QUERYING WORKFLOWS OVER DISTRIBUTED SYSTEMS

Hanna Kozankiewicz
Institute of Computer Science, Polish Academy of Sciences, Ordona 21, 01-237 Warszawa, Poland

Krzysztof Stencel
Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland

Kazimierz Subieta
Polish-Japanese Institute of Information Technology, Koszykowa 86, 02-008 Warszawa, Poland

Keywords: Workflow, query language, object-oriented, distributed system, pre-condition, post-condition, SBA, SBQL.

Abstract: We describe a new paradigm of declarative workflow, where an activity is modelled as an object which can have other activities as subobjects. All activities act in parallel and have pre- and post-conditions determining when they are to be started and when they are to be terminated. The proposal is based on the Stack-Based Approach (SBA) to object-oriented query languages, query language SBQL and updatable SBQL views. The idea is currently implemented within the European project eGov Bus.

1 INTRODUCTION

During our previous experience with workflow management software (Momotko, 2004), (Momotko, 2005) we have made important observations concerning the nature of querying a workflow data environment and the nature of workflows. Workflows are usually perceived as activities and transitions (WfMC, 1995) in the spirit of Petri nets. This view is common and quite successful, but has disadvantages. There are problems concerning parallel execution of sub-processes in a distributed environment. Usually the parallelism is supported by split/join operations build into the control of the workflow. However, such synchronization of parallel processes might be hard in many practical situations, e.g. when splitting/joining concerns statically unknown number of sub-processes. Such situations are frequent in distributed and heterogeneous workflow environment. There are other disadvantages of too stiff control flow within some graph of activities.

In the current projects we investigate an alternative paradigm, when activities are objects in an object-oriented model and sub-activities are sub-objects. By definition, all such objects have some attributes which can be queried, but they have also executable parts which by definition run in parallel. We provide synchronization means known as pre-conditions and post-conditions that make it is possible to start and terminate particular activities. Sequential execution of activities can be forced by a pre-condition of a next activity that depends on the state of a previous one. For a complex activity a post-condition determines when it has to be terminated. Pre-conditions and post-conditions are formulated in a query language which may work on the entire workflow environment, the state of all the population of workflow processes and the state of resources (e.g. workflow performers). The advantages of the approach concern, in particular, inherent parallelism of activities, easier distribution of parallel activities among many machines and dynamic changes of executed processes (e.g. by runtime inserting a new activity or removing an activity). Synchronization of workflow sub-processes by a post-condition corresponds to join operators in a typical workflow definition language, but a post-condition expressed as a query is incomparably more powerful and flexible.

The proposed paradigm makes workflows close to PERT nets that are used in project planning to

Kozankiewicz H., Stencel K. and Subieta K. (2007).
QUERYING WORKFLOWS OVER DISTRIBUTED SYSTEMS.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Society, e-Business and e-Government / e-Learning*, pages 251-254
DOI: 10.5220/0001285102510254

analyse critical time paths and other features of available resources. PERT nets assume inherent parallelism of processes, constrained however by their in-out dependencies and availability of resources necessary for execution. We argue that this paradigm is worth investigation concerning new qualities that it may bring for definition and execution of workflow processes and to make them more flexible for dynamic changes and parallel execution.

Our approach is based on the Stack Based Approach (SBA) to query languages and its powerful query/programming language SBQL (Subieta, 2004), (Subieta, 2006). The language can be used to perform efficient tracking and monitoring tools for population of workflows, just like in BPQL (Momotko, 2004). We propose a new metamodel for workflow processes and then show how SBQL can be used in workflow-specific applications. An important element of our architecture is that a workflow system can work on resources that are distributed among different organizations.

2 ARCHITECTURE OF A DISTRIBUTED WORKFLOW

A workflow mechanism operates on top of a *virtual store* which integrates data and services from different distributed sources. The details concerning the store are described e.g. in (Kozankiewicz, 2005). It is implemented by means of updatable views.

The resources are supplied by local servers. Local schemata have to be adapted to the requirements of the virtual store. To this end, the administrators of local servers have to implement *wrappers,* which provide mappings of local data/services to the global canonical data/service model assumed by the virtual store. These wrappers can be implemented as updatable views. A basic integration mechanism is on an intermediate server, which contains *integrators* that virtually fuse resources supplied by local servers. Integrators resolve heterogeneities and redundancies and join fragmented collections. Integrators are also to be implemented as updatable views. On a client level there are *customization views* that adopt the global virtual schema to the need of a particular client application and determines client's access privileges. A workflow mechanism is on a special server which through the virtual store can access all the distributed resources. The workflow server is connected to its clients (workflow participants).

Such integration of e-Gov resources supported by various partly independent institutions allows building a workflow system over a distributed e-Gov infrastructure. When resources are virtually integrated, one can use the query/programming language SBQL to write applications, with the use of abstractions such as classes, methods, procedures and updatable views. SBQL is also used in definitions of workflow processes to query all the workflow environment (through the workflow metamodel) and to formulate pre- and post-conditions for particular workflow activities.

3 DECLARATIVE WORKFLOWS

Our basic assumptions for workflow systems built on top of a virtual store are as follows. Each activity is modelled as an object. It can consist of sub-activities, i.e. sub-objects. There is no restriction on the size and the number of levels of the hierarchy of the activity nesting. The top level activity can be considered a workflow process.

Activities can be dynamically inserted into a super-activity or removed from it. In such a way we provide a flexible mean to adapt processes/activities to dynamically changing complex application requirements. There is no explicit graph describing control flow between activities, because for many workflow types such graphs are not enough flexible concerning parallel execution, dependency on resources necessary for an activity and the possibility of runtime workflow changes. Instead, all activities have pre- and post-conditions defining when they are to be started and when they are to be finished. Such conditions are expressed in SBQL.

Activities can include inner objects, procedures, views, etc. Atomic activities have executable parts that work concurrently to other activities. For complex activities an executable part can be treated as sub-activity. A sub-activity can be initialised if and only if its super-activity has been initialised. All activities are described by definitions, which can be used e.g. to create activity instances and to type checking. Not all sub-activities must be terminated to terminate the given activity – this depends on its post-condition only. Similarly, terminating all sub-activities does not necessarily mean that the given activity must be terminated too.

One of possible goals of our idea is to support life events of citizens or enterprises. In Figure 1 we show how activities related to a life event 'family location change' are represented using our workflow model. On the first part we show sub-activities such as changing residence, changing schools, changing home MD, and so on. On the second part we show

how the process of changing location can be modelled as a hierarchy of workflow activities. Note that the activity 'Change school' can appear zero, one, or more times, depending on the number of children in the family.

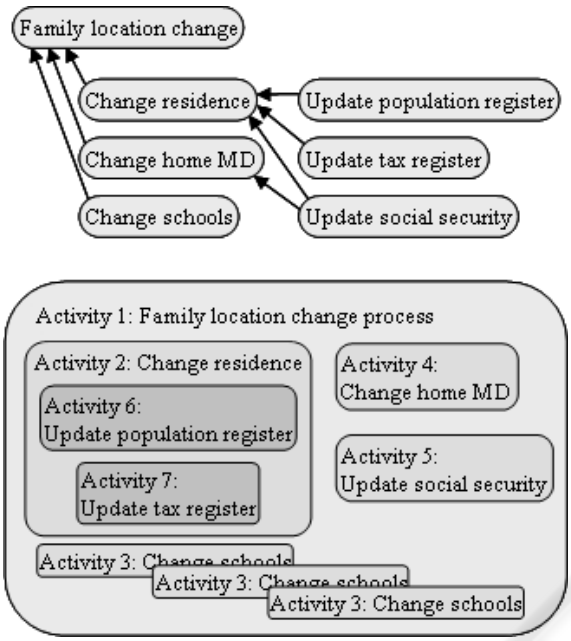

Figure 1: Activities as objects.

Activity instances are complex objects with the following example structure:

ActivityName{ ontology, priority, …
activityState,
precondition, **action**,
cost, duration, deadline,…
refs to **performers**,
refs to **subactivities**,
postcondition,
ref to **ActivityDef**,
local objects, procedures, …}

Each activity is identified by its name, which bears its business semantics. An activity can be in several states (waiting, pending, terminated, etc.). Complex activities have references to their subactivities and post-condition. An activity may have a reference to its definition in some metamodel.

Activity instances can generate activity instances. In this way, in particular, one can organize loops of some activities. As objects, activities can be connected to classes containing some of their invariant properties. The code of activity actions may require special workflow-oriented functions, such as an access to workflow client software.

4 SBQL FOR WORKFLOWS

In SBQL we can easily define workflow-specific procedures and functions. For simplification we skip typing information and make some simplifications concerning right object naming and SBQL syntax. The following function returns activities which are in one of the states specified by the parameter and have the highest priority.

procedure *HighestPriorityActiv*(*states*) {
create *maxPriority* = **max**(*Activity.priority*);
return *Activity* **where** *priority* = *maxPriority*
and *activityState* ∈ *state*; }

The function can be called in the following query, that selects all suspended and pending actives with deadline during the next 10 days:

HighestPriorityActiv(**bag**('suspended', 'pending'))
where *deadline – currentDate* < 10

By SBQL recursive procedures one can easily process nested activity structures, e.g. evaluate cost of an activity and all its sub-activities:

procedure *getActivityCost*(*act*) {
create local c = *act.cost*;
for each *act.subactivity.Activity* **as** *a* **do**
c := *c* + *getActivityCost*(*a*));
return *c*; }

A workflow system can offer a large set of predefined workflow procedures and functions, for instance, a function returning the best workflow participant for the given task, a function that recursively returns all sub-activities, a procedure that insert a new activity into the given one, etc.

SBQL can also be used to write pre- and post-conditions. Below, we present a query which checks whether we can find an employee with the current workload less than 10 activities:

exists (*Performer* **where**
count(*performs.Activity*) < 10)

The next query is a post-condition of an activity that verifies whether all its sub-activities with the name 'Order Approval' have been successfully finished:

forall (*self.subactivity.OrderApproval*) **holds**
activityState = 'completed'

Pre- and post-conditions can be the subject of manipulations during runtime, in particular, new pre- and post-conditions can substitute the previous ones by using a proper SBQL statement. Such property of programming languages is known as *reflection*, i.e. dynamic modification of a running program. Reflexive capabilities require careful

treatment in the programming language, in particular, they are more difficult for strong type checker. For this reason the function that modifies pre- and post-conditions is not a regular assignment function but the function that calls parser, compiler and optimizer and then converts a condition into an executable bytecode.

Another application of SBQL queries and procedures is monitoring and tracing of the entire population of workflow processes. Below we present some examples.

- List all pending activities with the deadline expiring during the next 3 days:

  Activity **where** *activityState*= 'pending' **and** *deadline* – *currentDate* < 3

- List all pending activities which can be started, i.e. all their pre-conditions are valid. In this example we use a function *valid* that checks if a set of conditions is fulfilled.

  Activity **where** *activityState* = 'pending' **and** *valid*(*precondition*)

- List all active activities which have already missed their deadlines.

  Activity **where** *activityState* = 'active' **and** *deadline* < *currentDate*

- List all activities which are close to their deadlines and not assigned to anyone yet.

  Activity **where** *count*(*performers*) = 0 **and** *deadline* – *currentDate* < 3

- List all activities performed by overloaded employees, i.e. employees assigned to instances of activities which must be completed in following 7 days while their total duration is greater that 40 working hours:

  Activity **where** *sum*(*performers.Performer.* *performs*. (*Activity* **where** *deadline* < *currentDate* + 7) . *duration*) > 40

5 CONCLUSIONS

We have shown how a declarative workflow systems can be implemented on the top of distributed object-oriented systems. The case covers most of e-Government applications where complicated routines should be performed at multiple institutions. If a citizen or a company faces a life event, a number of governmental agencies is to be involved in serving it. The goal of an e-Government system should be to let this citizen carry out a single visit to one governmental web page in order to act upon this life event. He/she should not be forced to visit many web pages with different rules of the user behaviour.

A carefully designed workflow system on the top of integrated e-Government resources is able to achieve this goal. Such a system will facilitate the concurrency of processes by using more powerful synchronization mechanisms in comparison to *join* and *split* operators. The critical feature is flexibility of dynamic changes of processes. Furthermore, such a system should support effective monitoring and tracing of the entire population workflow processes and activities, including resources and documents necessary to perform them. For such complex situation it seems that a high level database query and programming language like SBQL is the best choice.

ACKNOWLEDGEMENTS

This work is supported by European Commission under the 6th Framework Programme project e-Gov Bus, IST-4-026727-ST

REFERENCES

eGov Bus, 2006. Advanced eGovernment Information Service Bus Project. http://www.egov-bus.org/web/guest/home

Momotko, M., Subieta, K., 2004. Business Process Query Language - a Way to Make Workflow Processes More Flexible. Proc. of ADBIS 2004, Springer LNCS 3255, pp. 306-321

Momotko, M., 2005. Tools for Monitoring Workflow Processes to Support Dynamic Workflow Changes. PhD Thesis, 2005. http://www.ipipan.waw.pl/~subieta/ → Finished PhD-s

Subieta, K., 2004. Theory and Construction of Object-Oriented Query Languages. Editors of the Polish-Japanese Institute of Information Technology, 520 pages (in Polish)

Subieta, K., 2006. Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL). http://www.sbql.pl

WfMC, 1995. Workflow Management Coalition: The Workflow Reference Model, WfMC-TC-1003 issue 1.1

Kozankiewicz, H., Stencel, K., Subieta, K., 2005. Implementation of Federated Databases through Updatable Views. Proc. of the European Grid Conference, Springer LNCS 3470, pp. 610-620