Prototype of Object-Oriented Declarative Workflows[*]

Marcin Dąbrowski[1], Michał Drabik[1], Mariusz Trzaska[1], Kazimierz Subieta[1,2]

[1] Polish-Japanese Institute of Information Technology
{mdabrowski, mdrabik, mtrzaska, subieta}@pjwstk.edu.pl
[2] Institute of Computer Science Polish Academy of Sciences

Abstract. While in the traditional workflow processes the control flow is determined statically within process definitions, in declarative workflow processes the control flow is dynamic and implicit, determined by conditions that occur in the workflow data and the service environment. The environment consists of so-called *active objects*, which play a double role. On the one hand, they are persistent data structures that can be queried and managed according to the syntax and semantics of a query language. On the other hand, active objects possess active parts that are executable and represent workflow processes or tasks. The approach is motivated by features that are desirable in complex and less regular business processes: (1) the possibility of dynamic changes of process instances during their run, (2) mass parallelism of process instances and their components and (3) shifting the availability of resources that workflows deal with on the primary plan as a mean for triggering instances of process tasks. The paper presents the prototype of an object-oriented declarative workflows on a comprehensive example with roots in a real business case.

Keywords: workflow, object-oriented, declarative, query language, active object, dynamic workflow change, ODRA, SBQL

1. Introduction

The workflow technology is a well developed domain with many commercial successes, which include such standards as BPEL [2], BPMN [13] and XPDL [21]. Nevertheless, there are still problems that undermine applications of workflow management systems in important business domains. We mention the following features that are insufficiently developed in popular workflow systems:

- Dynamic changes in workflow instances during their run. For complex workflow processes it may happen that their behavior must altered without altering their definitions. Dynamic workflow changes are the subject of many research papers, see e.g. [1, 5, 6, 9, 10,14, 15,16]. Although valuable results are achieved (especially concerning workflow patterns) the problem in general still remains unsolved.
- Parallel execution of tasks within workflow processes. Currently, the parallelism is achieved by explicitly programmed splits and joins (AND, OR, XOR). In many cases such a parallelism is insufficient, for instance, when a process is to be split into many subprocesses, but their number is unknown in advance. For instance, a workflow is

---

processing a package of documents, processing of a document needs a parallel subprocess, but the number of documents within packages varies.

- Aborting a process or some of its parts. Currently, such situations are handled manually, with possibilities of inconsistencies and non-optimal human action.
- Resource management. In currently developed workflows the work control flow (a la Petri net) is on the primary plan and the resources (people, money, time, work power, equipment, infrastructure, offices, vehicles, etc.) are secondary and sometimes not taken into account at all. This is unnatural for workflow processes because just availability, unavailability, planning and anticipating required resources are the main factors that should determine the process control flow.
- Tracking and monitoring. In current workflows tracking and monitoring of processes is done by some specialized predefined tools with limited functionality. However, tracking and monitoring should concerns the entire workflow environment and all running process instances, including databases that support workflows, the state of resources, anticipation of availability of resources, etc. For this reason the core for tracking and monitoring should be a query language (such as BPQL [11]) with the full algorithmic power rather than predefined tools.
- Parallel execution of workflow instances and their parts on many servers (hundreds or thousands).
- Transaction processing. Classical implementations based on ACID properties and 2PC/3PC protocols are insufficient for workflows. Business processes cannot be reversed, hence the attitude to transaction aborting should be changed. For various reasons (e.g. long transactions, priorities, performance) the attitude to the transaction isolation should be changed too. Moreover, if process instances can be changed during their run by many independent agents, then the process instances and their parts should be also the subject of the transaction discipline.

In the project funded by the Polish Ministry of Science and Higher Education we have investigated a new workflow paradigm that has the potential to overcome the above difficulties. We assumed that workflow instances can be changed during their run, hence they should possess a double nature. On the one side they are to be executable processes. On the other side, they should be considered database structures that are described by some conceptual schema and can be queries and manipulated as usual (nested) database objects.

The second assumption was inherent parallelism of all workflow processes and their parts. We avoid explicit splits and joins. Instead, we assume synchronization of parallel processes by special constructs of a query language. In this way our workflow instances remind PERT (Program Evaluation and Review Technique) networks rather than Petri nets. PERT naturally describes dependencies between tasks within non-computerized human activities and can be formalized using the object-oriented approach. Such a workflow system we describe as "declarative", because the control flow is not determined explicitly, but through declarative queries. Sequences of tasks can be supported by tasks' states and conditions on the states.

The third assumption is shifting the resource management on the primary plan. Resources (available, planned, anticipated) are reflected in the database. The control flow of process instances can be determined by conditions addressing resources.

In this way we came to the idea of *active objects*, which have the mentioned above double nature. Active objects are persistent data structures that are described by a database schema and can be queried and manipulated according to the syntax and semantics of a query language (in this role SBQL [19, 20]). On the other hand, active objects possess active (executable) parts. We distinguish four kinds of such active parts: *firecondition*, *execution*

code, *endcondition* and *endcode* (in this role SBQL too). An active object waits for execution until the time when its firecondition becomes true. After that, the object's execution code is executed, and all its active sub-objects are put into the *waiting-for-execution* state (and perhaps executed if their fireconditions become true). Execution of the execution code of a given active object is terminated when either all the actions are completed (including active sub-objects) or its endcondition becomes true. After fulfillment of an endcondition some actions might be necessary (e.g. aborting transactions), thus an optional endcode. Each active object is an independent unit that can be manipulated by SBQL functionalities (updated, deleted, etc.). Active objects can be nested. In this way they can represent workflow processes, their tasks, subtasks, etc. Active parts can also be updated; their parsing, type checking, optimization and compilation are performed on-the-fly. Bindings are mostly dynamic.

The widely recognized paper devoted to dynamic workflow changes is [14]. It presents some framework for formalizing process graphs and updating operations addressing such a graph. There are very valuable observations concerning the necessity of dynamic workflow changes for real business processes and the necessity of strong discipline within the changes to avoid violation the consistency of the processes. Numerous authors follow the ideas of this paper (a more complete citation list is presented in [3]). The fundamental difference of our approach is that we do not determine explicitly the process control flow graph. It is on the secondary plan, determined dynamically and implicitly by fireconditions and endconditions. In majority of cases the control flow graph can be different depending on a runtime state of the workflow, database and computer environment. The problem of the necessity of various control flow graphs for the same business process is one of the motivations for the research presented in [14], but it is not easy to see how such a feature can be achieved within the proposed formal workflow model. In our case the feature is an inherent property of the idea.

The primary plan of declarative workflows deals with a database schema that describes executable data structures representing the processes, thus by definition enabling all updating operations that are provided within the assumed programming language SBQL. To restrict undesirable changes that may violate the consistency of processes we can use the semi-strong typing system [18] that is implemented for SBQL. This of course may not be enough for more sophisticated situations. For this goal we plan to implement facilities that are well-know from relational systems, such as user rights, integrity constraints, business rules and triggers.

Our active objects remain agents considered within the agent-oriented research school. However, we avoid to use this association and terminology because.

The detailed description of the project assumptions and preliminary results from the prototype [7] are presented in [3]. In [4] we describe in detail the concept of active object and related issues. To check the concept we have implemented three different prototypes. The first prototype [8] was focused on the orchestration of Web Services. After this experience we concluded that not all goals from the mentioned above are addressed. The second prototype [7] was based on Web Services too and on the ODRA infrastructure (objects, queries), but it was still limited. This paper is the first description of the third prototype [17], most advanced and with no previous tradeoffs concerning the new idea and current workflow technologies. The prototype is still a proof-of-a-concept rather a usable tool. More research and financial support is necessary to turn it into a product.

The rest of the paper is organized as follows. Section 2 presents basic assumptions and the architecture of the prototype. Section 3 presents how dynamic instance modifications can be performed. The presentation is based on a real example of a workflow that was taken from the experience in developing a bank system supporting credit processes. Section 4 concludes.

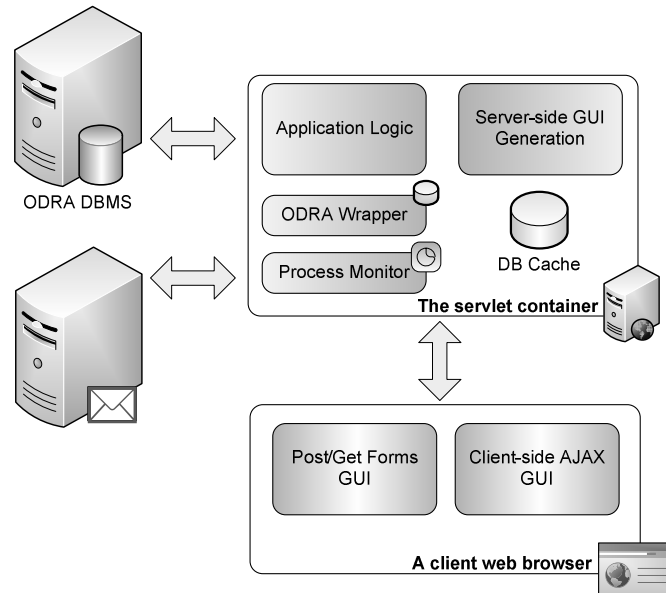## 2.   Prototype of Object-Oriented Declarative Workflow

Fig.1. Architecture of the prototype

The prototype [17] is built upon the ODRA system [12] and a Web-based application for manipulating prototype functionalities. The Web part uses the Groovy and Grails technologies. A workflow server part is written in Java. Fig.1 presents the overall prototype architecture.

The prototype can be tested using a Web application called SBQL4Workflow, Fig.2. It allows for all administrative tasks like creating process definitions, manipulating them, instating processes, freezing parts of a running workflow and more. A GUI generation module is based on the core Grails framework technology called GSP (Groovy Server Pages). It is similar to JSP (Java Server Pages). A client side is equipped with advanced AJAX controls to allow dynamic loading of a process tree and manipulating workflow objects minimizing the need to reload web pages. The SBQL code editor with syntax highlighting that is included into GUI makes the work with workflows much easier.

The ProcessMonitor is a Java application that can be run as a separate thread on a separate machine. It periodically checks (basing on timeouts) each ProcessInstance. Then, according to the values retrieved from condition codes, the ProcessMonitor executes the execution code of the process.

The prototype is build using the standard three layer approach. A middle layer consists of the Application Logic and ODRA Wrapper. The corresponding API allows for work with workflow objects. It is used not only by GUI and the ProcessMonitor but can be used by any Java program, so writing a different client application is possible. The ODRA Wrapper is a wrapper between the JOBC library that is used to access the ODRA DBMS through queries and Java business objects used by the Application Logic. All workflow data are stored on the ODRA DBMS []. The database schema is presented in the Fig.3.
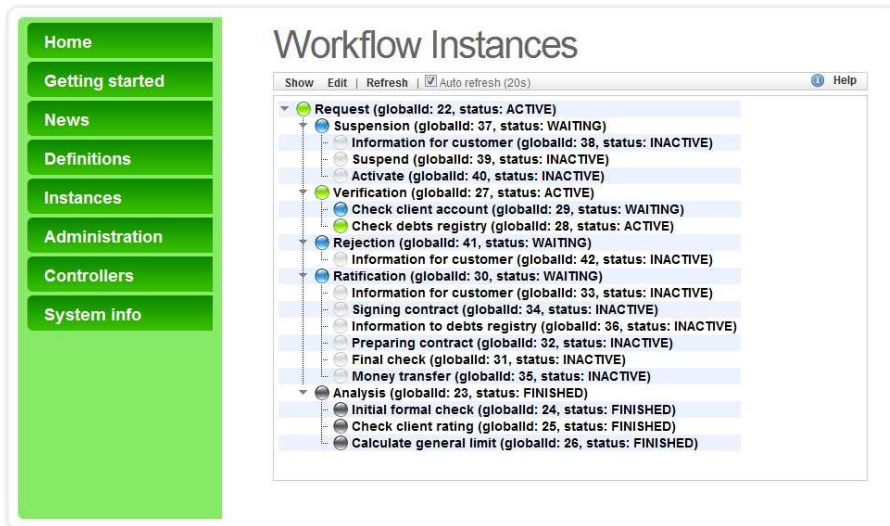
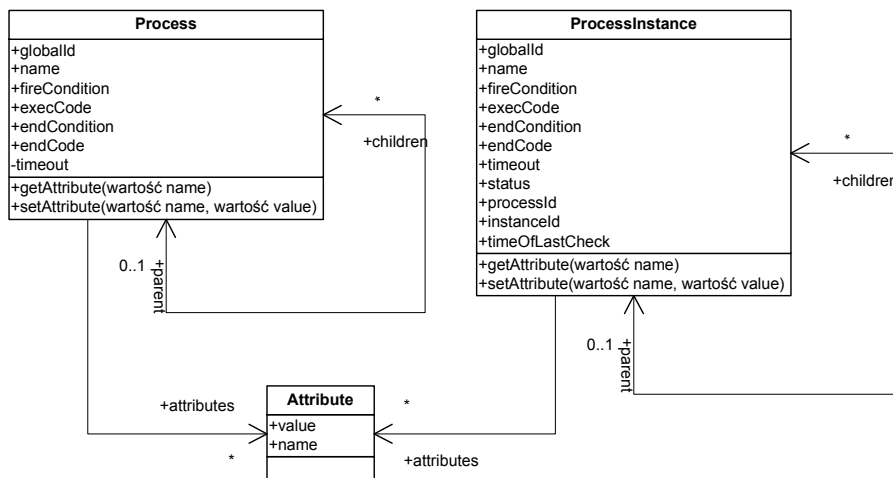**Fig.2.** A screen shot of the SBQL4Workflow process hierarchy

Fig.3. Workflow database schema

The process objects represent structures created by the workflow programmer. Once a process is initiated, all data, including the data of sub-processes, are copied to the corresponding ProcessInstance objects. The Parent-Children bidirectional pointer, combined with other SBQL query operators, gives a great flexibility in expressing conditions and codes. For instance:

- Find all my children (the code is written with regard to one particular Process).
- Find my parent.

- Find a process with a given status.
- Find a process with a given name.
  These constructs can be easily combined for more complex search, for instance:
- Find a child that has a certain name and status.

```
children where name = 'foo' and status = ProcessStatus.FINISHED
```

- Check if all my children have the status 'Finished'.

```
exists(children where status = ProcessStatus.FINISHED)
```

- Find my "brother" (using parent.children).
- Find all my "nephews" (using parent.children.children).

To allow processes to store ad-hoc additional data we have provided the Attribute class with a set of methods in the Process and ProcessInstance classes addressing attributes. The attributes can be easily used to control the flow (when the conditions are based on them) and allow communication between the processes (as one Process can query another Process attributes and change their values).

```
getAttribute('contractSigned')='true'
```

The code example presents the access to the attribute named 'contractSigned'.

```
setAttribute('mailSent'; sendMail('foo@bar.com'; 'Mail content'))
```

The code sends an email and stores the result (success or failure) as a process attribute.

3.   Dynamic Instance Modifications

After creating a process instance for any business reason it can be the subject of changes (without changing the corresponding process definition). Changes can be performed after launching an instance. Changes can also concern process definitions and this case is rather typical for all workflow systems. Our prototype has the following options concerning changes within workflows:

- Editing and modifying process definitions;
- Instantiating process instances according to the definitions;
- Editing and modifying a process instance by editing its core attributes such as name, fire condition, end condition, execution code, end code, etc.;
- Running any SBQL program (having updates, inserts, deletions, etc.) in order to manipulate the entire workflow environment, including nested active objects representing processes, a resource database and any other persistent or volatile objects that are available within the environment.. The programs include SBQL queries as expressions.

Changing a process instance may require further changes of other instances to ensure consistency of the corresponding business process. Our prototype offers much flexibility in controlling process instances without altering other instances, mainly by preparing more generic fireconditions and endcondition that are insensitive to some changes of active objects. For instance, an endcondition can test completion of all corresponding sub-processes with the use of a universal quantifier. In many cases, however, altering a process instance may require some actions on other instances. These actions can be nested within a transaction.

To demonstrate the possibility of dynamic instance modification we have created a comprehensive example of real business processes concerning issuing and granting bank credits for customers. The structure (schema) of the process presented in Fig.4. All presented SBQL codes are tested on the prototype.
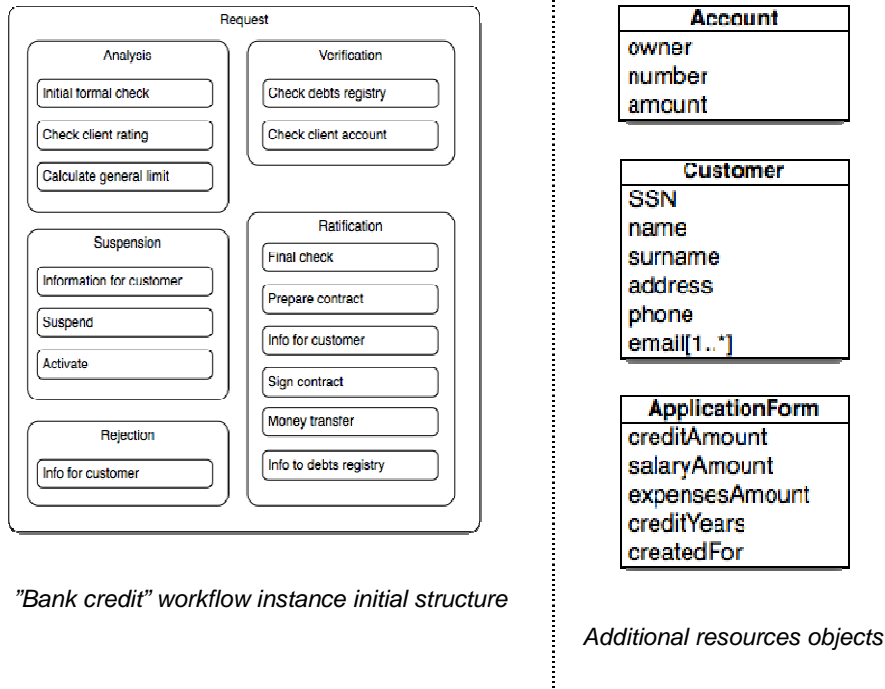
Fig.4. A bank credit process hierarchy and resource objects schemas

Example 1. *It demonstrates how to insert a new process into a workflow instance structure, without the need of changing the already working process instances details.*
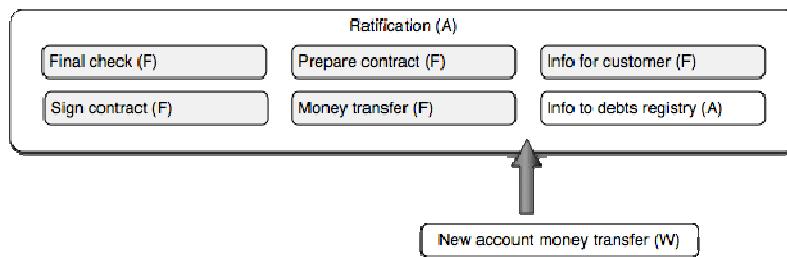

Fig.5. *Adding a „New account money transfer" process.*
Letters in brackets correspond to status of a process instance: FINISHED, ACTIVE.

Assume a bank credit process in progress. At the end of it the money that the customer has requested is transferred to his/her account. However after the transfer the customer has decided to change the target account. In this situation we can correct the working workflow instance by inserting an additional subprocesses that will do the requested operation, Fig.5. To achieve the goal we have to find a workflow instance that should be modified and create a new process called „*New account money transfer*" in it. It will have two attributes to store the value of an old and new account number, named respectively „*oldAccountNr*", "*newAccountNr*".

After inserting the new process its status is set to „*Waiting*". Its firecondition requires finishing the subprocess „*Money transfer*", hence it looks as follows:

```
exists(parent.children.ProcessInstance where name =
'Money transfer' and status=ProcessStatus.FINISHED)
```

Fire condition of „New Account money transfer".

The token *parent* is a navigation from the new subprocess to its parent process (*Ratification*), then *children* is a navigation to all the parent's subobjects, from which we select *ProcessInstances* with proper conditions.

The purpose of this process is to withdraw the money from the old account and transfer it into the new one. First a new account should be created:

```
create Account (ref (Customer where SSN =
parent.parent.ProcessInstance.getAttribute('customerSSN')) as
owner, getAttribute('newAccountNr') as number, 0 as amount)
```

Part of execution code that creates new account.

The code creates a new account object with the numbered delivered from the „*newAccountNr*" attribute of this process instance. Now we should find out the information about the amount of money that should be transferred. This information is a part of the „*ApplicationForm*" object which is available, so the task will be to find the application form assigned to the current customer and obtain the „creditAmount" value. To make this value available for further processing it will be saved as a value of a newly created attribute called „*amount*":

```
(self as p).(p.setAttribute('amount';(ApplicationForm where
createdFor.Customer.SSN =
p.parent.parent.ProcessInstance.getAttribute('customerSSN')).
creditAmount))
```

Part of execution code that obtains a value of credit amount and creates a new attribute for it.

Now it is possible to withdraw the money from the old account. To do that the right Account object should be found (the account number is the value of the process instance „*oldAccountNr*" attribute), and the value of its „*amount*" attribute should be decreased by a value of this process instance „*amount*" attribute:

```
(self as p).(((Account where
number=p.getAttribute('oldAccountNr')).(amount:=amount-
(p.getAttribute('amount') as a).((integer)a))))
```

Part of execution code that withdraws the money from the old account.

Then the new Account object should be found (the account number is the value of the process instance „*newAccountNr*" attribute) and its „amount" attribute should be increased by the value of process instance „*amount*" attribute.

```
(self as p).(((Account where
number=p.getAttribute('oldAccountNr')).(amount:=amount-
(p.getAttribute('amount') as a).((integer)a))))
```

Part of the execution code that transfers the money into a new account.

Before making any changes to a working workflow instance it should be suspended so that the state before and after applying the change is consistent. Knowing the current state of all process instances we can assume that a newly created process should start when the „*Money transfer*" is finished, and should end when the transfer operation between accounts is complete. In this case the insertion of a new process instance doesn't influence any other process, the construction of „*Ratification*" end condition ensures that it will not finish before every of its child finishes.

```
not exists(children.ProcessInstance where status <>
ProcessStatus.FINISHED)
```

Part of the „Ratification" end condition.

Example 2. *This example demonstrates the possibility of modification of a running workflow instance structure in order to meet new requirements. It shows how the proper written execution code can modify behavior of the workflow instance and how the workflow administrator can influence the behavior.*

The customer has decided to increase the credit amount just before signing the contract. In that case there is no need to restart the whole workflow instance, but only some of the processes. The activities that the workflow administrator have to perform are the following:
1. Suspend a chosen workflow instance.
2. Add new process instance „*Increase credit*" (as a child of „*Ratification*").
3. Delete process instances that are no longer required („*Verification*" - child of „*Request*", and „*Initial Formal Check*" - child of „*Analysis*").
4. Change the conditions of other involved process instances to conform to the new structure.
5. Resume workflow instance.

The purpose of the newly created „*Increase credit*" process instance is to change the „*creditAmount*" attribute of the „*ApplicationForm*" object associated with the customer.

```
(self as p).(ApplicationForm where createdFor.Customer.SSN =
p.parent.parent.ProcessInstance.getAttribute('customerSSN')).
creditAmount := 200000;;
```

Part of execution code that increases the credit amount of the application form associated with the current customer.

It also resets the „Analysis" and „Ratification" process instances in order to perform their tasks once more. It is done by changing their status to „*Waiting*" so the process monitor will include them when checking the candidates to activate. The children of this processes should also be included with this difference that their status will be changed to „*Inactive*".

```
(parent.parent.children.ProcessInstance where name =
'Analysis').status:=ProcessStatus.WAITING
```

Part of execution code that changes the status of „Analysis" process instance into WAITING.

```
((parent.parent.children.ProcessInstance where
name='Analysis').children.ProcessInstance).(status:=ProcessStat
us.INACTIVE)
```

Part of execution code that changes the status of „Analysis" children into INACTIVE.

```
parent.ProcessInstance.status:=ProcessStatus.WAITING
```

Part of execution code that changes the status of „Ratification" process instance into WAITING.

```
(parent.children.ProcessInstance).(status:=ProcessStatus.
INACTIVE)
```

Part of execution code that changes the status of „Ratification" children into INACTIVE.

Apart from process instances there are also attributes that values should be set to the previous state. This concerns the „*state*" attribute of the „*Request*" process instance, which holds the information about the current status of the application form.

```
parent.parent.ProcessInstance.setAttribute('state';'')
```

Part of execution code that sets the value of „Request" „state" attribute to an empty string.

When the „*Increase credit*" will perform the given task it's no longer needed in the system and to ensure that it will act only once, we can create such an end code that will delete it.

```
(self as p).(delete ProcessInstance where globalId =
p.globalId)
```

The next step is to get rid of unnecessary process instances such as „*Initial formal check*" and „*Verification*", because there is no need to repeat them when only the amount of the credit is changed. After that the conditions of some process instances have to be adjusted. The „*Check client rating*" will start as soon as „*Analysis*" is active instead of start after the „*Initial formal check*" finishes.

```
exists(self.parent.ProcessInstance where status =
ProcessStatus.ACTIVE)
```

Updated endcondition of „Check client rating".

All of the statements that concern „*Initial formal check*" should also be removed from „*Analysis*" end condition.

```
not exists(self.children.ProcessInstance where status <>
ProcessStatus.FINISHED)
```

Updated endcondition of „Analysis".

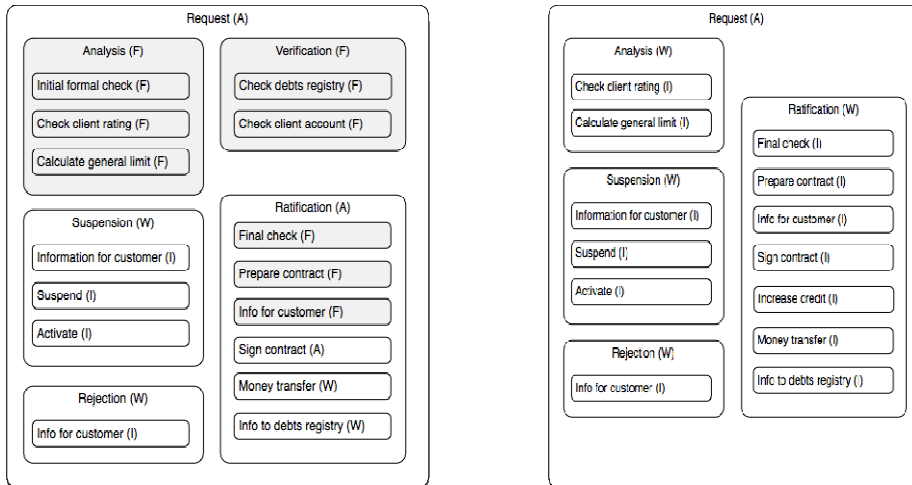The „*Ratification*" will no longer start after the „*Verification*" but as soon as the „*Analysis*" finishes.

| ```
exists(parent.children.Process
Instance where
name='Verification' and
status=ProcessStatus.FINISHED)
``` | ```
exists(parent.children.Process
Instance where name='Analysis'
and
status=ProcessStatus.FINISHED)
``` |
|---|---|
| *Part of the firecondition of „Ratification" that needs to be changed.* | *New part of the firecondition of „Ratification"* |

Now the workflow instance is ready to properly handle the updated application form and perform suitable tasks in order to complete the request.

(I) - Inactive, (W) - Waiting, (A) - Active,
(F) – Finished

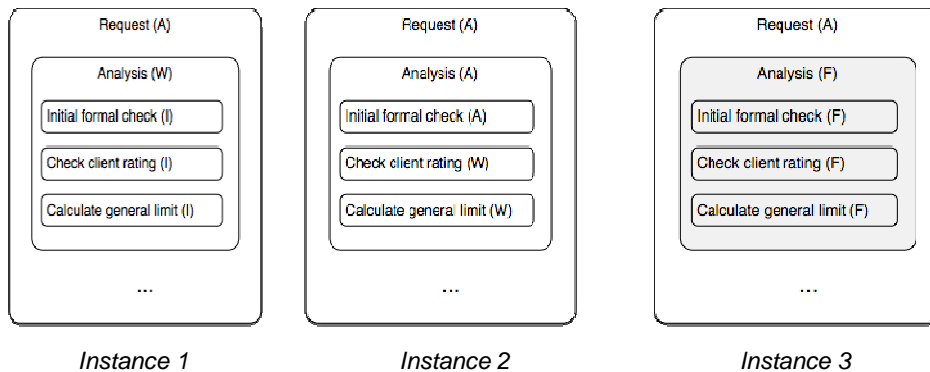*„Bank credit" workflow instance state before applying changes.*

„Bank credit" workflow instance state after applying changes.

Example 3. *It demonstrates how to apply a modification that affects several process instances of the same kind.*

The modifications are to be applied to the „*bankLimit*" attribute of the „*Calculate general limit*" process to 700000. Changing the process definition is straightforward through the GUI tool. However, changing manually all of the working instances in this way is awkward and can be error prone. For this reason we create an SBQL statement which will access the workflow environment and will do the necessary modifications. The statement finds all of the instances of the „*Calculate general limit*" subprocess, which has an „*Inactive*" or „*Waiting*" status, and then updates the value of the „*bankLimit*" attribute to the new value.

```
(ProcessInstance where name = 'Calculate general limit' and
(status=ProcessStatus.INACTIVE or
status=ProcessStatus.WAITING)).(setAttribute('bankLimit';'70000
0'))
```

SBQL statement which updates the „Calculate general limit" „bankLimit" attribute in the proper workflow instances.

| Instance 1 | Instance 2 | Instance 3 |

Update of the „Calculate general limit" „bankLimit" attribute will only apply to the instance number one and two.

Example 4. *It demonstrates how the working process instance can dynamically create new process instances.*

The commonly considered case in a process definition with parallel subprocesses is that a process instance is to be split into a fixed number of subprocesses. In many business situations the case leads to severe limitations, because the number of the subprocesses is known only during the execution of the instance. In such a case we should provide an option to create new subprocess instances dynamically, within the execution code of the process instance. To show this possibility we consider example where there is a need to send an e-mail with some information to the customer. During filling a request form a customer can provide some alternative e-mail addresses and we want to ensure that our e-mail will be delivered to all of them. The process responsible for the contact with the customer is „*Information for customer*" so we will modify it to provide required functionality. During the execution of this process instance it will create as many children process instances sending e-mails as required.

„Information for customer" process instance before running its execution code.

„Information for customer" process instance after running its execution code.

The execution code of the „*Information for customer*" creates a process instance for each of an e-mail address of a current customer:

```
(self as p).(((Customer where
SSN=parent.parent.ProcessInstance.getAttribute('customerSSN')).
email as e).(create ProcessInstance(...)))
```

Shortened execution code of „Information for customer"

12

Then we can populate the execution code for the newly created process instances in such way that it will send mail for one given address. If an e-mail was sent successfully it will create an attribute „*mailSent*" with value 1 to hold the information which will be used later to decide if the process should end or restart.

```
setAttribute('mailSent';sendMail('xxx@xxx.xx';'Dear Customer,
Your application should be corrected.'));;
```

Part of the execution code of a dynamically created process instance that sends e-mail to a given address and sets an attribute value depending on the result.

```
if(getAttribute('mailSent')=0) then (status :=
ProcessStatus.WAITING)
```

Part of the execution code of a dynamically created process instance that restarts it when sending mail has failed.

4.   Conclusion

We have presented the idea of an object-oriented declarative workflow management system that is especially prepared to achieve an important goal: the possibility of changing process instances during their run. We have discussed consequences of such a requirement and have argued that such a revolutionary feature cannot be achieved on the ground of traditional approaches to workflows based on specification of control flow graphs. Our idea allows to achieve next important features, such as mass parallelism of processes and flexible resource management. The idea is supported by the working prototype that shows its feasibility. The prototype is implemented on the basis of ODRA, an object-oriented distributed DBMS, and SBQL, a query and programming language designed and implemented for ODRA. In the paper we present comprehensive examples showing how a declarative workflow can be defined and how it can be dynamically changed. The examples have shown the feasibility of the idea of declarative workflows for real business cases.

The prototype is still under development. We are applying for next grants that will allow us to turn it into a commercial (open source) product. Independently from the ODRA-oriented project, the idea is being implemented as a proprietary commercial tool for Small and Medium Enterprises.

5.   References

1. W.M.P. van der Aalst. *Generic workflow models: How to handle dynamic change and capture management information?* Proc. 4th Intl. Conf. on Cooperative Information Systems (CoopIS'99), Los Alamitos, CA, 1999

2. 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.* OASIS, 2003.

3. M.Dąbrowski, M.Drabik, P.Habela, K.Subieta. *Object-Oriented Declarative Workflow Management System*. Editors of the Institute of Computer Science, Polish Academy of Sciences, ISBN 978-83-922508-3-8, 2009, 176 pages. http://www.ipipan.waw.pl/~subieta/SBA_SBQL/articles/ContentAndAbstract.htm

4. M.Dąbrowski, M.Drabik, M.Trzaska, K.Subieta. Dynamic Changes of Workflow Processes. Submitted to publication, April 2010.

5. C.A.Ellis. K.Keddara, GRozenberg. *Dynamic change within workflow systems*. Proc. ACM Conf. on Organisational Computing Systems (COOCS 95)

6. C.A.Ellis. K.Keddara, and J.Wainer. *Modelling workflow dynamic changes using time hybrid flow*. In Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98), 98(7), Computing Science Reports, pp.109-128. Eindhoven University of Technology, 1998

7. A.Juszkiewicz, A.Aksamit, M.Jaworski. *Workflow system based on the object-oriented DBMS ODRA*. MSc. Thesis. Polish-Japanese Institute of Information Technology, September 2009 (in Polish). http://www.ipipan.waw.pl/~subieta/prace%20magisterskie

8. Ł.Korneluk, K.Jasiołek. Web Services choreography based on an object-oriented process model. MSc. Thesis. Polish-Japanese Institute of Information Technology, September 2009 (in Polish). http://www.ipipan.waw.pl/~subieta/prace%20magisterskie

9. D.C.Ma, J.Y.-C.Lin, M.E.Orlowska. *Automatic merging of work items in business process management systems.* Proc. 10th Intl. Conf. on Business Information Systems (BIS2007), Poznań, Poland, 2007

10. M.Momotko, K.Subieta. *Dynamic Changes of Workflow Participant Assignment*. Proc. of. 6th ADBIS Conf., Vol.2: Research Communications, pp.175-184, 2002

11. M.Momotko, K.Subieta. *Business Process Query Language - a way to make workflow processes more flexible*. Proc. 8th ADBIS'04, Springer LNCS 3255, pp.306-321, 2004

12. ODRA (Object Database for Rapid Application development): Description and programmer manual. http://www.sbql.pl/various/ODRA/ODRA_manual.html , 2008

13. OMG. Business Process Modeling Notation (BPMN) specification. Final Adopted Specification. Technical Report, 2006

14. M. Reichert and P. Dadam. *ADEPTflex: Supporting dynamic changes of workflow without loosing control*. Journal of Intelligent Information Systems, 10(2), pp.93-129, 1998

15. S.Sadiq, O.Marjanovic, M.E.Orlowska. *Managing change and time in dynamic workflow processes.* Intl. Journal of Cooperative Information Systems (IJCIS), 9(1-2), 2000

16. S.Sadiq, M.E.Orlowska. *Architectural considerations in systems supporting dynamic workflow modification*. Proc. 11th Conf. on Advanced Information Systems Engineering, CAiSE99, Heidelberg, Germany, 1999

17. SBQL4Workflow Prototype Implementation. http://tomcat.pjwstk.edu.pl:8080/ProjectWorkflow/newsitem/list , May 2010

18. K.Stencel. *Semi-strong Type Checking in Database Programming Languages*. Editors of the Polish-Japanese Institute of Information Technology, 2006, 207 pages (in Polish)

19. K.Subieta. *Theory and construction of object query languages*. Editors of the Polish-Japanese Institute of Information Technology, 2005, 522 pages (in Polish)

20. K.Subieta. Stack-Based Architecture (SBA) and Stack-Based Query Language (SBQL). http://www.sbql.pl/ , 2008

21. WfMC, WorkFlow process definition interface – XML Process Definition Language. WfMC-TC-1025 (Draft 0.03a); May, 22, 2001