Business Process Query Language a Way to Make Workflow Processes More Flexible[1,2]

Mariusz Momotko[*] and Kazimierz Subieta[+#]

[*)] Rodan Systems S.A., Warsaw, Poland
[+)] Institute of Computer Science PAS, Warsaw, Poland
[#)] Polish-Japanese Institute of Information Technology, Warsaw, Poland
{Mariusz.Momotko@rodan.pl, subieta@pjwstk.edu.pl}

Abstract

Many requirements for a business process depend on the workflow execution data which include common data for all the population of processes, state of resources, state of processes, etc. The natural way to specify and implement such requirements is to put them into the process definition. In order to do it, we need: (1) a generalised workflow metamodel that includes data on the workflow environment, process definitions, and process execution; (2) a powerful and flexible query language addressing the metamodel; (3) integration of a query language with a business process definition language. In this paper the mentioned workflow metamodel together with the business process query language BPQL is presented. BPQL is integrated with the XML Process Definition Language (XPDL) increasing significantly its expressiveness and flexibility. We also present practical results for application of the proposed language in the OfficeObjects® WorkFlow system.

1. Introduction

During the last decade workflow management systems (WfM systems) made a successful career. The WfM systems have been used for implementing various types of business processes. Despite many advantages resulted from application of such systems, there were also observed significant limitations. One of the major restrictions was assumption that business processes do not change too often during their execution. While such assumption may be satisfied for majority of production processes, for less rigid processes, such as administration ones, this is not true. Because of the nature of the latter processes, they need to adapt frequent changes in workflow environment (i.e. resources, data and applications) as well as workflow itself (e.g. the current workload of participants) [21], [1], [23].

An approach to increase processes adaptability is to make their definition more flexible. In this context flexible means that it is possible to express complex dynamic requirements that depend on process execution history as well as current organisational and application data (referred further to as relevant data). An alternative is manual control of processes and their resources at run time and this may be the only way for complex and unpredictable cases. However for many quite complex requirements, such as workflow participant assignments and transition conditions, this approach seems to be the closest to the real business process behaviour.

In order to express the mentioned requirements, we need to define an appropriate process metamodel, then to develop a language to query this model, and finally to integrate this language with a process definition language. The process metamodel has to be generic and include both process definition as well as process execution entities. So far, there is at least one widely known process definition metamodel proposed in [24] standard by Workflow Management Coalition (WfMC). While this metamodel is well defined, there is no standard process execution metamodel provided neither by WfMC standards nor by other process management body (e.g. BPMI). In addition, process execution models provided by WfM systems seem to be tool oriented and mainly focusing on entities implemented within a given system. Recently, there is some effort in defining such generalised process execution model (e.g. [11]). However, it does not include some process features proposed within the last workflow

[1] This work was supported by the European Commission project ICONS, IST-2001-32429
[2] Extended version of the paper accepted to the 8th East-European Conference on Advances in Databases and Information Systems ADBIS'2004.

research such as advanced time management [9] or flexible workflow participant assignment [14].

In order to make process definition more flexible, in the next step we need to develop a language to query the mentioned metamodel. This language should be able to express all possible queries on the metamodel, and should be readable and clear for process designers which are not necessarily software programmers.

In Section 2 we propose a generalised process metamodel as an extension of the WfMC's metamodel; the extension concerns entities related to process execution. In Section 3 on the basis of this metamodel we define a Business Process Query Language (BPQL). First we specify process definition elements where this language may be used and then define its syntax and semantics followed by some aspects of its pragmatics. In Section 4 we present integration of BPQL with XML Process Definition Language (XPDL). In Section 5 we present practical results of application the mentioned language in a commercial workflow management system that is OfficeObjects® WorkFlow. In Section 6 we discuss related work. Section 7 concludes.

2. Process Metamodel

In order to specify a workflow query language in the first stage an appropriate workflow process metamodel should be defined. It should represent two parts of 'workflow process puzzle': process definition and process execution. The former part is mainly used by workflow engines to execute workflow processes while the latter helps monitoring and analysing workflow process execution.

Since WfM systems are only a part of IT applications, there is a need to specify requirements for the workflow environment. From the workflow point of view such systems have three dimensions: processed data, provided services and registered resources that may execute the services operating on the data. A part of these data is used by WfM systems to control execution of workflow processes (i.e. flow conditions, and workflow participant assignment). WfM systems have rights only to read these data, not to modify them. In WfMC terms these data are referred to as *workflow relevant data*.

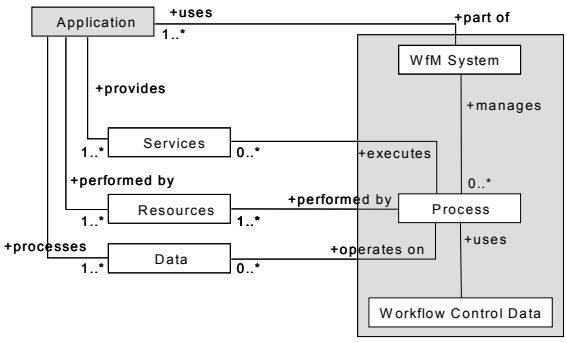

Figure -1 Workflow system as a part of IT application

Services provided by information systems may be used to express process activities. During execution of activities WfM systems call these services with appropriate parameters In the WfMC terminology these data are called *applications*. There are also resources that include users or automatic agents that may perform some activities within workflow processes.

Resources may be also selected using roles, groups or organizational units. A resource that may participate in process execution is called a *workflow participant*. In addition to the mentioned elements WfM systems use *workflow control data*. These data are managed only by WfM systems and store workflow specific information such as number of active process instances, international setting, etc.

The workflow process metamodel defines workflow entities, their relationships and basic attributes. It consists of two parts, namely a workflow process definition metamodel and a workflow process instance metamodel. The former part defines the top-level entities contained within process definition, their relationships and basic attributes. It is used to design and implement a computer representation of business processes. The latter part defines the top-level entities contained within process instantiation, their relationships and basic attributes. This metamodel is used to represent process execution that is done according to the process definition model. The workflow process metamodel also shows how individual process definition entities are instantiated during process execution.

The main entity of the definition metamodel is *process definition*. It provides basic information about the computer representation of a business process. For every process a set of *data container attributes* is defined. This attributes are used during process execution in the evaluation of conditional expressions, such as transition conditions or pre and post conditions. The set of container attributes (i.e. number, types, and names) depends on individual process definitions.

Process definition consists of activities. An *activity* defines a piece of work that forms one logical step within a process. There are three types of activities: atomic, route and compound ones. An *atomic activity* is a smallest logical step within the process that may not be further divided. For every atomic activity it is possible to define *who* will perform it, *how* it will be executed and *what* data will be processed. An atomic activity may be performed by one or more workflow participants. Basically, a workflow participant is a user (or their group), a role or an organizational unit. In addition, the system can be treated as a special workflow participant. Specification of workflow participants that may perform a given activity is called *workflow participant assignment*. The way how activity will be performed is specified by *application* which is executed on behalf of it.

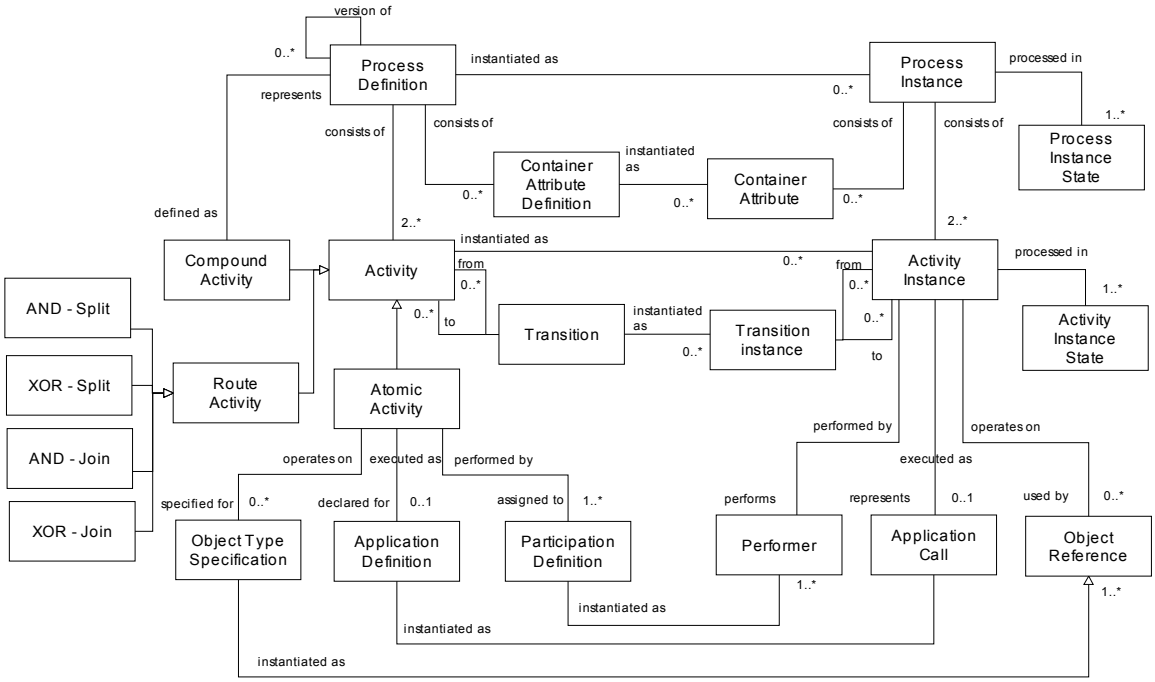

Figure -2 Process metamodel

Such specification also includes a set of parameters that will be passed to the application. Since the mentioned application operates on data, also object types that will be processed (i.e. created, modified, read or deleted) by this activity has to be defined. Object types may be considered as workflow relevant data that is a part of application data related to and processed by workflow processes.

The second type of activity is a compound activity. This type of activity represents a sub process and usually is defined to simplify process definition and make use of common activities that exist in many business processes. The last type of activity is a route activity. It is used to express control flow elements, namely split and join operations. For both split and join operations two basic control flow operators are defined: AND and XOR. On the basis of these operators it is possible to express more complex flow operations. A route activity is performed by the system. Since this type of activity is a skeletal one, it performs no work processing, neither object types nor application is associated with it.

The order of activities within the process is defined by transitions. A transition defines the relation between two activities. Transition from one activity to another one may be conditional (involving expressions which are evaluated to permit or inhibit the transition) or unconditional.

When a workflow process has been defined, it may be executed many times. Execution of a workflow process according to its definition is called a process enactment. The context of such an enactment includes real performers (workflow participants), concrete relevant data, and specific application call parameters.

The representation of a single enactment of a workflow process is called a process instance. Its behavior is expressed by states and described as a state chart diagram. The history of states for a given process instance is represented by process instance state entities. Every process instance has its own data container. This container is an instantiation of a data container type defined for workflow process and includes container attributes that are used to control process execution (e.g. in flow conditions).

Execution of a process instance may be considered as execution of a set of activity instances. Every activity instance that is an atomic activity is performed by one workflow participant. If more that one participant is assigned to the activity, then this activity is instantiated as a set of activity instances with one performer for each of them. Such activity instance may be executed as an application called with specific parameters and operates on data objects that are instances of data types assigned to the activity during process definition.

If an activity instance is a route activity it is performed automatically by the system and there is neither application nor data objects assigned to it. In the case when activity instance is a *sub-process*, it is represented by another process instance executed according to the definition of the mentioned sub-process.

Similarly to the process instance, behaviour of an activity instance is represented by state diagram and stored as activity instance state entities.

Flow between activity instances is represented by *transition instances*. When an activity instance is finished, the system checks which transitions that are going from this activity may be instantiated. If a transition has no condition or transition condition is satisfied, it is automatically instantiated by the system. A transition instance may be considered as a relation 'predecessor-successor' between two activities.

3. Business Process Query Language

The process metamodel described in the previous section specifies basic workflow entities and relationships between them. In order to extract information from this metamodel, it is necessary to define a query language addressing business processes.

This language like other standard query languages should have clear syntax and unambiguous semantics. It should be able to express complex queries. It also needs to be coherent and complete to ask all possible queries to the process model. It should be an object-oriented language rather than a relational one since the process metamodel is much easier to represent through objects and associations.

This language should also provide core workflow specific functionality, namely:

- functions to simplify operations on process definition/instance graphs, especially to retrieve information about the first activity, current activity, its predecessors as well as possible successors,
- functions to extract some information about the context of calling a given query, e.g. in the case of using function to retrieve the activity that is currently processed it is required to know which process that query is asking for.

It seems that these two kinds of requirements for business process query languages may be satisfied by selecting the most suitable standard query language and extend it by a set of process specific features/functions.

Because of popularity of XML the family of XML query languages (e.g. XQuery) comes as the first candidate for the mentioned standard language [22]. They have quite clear syntax and reasonable set of operators. However, these languages address a hierarchical data structure with no cycles and many-to-many relationships. This kind of relationship is used quite often in the process model, for instance, the predecessor-successor association between activities or activity instances is a many-to-many one. The next candidate is the family of SQL query languages. These languages are very popular and widely used. They are able to operate on relational data and query various types of relationships between entities. Despite many advantages of these languages their syntax is quite complicated (see successive specifications of SQL-89, SQL-92 [12] and SQL-99 [13]). Some constructs of SQL introduce limitations and semantic reefs (e.g. group by operator or null values) and sometimes are criticised for their ambiguous semantics. While SQL-99 covers not only relational, but also object-relational structures, implementation of it presents too big challenge (ca. 1500 pages of specification), clearly out of the potential of a small software enterprise.

The last alternative is the family of object-oriented query languages. They seem to be the most appropriate to express various types of queries to the business process metamodel. As far as well known specifications of object-oriented or object-relational specifications are concerned, it seems that there is no good solution for the requirements specified at the beginning of this section. There are many opinions that the most known ODMG OQL [15] does not provide clear and coherent specification [2, 19]. Moreover, lack of convincing formal semantics causes that query optimisation in OQL is still an open question. Fortunately, there is one more candidate that has recently joined the group, that is the Stack-Based Query Language (SBQL) [17, 18, 20] Unlike other presented candidates this language has simple, formal and coherent semantics. SBQL has very powerful query optimisation methods, in particular, methods based on query rewriting, methods based on indices and methods based on removing dead queries. Currently it has several implementations, in particular, for the European project [9], for XML repositories based on the DOM model, and for the object-oriented DBMS Objectivity/DB. SBQL syntax (at it is shown in the next sections) is very simple and seems to be easier to understood and use.

3.1 Syntax, Semantics and Pragmatics of BPQL

The BPQL syntax is defined as a context-free grammar specified in the EBNF notation. Semantics of BPQL follows the semantics of SBQL and it is based on the operational method (abstract implementation machine). Pragmatics of BPQL concerns how it can be used properly and what is the reason to use it. Pragmatics includes some visualisation of the process metamodel and rules of developing BPQL queries according to the metamodel and according to the assumed business ontology. In this section, BPQL pragmatics will be illustrated by examples showing definitions of a business processes and corresponding queries.

The fundamental concepts of BPQL are taken from the stack-based approach (SBA) to query languages. In 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 (call) stack. 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 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 the formal query language SBQL (Stack-Based Query Language) is defined. SBQL is based on abstract syntax and full orthogonality of operators, hence it follows the mathematical flavour of the relational algebra and calculi. SBQL, together with imperative extensions and abstractions, has the computational power of programming languages. Concrete syntax, special functionality, special features of a store model and a concrete metamodel allow one to make from SBQL a concrete query language, in particular BPQL.

SBA respects the naming-scoping-binding principle, which means that 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. The classical stack concept implemented in almost all programming languages is extended to cover database collections and all typical query operators occurring e.g. in SQL and OQL. The stack also supports recursion and parameters: all functions, procedures, methods and views defined by SBA can be recursive by definition. Rigorous formal semantics implied by SBA creates a very high potential for query optimization. Full description of SBA and SBQL is presented in [20].

Following SBQL, a BPQL query can return a simple or complex value that is constructed from object identifiers, atomic values and names. The definition of a query result is recursive and involves basic structure and collection constructs (struct, bag, sequence). In particular, a query can return an atomic value (e.g. for query 2+2), a bag of references to attributes (e.g. for query *Performer.Name*), a collection of named references to objects (e.g. for query *Performer as p*). BPQL is based on the modularity rule which means that semantics of a complex query is recursively composed from semantics of its components, up to atomic queries which are literals, object names and function calls.

BPQL includes all basic non-algebraic operators (SBA term), such as quantifiers, selection, dependent join, projection, navigation (path expressions). There is also many algebraic operators, among them a conditional query *if* q_1 *then* q_2 *else* q_3, aliasing (operator *as*), typical arithmetic/string operators and comparisons, boolean operators, and others. A simplified syntax of BPQL is presented in Appendix.

A BPQL query can also include a function call. A function may have arguments, which are BPQL queries too. A function returns a result, which is compatible with query results (e.g. can be a collection of OIDs) hence function calls can be freely nested in BPQL queries. BPQL introduces a set of core functions, both standard ones and workflow specific ones. The standard functions includes mathematical functions (e.g. COS, SIN, SQRT), string functions (e.g. CONCAT, SUBSTRING), date and time functions (e.g. CURRDATE, YEAR, MONTH), and aggregate functions (e.g. AVG, COUNT, MAX).

So far, there are four workflow-specific functions. They were implemented in BPQL in advance to simplify queries.

- *FirstActInst(ProcessInstance): ActivityInstance* – returns an object that represents the start activity for a process instance which is passed as the argument of the function.
- *PrevActInst(ActivityInstance): ActivityInstance []* – returns a list of objects that represent direct predecessors of the activity passed as the argument of the function.
- *ActInst(ProcessInstance, ActId): ActivityInstance []* – returns a list of activity instances that are instantiation of the activity within a process instance. Both process instance object and identifier of the activity are passed as the function arguments.
- *NextActInst(ActivityInstance): ActivityInstance []* – returns a list of objects that represent direct successors of the activity passed as the argument of the function.

In addition to the above functions, two context-related functions are provided:

- *ThisProcessInst: ProcessInstance* – returns the object that represents the process instance on behalf of which a given query is executed.
- *ThisActivityInst: ActivityInstance* - returns the object that represents the activity instance on behalf of which a given query is executed.

All the class and attribute names as well as function names and their parameters are stored in the internal BPQL dictionary. A couple of examples of using BPQL are presented below.

Example 1 – Optional activity

The 'advanced verification' activity should be performed only if the deadline for a given process instance is greater then 2 days and exists an expert which the current workload is less than 8 working hours.

```
(ThisProcessInst.deadline – Currdate) > 2 and
  exists (Performer as P where
                    sum( (P.performs.ActivityInst where
status = 'open').duration) < 8)
```

Example 2 – Participant assignment
An activity should be executed by the performer of the first activity or, if this performer has more than five delayed tasks to perform, by the performer of the previous activity.

```
if
  count(FirstActInst(ThisProcessInst).performedBy.
   Performer.performs.ActivityInst where
     (delayed = 'yes' and status = 'open') <= 5
then FirstActInst(ThisProcessInst).performedBy.Performer
else  PrevActInst(ThisActInst).performedBy.Performer
```

3.2 BPQL and process definition

BPQL may be used to generalise process definition. It is able to express some requirements on the process definition that depends on process execution data (e.g. the performer of this activity is a seller with the minimal work-load). As will be shown in the next section, BPQL queries may simplify process definition reducing the number of defined activities which have to be 'artificially' introduced to cope with the mentioned requirements.
BPQL may be used in the process definition in all the elements that operate on relevant or workflow control data such as transition condition, workflow participant assignment, pre and post-activity conditions, and event handling (condition part).

Moreover, BPQL exposes the mentioned requirements directly in the process definition giving more knowledge of its elements to the process designers. Before, these element had to be hidden to the process designers and written as programming procedures. Such situation makes process modification harder since very often it is necessary to modify the code of procedures instead of modifying the process definition itself. BPQL gives the chance to the process designers to change process definition without (or with less) interfering in the programming stuff.

3.3 How It Works – an Example

Let us assume that there is a simplified version of a process for ordering laptops. Every registered customer may order any number of laptops. An order made by a customer is then accepted by a company seller which is responsible for verifying financial status of the customer and ability to meet the order at the requested time. For a bigger order or if there is not too much time for its acceptance, the order is served by a senior-seller. Otherwise, it is served by a plain seller. In addition, the order should be served by a seller that has minimal work-load. At this stage of process implementation 'minimal' means a person which has minimal number of tasks assigned. If the order was accepted, it is sent to the production department for completing. It is an assumption that all the orders are processed by the company.

Even in this simplified example some of the requirements for the process have to be defined on the basis of process execution data. For instance, the 'minimal work-load' requirement may be only expressed using a query on the current task assignment. Yet, to select a kind of a seller for accepting the order may be defined by a condition on process relevant data (workflow environment). The requirement may be easily expressed using BPQL. What is more, their definition in BPQL simplifies the definition of the process making it more generalised. Instead of defining two activities for accepting the order: one for 'Senior-seller' and another for 'Seller', it is possible to define only one for both mentioned workflow participants. Such approach seems to be more adequate the real business process that is modelled and computerized. The role 'seller' may be defined in BPQL as follows[3]:

```
1.  if Order.Value > 30000 or (Order.DeliverDate – CurrDate) ≤ 2
2.  then
3.    User where position = 'Senior-Seller' and
4.     count(is.Performer.performs.ActivityInst  where status='open')
5.       =
6.     min((User where position = 'Senior-Seller').
7.     count(is.Performer.performs.ActivityInst where status='open'))
8.  else
9.    User where position = 'Seller' and
10.    count(is.Performer.performs.ActivityInst where status='open')
11.      =
12.   min((User where position = 'Seller').
13.    count(is.Performer.performs. ActivityInst where status='open'))
```

Line 1 defines the condition to select either a senior-seller (lines 3-7) or a seller (lines 9-13). A senior-seller that will perform a given activity is a person employed at the position 'Senior-seller' (line 3) and which has the minimal work-load (line 4) among all senior sellers (lines 6-7). Similarly for the users with the 'Seller' position. The condition *status='open'* specifies only those tasks that are currently being performed. The function *count* determines the number of activity instances that are currently performed by a given user. Similarly, the function *min* determines a minimal number of tasks assigned to 'Senior-seller' (lines 6-7) and a 'Seller' (lines 12-13). Association *is* connects User objects to Performer objects, and association *performs* connects Performer objects to ActivityInst objects (note corresponding path expressions). Because in general the query may return several users (all with minimal workload), the additional function is necessary that takes randomly one of them. The query can be optimised by using an index on the User

[3] The object *User* represents an application user. The corresponding class is a part of application resource data.

position attribute and by factoring out independent sub-queries (in this case both sub-queries starting from *min*).

4. Integration BPQL and XPDL

It seems that the best way of using BPQL in process definition is to integrate it with a well known and widely used process definition language. Nowadays, there are several standard process definition languages such as XML Process Definition Language (XPDL) [25], Business Process Modelling Language [6] or more web-oriented languages such as Business Process Execution Language for Web Services (BPEL4WS) [5] and Web Service Description Language [26].

So far it seems that XPDL and BPEL(4WS) are the most mature and complete process definition languages. Both these languages may be easily extended of BPQL. This integration will much extend the functionality of the existing language. As an example we present in the next sub-sections how it may be done in XPDL.

4.1 Transition condition

In XPDL [25] a transition condition is expressed by the XML tag *Transitions/Transition/Condition* with type *CONDITION*. However there is no specification how this condition should look like. Usually it is represented as a text. In this situation, the XPDL definition may be extended by more precise definition which requires a condition to be written in BPQL. If a BPQL query returns one or more objects as the result, then the condition is satisfied. Otherwise, if the result is empty, it is not. An example written in XPDL may look like:

```
<Transition Id="b1" From="ChckBalance" To="ProcRequest">
   <Condition Type="CONDITION">
    Order where
      (id = ThisProcessInstance.hasDataContainer.orderId  and
       (value > 30000 or quantity > 100))
   </Condition> <Transition/>
```

4.2 Workflow Participant Assignment

According to the WfMC's definition [24], a Workflow Participant Assignment (WPA) defines the set of participants that will perform a given workflow activity. A participant can be one of the following types: a resource (specific resource agent), resource set, organisational unit, role (a function of a human within an organisation), human (a user) or system (an automatic agent).

To be coherent with the above definition it is suggested to use BPQL to define a participant. BPQL definition could be included as an extended attribute of the participant specification, if the participant is represented as a role. In this case, the WPA definition would remain the same while the participant definition would be expressed as a function that returns a set of participants. A BPQL query would return a set of workflow participants that would satisfy it. In addition, also WPA decision and the modifier introduced in [14] could be used to specify a workflow participant. An example written in XPDL may look like:

```
<Participant Id="p1" Name="Seller">
    <ParticipantType Type="ROLE">
    <Description>Seller</Description>
    <ExtendedAttributes>
     <ExtendedAttribute Name="Definition">
        User where (position = 'Seller')
     </ExtendedAttribute> </ExtendedAttributes>
</Participant>
```

4.3 Pre and post-activity condition

So far, pre and post conditions for workflow activities are not defined directly in XPDL. However, it is possible to use the tag *ExtendedAttribute* to express these conditions which, once again, would be defined in BPQL. If a BPQL query returns one or more objects as the result, then the condition is satisfied. Otherwise, if the result is empty, it is not. An example written in XPDL may look like:

```
<Activity Id="56" Name="Compose Acceptance Message">
  <Implementation>
    <Tool Id="composeMessage" Type="APPLICATION">
      <ActualParameters>
        <ActualParameter>status</ActualParameter>
        <ActualParameter>orderNumber</ActualParameter>
      </ActualParameters>
    </Tool>
  </Implementation>
  <ExtendedAttributes>
     <ExtendedAttribute Name="Pre-Condition">
       Order where (id =
ThisProcessInstance.hasDataContainer.orderId and status =
'closed')
     </ExtendedAttribute>
  </ExtendedAttributes>
</Activity>
```

5. Practical results

The first version of BPQL has been implemented for workflow participant and integrated in *OfficeObjects*® *WorkFlow* (*OO WorkFlow*) from Rodan System. This system has been deployed at major Polish public institutions (e.g. Ministry of Labour and Social Policy, Ministry of Infrastructure) as well as at private companies (e.g. Sanplast Ltd.).

XPDL, used in *OO WorkFlow* to represent process definition, has been recently extended of BPQL according to the suggestions about workflow participant assignment presented in the previous section.

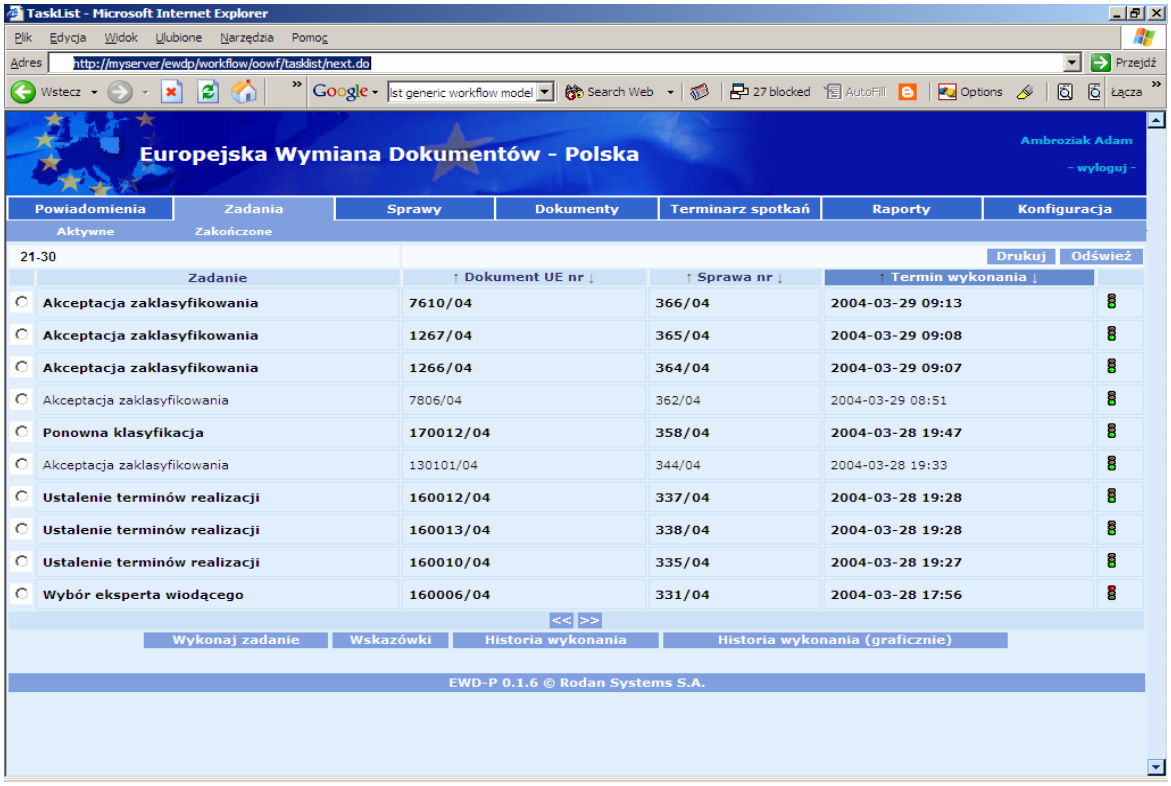

Figure -3 list for an EWDP user (in Polish)

First practical verification of BPQL was done for the system for electronic document exchange between Poland and European Council (referred further to as EWDP). In this system *OO WorkFlow* was used to implement the process for preparation of the Polish standpoint concerning a given case that was discussed at the European Council, COREPERS or its working groups. The process consists of more than 40 activities and includes about 10 process roles. It is going to be used by all nineteen Polish Ministries and central offices with about 12000 users registered. Daily, there are about 200 documents processed. On the average, preparation of the Polish standpoint lasts about two or three days.

Owing to BPQL, it was possible to generalise this process and make it suitable for all the offices. Complex rules to assign appropriate workflow participants have been quite easily expressed in BPQL, especially for (1) main coordinator which assigns Polish subjects to individual EU documents, (2) leading and supporting coordinators which assign experts to the processed document, (3) leading expert and supporting experts. These workflow participants are selected on the basis of possessed roles, their competence, current work-load and availability. Competence data are extracted from the system ontology.

In addition, also the process owners got the better chance to modify the process definition without modifying the code of the system. So far, there were twelve medium-scale changes of the process that were done only by modification of the process definition (early production phase).

6. Related Work

So far, there are a few other approaches to define a business process query language and use it in process definition. Firstly, in [7] the authors presented a language to model web applications – WebML. This language provides functionality to define business processes and make them more flexible. To define a condition or a workflow participant it is possible to use an attribute whose value may be calculated by a complex program. Despite its huge flexibility (every algorithm may be written as a procedure), this approach, however, seems to be less appropriate for non-programming process owner and to hide the algorithms to calculate these attributes inside the program code. In addition, at the best our knowledge this language is not compliant with any of the existing well known standard process definition languages.

In [14] the authors proposed WPAL – a functional language to define workflow participant assignment. This language is able to use workflow control data (e.g. the performer of a given activity, reference to the objects represents the start activity). Despite its flexibility, it has similar problems as WebML has. BPQL defined in this article may be treated as a continuation and significant extension of WPAL.

Finally, there is some work on a business process query language carried out by BPMI. It is promised that this language will offer facilities for process execution and process deployment (process repository). However, after two years of this work, still there is no official, even a draft version available [3].

On the other hand there are many workflow management systems that provide some selected elements of business process query languages. For example in Staffware [16], there is a set of workflow functions (i.e. SW functions) that can be used to define a conditions and a workflow participant. In Lotus Workflow [8], it is possible to call a lotus script function in order to define the above elements. *OfficeObjects WorkFlow* in her previous version implemented WPAL [14]. Unfortunately, all these examples of query languages do not provide clear and coherent semantics. In addition, the problem with moving algorithms form process definition into application still remains.

7. Conclusion

In this paper we have defined a workflow process metamodel and a business process query language BPQL to operate on this metamodel. In order to assure clear syntax and complete semantics of the language, SBQL has been used as the core specification. On the top of it BPQL was defined. The article also shows how BPQL may be used to make process definition more flexible and easy to modify. BPQL, following SBQL, can work on all data models that can be used for a workflow process metamodel, starting from the relational one, through XML-oriented, up to most advanced object oriented models.

Despite these advantages the presented approach leaves several open issues. In some cases where relevant data come from several data sources it is impossible to make a BPQL query. Also BPQL is not helpful when the current control of process resources depends on factors that are not present in the metamodel, or factors that are too random. In such cases the workflow processes must be controlled manually.

The first version of BPQL has been developed in *OfficeObjects® WorkFlow* and implemented in the system for electronic document exchange between Poland and European Council to define process for answering European documents. Owing to BPQL it was possible to reduce flow complexity of the process and make it easier for further modifications.

References

1. Van der Aalst, W.M.P.: Generic Workflow models: How to Handle Dynamic Change and capture management Information?, CoopIS'99, Edinburgh, Scotland
2. Alagic, S.: The ODMG Object Model: Does it Make Sense?, OOPSLA'97, Atlanta, USA
3. Business Process Management Initiative: Business Process Query Language Web Page, http://www.bpmi.org/bpql.esp
4. Bussler, C.: Keynote – Semantics Workflow Management, presentation at BPM'2004, Potsdam, Germany
5. IBM developer works: Business Process Execution Language for Web Services, ver. 1.0, Jul 2002
6. Business Process Management Initiative: Business Process Modelling Language, Nov 2002
7. Ceri, S., Manolescu I.: Constructing and integrating Data-Centric Web Applications; Methods, Tools and Techniques, VLDB'2003, Berlin, Germany
8. Lotus Development Corporation, Lotus Workflow Process Designer's Guide, ver. 3.0, 2001
9. Eder, J., Paganos, E.: Managing Time in Workflow Systems, in Workflow Handbook 2001, Layna Fischer (Ed.), Future Strategies Inc., USA
10. Intelligent Content Management System, IST-2001-32429, 5th EC Framework Programme, www.icons.rodan.pl
11. List B., Schiefer J., Bruckner R.: Process Data Store: A Real-Time Data Store for Monitoring Business Processes, DEXA'2003, Prague, Czech Republic
12. Melton, J., Simon, A.: Understanding the New SQL: A Complete Guide. Morgan Kaufmann, 1993
13 Melton, J., Simon, A., R., Gray, J.: SQL:1999 - Understanding Relational Language Components. Morgan Kaufmann Publishers, 2001
14 Momotko, M., Subieta. K.: Dynamic change of Workflow Participant Assignment, ADBIS'2002, Bratislava, Slovakia
15. Object Data Management Group: The Object Database Standard ODMG, Release 3.0. R.G.G.Cattel, D.K.Barry, Ed., Morgan Kaufmann, 2000
16. Staffware, Procedure Definer's Guide; Staffware Plc, Issue 2, March 1999
17. Subieta K., Beeri, C., Matthes, F., Schmidt, J.,W., A Stack-Based Approach to Query Languages. East-West Database Workshop, 1994, Springer Workshops in Computing, 1995
18. Subieta, K., Kambayashi, Y., and Leszczyłowski, J.: Procedures in Object-Oriented Query Languages. VLDB'1995, Zurich, Switzerland
19. Subieta, K.: Object-Oriented Standards: Can ODMG OQL be Extended to a Programming Language? (In) Cooperative Databases and Applications, World Scientific, 459-468, 1997
20. Subieta, K.: Theory and Construction of Object-Oriented Query Languages. Editors of the Polish-Japanese Institute of Information Technology, 2004, 520 pages (in press)
21. Sadiq, S., Handling Dynamic Schema Changes in Workflow Processes, 11th Australian database Conference, Canberra, Australia, 2000
22. W3C: XQuery 1.0: An XML Query Language. W3C Working Draft 12, Nov 2003, http://www.w3.org/TR/xquery
23. Weske, M., Vossen, G., Flexibility and Cooperation in Workflow Management Systems, Handbook on Architectures of Information Systems., pp 359–379. Berlin: Springer 1998
24. Workflow Management Coalition: The Workflow Reference Model, WfMC-TC-1003 issue 1.1, Jan 1995
25. Workflow Management Coalition: Workflow process definition language – XML process definition language, WfMC-TC-1025, ver. 1.0, Oct 2002
26. W3C Consortium: Web Service Description Language, ver. 1.1, W3C, March 2001

Appendix: the (simplified) Syntax of BPQL

<query>	::=	<literal>
	|	<name>
	|	**exists** <query> **(** <query> **)**
	|	**all** <query> **(** <query> **)**
	|	<query> **.** <query>
	|	<query> **where** <condition>
	|	<query> **join** <query>
	|	<query> **as** <aliasName>
	|	<query> **group as** <aliasName>
	|	<function>
	|	(queryList)
	|	**if** <query> **then** <query> **else** <query>
	|	<algExpression>
queryList	::=	<query> {, <query>}
<condition>	::=	<logExpression>
<logExpression>	::=	<logSum>
<logSum>	::=	<logProduct> { **or** <logProduct>}
<logProduct>	::=	<logSExpression> { **and** <logSExpression> }
<logSExpression>	::=	**not** <logSum> | **(** <logSum> **) |** <logCondition>
<logCondition>	::=	<leftSide> <opComp> <rightSide>
<opComp>	::=	**<** | **<=** | **=** | **=>** | **>** | **<>**
<leftSide>	::=	<algExpression>
<rightSide>	::=	<algExpression>
<algExpression>	::=	<algSum>
<algSum>	::=	<algProduct> { [**+** | **-**] <algProduct>}
<algProduct>	::=	<algExpression> { [* | / | %] <algExpression>}
<algExpression>	::=	**(** <algExpression> **)**
	|	<symbol> | <literal> | **(** <query> **)**
<function>	::=	<fName> | <fName> **(** <queryList>**)**
<symbol>	::=	<objName> {. <objName>}
<literal>	::=	<text> | <integer> | <float> | <boolean>
<boolean>	::=	**true | false**