow link leading at *Choose Shirt* has been redirected to *Absorb Tokens*. The inner dashed portion of the workflow in Figure 5 delimits the cancellation region [24] associated with the process *Absorb Tokens*. Informally, this means that whenever *Absorb Tokens* is executed, all tokens enabling processes in its cancellation region are removed.

If a client of the *Trousers Boutique* service requests a *shirt*, then *Choose Manufacturer* will enable *Absorb Tokens* for execution and further *Choose Item* which will ask the client for another input. Moreover, *Absorb Tokens* will clear remaining tokens in the workflow so as to avoid (possibly) multiple executions of the other processes in the workflow (e.g., *Delivery Information*). Outputs that are not desired by the requester are hidden by mapping them to local net variables.[3] This is the case for the *estimatedDeliveryTime* output of the *Delivery Information* process.

3 Related Work

Different approaches to *service discovery* have been proposed, ranging from UDDI-based approaches to proposals which take into account semantics and/or

[3] The reason why we do not remove them is, for example, that the respective process may correspond to a YAWL task that invokes a WSDL service and then a mapping between its parameters and the ones of the WSDL service is necessary.

behaviour information. Liang et al. [14] use UDDI registries and constraints over services to semi-automatically discover (composed) services. Kawamura et al. [11] enhance UDDI matching by extending WSDL service descriptions with semantic information in the style of DAML-S profiles. Kifer et al. [13] employ WSMO to define a logical framework for service discovery. Some proposals use DAML-S/OWL-S and address service matching either at the service profile level (e.g., [3,18]) or at the service model level (e.g., [4,7]).

Web *service adaptation* is in its early stages and current approaches feature only partial solutions to the issues of adaptation. Hau et al. [9] propose a framework for semantic matchmaking and service adaptation, which deals with signature mismatches yet not with behavioural ones. Syu [22] proposes an OWL-S based approach to deal with only three cases of adaptation of input parameters (permutation, modification, and combination). Iyer et al. [10] employ XML scripts and XSL to (manually) achieve the signature-level interoperability of SOAP services. Kaykova et al. [12] show how to semantically adapt heterogeneous industrial resources. Yet their approach – as [9] – relies on black-box views of services and on semantically annotated signatures. A methodology for generating adapters to solve behavioural mismatches was presented in [6], yet ensuring the availability of an adapter specification to be manually generated.

To the best of our knowledge, our service adaptation approach is the first to take into account both semantics and behaviour information, and to feature a fully automated generation of a service adapted to the client's needs.

4 Concluding Remarks

Our long-term objective is to develop a general methodology for service adaptation capable of suitably overcoming both semantic and behaviour mismatches.

The main features of our approach can be summarised as follows:

- it is a *fully automatic* approach capable of generating service contracts tailored to client requests – given a registry of service contracts (containing both semantics and behaviour information) and a query,
- it supports both service discovery and adaptation at the level of sub-services (and not only of entire services),
- it is amenable to efficient implementations, as it relies on the inspection of execution traces that can be generated off-line,
- it can be exploited to discover and adapt services written in different languages, and to generate multiple deployments of the adapted contract – given that it relies on intermediate YAWL descriptions of the behaviour of services.

In this paper we have illustrated how to automatically discover and adapt a service so as to match a client query requesting a service with certain inputs and outputs. We intend to devote our future work to extending the adaptation methodology so as to cope with queries specifying (part of) the behaviour of the desired service (i.e., not only its inputs and outputs), and to experimenting the deployment of adapted services with BPEL and OWL-S.

References

1. R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor. Constraint Driven Web Service Composition in METEOR-S. In *IEEE SCC*, pages 23–30. IEEE Computer Society, 2004.
2. R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S Version 1.0. `http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html`.
3. L. Aversano, G. Canfora, and A. Ciampi. An algorithm for web service discovery through their composition. In *IEEE International Conference on Web Services*, pages 332–, 2004.
4. S. Bansal and J. Vidal. Matchmaking of Web Services Based on the DAML-S Service Model. In T. Sandholm and M. Yokoo, editors, *Second International Joint Conference on Autonomous Agents (AAMAS'03)*, pages 926–927. ACM Press, 2003.
5. BPEL4WS Coalition. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1. `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`.
6. A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing Web Service Choreographies. In *Proceedings of First International Workshop on Web Services and Formal Methods*, 2004. Available from `http://www.lcc.uma.es/~av/Publicaciones/04/ws-fm04.pdf`.
7. A. Brogi, S. Corfini, and R. Popescu. Composition-oriented Service Discovery. In F. Gschwind, U. Assmann, and O. Nierstrasz, editors, *Proceedings of Software Composition '05, LNCS, vol. 3628*, pages 15–30, 2005.
8. U. Coalition. The UDDI Technical White Paper. `http://www.uddi.org/`.
9. J. Hau, W. Lee, and S. Newhouse. The ICENI Semantic Service Adaptation Framework. In UK e-Science All Hands Meeting, 2003. `http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/017.pdf`.
10. A. Iyer, G. Smith, P. Roe, and J. Pobar. An Example of Web Service Adaptation to Support B2B Integration. `http://ausweb.scu.edu.au/aw02/papers/refereed/smith2/paper.html`.
11. T. Kawamura, J. D. Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. Public Deployment of Semantic Service Matchmaker with UDDI Business Registry. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of The Semantic Web ISWC'04, LNCS, Volume 3298*, pages 752–766, 2004.
12. O. Kaykova, O. Khriyenko, D. Kovtun, A. Naumenko, V. Terziyan, and A. Zharko. An Approach to Semantic Adaptation of Heterogeneous Industrial Web Resources. `http://www.cs.jyu.fi/ai/papers/SJIS-2005.pdf`.
13. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, volume 119, Hiroshima, Japan, 2004. CEUR Workshop Proceedings.
14. Q. Liang, L. N. Chakarapani, S. Y. W. Su, R. N. Chikkamagalur, and H. Lam. A Semi-Automatic Approach to Composite Web Services Discovery, Description and Invocation. *International Journal of Web Services Research*, 1(4):64–89, 2004.
15. D. McGuiness and F. van Harmelen (Eds). OWL Web Ontology Language Overview. Web guide, February 2004. `http://www.w3.org/TR/owl-features`.
16. L. Meredith and S. Bjorg. Contracts and Types. *CACM*, 46(10), 2003.

17. OWL-S Coalition. OWL-S: Semantic Markup for Web Services Version 1.1. http://www.daml.org/services/owl-s/1.1/overview/.
18. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matchmaking of Web Services Capabilities. In I. Horrocks and J. Hendler, editors, *First International Semantic Web Conference on The Semantic Web, LNCS 2342*, pages 333–347. Springer-Verlag, 2002.
19. M. P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Communication of the ACM*, 46(10):24–28, 2003.
20. P. Rajasekaran, J. A. Miller, K. Verma, and A. P. Sheth. Enhancing Web Services Description and Discovery to Facilitate Composition. In J. Cardoso and A. P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 55–68. Springer, 2004.
21. SWSO Coalition. Semantic Web Services Ontology (SWSO) Version 1.0. http://www.daml.org/services/swsf/1.0/swso/.
22. J.-Y. Syu. *An Ontology-Based Approach to Automatic Adaptation of Web Services*. Department of Information Management National Taiwan University, 2004. http://www.im.ntu.edu.tw/IM/Theses/r92/R91725051.pdf.
23. W. M. P. van der Aalst. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype, 2004. Available from http://tmitwww.tm.tue.nl/staff/wvdaalst/pi-hype.pdf.
24. W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275, 2005.
25. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.
26. P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.-Y. Song, S. W. Liddle, T. W. Ling, and P. Scheuermann, editors, *Proceedings of the 22nd International Conference on Conceptual Modeling*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2003.
27. WSCDL Coalition. Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/.
28. WSDL Coalition. Web Service Description Language (WSDL) version 1.1. http://www.w3.org/TR/wsdl.
29. WSMO Coalition. Web Service Modeling Ontology (WSMO) D2v1.2. http://www.wsmo.org/TR/d2/v1.2/.
30. M. T. Wynn, D. Edmond, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In G. Ciardo and P. Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443. Springer, 2005.
31. J. Yang. Web Service Componentization. *Communications of the ACM*, 46(10):35–40, 2003.

Complex Adaptive Services

Jean-François Mascari[1] and Giuseppe A. Cavarretta[2]

[1]Istituto per le Applicazioni del Calcolo – (National Research Council) CNR
Viale del Policlinico 137, 00161 Rome, Italy,
mascari@iac.rm.cnr.it

[2]Servizi Sistemi Informativi – (National Research Council) CNR
Piazzale Aldo Moro 7, 00185 Rome, Italy,
giuseppealfredo.cavarretta@cnr.it

Abstract. The duality between service providers and services consumers is a basic pattern of service oriented computing. A service oriented approach to business processes and to adaptive interacting processes requires an additional pattern based on the, possibly online, interaction between specification, execution and evaluation basic processes. These two patterns combine to composed processes: the foundations of complex adaptive services. The management of the dynamics of such services is then obtained by additional processes distributed over the network of interactions of the basic and composed processes. The double triad architecture so obtained is inspired from quark-antiquark models of particle physics.

1 Introduction

A new paradigm in various applications areas is emerging from the convergence of Grid technologies, semiotics methodologies and knowledge applications. This convergence is enabled by the duality between service providers and service consumers. Essentially what is happening can be explained by three related phenomena.

- The logical phenomena realized by Grid computing consists in a "quantifier inversion" process from a "*for every* data and processing request, *there exists* a specific data and processing supply technology" to a "*there exists* a general data and processing request – supply interaction Grid, *for every* data and processing request and supply";
- The semiotics phenomena realized by Semantic Grid and implicit also in the data Driven Dynamic Application Simulation (DDDAS) approach "*for every* node of the

semiotic triangle (concept-model, symbol-simulation, object-reality), *there exists* an offline space-time of activation" to a "*there exists* an online space-time of activation, *for every* node of the semiotic triangle";

- The complexity science phenomena being realized by Knowledge Grid "*for every* role in a Knowledge Grid played by interacting objects of a Semantic Grid, *there exists* a local dynamics such that the global dynamics of the emerging Knowledge Grid is obtained by composing such local dynamics" to a "*there exists* a global dynamics of a Knowledge Grid, *for every* role played at a local level by interacting objects of the underlying Semantic Grid such that the global dynamics could not be obtained by the set of such local dynamics ".

The impact of such a paradigm on business process integration and management will require to reason in terms of "complex adaptive services" based on the negotiation patterns between:

- Consumer objectives specification, realization and evaluation, and
- Providers' constraints specification, satisfaction and evaluation.

These patterns are captured by the proposed fractal double-triads modeling approach derived from [1], which has been inspired by the quark-antiquark model of particle physics, is presented for the architectural design of complex adaptive services.

Usually an object oriented paradigm is used to represent organization and functional rules, describing who makes what and how, using processes to reach the company objectives.

However a service oriented modelling can be used to represent a dynamics of an organization independently of its specific organization model.

A service oriented approach is useful to a specification level and the object oriented approach to an implementation level.

The negotiation patterns are based on the following processes:

- Consumer Objectives Specification: Request preparation,
- Consumer Objectives Realization: Request execution,
- Evaluation of Consumer Objectives Realization: Request evaluation,
- Providers Constraints Specification: Supply preparation,
- Providers Constraints Satisfaction: Supply execution,
- Evaluation of Providers Constraints Satisfaction: Supply evaluation.

Just an example from Italian National Research Council internal regulation about planning, execution and evaluation of research activity:

1. The Board of Directors, following a President proposal, deliberates the guide lines for a three-years research activity plan...

2. The director of the DEPARTMENT produces guide lines for the DEPARTMENT, as proposal of contribution to the three-years plan…. in particular, also through a negotiation among the DEPARTMENT DIRECTORS…

3. The INSTITUTES, in coherence with the guide lines and based on the current activities and the opportunities of development, they formulate proposals of research activity to related DEPARTMENT and to other DEPARTMENTS,. ...

4. A negotiation between each DEPARTMENT and related INSTITUTES produces research activity plan for each INSTITUTE….

In the drawing up of the regulations is pointed "who makes what" about document or activities, "how it works" and "how monitoring and evaluations".

In an object-based model all these phases are spread out, as we could see, and consequently the dependency of the organization dynamics is strongly dependent to the specific organization structure.
In a service-based model, on the contrary, these phases are semantically "typed", i.e. grouped together and consequently the organization dynamics is independent of the specific organization structure.

Several organization units of the agency supply in a "recursive" way their elementary contribution which is negotiated with the other units at different hierarchy levels.
Several organization units inside of the organization (Institutes, Departments, Council Directors of Department, President, Board of Directors), play therefore a dual role.
They play Request roles when they make a CALL inside their competences, and Supply roles when they SUPPLY the result of their job to the higher hierarchical level.
Each interaction shows in a recurrent way the matching of the Request and the Supply with respect to the preparation, the execution and the evaluation/monitoring for every phase.

Aim of this model is to represent complex structures through decomposition in nested patterns.

2 Services Consumers and Providers Duality

Services Consumer Basic Processes consist of:

- Consumer Objectives Specification,
- Consumer Objectives Realization and
- Evaluation of Consumer Objectives Realization.

Their interaction is summarized in the following

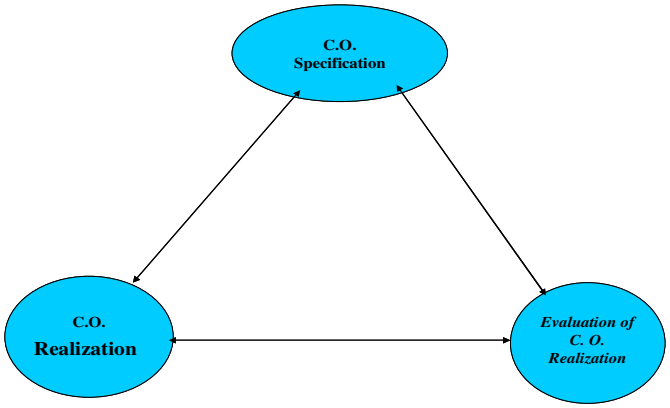


Fig. 1. Dynamics of Services Consumers Objectives

Services Provider Basic Processes consist of:

- Providers Constraints Specification,
- Providers Constraints Satisfaction and
- Evaluation of Providers Constraints Satisfaction.

Their interaction is summarized in the following

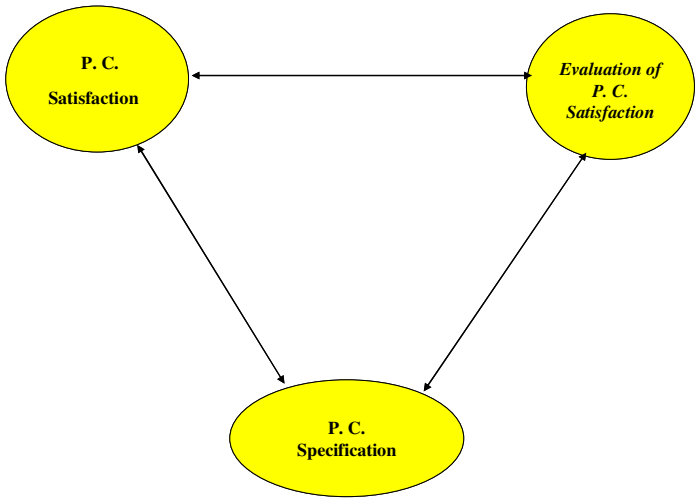


Fig. 2. Dynamics of Service Providers Constraints

We distinguish two types of adaptation processes between basic processes.

i) Adaptation Processes between Services Consumer Basic Processes summarized in the following

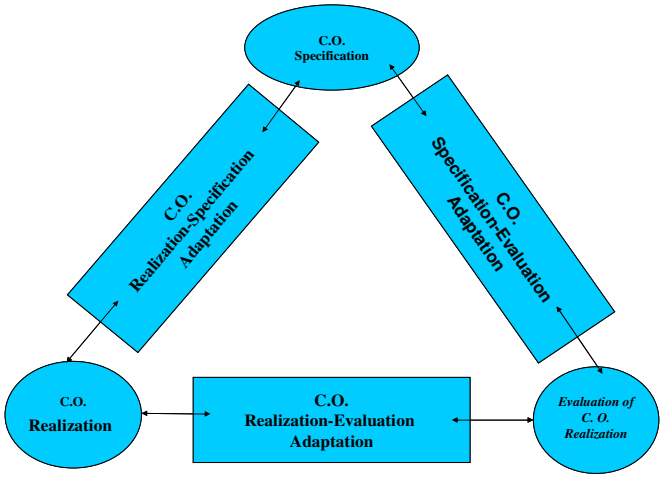


Fig. 3. Adaptation of Service Consumers Objectives

ii) Adaptation Processes between Services Provider Basic Processes summarized in the following

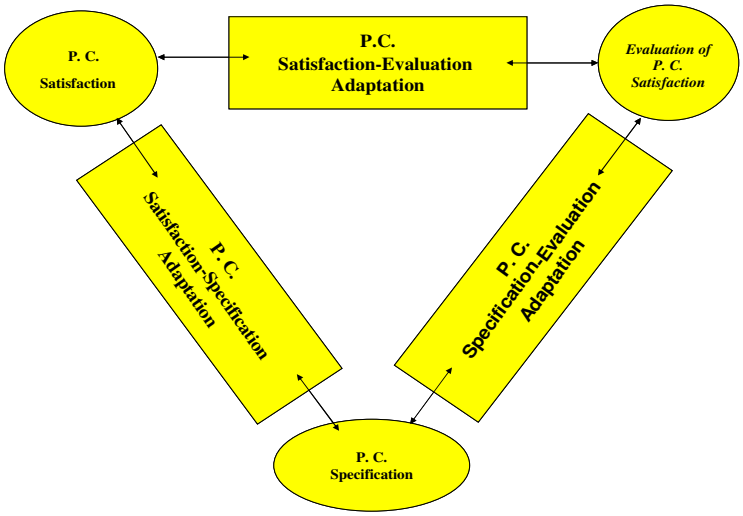


Fig. 4. Adaptation of Service Providers Constraints

A fractal double triad is obtained since at a local level each of the above six basic processes (and the corresponding six adaptation processes) consist of a negotiation between two dual roles:

- a provider of the basic process (of the adaptation process)
- consumer of the basic process (of the adaptation process)

as represented in the following

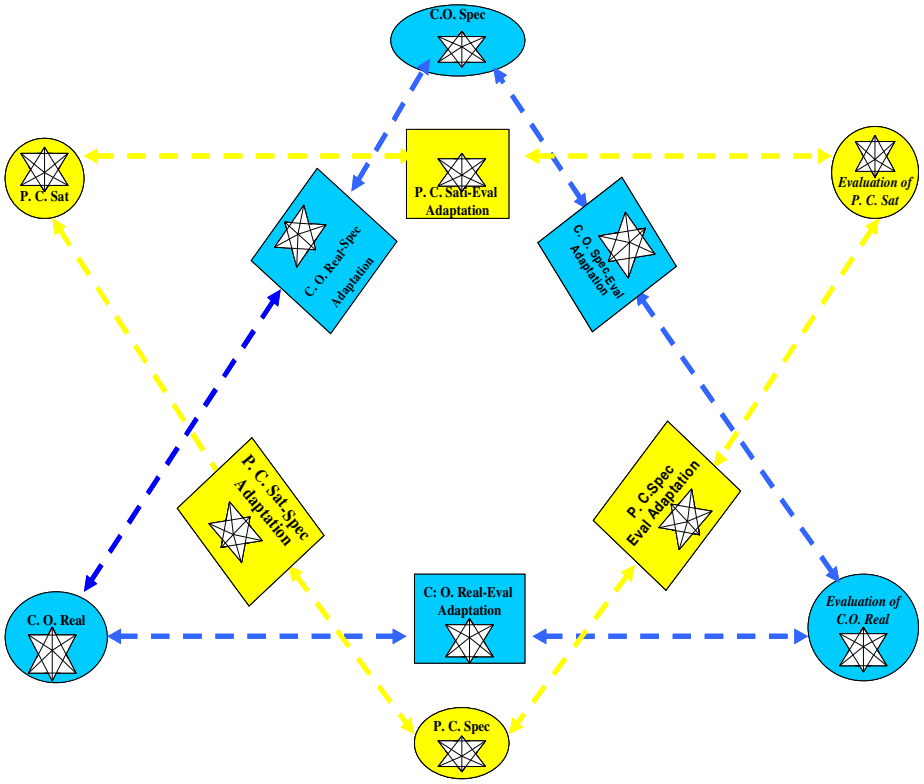


Fig. 5. Consumers Objectives and Providers Constraints Adaptation

3 Composed Processes and Adaptation

We consider composed processes associated to Services Consumers and to Services Providers interaction.
Composition of Services Consumer Basic Processes and Services Provider Basic Processes consist of:

- Evaluation of Providers Constraints Satisfaction & Consumer Objectives Specification,
- Consumer Objectives Specification & Providers Constraints Satisfaction,
- Providers Constraints Satisfaction & Consumer Objectives Realization,
- Consumer Objectives Realization & Providers Constraints Specification,
- Providers Constraints Specification & Evaluation of Consumer Objectives Realization and

- Evaluation of Consumer Objectives Realization & Evaluation of Providers Constraints Satisfaction.

Adaptation Processes between Composed Processes consisting of

- C.O. Eval ↔ P.C. Eval ↔ C.O. Spec,
- P.C. Eval ↔ C.O. Spec ↔ P.C. Sat,
- C.O. Spec ↔ P.C. Sat ↔ C.O. Real,
- P.C. Sat ↔ C.O. Real ↔ P.C. Spec,
- C.O. Real ↔ P.C. Spec ↔ C.O. Eval and
- P.C. Spec ↔ C.O. Eval ↔ P.C. Sat

are summarized in the following

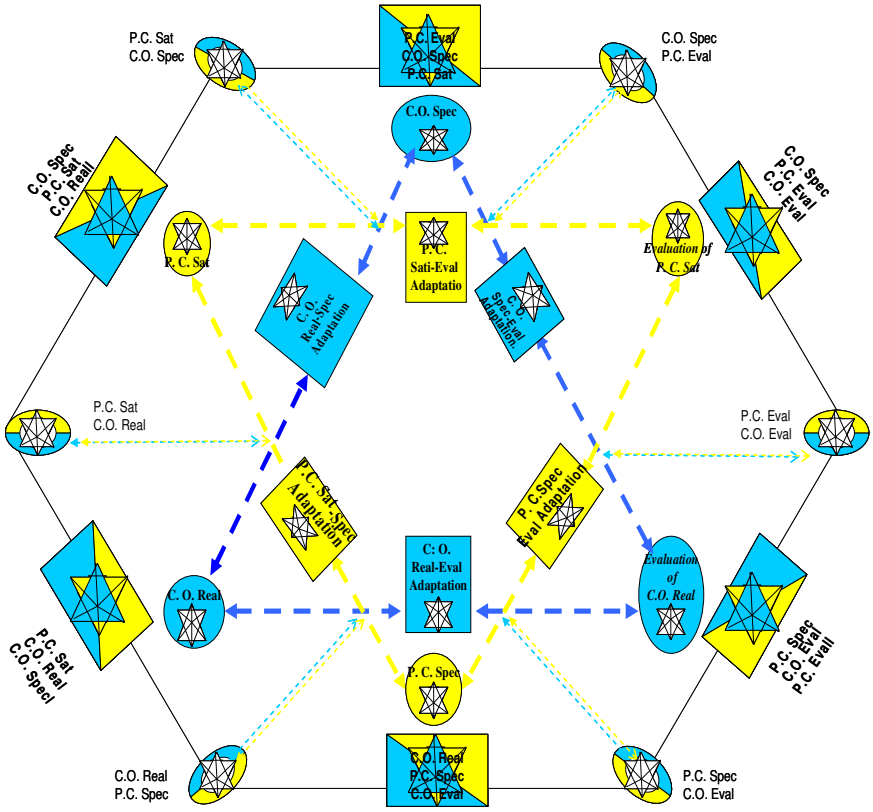


Fig. 6. Dynamics of Composed Processes

4 Management Services, Adaptation and Basic Services

We consider management services supporting services consumers and service providers.
Management services for

- Services Consumers Basic Processes and
- Adaptation Processes between S.C. Basic Processes

consist of

- Consumer Objectives Specification ➔ C.O. Spec-Real Adapt,
- C.O. Spec-Real Adapt ➔Consumer Objectives Realization,
- Consumer Objectives Realization ➔ C.O. Real-Eval Adapt,
- C.O. Real-Eval Adapt) ➔Evaluation of Consumer Objectives Realization,
- Evaluation of Consumer Objectives Realization ➔ C.O. Spec. Eval,
- C.O. Spec. Eval ➔ Evaluation of Consumer Objectives Realization.

Management services for

- Services Providers Basic Processes and
- Adaptation Processes between S.P. Basic Processes

consist of

- Providers Constraints Specification ➔ P.C. Spec-Sat Adapt,
- P.C. Spec-Sat Adapt ➔ Providers Constraints Satisfaction,
- Providers Constraints Satisfaction ➔ P.C. Sat-Eval Adapt,
- P.C. Sat-Eval Adapt ➔ Evaluation of Providers Constraints Satisfaction,
- Evaluation of Providers Constraints Satisfaction ➔ P.C. Spec-Eval,
- P.C. Spec-Eval ➔ Providers Constraints Specification.

These management services are summarized in the following

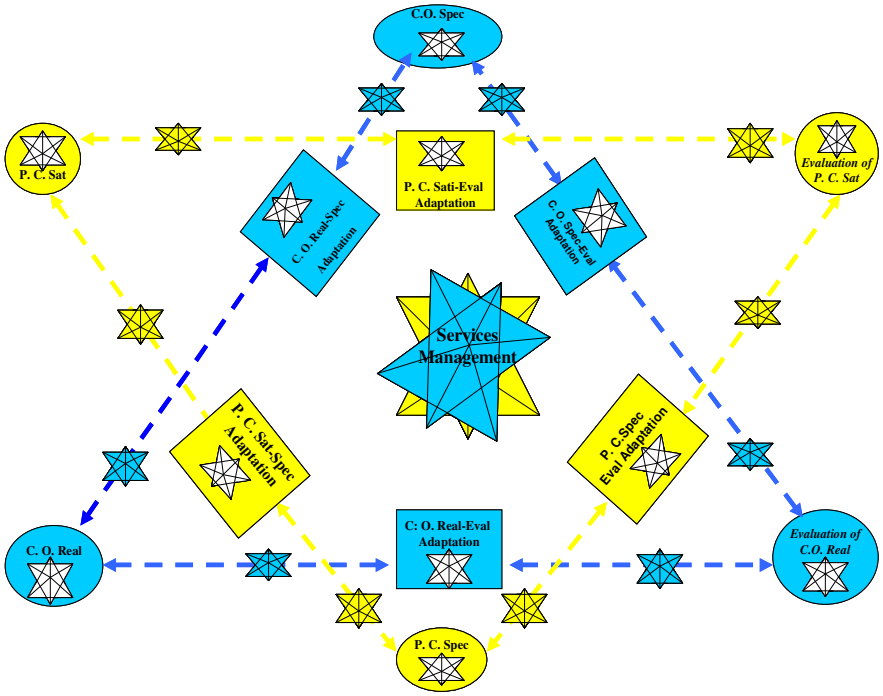


Fig. 7. Management and Basic Processes

5 Management Services, Adaptation, Basic and Composed Services

Management services for

- Adaptation Processes between Basic Processes and
- Adaptation Processes between Composed Processes

consist of

- (C.O. Eval ↔ P.C. Eval) & (C.O. Eval ↔ P.C. Eval ↔ C.O. Spec)
- (C.O. Eval ↔ P.C. Eval ↔ C.O. Spec) & (C.O. Eval ↔ C.O. Spec)
- (P.C. Eval ↔ C.O. Spec) & (P.C. Eval ↔ C.O. Spec ↔ P.C. Sat)
- (P.C. Eval ↔ C.O. Spec ↔ P.C. Sat) & (C.O. Spec ↔ P.C. Sat)
- (C.O. Spec ↔ P.C. Sat) & (C.O. Spec ↔ P.C. Sat ↔ C.O. Real)
- (C.O. Spec ↔ P.C. Sat ↔ C.O. Real) & (P.C. Sat ↔ C.O. Real)
- (P.C. Sat ↔ C.O. Real) & (P.C. Sat ↔ C.O. Real ↔ P.C. Spec)
- (P.C. Sat ↔ C.O. Real ↔ P.C. Spec) & (C.O. Real ↔ P.C. Spec)
- (C.O. Real ↔ P.C. Spec) & (C.O. Real ↔ P.C. Spec ↔ C.O. Eval)
- (C.O. Real ↔ P.C. Spec ↔ C.O. Eval) & (P.C. Spec ↔ C.O. Eval)
- (P.C. Spec ↔ C.O. Eval) & (P.C. Spec ↔ C.O. Eval ↔ P.C. Sat)
- (P.C. Spec ↔ C.O. Eval ↔ P.C. Sat) & (C.O. Eval ↔ P.C. Sat)

and are summarized n the following

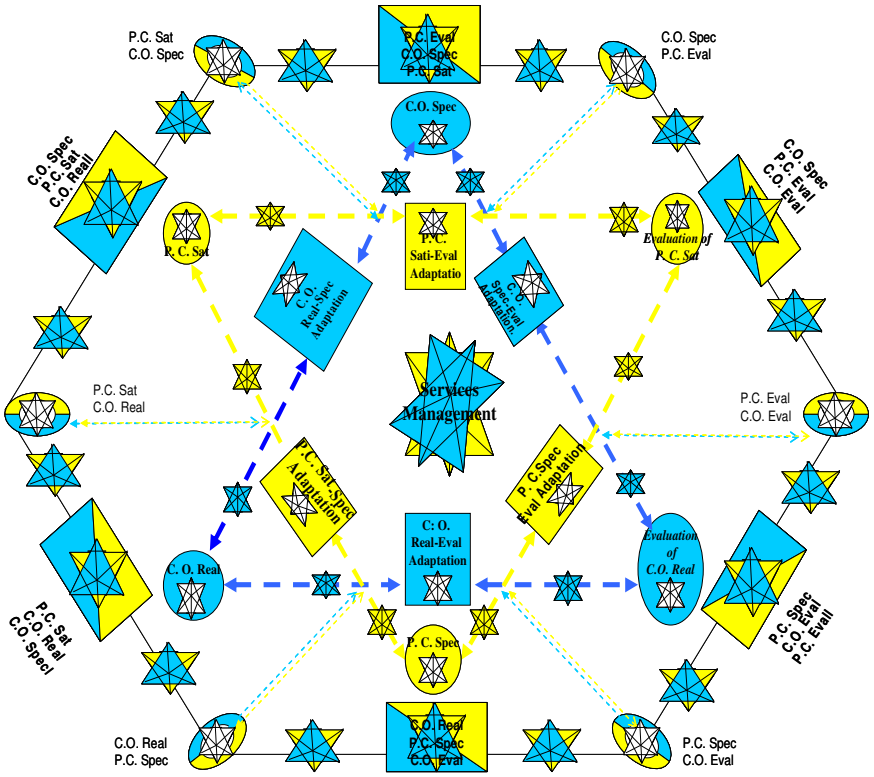


Fig. 8. Global Dynamics and Management

6 Conclusions

The proposed modeling approach to the integration and management of adaptive business processes is being experimentally tested with respect to the organization framework of the Italian National Research Council based on demand and supply interaction mechanisms for planning, execution and evaluation of research activities.

References

1. Mascari, J.-F.: Complex Autonomic Knowledge Systems. 1rst International Conference on Semantics, Knowledge and Grid (SKG2005) Beijing, China – (to appear)

2. Singh M. P. and Huhns M. N., Service Oriented Computing . John Wiley & Sons Ltd (2005)
3. Zhuge, H., The Knowledge Grid. World Scientific (2004)

Acknowledgment

Thanks to Barbara Dragoni, Paolo De Gasperis, Angelo Guerrini, Maurizio Lancia, Raffaele Pelliccia, Alberto Salvati and Maurizio Vitale for their collaboration, support and interest in the application of the proposed modeling method to the specification and design of a service oriented architecture supporting the new CNR organization framework.

Towards the Autonomic Composition of Business Processes

Pierluigi Lucchese[1], Marco Pistore[2], Michele Trainotti[1], and Paolo Traverso[1]

[1] ITC-irst, Via Sommarive 18, I-38050, Trento, Italy
[2] Dept. of Information and Communication Technology, University of Trento
Via Sommarive 14, I-38050, Trento, Italy
[1]{traverso, mtrainotti, lucchese}@itc.it
[2] pistore@dit.unitn.it

Abstract. One of the main ideas of service oriented applications is to abstract away the logic at the *"business level"* from its non-business related aspects, which we call the application at the *"system level"*, e.g., the implementation of transaction, security, and reliability policies. This abstraction should make easier and effective the composition of distributed business processes. However, the provision of automated composition techniques, which make this potential advantage real, is still an open problem. We address this challenge by proposing the idea of *Autonomic Composition of Distributed Business Processes* that exploits the business and system level separation in service oriented development. According to this view, composition at the business level poses the requirements and the boundaries for the automatic and autonomic composition at the system level. While the former is supported by user-centered and highly interactive techniques, the latter is fully automated and hidden to the humans. System level compositions are generated automatically and autonomically monitored and self-modified and adapted. In this paper we provide a first step towards this vision. We describe an architecture for the execution of distributed business processes that clearly separates the two levels, and envisage techniques for the automated generation, the automated monitoring and self-modification and adaptation of the system level. We describe the proposed architecture and the devised autonomic composition techniques in the case of an e-banking application, where both the business level as well as system level aspects, like transactions and security, are extremely important.

Introduction

One of the main ideas of service oriented applications is to abstract away the logic at the *"business level"* from its "non-business related aspects", what in this paper we call the application at the *"system level"*, e.g., the implementation of transaction, security, and reliability policies. This abstraction and this two-level separation should make easier and effective the composition of distributed business processes, thus enabling the interoperability among business processes that are distributed within and across organizational borders. The business level should define platform-independent and system/architecture-independent business processes. It should consist, for instance, of the definition of how a company interacts with clients and/or business part-

ners, the definition of the organization-dependent procedures and workflows that realize its business needs and strategies. The system level should instead implement the business processes taking into account requirements such as transaction, security, presentation, and reliability policies. As an example, consider an e-banking application that allows for an electronic payment of products available on-line. The business level should define the flow of interactions between the client, the electronic store and the bank. It should define when and how the billing in a bank account can be processed, the fact that the operation selling a product can be completed only if the banking operation charging money in the bank account has been completed too. The architectural level should instead implement, e.g., the transactional policy that is defined at the business level and that allows us to commit and roll-back the selling process according to the payment process. It should as well implement the security policies by encrypting messages and using appropriate security key mechanisms.

The potential advantage of the "two-level view" of service oriented applications is straightforward: an effective, easy, low cost, time-to-market development, management and composition of distributed business processes that can cope with the dynamic evolution and required adaptation to changes in, e.g., business strategies and markets, customers and providers relationships, etc. Service oriented infrastructures and standards for the development of distributed business process provide the starting point to face this challenge. A set of emerging standard languages and techniques is available for the definition of the business level and of the system level, e.g., from BPEL [1] to WS-Transaction [8] and WS-Security [6]. However, so far, an automated composition of distributed business processes that actually exploits the two-level separation between the business and the system level is still far from being achieved: no support is provided that can allow for composing distributed processes at the business level in an easy way, e.g., by releasing the developer from caring about the implementation at the system level, e.g., the implementation of the transactional and security aspects. All of this still remains an open challenge.

We address this challenge by proposing the idea of the ***Autonomic Composition of Distributed Business Processes***. We envision a service-oriented development process where composition itself is done at two different levels of abstractions:

- **Business-level composition:** The business level of the composed service is decided by business analysts, who define the business logic that composes distributed processes. We claim that this composition must be *highly transparent and user centered,* i.e., it must be highly interactive and the human, the business analysts, must be "in the loop", since this is related with critical issues such as the company's business needs and strategies, the company's organizational assets, etc.. In terms of the vision of "Autonomic Computing" [5], this is the part "outside" of the autonomic functioning of the human nervous system, i.e., the part that corresponds to a conscious control by the human brain.
- **System level composition:** the business level composition poses the requirements and the constraints for the composition at the system level, which is instead performed automatically, by hiding to and releasing the human from the implementation details that realize the electronic business process, e.g., the implementation of transactions and security policies. According to the autonomic computing vision,

this is the autonomic non-conscious functioning of the composition, where the human is released from the effort and time consuming, as well as difficult interaction with the computer.

According to this vision, the business level composition, which is highly transparent and human centered, poses the boundaries, the requirements, and the constraints for the autonomic composition at the system level, the boundaries for the human-opaque and user-hidden self-composition, self management, self-adaptation, self-optimization, etc. of distributed business processes.

In this paper, we provide a first step towards the vision of the Autonomic Composition of Distributed Business Process:

- We devise a service-oriented architecture for the execution of distributed processes that clearly separates the business level from non-business related aspects, like the implementation of transactional, security, presentation and reliability aspects. A business process manager interacts with, e.g., a transaction and a security manager that inject in the implementation layer the messages for managing the atomicity of operations, their commit and rollback, as well as the encryption of messages.
- Given a set of available business processes and the business level of the composed process, we devise techniques for the automatic generation of the system level composed service. The business level constitute the composition requirements for the non-business related composition.
- We provide techniques that support the monitoring of the composed business process, including its business as well as non-business level. For instance, we can monitor that process interactions actually run as expected, as well as that transactions requirements are satisfied. Monitoring at the business level provides information to business analysts, information that can reveal problems or changes in markets and business needs. Business analysts can use this information to re-define the business process reflecting the market evolutions. The result of monitoring at the system level can instead result in an autonomic automatic self-modification and adaptation of the system level itself.

We describe the proposed architecture and the devised autonomic composition techniques in the case of an e-banking application, where, beyond the business level, architectural aspects such as transactions management and security issues are extremely critical and important, thus motivating the approach to autonomic composition.

The paper is organized as follows. We first describe the e-banking application. We then describe the model for the components and composed business processes at the conceptual and architectural level. We finally provide an high level description of the techniques for autonomic compositions.

Problem Scenario: the integration of bank and store processes

In this section we review the case of a real e-banking application we have been involved in the adoption of the ideas proposed in this paper. In these kinds of applications, the regulatory pressure (e.g.: Basilea2 regulation), the need for transparency towards customers as well as for security and privacy pose critical system level requirements, which are as important as those at the business level. The result is that in many applications we find that the code implementing the business logic is "mixed" with system level implementation code.

The application under consideration is a service for on-line payment offered by a bank for an electronic store. The e-store publishes its catalog and a customer can select items from the catalog. When the customer decides to purchase the selected items, he/she is connected with an e-bank service that verifies the identity and the credit of the customer, charges the amount to his/her credit card or possibly to his/her account (if he/she has one account with the bank) and then credits the value to the current account of the e-store (see Figure 1).

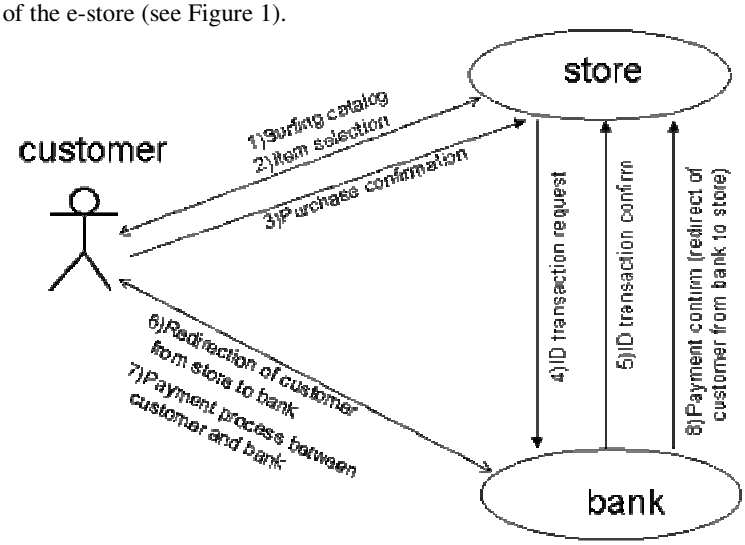


Fig. 1. High level interaction schema between customer, store and bank

The application must guarantee:

- The *privacy* of each actor involved in the process. The customer banking data – e.g., its credit card number – should not be visible to the shop. Moreover, the item purchased by the customer should not be visible to the bank. Finally, the data exchanged between the store and the bank should not be visible to the customer.
- The *safety* of the process: the bank is a kind of broker between the customer and the shop. The shop is guaranteed by the bank about the customer identity and accountability, the customer is guaranteed by the bank about the shop trustworthiness.

The previous service oriented application for this case study, was developed as a packaged solution, where the business process integration was technically achieved with a set of java files and libraries that were installed and configured in the web application server of the shop. A configuration module allowed for the connection with the e-banking application, and a start-up module managed the operation of creation, authentication and update of the security mechanism. A set of procedures had to developed by the store in its application server in order to integrate with the e-banking process (see Figure 2).

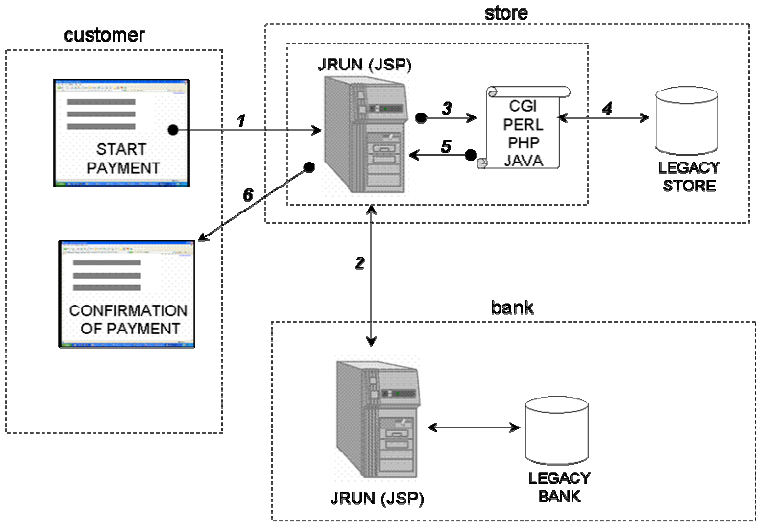


Fig. 2. High level architecture of the existing system.

The store had to deploy in its environment an application server where the JSP pages had to run with the java code that managed the process of payment between the customer, the store and the bank. This java code contained also the code that managed the *"system level"* of the application.

This kind of solution has some problems, since it is invasive (the shop has to adopt the technology chosen by the bank), a development effort is required to the shop, and finally the application is not flexible and easy adaptable to any change of the process. Having hard-coded the logic of the process in a java program, it is not easy to adapt the process to any possible evolution of the communication protocol between the bank and the store. For instance, if the bank would like to extend the process to deal with the possibility of subscribing an insurance related with the purchase, the application must be re-written.

In order to address these problems, we propose a new architecture, based on the adoption BPEL. As explained in detail in the rest of the paper, our solution implements the idea of the separation between the business and the system level. The business logic implementing the payment is executed by a BPEL engine that interacts with different managers for the "non-business related aspects".

Two level view of a service

The adoption of the new architecture described in this paper requires a modular description of each service involved in the composition. A central role in this modular description is played by the **Business Logic** of the service, which describes how one has to interact with the service in order to exploit it. This description is at the "business level" in the sense that we describe only those interactions that are relevant for a logical (or business) point of view, and we rule out the additional interactions that occur in the actual executions, but that are relevant only for managing "system level" aspects such as security and reliability.

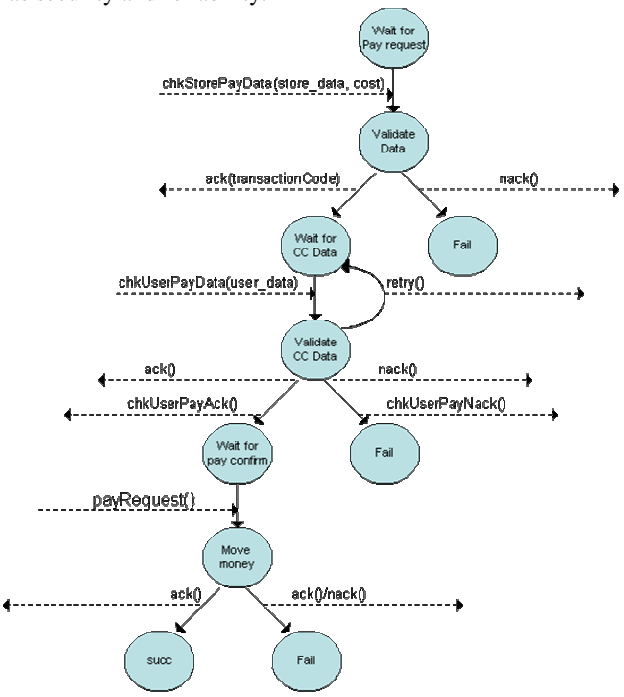


Fig. 3. The business logic of the bank

For instance, the business logic associated to the bank is represented in Figure 3. According to it, in order to execute a payment, the bank requires first to receive the payment request, containing the amount and destination of the money transfer. The bank has then to execute the internal checks in order to verify the validity of the received payment data. If the request is valid, the bank sends an acknowledge message and waits for the credit card data (or bank account coordinates) where to charge the payment. Again the bank checks the credit card data validity and answer with an acknowledge. Finally it needs to receive a confirmation message after which the bank triggers the money transfer and notifies the operation result.

Among the different languages that can be used to express the business logic, we adopted BPEL [1]. BPEL provides a core of process description concepts that allow for the definition of business processes interactions. This core of concepts is used both for defining the executable process implementing a given service and, what is more relevant here, for describing and making available to potential partners the interaction protocol. In particular, abstract BPEL specifications (i.e., BPEL specification from which the details of the internal operations of the service are omitted) can be used to describe protocols such as the one presented in Figure 3.

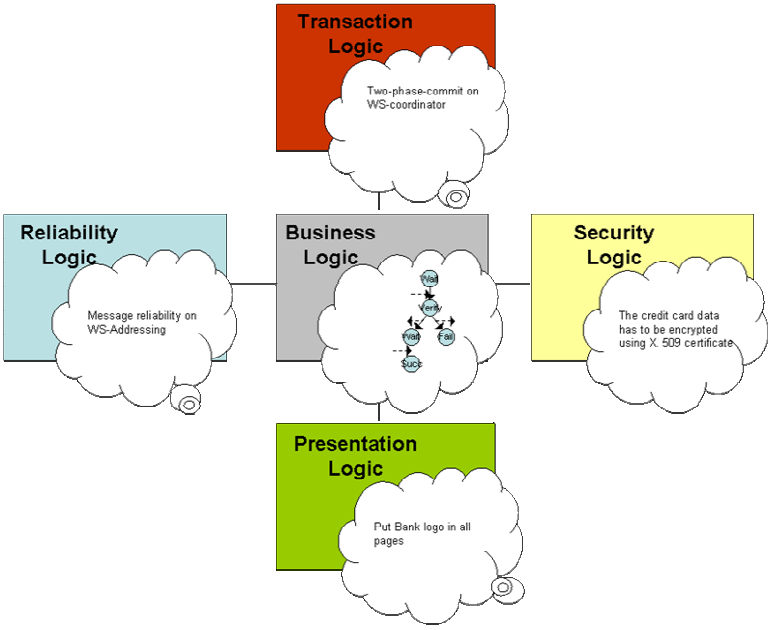


Fig. 4. Two level view of a service

The business logic of a service is complemented by the definition of a set of other "logics". These describe additional aspects that need to be addressed in order to exploit the service in a composition. Clearly, there are several of these additional aspects, and they may vary from domain to domain. In our e-banking domain, we identified four such logics that are necessary to complete the business logic of Figure 3: the security logic, the reliability logic, the transaction logic, and the presentation logic. In the following, we give a short description of these four logics (see also Figure 4).

The **security logic** describes the policy to provide quality of protection through message integrity, message confidentiality, and single message authentication. The security logic also provides a general-purpose mechanism for associating security tokens with messages. Additionally, the security logic could describe how to encode security tokens, e.g., it can specify to use the WS-Security standard and X.509 certificates.

The security logic can also specify some specific requirements on parts of messages (e.g., the bank coordinates and the credit card numbers) that need to be encrypted in every communication.

The **transaction logic** covers the aspects related to the transactionality of certain operations. In the transaction logic, the bank describes standards (e.g., WS-Transaction [8]), tools (e.g., Choreology [4]) and methods (e.g., two face commitment) that are required or supported in the management of transactional activities.

Similarly to the security and the transaction logic, the **reliability logic** defines the policy and standards (e.g., WS-Addressing) used in order to make the message delivery reliable.

The **presentation logic** describes how the back should appear to the end-user interacting with it through the web or through other channels. Elements defining the visual identity of the bank (colours, logos,…), as well as active web objects that are used in the definition of the web pages prepared for the user who interacts with the bank are defined here.

Composition of business processes

The modular definition of the business processes described in the previous section is exploited when several such business processes are composed in applications such as the on-line payment of an e-store described in Figure 1. This composition is done in two steps, first of all at the "business level", and then at the "system level".
The **business level composition** has the objective of defining the shared, composed business logic of the distributed business process. Several approaches, also from our group [9], propose to make this composition fully automatic. However, in several domains, such as our e-banking domain, an automated composition is not realistic: the business level composition has to be performed by system analysts, in many cases through a negotiation process, since it is related with critical issues such as the company's business needs and strategies, the company's organizational assets, etc.. Tools can offer support to the analysts, but we claim that this support will be effective only if it is *highly transparent and user centered,* the analysts will be able to control the composition process and to analyze and modify the outcome.
The most important outcome of the business level composition is a **choreography** defining, at the business level, how the different partners are supposed to interact in order to carry out the distributed business process. In the case of our e-banking domain, such choreography is defined in Figure 5. According to it, the user starts the interaction sending to the store a checkout message confirming the intention to buy the selected items. Then the store sends to the bank the payment data (store's account data and order cost). The bank waits for the credit card data from the user and checks its validity, After being notified that the user is able to pay the order, the store confirms the payment to the bank, finalizes the order (e.g., forwarding it to the internal warehouse department), and finally notifies the user.

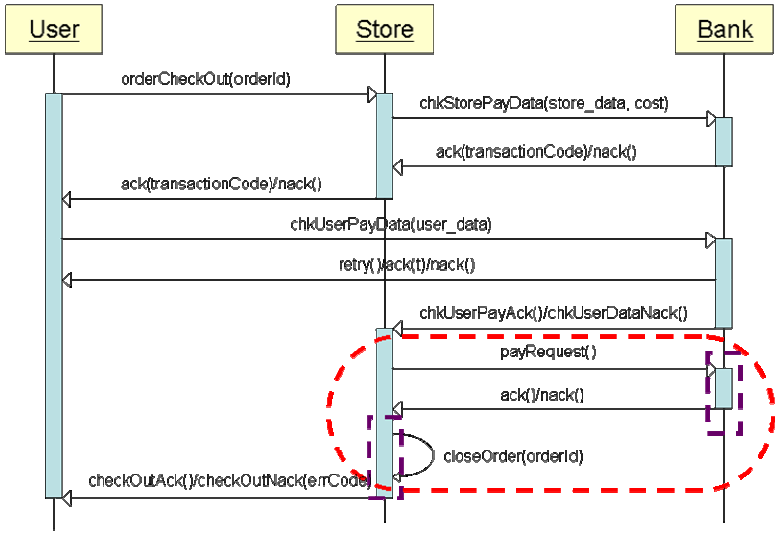


Fig. 5. The business level of the e-banking application

We remark that this choreography respects, and somehow implements, the business logic of the back shown in Figure 3. Indeed, all the data and interactions required by the bank in order to perform a payment are fulfilled either by the shop (e.g., payment data) or directly by the user of the e-banking application (e.g., credit card data).
Besides defining the interaction protocol, the choreography can express additional requirements identified during the negotiation. For instance, the choreography in Figure 5 identifies two activities, the payment of the bank and the finalization of an order of the store (both bordered by the dashed rectangles) that need to be executed in a transactional context (represented by a dashed oval). The transaction is here required because either both or none of them have to succeed. We remark that, in the business level choreography, also such additional requirements are given at a "business" level, without defining how such transactional behavior has to be implemented in the system.

The shared choreography defined in the business level composition defines the requirements and the constraints for the second phase of the composition, namely the **system level composition**. The objective here is to define the systems to de deployed and executed by the participants in order to execute the distributed business process. In the case of our e-banking application, this corresponds, for instance, to define system components such as executable BPEL processes to be deployed by the bank and by the store, as well as the servlets for the web interactions with the end user. The definition of these systems has to implement the choreography, and has to take into account the "system level" specifications given by the different partners.
Consider for instance the transactionality requirement in Figure 5. Its implementation requires the exchange of additional messages between the bank and the store, in order to share a transaction context, and to manage the start and the finalization of the

transaction. The implementation of these messages depends on the mechanisms and standards supported by the participants to the transaction: e.g., if both bank and store use transaction engines such as choreology, then the management of the transaction is delegated to these engines; if such engines are not available, then the transaction has to be managed explicitly, e.g., by extending the executable BPEL processes with the extra messages necessary to set up and exploit a transaction context. In Figure 6, we see a fragment of the executable processes of bank and store extended in order to implement the transaction among money transfer in the bank and finalization of the order in the store. The implementation of such requirement requires the introduction of a new participant to the interaction, the coordinator of the transaction, and the introduction of new communications between store and bank and with the coordinator in order to manage the transaction (these additional communications are represented by the dashed arrows in Figure 6).

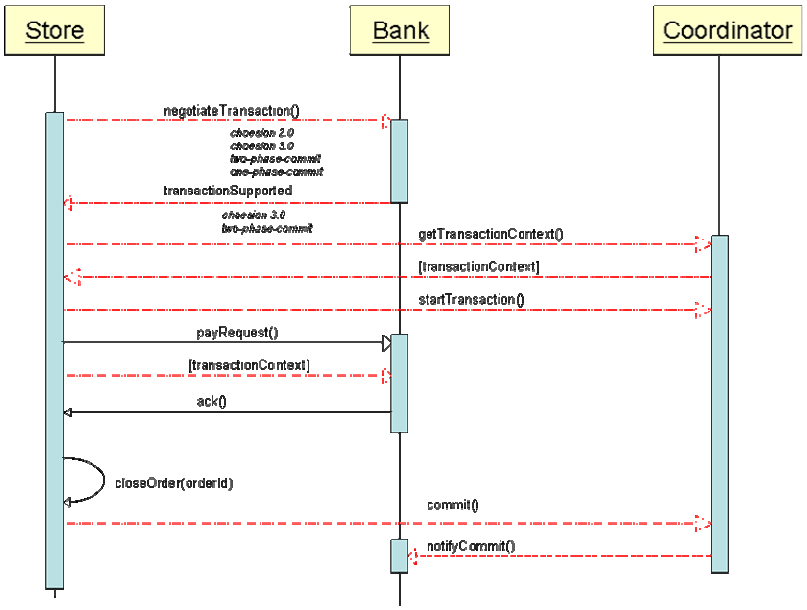


Fig. 6. A fragment of executable processes enriched with transaction management

Our claim is that, differently from the business level composition, the system level composition can be performed automatically, by hiding to and releasing the human from the implementation details that realize the electronic business process. This of course requires advanced techniques: adding automatically the additional communications for implementing the transactions as shown in Figure 6, requires sophisticated techniques for automated program synthesis and adaptation. Our preliminary experiments show that automated composition techniques such as those described in [9], while not adequate for a fully automated business level composition, are successful at

the system level for automatically synthesizing the code implementing system level features such as transaction and reliability requirements.

System Architecture

In this section we present a possible architecture for supporting the execution of composite business processes defined as in the previous section. A high level representation of the architecture is given in Figure 7.

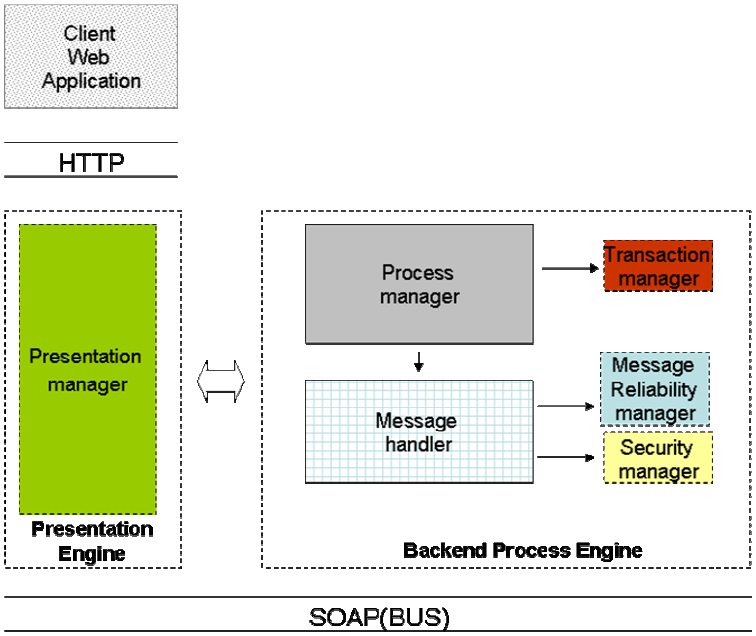


Fig. 7. Architecture

The two main modules of this architecture are: the **Backend Process Engine**, which is responsible of executing the (BPEL) processes implementing the business logics as well as the transaction, security and reliability logics; and the **Presentation Engine**, which is responsible of providing the web pages to the end user and of implementing the presentation logic. These two modules correspond to two different busses that are exploited in the execution of the distributed business processes: the **Application Bus**, which transports the (SOAP) messages between process engines, and the **HTTP Bus**, that is used to provide the web pages to the end user.

The two key components of the backend process engine are the **Message Handler**, responsible of managing the communications and invocations among web services,

and the **Process Manager**, responsible of executing the distributed business processes. Examples of existing tools implementing these two modules are, respectively, the Apache Axis server and BPEL engines such as Oracle BPEL Process Manager [3] or Active BPEL [2]. The other modules of the backend process engine are support these two main components offering additional services. More precisely, the **Security** and the **Message Reliability Managers** are linked to the message handler; such modules can be implemented as Axis plug-ins. The **Transaction Manager** is linked instead to the process manager and supports it in the implementation of the transaction logic. An example of a tool that can be exploited as transaction manager is Choreology [4].

The main goal of the presentation engine is to mediate the interaction of the user with the backend process engine and to synchronize the operations performed by the user through web pages with the processes executed by the backend process engine. Standard tools such as Tomcat can be used in the presentation engine.

Of course, the integration of all these engines and modules in order to implement a seamless execution of the distributed business processes is far from trivial: just to make an example, a seamless management of the transaction requires a strong coupling between the processes executed on BPEL engines and transaction managers such as cohesion. The automated generation techniques we foresee for the system level composition are a key technology for supporting such an integration.

Towards Autonomic Composition

Several business process integration domains are highly dynamic and evolving environments. Service oriented applications have to cope with the dynamic evolution and required adaptation to changes in, e.g., business strategies and markets, customers and providers relationships, etc. External partners can autonomously change the way they interact with each other. This kind of evolution has to be tracked and monitored at *"run time",* i.e., during the execution of a business process. Monitoring is indeed the key task that can reveal a change, e.g., in interactions with other partners, a service oriented application has to cope with.
Monitoring can reveal evolutions that require adaptation either at the system or at the business level:

- **Monitoring at the system level.** Monitoring at the system level can track unexpected changes in the implementation of, e.g. the transaction, security, presentation, as well as reliability mechanisms. The result of monitoring at the system level can activate an autonomic self-modification and adaptation of the system level itself. For instance, monitoring can reveal that the system level implementation of the transaction or security mechanisms do not work properly, e.g., since the transaction/security managers of two organizations do not exchange messages as expected. This can be due, e.g., to a changes in the transaction/security manager policies of an external organization.

- **Monitoring at the business level.** Monitoring at the business level can reveal changes in business processes, or situations that require a rethinking of the business strategies. It provides information to business analysts, information that can reveal problems or changes in markets and business needs. Business analysts can use this information to re-define the business process reflecting the market evolutions. For instance, in our e-banking application we can monitor whether business process interactions actually run as expected. A monitoring module at the bank could for instance reveal that the store does not send the right information at the right moment to the bank, e.g., the payment request. As a further example, the monitor could reveal that the client in several occasions does not complete the payment procedure, revealing a serious problem in the perception of trust of the system. Strategic actions must be devised in order to cope with these kinds of problems.

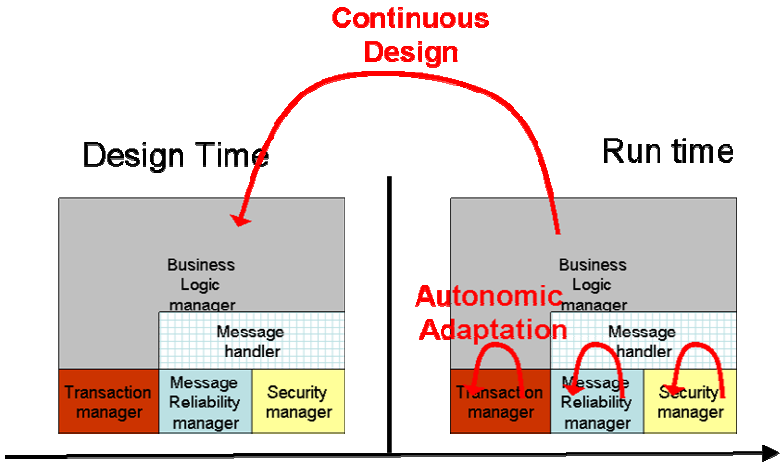


Fig. 8. Continuous Design and Autonomic Adaptation

While monitoring at the system level can be coped with an autonomic computing mechanism, which is opaque to humans and allows for self-adaptation and self-repairing, changes revealed by monitoring at the business level require highly transparent and human centered support. While the former is a process that can be completed autonomically at run time, the latter must return control to the business analyst at the design phase (see Figure 8).

In this way, we envision an environment for business process modeling where the designer can defer to the run-time decisions on how to carry out a given process; conversely, the run-time environment can detect failures in performing the business process, for instance due to unforeseen changes in the business domain, and can trigger a redesign phase. During redesign, the overall model of the business application is updated, in order to reflect the changes in the domain (or in the strategic goals of the partner); decisions and procedures may now be fixed by the designer and, conversely, design decisions may be relaxed in order to give more flexibility to the run-time. In this framework, a redesign is not destructive with respect to the run-time. Rather, we

envision *a "continuous design" environment,* where the human driven re-design and the supporting automated techniques concur to the achievement of the strategic goals in a service oriented world.

Conclusions

In this paper, we have described an architecture for the execution of distributed business processes that clearly separates the business logic from the system level implementations of functionalities like transaction and security management. We have discussed techniques for the automated generation and the automated monitoring and self-modification and adaptation of the system level. We have explained these ideas in the case of an e-banking application we have been involved in. We see this work as a first step towards a vision of autonomic business processes composition where the business logic, the goals and the strategic requirements, represent the boundaries for autonomic self-adaptation.

Acknowledgements

The authors would like to thank Fausto Giunchiglia for the interesting discussions and suggestions on Autonomic Business Process Composition. We would also like to thank the ASTRO lab (http://www.astroproject.org/) for their collaboration and their feedback.

References

1 T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. Business Process Execution Language for Web Services (Version 1.1), 2003.
2 http://www.active-endpoints.com/products/activewebflow/active_bpel/
3 http://www.oracle.com/technology/products/ias/bpel
4 http://www.choreology.com/
5 J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. Computer magazine, January 2003.
6 Web Services Security (WS-Security), IBM, Microsoft & VeriSign.
7 Web Services Coordination (WS-Coordination), Microsoft, BEA & IBM.
8 Web Services Transaction (WS-Transaction), BEA, IBM &Microsoft.
9 Supporting the Composition of Distributed Business Processes: http://www.astroproject.org.

A Development Methodology for Improving Cost Estimates in Process Automation Projects

Doug Tidwell

IBM Corporation, University Relations, 4400 Silicon Drive,
Research Triangle Park, NC 27709 USA
dtidwell@us.ibm.com

Estimating the cost of a software project has long been more of an art than a science. In this paper, we present a methodology that relies on the knowledge of business experts and the power of process simulation tools to create much more accurate cost estimates. With this methodology, business experts (typically non-programmers) can model and simulate business processes to create accurate, detailed estimates of the benefits of the updated process. The business experts deliver a complete, implementable process model to the IT department. The IT department can then create a detailed assessment of the exact changes needed for the updated process. The combination of the benefits analysis from the business experts and the cost analysis from the system developers give decision makers more insight than ever before. We close with a short list of best practices for managing process automation projects.

An Overview of our Methodology

Our methodology addresses two distinct audiences involved in process automation projects: Business experts and developers. Our assumption is that business experts have rudimentary programming skills, if any, and that developers' knowledge of business processes is limited to their impact on IT systems. In describing our methodology for improving cost estimates, we will consider the particular skills and perspectives of both groups.

We refer to business experts as people who understand how goods and cash flow through a business process. These experts have a high-level understanding of how goods and services are delivered to customers, how those customers are billed, how goods are delivered from suppliers, how suppliers are paid, and so forth.

Developers are programming experts who understand how business systems work, including network topologies, data formats and database schemas, security, audit trails, and other details of the actual system that implements the business process.

The Business Experts' Perspective

The first step in estimating the cost of a process automation process should be driven by the business experts. The business experts likely have the most knowledge of

"low-hanging fruit," the obvious process inefficiencies that are the normal starting point for automation projects. In addition, the business experts are aware of marketplace opportunities that could deliver significant competitive advantages if processes were automated correctly.

An important part of our methodology is that *business experts should be the only stakeholders involved in the process automation design phase*. Developers involved in the discussion are likely to point out impediments to developing process improvements. "We could do that, but we'd have to update the database schema for the payroll system" might indeed be a reason to reject a process improvement, but that decision should not be made until our methodology has run its course.

In the design phase, the business experts should focus on their ideal system, disregarding practical considerations to some extent. There will always be process changes that are completely impractical and will never be implemented; those of course should not be used in any design. However, the business experts are encouraged to "think outside the box" as much as possible. Our methodology uses process modeling to create the most accurate estimate of the benefits of the proposed changes.

The business experts' perspective includes the process steps that involve the most pain to the organization. Does the current process involve human interactions that could be implemented with business rules in many cases? Does the current process take into account the expiry date of inventory when selecting stock from different warehouses? How does the current process balance the possibly competing goals of faster order processing and reduced shipping costs? The business experts have the knowledge and experience to address these questions.

Our Sample Business Process

The business process we will use as an example is based on the Supply Chain Management sample application developed by ws-i.org. (See http://ws-i.org/ for more information.) Our sample process is concerned with processing purchase orders that arrive from our customers. This is certainly not an enterprise-ready application or a real-world design, but it has enough complexity to illustrate our methodology. Here is a high-level description of the process:

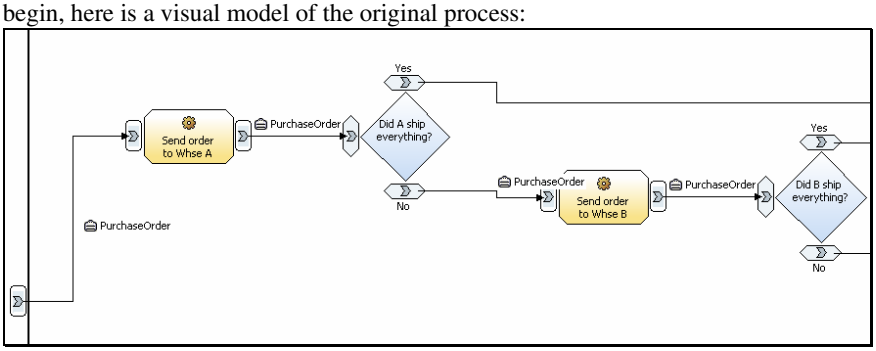
- A purchase order is received. It contains a customer number and the requested quantities of some number of items. For each item ordered, the purchase order has the quantity and the product number.
- The purchase order is sent to the first warehouse. That warehouse ships as many of the requested items as possible. What the warehouse system returns is an updated purchase order in which the quantities of all shipped items are set to zero.
- The updated purchase order is examined. If there are any more items to ship (at least one item has a quantity greater than zero), the order is sent to the second warehouse.
- The second warehouse processes the order just as the first warehouse does. It returns an updated purchase order.
- The updated purchase order is examined. As before, if there are any items left to ship, the purchase order is sent to the third warehouse.

– The third warehouse processes the order just as the first two warehouses.
– The updated purchase order is examined. If there are any items left to ship, the unshipped items are stored are logged.

Looking at the process as a whole, the input to the process is a purchase order and the output from the process is a purchase order. Similarly, the input to a particular warehouse is a purchase order and the output from a warehouse is an updated purchase order. For the purposes of our example, this symmetry greatly simplifies process reuse. This aspect of our example allows us to demonstrate our methodology more fully.

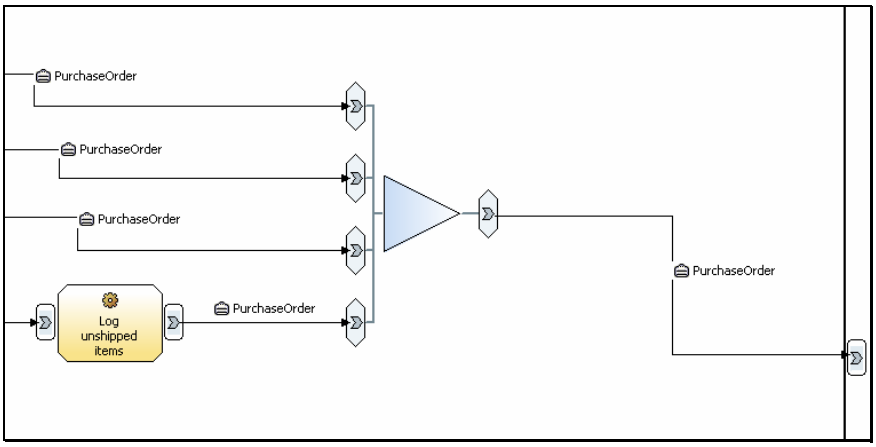
Business Process Modeling

The heart of our methodology is a comprehensive business process modeling tool. The example we use here is IBM's WebSphere Business Integration Modeler product, although any tool with similar modeling and simulation techniques should work. To begin, here is a visual model of the original process:



This screen capture shows the purchase order being sent to the first two warehouses. The PurchaseOrder icon on the arrows indicates that the PurchaseOrder data item is being sent from one node of the process to another.

At the end of the process, we merge all the branches of the process and return the updated PurchaseOrder:



One of the four branches comes from the "Log unshipped items" task, while the other three branches represent empty purchase orders returned from each warehouse.

The tool's ability to diagram a business process is a nice feature, but its real value lies in its process simulation capabilities. Business experts can define the resources used to perform each task. In our example, the three warehouses are resources, as is the logging facility. Business experts can define data items as well; a purchase order is defined in terms of the data it contains (customer number, items, etc.).

The modeling tool allows business experts to build a significant amount of logic into the process. For example, a decision node can include logic to branch in one direction or the other if the number of items in the purchase order is zero or not. Although this verges on programming, it is important to define the branches through the process for any process simulations.

Creating a Process Baseline

It is important for the business experts to model a process baseline that represents the current state of the process. The more accurate the baseline process, the more accurate the resulting cost estimates will be. This is primarily a manual task, although there are network monitoring tools that can capture data useful for the baseline model. It is especially important that the bottlenecks and inefficiencies in the existing process be modeled accurately, for those are likely to be the areas of focus as process improvements are discussed. The experience of the business experts is invaluable in constructing the model of the existing process because they are most familiar with those trouble spots.

Running simulations of the baseline process and comparing the results of the simulation to the results of the real-world process is crucial. As the business experts develop enhancements to the business process, the simulations of the new process will be compared to the results from the existing process. If the model of the existing process is flawed, the resulting estimates of time or cost savings from process changes will be flawed as well.

Simulating the Business Process

After the business process has been defined, the next step for the business experts is to simulate the process. The modeling tool we use has many sophisticated simulation techniques, including the ability to define the statistical distributions of events, the ability to define when resources are available, and the ability to different work schedules for the humans involved in the process.

Our simulation tool can define many different result sets for different versions of the business process. This allows the business experts to compare the versions of the business process and decide which version is preferable. As the simulations are more detailed, chances are there will be several different versions of the process that are viable. The business experts can decide which version they prefer based on the needs of the business. For example, one version of the process might be best when orders are above a certain threshold. Another version might be best if the company's focus shifts from keeping shipping costs low to shipping orders as soon as possible. A third version might be best if labor costs at a particular warehouse rise above a certain level.

The Business Experts' Deliverables

Based on the preferred business process model(s), the business experts deliver their design to the development shop. Our simulation tool delivers those designs in three important formats, all of which can be used directly by the IT shop:

- XML schemas that describes the data items used by the process
- A BPEL (Business Process Execution Language) document that defines the steps in the process and the logic for moving from one step to another
- A Web service interface that describes how to invoke the business process.

The formats of the three files exported by the WebSphere modeling tool are both standards-compliant and directly usable inside development tools. This is a significant step forward in getting business experts and developers to work together. It can be argued that all of the effort that has created the Unified Modeling Language (UML) is an attempt to define a non-ambiguous way for business experts and developers to communicate. These standards-based files automate this communication even further. We'll discuss these three files in more detail here.

XML Schemas

The XML schemas are generated from the definitions of business data items. Our modeling and simulation tool helps business experts define business items such as purchase orders without programming skills. When those items are exported, they are converted into XML schemas. Developers can use their tools of choice to convert the XML schemas into Java beans, .Net objects, or similar structures in other programming languages. Most significantly, the XML schemas are used directly by the developers in their implementation. No human interpretation is necessary; the XML

schema is not a high-level description or diagram that must be read and correctly understood by the developers before the project can move forward. The data item goes directly from its expression in the modeling tool to an XML schema, and from there into an object or data structure in the programmers' language of choice.

The BPEL document

The BPEL document is used directly in the implementation phase as well. The BPEL file defines all the steps in the business process, all of the data managed by the process, the logic for moving from one state to another, and descriptions of the services used in the process. A developers' job is to take the BPEL definition of the process and link the process steps to actual services in the business environment. In our sample application, for example, we have three warehouses that perform the "Process purchase order" step in the process. In implementing the process as defined in the BPEL file, the step that sends a purchase order to Warehouse A would be linked to the (Web) service provided by Warehouse A. Finding the necessary services to implement the process is a crucial part of the cost estimate; more on this soon.

The Web service interface

The final artifact of the business experts' model is a Web service interface. This interface is defined in a WSDL (Web Services Description Language) file. The description includes all of the methods provided by the service, the XML format of the data passed to the service, and the XML format of the data returned from the service. The Web service interface defines how to access the business process itself as a Web service. A side effect of implementing the process is that the business can then invoke the process from any Web services client. The universal access provided by Web services means the process can be started (or, if the interface allows it, monitored or controlled) from any device or protocol needed.

The Developers' Perspective

In our methodology, developers get involved after the business experts have finished designing the new process. Their first task is to assess the BPEL file to estimate the costs of implementing the new process. Our methodology makes this estimate more straightforward because the BPEL file specifies the services only, not their implementation. At a high level, estimating the costs of implementation involves determining the services required for the new process. The focus of this effort is to find how many of those services exist and sizing the effort to create the services that do not.

The use of BPEL has two distinct advantages. First, it merely defines the interfaces between the process steps and their implementations. In our example, each of our three warehouses might be completely different, using different operating systems, database vendors, database schemas, and so forth. As long as each warehouse supports the interface we need, those differences are irrelevant.

Second, the BPEL document defines the areas developers need to focus on in their cost estimates. For a given step in a BPEL process, it should be obvious which systems are affected. In a simple world, all of those systems are controlled by the enterprise; in the real world, many of those systems will be controlled by suppliers or partners. Regardless, the focus provided by the BPEL process helps developers specify exactly what needs to be modified.

It is possible, though extremely unlikely, that developers will find that all of the services needed by the BPEL process are available. In our existing process, warehouses are capable of processing purchase orders. Assuming all of the warehouses support the interface defined in the BPEL file, implementing the process steps that involve sending a purchase order to a warehouse is extremely simple.

Assessing a Process Enhancement

In our sample application, the business experts determine that a change to the process would result in significantly reduced shipping costs. The original process sent the purchase order to Warehouse A, then Warehouse B, then Warehouse C, until the entire order was processed. The business experts proposed that each warehouse should first be queried to see if any warehouse can ship the entire order. Modeling and simulating the changed process indicates that shipping costs can be reduced by 23%. In addition, the simulation results indicate that shipments will reach customers an average of .38 days faster. Although the shorter delivery time does not apply to costs directly, business experts can estimate how that improved service will help the business in the field.

Continuing with our sample, when the business experts deliver the enhanced process to the developers, the developers quickly realize that the warehouses are not equipped to examine a purchase order without actually shipping it. Their cost estimates are then based on the amount of work required to change the business systems at each warehouse.

The ultimate goal of our methodology is for business executives to make unemotional business decisions based on accurate cost and benefit estimates. This is the end point of our sample application. After the business experts and developers have finished their assessments, the executives are presented with the following information:

- Implementing the new process will cut shipping costs by 23%.
- Implementing the new process will take 20 person-months of design, development, testing, and deployment, and can be completed in as little as 5 months.
- Implementing the new process will deliver customer orders an average of .38 days faster.

In addition to this kind of information, developers can assess the risks of changing the existing warehouse systems. If those systems are implemented in modern programming languages and designed intelligently, the risk of modifying the existing systems will probably be low. On the other hand, if those systems are implemented in COBOL code, much of which was written decades ago, the risk of modifying those systems could be quite high. It is possible, of course, that the risks involved outweigh any possible benefits.

Executives will also have to consider other, more intangible factors. If the sales force believes the faster shipping times will give the company a considerable competitive advantage, executives will likely be willing to take more risks and absorb more costs. The cost/benefit analysis for this particular process might not be particularly attractive, but if the costs are viewed as part of a company-wide reengineering effort, executives might be willing to approve the enhancements anyway.

Finally, given the many factors involved, it is possible that executives will ask for a less ambitious process enhancement. In our example, what if the business systems at Warehouses A and B are modern and relatively straightforward to enhance, while changing the system at Warehouse C is too risky? The business experts could go back to their model and run more simulations to see if the process enhancements would still make sense. The business experts might be asked to come back with a less ambitious process enhancement with lower costs or faster implementation times. The developers might be asked to consider alternate development approaches (a Web services gateway, for example) with similar benefits. In both cases, the ability of modeling and simulation tools to provide accurate benefits analysis is extremely useful in the executives' final decision.

Summary

In this paper we have examined a simple application that illustrates some of the issues involved in business process automation. When our example process was modeled, the business experts were able to simulate the process to determine its potential benefits. Once the business experts were satisfied with the design of the proposed process automation, what they delivered to development was a set of files that can be used in the final implementation. Finally, the decision-makers can weigh the detailed benefits analysis delivered by the business experts against the detailed costs analysis delivered by the developers to make the most informed decision possible.

Author Index