# Reactive workflows for visual analytics

Ioana Manolescu, Wael Khemiri, Veronique Benzaken, Jean-Daniel Fekete

# Reactive workflows for visual analytics

Ioana Manolescu[1,2] Wael Khemiri[1,2] Véronique Benzaken[2]
Jean-Daniel Fekete[1]
[1]INRIA Saclay–Île-de-France [2]LRI, Université de Paris-Sud

## 1 Introduction

The increasing amounts of electronic data of all forms, produced by humans (e.g. Web pages, structured content such as Wikipedia or the blogosphere etc.) and/or automatic tools (loggers, sensors, Web services, scientific tools etc.) leads to a situation of unprecedented potential for extracting new knowledge, finding new correlations etc. Typically, such analysis is performed by using *data visualization* techniques, to enable users to get a grasp on the data; and *data analysis programs* which perform potentially complex and/or time-consuming analysis on the data, enrich it with new dimensions, discover commonalities or clusters etc. The human expert carrying on the visual analytics task must be able to chose the range of data to analyze, trigger computations on this data, and visualize the results.

An important class of visual analytics applications has to deal with *dynamic data*, which is continuously updated (e.g. by receiving new additions) *while* the analysis process is running. For instance, we have been involved in the development of an application seeking to compute a "global picture" of INRIA research by analyzing a graph of co-publications and joint projects between INRIA researchers, within and across INRIA teams. This analysis involves some clustering algorithms, produces visual results which have interesting insight for the INRIA scientific managers, and has to proceed *while* new publications or contracts are added to the database.

We propose to demonstrate ReaViz, a *reactive workflow platform*, conceived and deployed in close connection with a database of application and workflow-related data. ReaViz enables the declarative specification of *reactive* data-driven workflows, which react in well-specified ways in the event of database updates.

This document is organized as follows. Section 2 describes our model, prototype, and demonstration sceario. We review related works in Section 3, then we conclude.

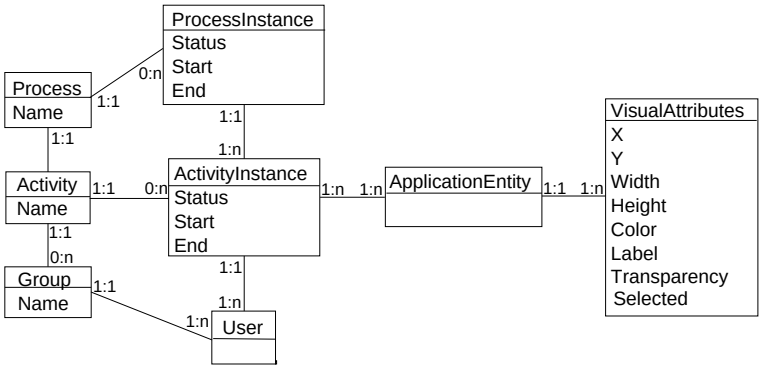


Figure 1: ReaViz data model.

# 2 A model and tool for reactive processes

In this section, we start by describing the ReaViz data model in Section 2.1, then we discuss the process model (Section 2.2). Section 2.3 outlines our prototype.

## 2.1 Data model

The data model of a ReaViz application (Figure 1) comprises three kinds of entities. *Application-dependent* entities model the data used by the specific visualization/analysis application. *Workflow-related* entities capture (to a certain extent) the definition and instances of workflows, while *visualization-related* entities capture the information items required by the data visualization modules. In Figure 1. At left, the entities *Process*, *Activity* and *Group* describe the process schemas; the latter entity corresponds to groups of users, e.g., lab operator, scientist etc. The entities *ProcessInstance*, *ActivityInstance*, and *User* are used to record specific enactments as part of running processes. An activity instance has a start date and an end date, as well as a *status* flag which can take the values: *not_started* (the activity instance is created, e.g. by a user who assigns it to another for completion, but work on the activity has not started yet), *running* (the activity instance has started but it has not finished) and *completed* (once the activity instance has finished executing. The status of a process instance can take similar values.

The *ApplicationEntity* entity refers in a generic manner to all entities which may actually be used by a given application (e.g. Person, Article etc.) The relationship between *ApplicationEntity* and *ActivityInstance* captures is to be instantiated according to the specifics of each application; it represents the way activity instances are manipulated during process execution (e.g. WrittenBy, ReviewedBy etc.)

The *VisualAttributes* entity encapsulates a set of attributes frequently used in data visualization, such as: $(x, y)$ coordinates, width, height, color, label (a string) etc. Each data item to be visualized is associated a tuple in this table, whose values are used by the data visualisation software [3].

We assume a relational enactment of this conceptual model. Thus, a relation is created

for each entity, endowed with a primary key; relationships are captured by means of association tables with the usual foreign key mechanism.

## 2.2 Process model

We consider a simple yet expressive model for describing reactive processes. We start by introducing some useful ingredients.

**Relations and queries** We consider available a set of relations denoted $R, S, T$ etc., and select-project-join queries over them.

**Procedures** A procedure is a computation unit implemented by some external, black-box software (realized in C++ or Matlab etc.). It takes as input $l$ relations which are read but not changed; $m$ relations which are read *and* changed by the procedure; and outputs data in $n$ relations.

**Delta handlers** Associated to a procedure may be *procedure delta handlers*, which given a set of deltas corresponding to updates in the procedure input relations, may be invoked to realize some compensation work, in order to reflect delta data in the procedure output. Two types of handlers may be specified. $(i)$ Compensate while the procedure runs, e.g., for a procedure which computes some data placement on a screen and must update the display to reflect the new data. $(ii)$ Compensate after the procedure has finished, e.g. for quantitative analysis on the procedure results. Each handler is a procedure, whose implementation is also opaque to our framework.

**Distributive procedures** An interesting family of procedures are those which distribute over union in all their inputs. More formally, let $X$ be one of the $R_i$ inputs of $p$, and let $\Delta X$ be the set of tuples addes to $X$. If $p$ is distributive then:

$$p(R_1, \ldots, X \cup \Delta X, \ldots, T_m) = p(R_1, \ldots, X, \ldots, T_m) \cup p(R_1, \ldots, \Delta X, \ldots, T_m)$$

There is no need to specify delta handlers for procedures which distribute over the union, since the procedure itself can serve as handler.

**Expressions** Queries are the simplest expressions. More complex expressions can be obtained by calling a procedure $p$, and retaining only its $j$-th output table:

$$e ::= Q \mid p(e_1, e_2, \ldots, e_n, T_1^w, T_2^w, \ldots, T_p^w).t_j, 1 \leq j \leq m$$

Observe that the first $n$ call parameters are expressions themselves, allowing complex expression composition.

**Activities** Building blocks of our processes, activities are specified as follows:

$$a ::= upd(R) \mid (S_1, S_2, \ldots, S_n) \leftarrow p(e_1, e_2, \ldots, e_n)$$

where $upd(R)$ is a declarative update of a table $R$, specified by a SQL statement. An activity may also consist of invoking a procedure $p$ by providing appropriate input parameters, and retaining the outputs in a set of tables.

**Processes** The process structure we consider is close to the widely adopted Workflow Management Coalition Model [7]. In a process, activities are combined by means of sequence, conjunctive split-join, or split-join, and conditional execution.

**Reactive processes** A *reactive process* $RP ::= R^*, p^*, P, CA^*$ can now be defined as a 4-tuple consisting of: a set of relations; a set of procedures; a process; and a set of *compensating actions*. A compensating action $CA$ specifies what should be done in the event that a set of tuples, denoted $\Delta R$, are added to an application-dependent relation $R$ *during the execution of a process instance*. Let $t_{\Delta R}$ be the moment when $\Delta R$ was received. Several options are possible.

**1.** Ignore $\Delta R$ for the execution of all *processes* which had started executing before $t_{\Delta R}$. The data will be added to $R$, but will only be visible for process instances having started after $t_{\Delta R}$. This recalls a process-granularity locking model, where each process operates on exactly the data which was available when the process started. We consider this to be the default behavior for all updates to the relations part of the application data model.
**2.** Ignore $\Delta R$ for the execution of all *activities* which had started executing (whether they are finished or not) before $t_{\Delta R}$. However, for a process already started, instances of a specific activity which start after $t_{\Delta R}$ may also use this data. This resembles some activity-level locking model. Observe that no compensation is performed in this case! $\Delta R$ is silently made available to a specific yet-to-start activity.
**3.** Execute compensation work for all the terminated instances of a given activity. This can apply to activity instances whose process instances have terminated and/or to those whose process instances are running.
**4.** Execute compensation work for all the running instances of a given activity.

A compensation action is thus defined by: a relation $R$, an activity $a$, and a specific action of one of the four types above. To support such actions, our system assigns *creation timestamps* to all tuples from application-dependent relations.

## 2.3 The ReaViz prototype

ReaViz is implemented in Java, on top of Oracle 11g. A reactive process is specified as an XML file, which Reaviz compiles into the corresponding *Process* and *Activity* tuples. Users may *instantiate* a ReaViz process using a control interface, which also allows them to move from one activity to another as they advance the process. All along the process instance execution, ReaViz issues the necessary data manipulation statements to (*i*) record in the database the advancement of process and activity instances, (*ii*) evaluate on the database queries and updates, allow external procedures to read and update the application-driven entities, (*iii*) record the connections between users and application instances, and between application instances and application-dependent data. To implement compensations, the addition of $\Delta R$ tuples to the relation $R$ is captured by a database trigger, automatically derived from the process specification. The trigger invokes ReaViz which performs the

necessary actions, by a combination of database manipulations (queries or updates, which may use the activity, process, and data timestamps), external handler invocation, and execution of ReaViz control code. The coupling between ReaViz and the InfoViz data visualization toolkit [3] is made via the VisualAttributes relation.

**Demonstration scenario** We will demonstrate ReaViz using the INRIA clustering application mentioned in the introduction, as well as an online Wikipedia author importance computation (if an Internet connection is available on site).

# 3 Related works and conclusion

Research in data visualization has produced several interactive platforms for data visualization [3, 4]. Such platforms do not facilitate the integration of data analysis programs (or procedures), and do not address dynamically changing data. Workflow specification and deployment is a rife area [7, 6, 1]; more recently, *scientific workflow* platforms have received significant attention, e.g.Taverna (taverna.sourceforge.net) or Kepler (kepler-project.org). Scientific workflows incorporate data analysis programs as a native ingredient, and are meant to be specified by scientists, their end users. However, they are not well adapted to our problem, because the relationship between the data and the process specification is not well formalized, hindering the definition of compensating actions with clean semantics. The focus in [2] is on adapting process instances to changes in the process structure, which is different from our problem.

The ReaViz approach is described in more detail in [5]. We view our work as a first step in combining data management and visual analytics, to build more powerful and flexible platforms for handling complex data.

# References


[1] M. Brambilla, S. Ceri, P. Fraternali, and I. Manolescu. Process modeling in web applications. *ACM Trans. Softw. Eng. Methodol.*, 15(4), 2006.

[2] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *Data and Knowledge Engineering*, pages 438–455. Springer Verlag, 1998.

[3] J.-D. Fekete. The InfoVis toolkit. In *IEEE Symposium on Info. Visualization*, 2004.

[4] J. Heer, S. K. Card, and J. A. Landay. Prefuse: a toolkit for interactive information visualization. In *SIG-CHI*, 2005.

[5] I. Manolescu, W. Khemiri, V. Benzaken, and J.-D. Fekete. Reactive workflows for visual analytics. Tech. report, 2009.

[6] Business Process Execution Language for Web Services version 1.1. Available at http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[7] The Workflow Management Coalition Reference Model. Available at http://www.wfmc.org/reference-model.html.