Data-intensive interactive workflows for visual analytics

Wael Khemiri

Université Paris-Sud 11 **École doctorale d'informatique de Paris-Sud**

Data-intensive interactive workflows for visual analytics

(Données en masse et workflows interactifs pour la visualisation analytique)

THÈSE

présentée et soutenue publiquement le 12 Décembre 2011

pour l'obtention du grade de

Docteur de l'Université Paris-Sud 11

(spécialité informatique)

par

Wael KHEMIRI

Composition du jury

Rapporteurs :	Dominique Laurent	Professeur, Université de Cergy Pontoise
	Guy Melançon	Professeur, LaBRI, Université Bordeaux I
Examinateurs :	Alain Denise	Professeur, Université de Paris-Sud 11
	Thérèse Libourel	Professeur, LIRMM, Université Montpellier II
Directeurs :	Véronique Benzaken	Professeur, Université de Paris-Sud 11
	Jean-Daniel Fekete	Directeur de recherche, INRIA Saclay Île de France
	Ioana Manolescu	Directeur de recherche, INRIA Saclay Île de France

Sans transmission de la pensée,
le langage n'est qu'une terre morte. [Ibn Khaldoun]

Remerciements

Je remercie en premier lieu tous les membres du jury de m'avoir fait l'honneur de leur participation à ma soutenance de thèse. Je remercie Monsieur Dominique Laurent et Monsieur Guy Melançon d'avoir accepté de rapporter sur mon travail de thèse. Je vous remercie pour l'intérêt que vous avez porté à ce travail et pour vos remarques pertinentes. Je remercie également Monsieur Alain Denise et Madame Thérèse Libourel de m'avoir fait l'honneur d'être présents à ce jury de thèse.

Je tiens à remercier très chaleureusement mes directeurs de thèse Véronique Benzaken, Jean-Daniel Fekete et Ioana Manolescu pour leur disponibilité et les moyens qu'ils m'ont accordés. Je suis également reconnaissant pour la richesse des enseignements scientifiques qu'ils m'ont transmis. Je vous remercie pour ces trois années de soutien et de patience pendant lesquelles vous avez guidé mes premiers pas de chercheur. Je retiendrai vos précieux conseils qui m'ont permis de prendre confiance en mon travail, de nourrir ma créativité, d'affiner mon sens critique et surtout d'affirmer mon esprit de contradiction.

Je remercie Pierre-Luc Hémery de l'équipe Aviz, pour toute l'aide qu'il m'a fournie dans l'implémentation des scénarios d'utilisations appliqués à EdiFlow. Merci pour ton coup de main.

Je remercie tous mes chers collègues des équipes Leo et Aviz que j'ai eu le plaisir de côtoyer durant ces quelques années. Un grand merci pour les échanges aussi bien culturels qu'académiques, sans oublier les moments de détente. Je remercie par ailleurs remercier tous mes amis de Tunis et Paris pour vos encouragements. Je vous remercie pour tous les moments de bonheur et de délire que j'ai partagé avec vous.

Je remercie également tous les membres de l'équipe Proval qui m'ont ouvert leurs bureaux pour terminer la rédaction de ma thèse dans de bonnes conditions.

Un grand merci à tous les membres de la famille SASSI et la famille GABSI qui m'ont considérés comme l'un des leurs. Merci pour tous les moments de bonheur et de délire que nous avons passé ensemble.

Finalement cette thèse ne serait pas sans le soutien des très proches. Je remercie tout particulièrement mes parents Houcine et Zohra, mes frères Ahmed et Fakhreddine et ma soeur Rihab. Votre soutien et votre amour m'ont aidé à dépasser les moments difficiles de ma vie. Je vous exprime mon éternelle reconnaissance et mon indéfectible attachement. J'espère que ce travail réalisé grâce à vous sera à la hauteur de la confiance et de l'apport que vous m'avez donné. Un grand merci également à ma chère Nesrine pour sa patience, son sens de l'écoute et son soutien tout au long de ces dernières années.

À tous, un grand merci, vous m'êtes très chers.

À la mémoire de ma grand-mère Fatma
et mes grands-pères, Hamadi et Mohamed.

Résumé

L'expansion du World Wide Web et la multiplication des sources de données (capteurs, services Web, programmes scientifiques, outils d'analyse, etc.) ont conduit à la prolifération de données hétérogènes et complexes. La phase d'extraction de connaissance et de recherche de corrélation devient ainsi de plus en plus difficile. Typiquement, une telle analyse est effectuée en utilisant les outils logiciels qui combinent: *des techniques de visualisation*, permettant aux utilisateurs d'avoir une meilleure compréhension des données, et *des programmes d'analyse* qui effectuent des opérations d'analyses complexes et longues.

La visualisation analytique (*visual analytics*) vise à combiner la visualisation des données avec des tâches d'analyse et de fouille. Etant donnée la complexité et la volumétrie importante des données scientifiques (par exemple, les données associées à des processus biologiques ou physiques, données des réseaux sociaux, etc.), la visualisation analytique est appelée à jouer un rôle important dans la gestion des données scientifiques.

La plupart des plateformes de visualisation analytique actuelles utilisent des mécanismes en mémoire centrale pour le stockage et le traitement des données, ce qui limite le volume de données traitées. En outre, l'intégration de nouveaux algorithmes dans le processus de traitement nécessite du code d'intégration ad-hoc. Enfin, les plate-formes de visualisation actuelles ne permettent pas de définir et de déployer des processus structurés, où les utilisateurs partagent les données et, éventuellement, les visualisations.

Ce travail, à la confluence des domaines de la visualisation analytique interactive et des bases de données, apporte deux contributions. (*i*) Nous proposons une architecture générique pour déployer une plate-forme de visualisation analytique au-dessus d'un système de gestion de bases de données (SGBD). (*ii*) Nous montrons comment propager les changements des données dans le SGBD, au travers des processus et des visualisations qui en font partie. Notre approche permet à l'application de visualisation analytique de profiter du stockage robuste et du déploiement automatique de processus à partir d'une spécification déclarative, supportés par le SGBD.

Notre approche a été implantée dans un prototype appelé EdiFlow, et validée à travers plusieurs applications. Elle pourrait aussi s'intégrer dans une plate-forme de workflow scientifique à usage intensif de données, afin d'en augmenter les fonctionnalités de visualisation.

Mots-clés: Visualisation analytique, systèmes workflow, gestion dynamique des données

Abstract

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 programs or analysis tools etc.) leads to a situation of unprecedented potential for extracting new knowledge, finding new correlations, or simply making sense of the data.

Visual analytics aims at combining interactive data visualization with data analysis tasks. Given the explosion in volume and complexity of scientific data, e.g., associated to biological or physical processes or social networks, visual analytics is called to play an important role in scientific data management.

Most visual analytics platforms, however, are memory-based, and are therefore limited in the volume of data handled. Moreover, the integration of each new algorithm (e.g., for clustering) requires integrating it by hand into the platform. Finally, they lack the capability to define and deploy well-structured processes where users with different roles interact in a coordinated way sharing the same data and possibly the same visualizations.

This work is at the convergence of three research areas: information visualization, database query processing and optimization, and workflow modeling. It provides two main contributions: (*i*) We propose a generic architecture for deploying a visual analytics platform on top of a database management system (DBMS) (*ii*) We show how to propagate data changes to the DBMS and visualizations, through the workflow process. Our approach has been implemented in a prototype called EdiFlow, and validated through several applications. It clearly demonstrates that visual analytics applications can benefit from robust storage and automatic process deployment provided by the DBMS while obtaining good performance and thus it provides scalability.

Conversely, it could also be integrated into a data-intensive scientific workflow platform in order to increase its visualization features.

Keywords: Visual analytics, scientific workflow systems, dynamic changes

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 The thesis in a nutshell

Current scientific data management applications involve huge and increasing data volumes. Data can be numeric, e.g., output of measure instruments, textual, e.g., corpora studied by social scientists which may consist of news archives over several years, structured as, in the case of astronomy or physics data, or highly unstructured as in the case of medical patient files. Data, in all forms, is increasingly large in volume, as a result of computers capturing more and more of the work scientists used to do based on paper, and also as a result of better and more powerful automatic data gathering tools, e.g., space telescopes, focused crawlers, archived experimental data (mandatory in some types of government funded research programs) and so on.

The availability of such large data volumes is a gold mine for scientists which may carry research based on this data. Today's scientists, however, more often than not rely on proprietary, ad-hoc information systems, consisting perhaps of a directory structure organized by hand by the scientist, a few specialized data processing applications, possibly a few scripts etc [37]. In addition, the increasing amounts of this data of all forms leads to a situation of unprecedented potential for extracting new knowledge, finding new correlations, or simply making sense of the data. Typically, such analysis is performed by using software tools which combine: *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 be analyzed, trigger analysis or various computations on this data, and visualize the results. Visualizing results leads to understanding which parts of the data should be further analyzed, which could be ignored etc.

A sample application would be: Social scientists are currently interested in analyzing online social networks such as Wikipedia, or the World Wide Web Consortium (W3C) standard-writing community, where new forms of group organization and interaction emerge.

In the case of Wikipedia, visualizing the hypertext network that connects articles together requires accessing the hypertext data, computing some "shape" to visualize the network and using visualization tools to navigate the representation effectively. Using a standard XML serialization, the French Wikipedia corpus is quite large (4.5 GB in its compressed format). Therefore, efficient data support for analysis and computation services is crucial.

Building expressive and efficient platforms for scientific data management raises several challenges as we explain further. Once such a platform is available, it could be developed and extended to help scientists in Digiteo, the surrounding scientific campus, and in other labs (e.g., the social sciences laboratories with whom we already interact) advance in their work and compete with their international colleagues. Meanwhile this would result in advancing the state of the art in our respective fields, where service to the scientific communities is perceived as an outstanding priority [37].

Visual analytics is on the border between visualization, HCI (Human Computer Interaction), databases, data analysis and data-mining [77]. Its aim is to enable users to closely interact with vast amounts of data using visual tools. Thanks to these tools, a human may detect phenomena or trigger detailed analysis which may not have been identifiable by automated tools alone. Visual analytics tools routinely include some capacity to mine or analyze the data; however, most applications require specific analysis functions.

Though, most current visual analytics tools have some conceptual drawbacks. Indeed, they rarely rely on persistent databases (with the exception of [36]). Instead, the data is loaded from files or databases and is manipulated directly in memory because smooth visual interaction requires redisplaying the manipulated data 10-25 times per second. Standard database technologies do not support continuous queries at this rate; at the same time, ad-hoc in-memory handling of classical database tasks (e.g., querying, sorting) has obvious limitations. We argue connecting a visual analysis tool to a persistent database management system (DBMS) has many benefits such as scalability, persistence, distribution and management capabilities of the data.

The visual analytics process can be modeled as a workflow process. Workflow systems are generally based on database management systems. Thus, the integration of a DBMS in a visualization platform must take into account the following prevalent aspects in today's visual analytics applications:

- Convergence of visual analytics and workflow: current visual analytics tools are not based on workflow (process) models. This fits some applications where datasets and tasks are always exploratory and different from one session to the next. Several visual analytics applications however, require a recurring process, well supported by a workflow system. The data processing tasks need to be organized in a sequence or in a loop; users with different roles may need to collaborate in some application before continuing the analysis. It also may be necessary to log and allow inspecting the advancement of each execution of the application. (Scientific) workflows platforms

allow such automation of data processing tasks. They typically combine database-style processing (e.g., queries and updates) with the invocation of external functions, implementing complex domain-dependent computations. Well-known scientific workflow platforms include Kepler [57], Taverna [51], or Trident [85]. These systems build on the experience of the data and workflow management communities; they could also benefit from a principled way of integrating powerful visualization techniques.

- Handling dynamic data and change propagation: 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; conversely, processes (or visualization) may update the data. The possible interactions between all these updates must be carefully thought out, in order to support efficient and flexible applications.

However, the use of a database also has several shortcomings for visual analytics applications:

- No guarantee on response time. For many applications, each microsecond account, response times is of paramount importance. The persistent databases store data on disk which can make a bottleneck associated with writing data. This response time increases with the number of I/O operations on disk.

- Not convenient management of notifications. In the current DBMS, databases do not play the role of notifier. Indeed, it is important that the user query the database and it makes him the result. Despite the available mechanisms such as triggers, notification is not a task ahead of DBMS.

- Incompatible isolation modes with long calculations. The SQL standard defines four levels of transaction isolation to prevent phenomena occur when concurrent transactions: (i) Uncommitted Read, (ii), Committed Read, (iii) Repeatable Read and (iv) Serializable. These logic levels can overcome various transactional anomalies that can be induced by the nature of the underlying lock (dirty reads, non-repeatable reads, phantom reads). However, all these isolation modes are not compatible with long transactions. Indeed, several scientific applications present the challenge of having very long calculations which must not be interrupted.

1.2 Contributions

We briefly sum up our contributions. We begin with the contributions related to the scientific workflow management, and then we give the contributions related to the visual analytics field in order to provide solutions to the points previously mentioned.

1.2.1 Scientific workflow management

In this thesis, we propose contributions related to the scientific workflow management field.

- *Generic architecture for workflow systems.* We present a generic architecture for integrating a visual analytics tool and a DBMS. The integration is based on a core data model, providing support for (i) visualizations, (ii) declaratively-specified, automatically deployed workflows, and (iii) incremental propagation of data updates through complex processes, based on a high-level specification.

- *Efficient communication protocol.* We present a simple yet efficient protocol for swiftly propagating changes between the DBMS and the visual analytics application. This protocol is crucial for the architecture to be feasible. Indeed, the high latency of a "vanilla" DBMS connection is why today's visual analytics platforms do not already use DBMSs.

- *Implementing a prototype.* We have fully implemented our approach in a barebones prototype called EdiFlow, and de facto ported the InfoVis visual analytics toolkit [33] on top of a standard Oracle server.

- *A well-specified execution semantic.* EdiFlow implements a process model inspired by the basic Workflow Management Coalition model. We use a set of variables, constants and attribute names, a set of atomic values, and a set of atomic data types. The main innovative ingredient of our model is the treatment of data dynamics, i.e., the possibility to control which changes in the data are propagated to which part(s) of which process instances.

- *Dynamic changes.* A mechanism to manage dynamic data efficiently, relying on standard features of persistent DBMS system. The main feature of EdiFlow is to react to dynamic data changes; reaction management complicates the logic of the system but also its implementation.

- *An isolation system.* A "snapshot isolation" semantic built on top of DBMS to allow modules to run isolated from changes made by other modules or from the external environment until they are ready to handle them. Applications may require different levels of sharing (or, conversely, of isolation) among concurrent activities and processes. EdiFlow implements various degrees of isolation between concurrent processes operating on top of the same database: (i) Process and activity-based isolation and (ii) Timestamp-based isolation.

1.2.2 Visualizations and interactions

Then, we present in the following the contributions which are related to the visual analytics field allowing to take into account the lacks described previously,

- *Unification mechanisms for dynamic data management and interaction management.* Interaction is implemented as data changes. According to the Information Visualization Reference Model, the interaction can impact the view, the visualization, the data and, in the case of visual analytics, it can also impact any analytical module. Since all of the modules read their input and parameters from DMBS tables, changing a parameter boils down to changing a value in a table and the propagation will be done by EdiFlow.

- *Database cache mechanism.* The use of a database cache mechanism to achieve the speed required for visualization and computation. The mechanism used to collect the updates for EdiFlow has been extended to manage cached tables that are kept consistent with tables in the DBMS. When a DBMS table is cached in an EdiFlow module, the in-memory table is kept consistent with the on-disk table in both directions: changing the in-memory table propagates to the on-disk table and vice-versa: changes done externally to the DBMS are propagated to the in-memory cache, potentially invalidating entries.

- *Polylithic visualization model.* The definition and implementation of a polylithic visualization model [12] relying on a persistent table that allows for rendering on any display (from small screen to wall-sized display) and collaboration. EdiFlow implements visualizations using a data table that contains graphic information, such as position (x, y), shape, color, and label.

1.3 Publications

In the context of this PhD work, the following articles were published.

1. Véronique Benzaken and Jean-Daniel Fekete and Pierre-Luc Hémery and Wael Khemiri and Ioana Manolescu: "EdiFlow: data-intensive workflows for visual analytics", in ICDE 2011

   In this article, we have described the design and implementation of EdiFlow, the first workflow platform aimed at capturing changes in data sources and launching a repair mechanism.

2. Jean-Daniel Fekete and Wael Khemiri and Ioana Manolescu and Véronique Benzaken: "Provenance Management in the EdiFlow VA Workflow", in Analytic Provenance workshop, Vancouver, 2011

   This article addresses the problem of managing provenance over EdiFlow since the workflow manager already maintains rough provenance information from the structure of the modules. We were interested on improving this information and on the feedback from the community.

3. Véronique Benzaken and Jean-Daniel Fekete and Pierre-Luc Hémery and Wael Khemiri and Ioana Manolescu: "EdiFlow: data-intensive workflows for visual analytics", in BDA 2010

   In this article, we presented a generic architecture for integrating a visual analytics tool and a DBMS. The integration is based on a core data model, providing support for (i) visualizations, (ii) declaratively-specified, automatically-deployed workflows, and (iii) incremental propagation of data updates through complex processes, based on a high-level specification.

4. Ioana Manolescu and Wael Khemiri and Véronique Benzaken and Jean-Daniel Fekete. "Reactive workflows for visual analytics", in BDA 2009 (software demonstration)

   In this article, we proposed to demonstrate a reactive workflow platform conceived and deployed in close connection with a database of application and workflow-related data. This platform enables the declarative specification of reactive data-driven workflows, which react in well-specified ways in the event of database updates. We presented the INRIA clustering application using our platform.

5. Ioana Manolescu and Wael Khemiri and Véronique Benzaken and Jean-Daniel Fekete. "ReaViz: Reactive workflows for visual analytics", in Data Management & Visual Analytics workshop, Berlin, 2009 (position paper)

   This article is a position paper aimed at presenting our work to the visual analytics community and get a feedback on it.

1.4 Thesis outline

This thesis is organized as follow:

I. In the second part, we study the state of art of both workflows and visual analytics fields.

   Chapter 2 provides a brief introduction to the key elements of the workflow field. We limit the description of the workflow definitions to those addressed in our contributions.

   Chapter 3 presents the main works in the visual analytics field. We start by presenting the main features and application domains related to this field; then, we present the various issues related to our work.

II. Part III provides the main contributions of this thesis.

   Chapter 4 provides the design and implementation of EdiFlow, the first workflow

platform aimed at capturing changes in data sources and launching a repair mechanism. EdiFlow unifies the data model used by all of its components: application data, process structure, process instance information and visualization data. It relies on a standard DBMS to realize that model in a sound and predictive way.

Chapter 5 addresses the problem of scalability in visual analytics. EdiFlow implements a novel mechanism for managing changes in data sources. Using this mechanism, EdiFlow provides a unified model that is both control-driven and data-driven.

III. Part IV concludes this dissertation.

Part II

State of the art

Chapter 2

Workflow systems and models

Contents

2.1 Introduction

This chapter provides the reader with background knowledge about the relevant technologies to this PhD work. In particular, an overview of workflow management and databases is presented. The remainder of this chapter is structured as follows. In Section 2.2, we provide an overview to workflow environment. In Section 2.3, a special type of workflow systems is provided: the scientific workflows. Section 2.4 focuses on the relation between workflow execution and database management systems. Section 2.5 concludes this chapter.

2.2 Workflow systems

2.2.1 Workflow system definition

The term "workflow" refers to the automation of processes. A process is defined as a coordinated sequence of processing tasks of information, following a certain pattern and leading to a definite result (for example, medical monitoring, reimbursement of expenses ...) During the execution of process, information, forms or documents are shared or are transmitted from one workstation to another for processing.

The design of a workflow is based on three interacting models: the organizational model, the information model and process model:

- The organizational model: structure resources, i.e humans or machines that can perform a task, roles;
- The information model: describes the structure of forms, documents and data that are used and produced by a workflow;
- The process model: defines the tasks components, coordination, information and actors involved in each task.

Informally, a workflow is a cooperative work involving a limited number of people to accomplish in a limited time, tasks centered around a procedure defined and with an overall objective. We can then think more specifically of the workflow as an object that can be described by a descriptive language in a file which can then be interpreted and executed by workflow management system.

The Workflow Management Coalition (WfMC) as the acknowledged professional association strives to reach standardization and also conducts widespread marketing to spread the concepts and methodologies linked to workflow technology.

A workflow is composed of three main components: Activity, Process and Instance.

- Activity: A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation,

or a workflow (automated) activity. A workflow activity requires human and/or machine resources(s) to support process execution; where human resource is required an activity is allocated to a workflow participant [5].

- Process: The representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc [5].

- Instance: The representation of a single enactment of a process, or activity within a process, including its associated data. Each instance represents a separate thread of execution of the process or activity, which may be controlled independently and will have its own internal state and externally visible identity, which may be used as a handle, for example, to record or retrieve audit data relating to the individual enactment [5]. Instances are process instances, which are representations of a single enactment of a process, or activity instances, which are representations of an activity within a single enactment of a process (within a process instance). Process instances are often called cases.

2.2.2 Workflow patterns

To compare the expressiveness of workflow languages, a set of 43 workflow control patterns has been identified in [69]. These patterns address the behavioral workflow perspective. They are classified into basic patterns and advanced patterns. The basic workflow control patterns are supported by most workflow languages. In the following, we present the patterns that will be used in this thesis:

Sequence pattern Sequence refers to a set of activities in the workflow process that are executed in sequential order. An activity in a workflow process is enabled after the completion of a preceding activity in the same process.

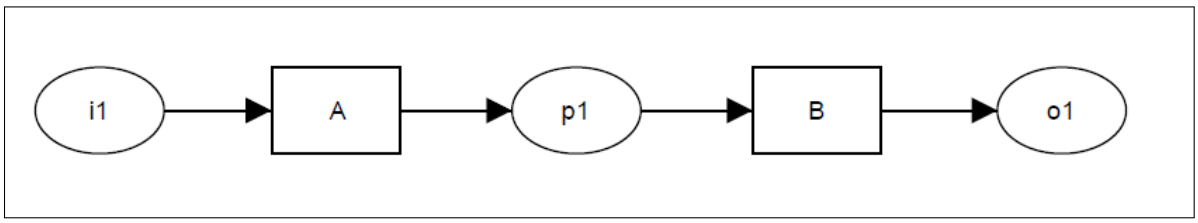

Figure 2.1: Sequence pattern.

Parallel split pattern Parallel split refers to a point in the workflow process where a single thread of control splits into multiple threads of control, which can be executed in parallel. The divergence of a branch into two or more parallel branches each of which execute concurrently.

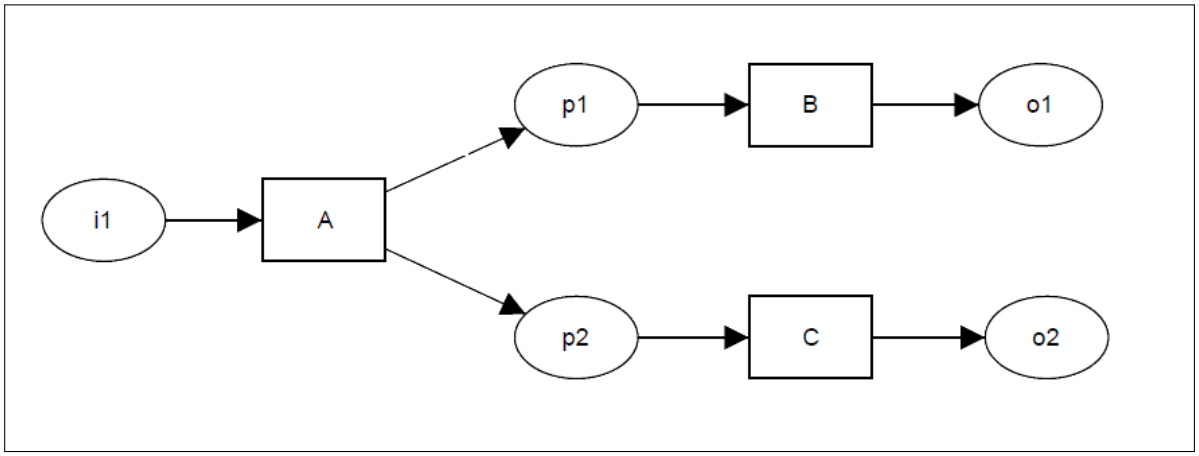

Figure 2.2: Parallel split pattern.

Synchronization pattern Synchronization refers to a point in the workflow process where multiple parallel activities converge into one single thread of control, thus synchronizing multiple threads. The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.

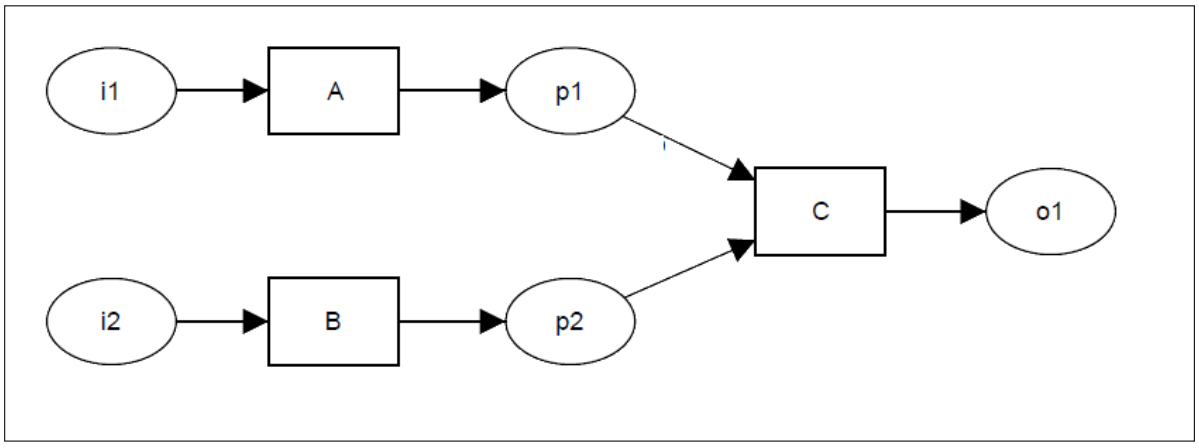

Figure 2.3: Synchronization pattern.

Exclusive choice pattern Exclusive choice chooses one execution path from several alternatives based on some decision. The divergence of a branch into two or more branches. When the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on the outcome of a logical expression associated with the branch.

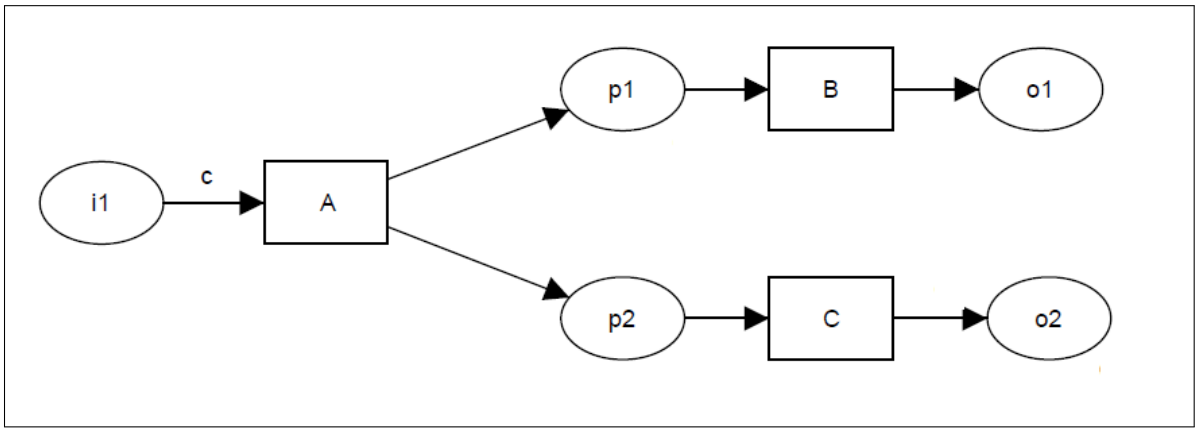

Figure 2.4: Exclusive choice pattern.

Simple merge pattern Simple merge refers to two or more alternative branches that come together without synchronization. The convergence of two or more branches into a single subsequent branch. Each enablement of an incoming branch results in the thread of control being passed to the subsequent branch

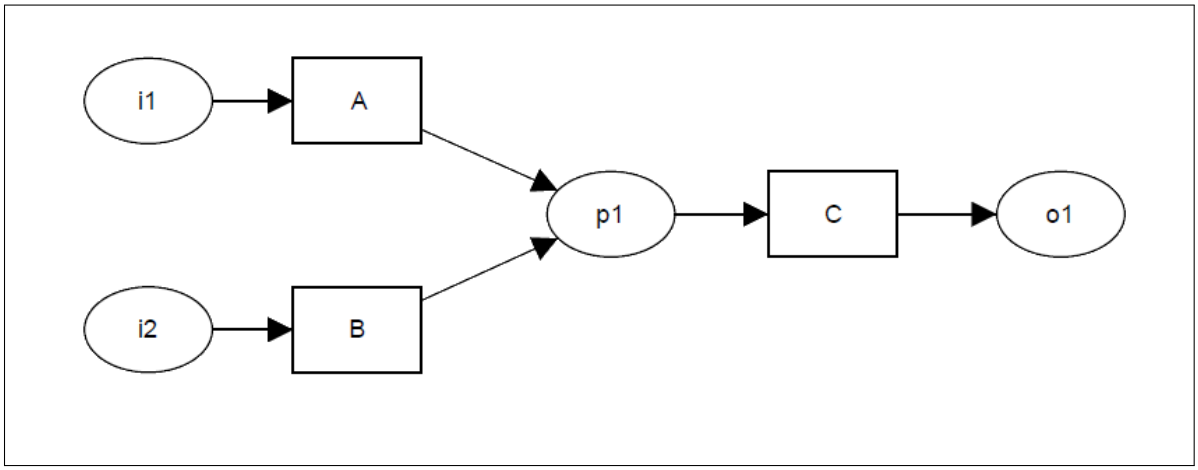

Figure 2.5: Simple merge pattern.

Multi-choice pattern Multi-Choice refers to two or more alternative branches that come together without synchronization. The divergence of a branch into two or more branches. When the incoming branch is enabled, the thread of control is passed to one or more of the outgoing branches based on the outcome of distinct logical expressions associated with each of the branches.

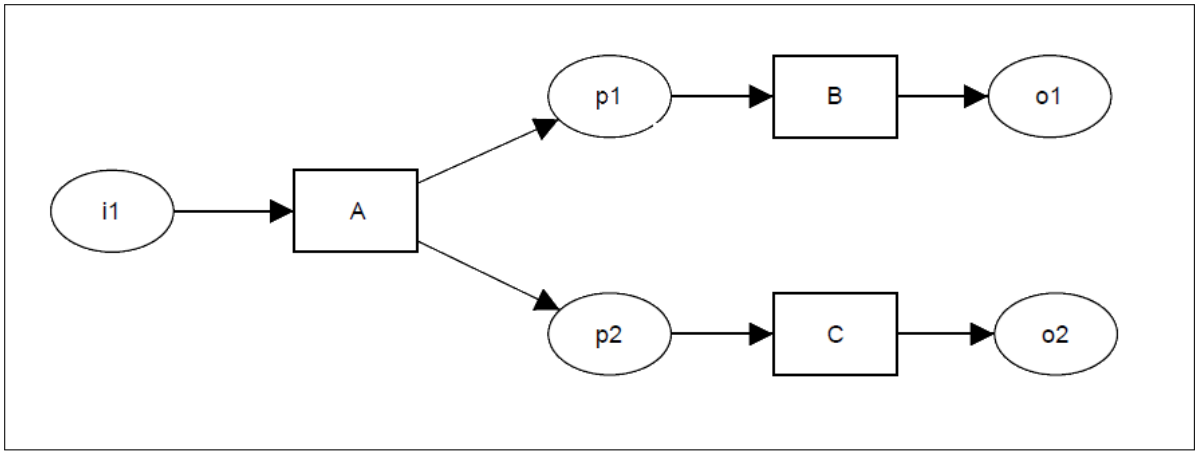

Figure 2.6: Multi-choice pattern.

In order to be able to describe workflows, several descriptions can be used. Among them, we identify three main classes of workflow languages. We describe below each of these classes.

2.2.3 Workflow description languages

There are several workflow description languages which are used to manage business procedures, resource coordination and oversight of the progress. In [59], workflow languages are classified based on the underlying methodologies into the following four classes. We describe the main features of each of these languages.

Graph-based languages They are the most traditional and intuitive way for specifying workflows. They use directed graphs, whereby nodes represent activities and edges represent the control and/or data flow.

The structure of the graph allows covering different aspects of the workflow. They have a nested structure as each node in the graph can be considered as a graph. In addition, this model has two types of oriented edges:

- Control-flow edges: Control flow edges belong to the behavioral aspect. In particular, a control flow edge $i \rightarrow j$ specifies that activity j can start only after activity i has terminated.

- Data-flow edges: Data flow edges belong to the informational aspect. They specify data dependencies between workflow activities; if activity i generates data which is required as input to activity j then there is a data flow edge connecting these activities.

The main shortcoming of the graph-based languages is the specification of informational and operational aspects which requires additional specifications, like data types of transferred data objects or information on the execution environment of application programs.

Petri Net-based languages A Petri Net is one of several mathematical representations of discrete distributed systems. It is represented by a bipartite graph (composed of two types of nodes which are not connected to two nodes of the same type) oriented (consisting of arc(s) that makes sense) connecting places and transitions (the nodes) [78]. Two places cannot be linked together, nor two transitions. Places may contain tokens, typically available resources. We distinguish several types of Petri Nets. In [13], the authors classify them in three different levels:

- Level 1: Petri Nets characterized by places that can represent Boolean values; i.e., a place is marked by at most one unstructured token. Examples of level 1 nets are Condition/Event systems and State Machines.
- Level 2: Petri Nets characterized by places that can represent integer values; i.e., a place is marked by a number of unstructured tokens. Examples of level 2 nets are Place/Transition Nets.
- Level 3: Petri Nets characterized by places that can represent high-level values; i.e., a place is marked by a multiset of structured tokens. Examples of level 3 nets are Colored Petri Nets.

Script-based languages Workflow programming (or script) languages are often used in projects where system development issues play a major role. Workflow programming languages are either used directly to specify workflow models or they are used as an internal representation with the aim of the controlled execution by a workflow management system or to allow the import and export of workflow models.

Script-based languages are closely related to workflow management system development. Workflow languages can also have multiple representations. For instance, there may be a graphical language for the specification of workflow models, which is translated into a script language, to be processed by a workflow management system.

State charts are an extension of finite state machines [40; 46; 50]. In this model, a transition moves the workflow from one state to another. State charts are complemented by activity charts to describe the events that trigger state transitions.

2.2.4 Workflow management systems

Once our workflow is specified in a given language, the workflow management system is in charge of implementing the whole process. A Workflow Management System is a complete system that is used to define, manage and execute procedures by implementing programs whose execution order is defined in a pre-representation. In addition, many management systems also provide the opportunity to measure and analyze the execution of the process so that continuous improvements can be made. Such improvements may be short-terms (e.g., reallocation of tasks to better balance the workload at any point in time) or long-term (e.g., redefining portions of the workflow process to avoid bottlenecks in the future). Most workflow systems also integrate other systems such as document management systems, databases, e-mail, office automation products, Geographic Information Systems, production applications, etc. This integration provides structure to a process which employs a number of otherwise independent systems. It can also provide a method (such as a project folder) for organizing documents from various sources.

The WfMC provides a Reference Model (cf. Figure 2.7) for designing workflow management systems. This model includes all requirements that must be included in a workflow management system and its interfaces.

The core component in the Reference Model is the Workflow Enactment Service which is responsible for creation, management and execution of workflow process instances according to process definition produced by process definition tools. The workflow enactment software consists of one or more workflow engines, which are responsible for managing all, or part of the execution of individual process instances. It interprets a process definition coming from the definition tool and coordinates the execution of workflow client applications for human tasks and invoked applications for computerized tasks. This may be set up as a centralized system with a single workflow engine responsible for managing all process execution or as a distributed system in which several engines cooperate, each managing part of the overall execution.

Process Definition Tools are used to analyze, model, describe and document a business process. The outcome of this process modeling and design activity is a process definition which can be interpreted at run time by the workflow engine(s) within the enactment service.

Workflow Client Applications involve the activities which require interaction with the human resources. In the workflow model, interaction occurs between the client application and a particular workflow engine through a well-defined interface embracing the concept of a worklist - the queue of work items assigned to a particular user by the workflow engine. Invoked Applications are the programs invoked by the workflow management system.

Administration and Monitoring Tools are used to track workflow process events during workflow process execution.

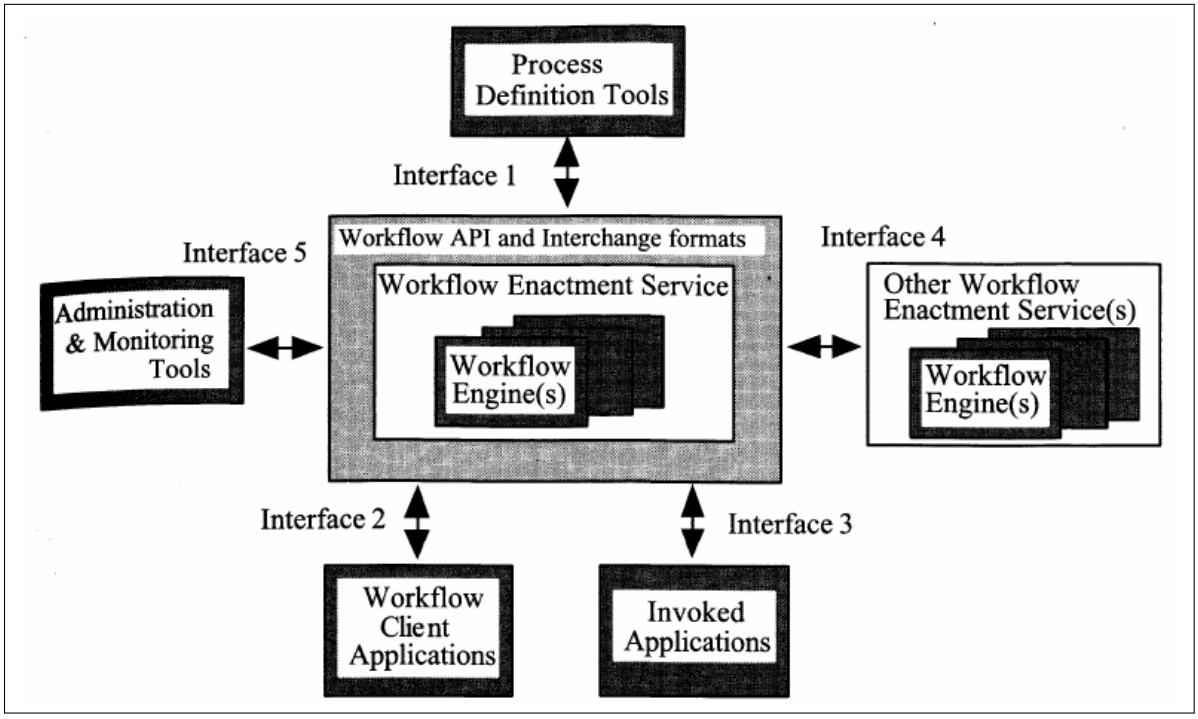

Figure 2.7: The workflow management coalition's reference model.

In order to achieve interoperability among various WFMS implementations, WfMC has defined five standard interfaces between the components. These interfaces are designated as workflow APIs and interchange formats. These interfaces are:

- Interface 1: Process Definition Tools Interface. The purpose of this interface is to integrate process definitions generated by different process definition tools, or to use a definition generated for a workflow system in another system.
- Interface 2: Workflow Client Applications Interface. Users of a workflow system utilize several types of client applications such as editors, CAD/CAM tools, and WWW browsers. This interface provides integration of these applications in order to participate in a workflow system.
- Interface 3: Invoked Applications Interface. WFMSs are expected to workflow with already existing software components such as legacy systems and DBMS applications.
- Interface 4: Other Workflow Enactment Services Interface. It may be necessary for different enactment services to interoperate because a process in one enactment service may invoke a process in another.
- Interface 5: Administration and Monitoring Tools Interface. An administration and monitoring tool may be a separately developed system, or it may be necessary to centrally monitor different workflow systems.

2.3 Scientific workflow systems

Among workflow, a specification is the scientific workflow. Scientific workflows are flexible tools for accessing scientific data, executing complex analysis, simulation and visualization tasks [18]. Scientific workflows often exhibit particular traits, e.g., they can be data-intensive, compute-intensive, analysis-intensive, and visualization-intensive, thus covering a wide range of applications from low-level plumbing workflows of interest to Grid engineers, to high level knowledge discovery workflows for scientists [7]. Each workflow consists of analytical steps that may involve database access and querying, data analysis and mining, and intensive computations performed on high performance cluster computers. Thus, scientific workflows systems are a formalization of the ad-hoc process that a scientist may go through to get from raw data to publishable results.

2.3.1 Scientific versus business workflows

Following [58], we compare features of scientific workflows and business workflows considering several aspects. We sum up them in the following:

- Implementation vs Modeling. Scientific workflows are developed with executability in mind, i.e., workflow designs can be viewed as executable specifications. In contrast, the primary goal of business process modeling is to develop a common understanding of the process that involves different persons and various information systems.
- Experimental vs Business-Driven Goals. A scientific workflow can be seen as a computational experiment, whose outcomes may confirm or invalidate a scientific hypothesis, or serve some similar experimental goals. In contrast, the outcome of a business workflow is known before the workflow starts.
- Multiple Workflow Instances. It is common that business workflows handle large numbers of cases and independent workflow instances at any given time. For example, each instance of an order workflow makes sure that the particular customer receives the ordered goods, and that billing is taken care of. In scientific workflows, truly independent instances are not as common. Instead, large numbers of related and interdependent instances may be invoked, e.g., in the context of parameter studies.
- Users and Roles. Business workflows usually involve numerous people in different roles. A business workflow system is responsible for distributing work to the human actors in the workflow. In contrast, scientific workflows are largely automated, with intermediate steps rarely requiring human intervention.
- Dataflow vs Control-Flow Focus. An edge A to B in a business workflow typically means B can only start after A has finished. Dataflow is often implicit or modeled separately in business workflows. However, A to B in a scientific workflow typically represents dataflow, i.e., actor A produces data that B consumes.

- Dataflow Computations vs Service Invocations. In scientific workflows data is often streamed through independent processes. These processes run continuously, getting input and producing output while they run. However, in business workflows, there are usually no data streams. An activity gets its input, performs some action, and produces output. An order arrives, it is checked, and given to the next activity in the process.

2.3.2 Main requirements on scientific workflow management systems

Following [39], we present the main requirements needed for a scientific workflow management systems:

- Usability: Scientists need the same support for services used in traditional workflows, i.e., scientists should be able to specify the services to be used in a scientific workflow themselves, or to delegate the service discovery to the scientific workflow management system. The need for automation basically includes the deployment of workflows, the provisioning of workflows, services and data, the instantiation and execution of workflows, and the service discovery.

- Flexibility: With flexibility, we denote the ability of a system to react to changes in its environment. Approaches to flexibility of workflows can be divided into two groups. First, a workflow can be modified automatically or manually according to the changed situation (known as adaptation). Second, a workflow can be modeled in a way that avoids its modification even in the presence of a changing environment (known as avoid change).

- Robustness: The term robustness denotes the ability of being error-resistant. The needed flexibility mechanisms mentioned above are a way to improve the robustness of a system. But additional approaches are needed to protect the execution progress from being lost in case of unforeseeable failures, to reach a consistent system state even in the presence of failures, and to proceed a simulation/experiment after a failure.

- Scalability: Scientific workflows should scale with the number of users, number of utilized services, data or calculation resources, and involved participants. Today, typical scientific workflows are mostly executed in a central manner on a single machine. A decentralized workflow enactment can help to scale via distributed process execution.

2.3.3 Data provenance in scientific workflow systems

Workflows can generate very large amounts of data. With these data, there are metadata that can describe them. The provenance is a type of metadata. It is also called "Genealogy

Lineage". This type of metadata follows the steps by which the data were derived and can make an important value to science. The provenance can be described by several different terms. This depends on the scope. Several definitions exist in the literature. In [21], Buneman et al define the data source in the context of databases. It is a description of the origin of the data and all processes that have sent this information to the database. Thus, the provenance is not only associated with the data but also the processes that led to the generation of such data.

In [38], Greenwood et al extend the definition of Lanter and define the source as a metadata that records the different processes, annotations and notes experiments in workflows. Finally, in [74], Simmhan et al define the source as the information that allows us to determine the history of a given from its origin. Data can be in different formats: tables, files, collections, etc.

Following [74], we sum up why we use the provenance as follows:

- Data Quality: The source can be used to estimate data quality and level of reliability. This estimate is based on the data source as well as the estimation process. This is an example of data from temperature sensors. Indeed, these temperature sensors can provide two totally different at times very close. This can be explained by a hardware failure or climatic factors. This data will have an accuracy which is the reliability of the sensor. One can refer to this in a wide range of literature in probabilistic databases.

- The audit (control): To extract data from databases and applications, analyze and report various forms, it is necessary to ensure consistency. Otherwise, the scientist may make choices based on misinformation. This step, which is fundamental, is not to be taken lightly. The source can be used to trace the audit data, determine resource usage and detect errors in data generation

- Reproduction of data: The detailed information of the provenance, may allow the reproduction of data and to help maintain their credibility. This information can be viewed as a creative recipe from the data.

- Attribution: The provenance may specify the copyright information and their citation. Indeed, in metadata, may have found, for example what is the person (or entity) that generated the data and the rights related to such information. For example, some data, we have the right to use them but not to sell, etc.

- Querying: The logical using basic of the provenance would to query a metadata database for discovering new knowledge.

2.3.4 Overview of scientific workflow systems

The increasing number of scientific applications involving complex analysis has motivated several proposals scientific workflow management systems in the academic community, including, Kepler, who have a general purpose, while others are intended for a particular type of application such as ecology, bio-informatics, etc.

One of the first integration of scientific workflows with DBMSs was supported by [6]. We present the following the main scientific workflow management systems.

2.3.4.1 SciRun

SciRun [66] is a Problem Solving Environment, for modeling, simulation and visualization of scientific problems. It is designed to allow scientists to interactively control scientific simulations while the computation is running. SCIRun was originally targeted at computational medicine but has, later, been expanded to support other scientific domains. The SCIRun environment provides a visual interface for dataflow network's construction. As the system will allow parameters to be changed at runtime, experimentation is a key concept in SCIRun. As soon as a parameter is updated, at runtime, changes will be propagated through the system and a re-evaluation is launched.

2.3.4.2 GPFlow

GPFlow [70] is a workflow platform providing an intuitive web based environment for scientists. The workflow model is inspired by spreadsheets. The workflow environment ensures interactivity and isolation between the calculation components and the user interface. This enables workflows to be browsed, interacted with, left and returned to, as well as started and stopped.

2.3.4.3 VisTrails

VisTrails [8] combines features of both workflow systems and visualization fields. Its main feature is to efficiently manage exploratory activities. The user interaction in VisTrails is performed by iteratively refining computational tasks and formulating test hypotheses. VisTrails maintains detailed provenance of the exploration process. Users are able to return to previous versions of a dataflow and compare their results. However, VisTrails is not meant to manage dynamic data. In VisTrails, dynamicity is performed by allowing users to change some attributes in order to compare visualization results. It does not include any model to handle data changes. Indeed, when the user starts its workflow process, VisTrails does not take into account the updated data in activities that have already started: there is no guarantee that the model for updates is correct.

2.3.4.4 Trident

Trident [11; 85] is a scientific workflow workbench built on top of a commercial workflow system. It is developed by Microsoft Corporation to facilitate scientific workflows management. Provenance in Trident is ensured using a publication/subscription mechanism called the Blackboard. This mechanism allows also for reporting and visualizing intermediate data resulting from a running workflow. One of the salient features of Trident is to allow users to dynamically select where to store results (on SQL Server for example) issued by a given workflow. However, it does not support dynamic data sources nor does it integrate mechanisms to handle such data.

2.3.4.5 Orchestra

Orchestra [2; 53] addresses the challenge of mapping databases which have potentially different schemas and interfaces. Orchestra is specially focusing on bio-informatics applications. In this domain, one finds many databases containing overlapping information with different levels of quality, accuracy and confidence. Database owners want to store a relevant ("alive") version of relevant data. Biologists would like to download and maintain local "live snapshots" of data to run their experiments. The Orchestra system focuses on reconciliation across schemas. It is a fully peer-to-peer architecture in which each participant site specifies which data it trusts in. The system allows all the sites to be continuously updated, and on demand, it will propagate these updates across sites. User interaction in Orchestra is only defined at the first level using trust conditions. Moreover, the deployed mechanism is not reactive. Indeed, there are no restorative functions called after each insert/update operation.

2.3.4.6 Kepler

Kepler is a software application for the analysis and modeling of scientific data. Using Kepler graphical interface and components, scientists with little background in computer science can create executable scientific workflows, which are flexible tools for accessing scientific data (streaming sensor data, medical and satellite images, simulation output, observational data, etc.) and executing complex analysis on the retrieved data [41].

Kepler system follows an actor-oriented modeling paradigm individual workflow components such as data base access and querying are abstracted into a set of generic, reusable tasks [60].

The visual interface of Kepler provides us the possibility to see each workflow step as an actor, a processing component that can be dragged and dropped into a workflow. Connected actors form a workflow allowing scientists to inspect and display on the fly as it is computed, make parameter changes as necessary and re-run and reproduce computer results. Kepler combines the advantages of all programs (Stella to model systems graphically, R or Matlab to perform statistical analyses) permitting users to model, analyze and

display data in one easy-to-use interface [41].

2.4 Databases and workflow systems

2.4.1 Database-based workflow systems

One approach for implementing a workflow management system is the so-called database-based approach. The database-based systems store the documents and the process information in a database. In [30], Eder et al describe the various advantages of this approach. We summarize below:

- The execution of a workflow is a typical task for a client-server application. Thus, several users can connect to the workflow server, which can cause malfunctions. The recovery system of databases ensures that even after a failure state the database remains consistent and each running process is restored.
- The integrity of data is ensured by having a central storage of all documents. An explicit versioning system can be handled by the database.
- The transaction mechanism in DBMS permits to increase concurrency in a safe way. It is possible that different users view a document concurrently, or different users edit different parts of the same document.
- Several database management systems provide application programming interfaces (APIs) to various languages and allows access over the network. These features are necessary, when coupling applications with the workflow system.
- The presence of all information on the status of processes and activities allows easy implementation of a monitoring component. This component allows users to inspect the state of the process, the content of documents, etc. All this operation can be recovered using SQL queries.

2.4.2 Active databases for workflow execution

Traditionally, database systems have been passive, storing and retrieving data in direct response to user requests without initiating any operations on their own. As the scale and complexity of data management increased, interest has grown in bringing active behavior into databases. Typically this behavior is described by event-condition-action (ECA) rules (cf. Figure 2.8).

ECA rules comprise three components: event E, condition C, and action A. The event describes an external happening to which the rule may be able to respond. The condition examines the context in which the event has taken place. The action describes the task

to be carried out by the rule if the relevant event has taken place and the condition has evaluated to true [67]. In sum, if the specified event E occurs and if the condition C is true then the specified action A is executed.

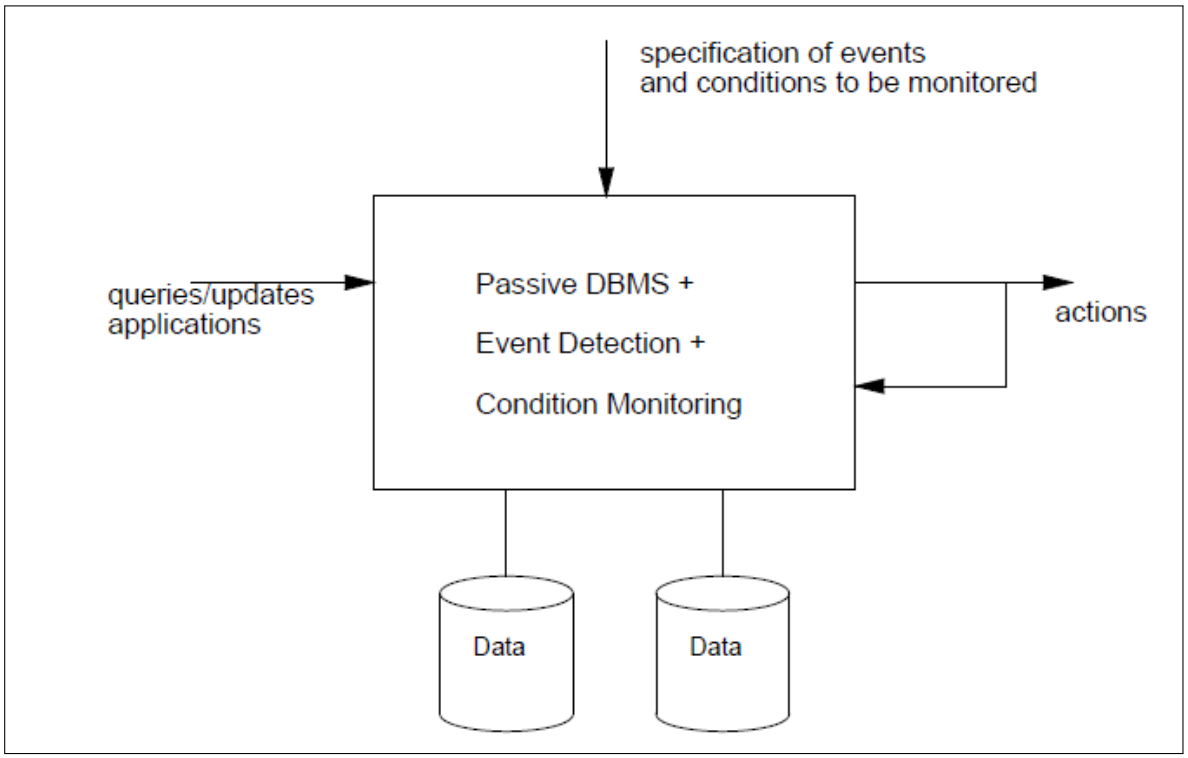

Figure 2.8: Principle of an active database.

While there is agreement that all active databases must detect event occurrences, support rule management, and execute actions [28], there is no consensus on how the events, conditions and actions are specified. Rule conditions may get arbitrarily complex and rule conditions may have to be monitored in one of many different ways [68].

Each operation performed over the database (insert, update, delete, select) is considered as an event, which can trigger the application of a rule. If a rule is triggered, the conditions of the rule are evaluated. If the conditions are satisfied, the actions of the rule are applied. The basic structure of a rule is:

```
create trigger name on table
after event
when condition
then action
```

In [30], the authors emphasize that the whole workflow manager simply consists of all the rules resulting from the compilation of workflow specifications. All other necessary features are already provided by the database management system.

The rules are automatically generated from the declarative descriptions of the tasks and flows by the compiler. Therefore, the active database management system is the workflow server and has the functionality described in the process specification. For each flow one rule is generated (called flow-rule), triggering when a task is completed, i.e. after the satisfaction of the postcondition. This corresponds to the third step of the above execution model. The following rule specifies a flow of a form of type form i from task A to task B, where the form is sent if the condition flow-condition is met.

```
create trigger flow n step3 on form i
after update status
when new.status='finished'
and form.type=form i and form.task = task A and flow-condition
then update new set task = task B;
```

2.4.3 Transaction support for workflow management systems

In [30], Eder et al extend the transaction concept to develop workflow transactions for consistent execution of workflows. The main characteristics of workflow transactions may be summarized as follows:

- Analogies: In DBMS, a transaction transforms a database from one state to another while preserving its consistency. Similarly, a workflow transaction transfers a workflow process from a consistent state A to the next consistent state B. We will refer to transaction-level databases as traditional transactions and those conducted on the workflow process as workflow transactions.

- Transaction structure: Workflow transactions should allow a hierarchical structure to be applicable for complex applications. A workflow usually consists of several activities that are themselves composed of (sub-) activities. Each activity is a transaction workflow.

- Atomicity: Application dependent (user-defined) failure atomicity is required given that workflow activities are in general of long duration. Instead to selectively roll back parts of the work until the most recent consistent state (within the transaction) is reached. The workflow transaction manager needs support from a human expert (e.g., in WAMO [31]) and more advanced recovery concepts [56] have to be provided to find such a consistent state the workflow transaction manager.

- Consistency: The commit of an activity is taken as a guarantee that the activity has produced a consistent result. If an activity fails then an inconsistent state may be the consequence. As for flat transactions, such situations should not occur and must therefore be removed.

- Isolation: It is not possible to execute workflow transactions fully isolated from concurrent transactions. Serializability as correctness criterion for concurrent processing is too restrictive. There exist several theoretical approaches to overcome this problem without compromising consistency, as for example semantic serializability [20]. The goal is to exploit the semantics of the activities by defining compatibility specifications between the activities.

- Durability: As soon as a workflow (sub-)transaction commits, its effects are persistent.

2.5 Conclusion

This chapter provides the necessary background knowledge for understanding the workflow concept and specially the meet between workflow systems and database management systems which is a main feature of our contribution in this thesis. The next chapter focuses on the visual analytics domain. Indeed, the relationship between scientific workflow systems and visualization systems becomes increasingly narrow. The visualization task is used to extract knowledge that cannot be released following only an automated reasoning.

To summarize, all workflow platforms presented in this chapter share some important features, which we also base our work on. Workflows are *declaratively specified, data-intensive* and (generally) *multi-user*. They include *querying and updating* data residing in some form of a database (or in less structured sources). Crucial for their role is the ability to invoke *external procedures*, viewed as black boxes from the workflow engine perspective. The procedures are implemented in languages such as C, C++, Matlab, Fortran. They perform important domain-dependent tasks; *procedures may take as input and/or produce as output large collections of data.* Finally, current scientific workflow platforms do provide, or can be coupled with, some *visualization* tools, e.g., basic spreadsheet-based graphics, map tools.

None of these platforms are currently able to propagate data changes to a running process and launch a repair mechanism.

Chapter 3

Visual analytics survey

Contents

3.1 Introduction

This chapter provides the reader with background knowledge about the visual analytics field. The remainder of this chapter is structured as follows. In Section 3.2, we introduce the visual analytics field and describe its process in section 3.3. Section 3.4 focuses on the scalability challenge in visual analytics which is the major concern of our contributions in this field. The Section 3.5 enumerates several visual analytics tools devised into generalized and specialized tools. Section 3.6 concludes this chapter.

3.2 Visual analytics definition

According to [77] "Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces". The standard approach for visualization is to aim at interactive visual systems that enable people to obtain insight in large data sets. This is also a central aspect of visual analytics, but here a broader view is taken. If we want to support massive data sets, other data analysis methodologies, such as statistics and machine learning, have to be integrated in order to reduce the data set size and to enable an optimal division of labor between man and machine. In complex, real-world cases the data sets to be analyzed are often not just plain tables with numbers, but are mixed collections of text, multimedia, and relational data. The knowledge discovery process is not just about obtaining insight, but encompasses a series of stages, from data collection via analysis to presentation. Also, large data analysis problems require teams of people, where each brings in his or her own expertise. As a result, visual analytics is an ambitious endeavor, requiring a variety of disciplines, and with many open ends and unanswered questions [79]. Additionally, visual analytics is the focus of a vibrant and active research community and this with the creation of an IEEE supported conference in 2006 and a growing number of articles and attendees.

3.2.1 Goals of visual analytics

In [54], authors define four goals that must be achieved by visual analytics. Visual analytics is the creation of tools and techniques to enable people to:

- Synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data.
- Detect the expected and discover the unexpected.
- Provide timely, defensible, and understandable assessments.
- Communicate these assessments effectively for action.

3.2.2 Applications of visual analytics

Visual analytics is essential in application areas where large information spaces have to be processed and analyzed. Major application fields are physics and astronomy because they offer massive volumes of unstructured data from continuous streams of terabytes of data that can be recorded and analyzed. We describe in the following three applications of visual analytics in three different areas.

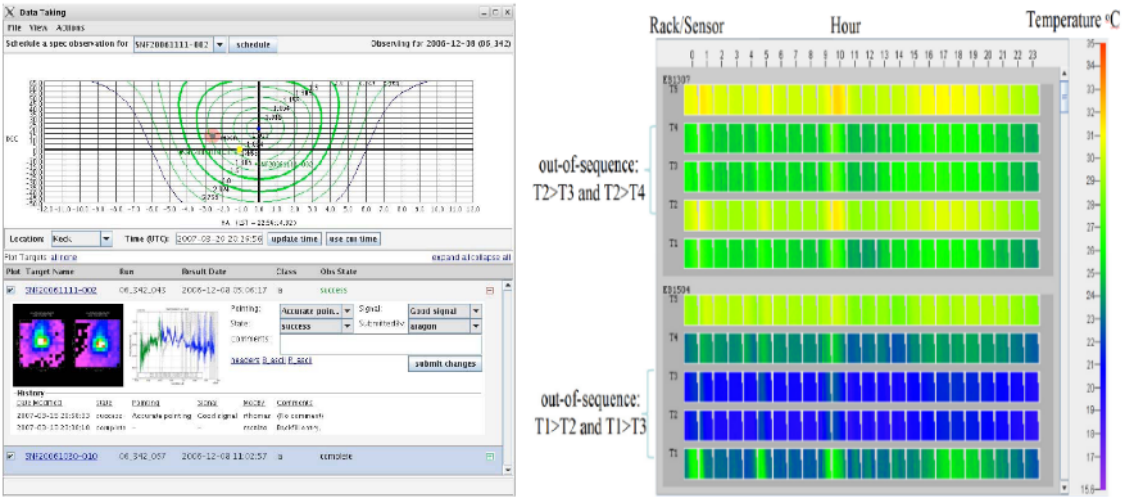

Figure 3.1: Visual analytics applications.

3.2.2.1 Economic decision-making

In [71], Savikhin et al addressed the problem of how visual analytics can improve economic decision making specially in Winner's and Loser's Curse problems in economics. They used visual analytics as a tool to consider all the information of the problem and make a decision that is closer to the optimal.

The article showed that all the information displayed via a visual can be useful for overcoming bounded rationality issues that arise from the cognitive limitation of considering all the information at hand [71]. This also suggests that the addition of an interactivity element is very important to the usefulness of visuals.

3.2.2.2 Thermal state management

In [45], Hao et al present visual analytics techniques for thermal state detection. The authors embedded the visual analytics capabilities into a mobility-enabled visualization platform, Data Center Mobile Studio, which is hosted on HP iPAQ 210. This empowers onsite administrators to visualize current thermal state information. The key technical

work is centered on the ability to perform progressive visual analytics on data either from a local database cache or a web service. The right of the Figure 3.1 illustrates visual time series for finding out of sequence sensors.

3.2.2.3 Astrophysics

In [9], Aragon et al described a case study involving a visual analytics system developed for astrophysicists collaboratively operating a large telescope for time-critical supernova observation. The authors showed the effectiveness of a simplifying visualization, projecting three-dimensional data to a rectilinear two-dimensional format, in increasing situation awareness for users needing to synthesize large amounts of streaming data and make critical decisions under time pressure. The left of the Figure 3.1 illustrates the Data Taking window where the observer can follow the targets on The Sky visualization, take notes on the success or failure of each observation.

3.2.3 Visual analytics disciplines

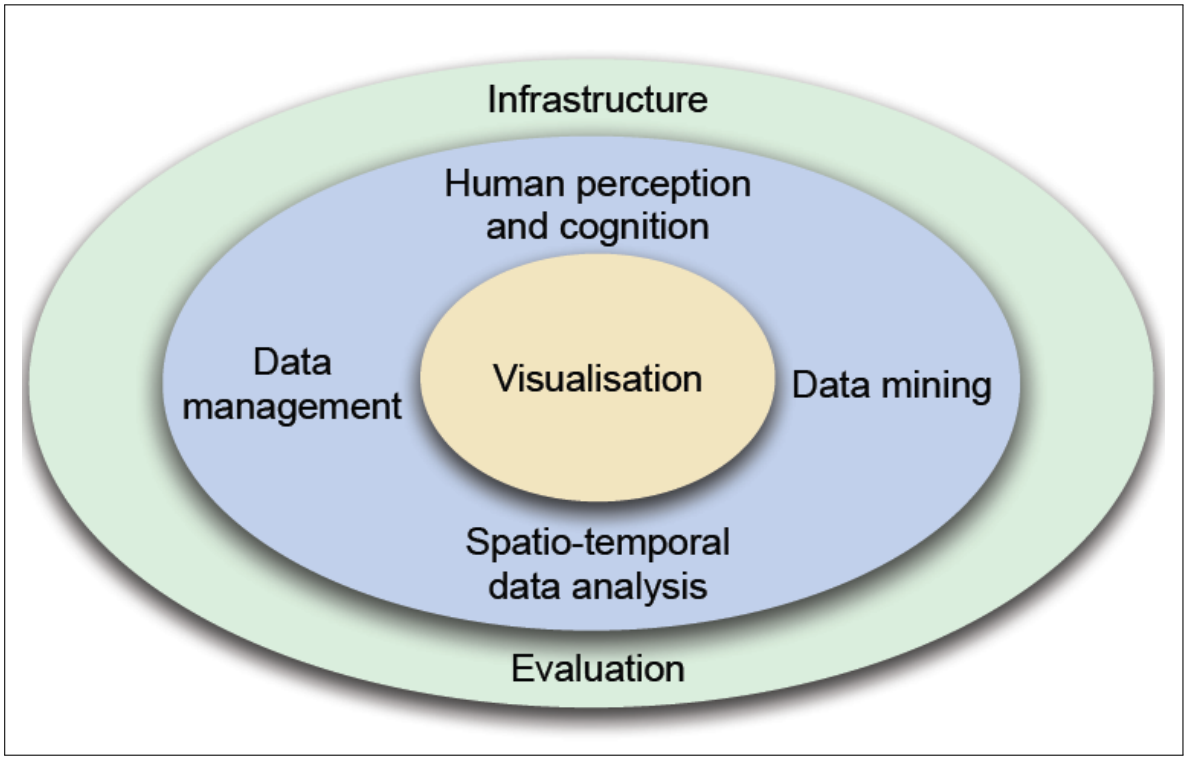

Figure 3.2: The visual analytics disciplines.

In [54], authors present various disciplines related to the visual analytics field. Figure 3.2 summarizes the visual analytics block. We describe in the following each discipline and its relation to visual analytics.

- Visualization. Information visualization has developed methods for the visualization of abstract data where no explicit spatial references are given. Typical examples include business data, demographics data, social networks and scientific data.

- Data Management. The efficient management of data of various types and qualities is a key component of visual analytics, as it typically provides the input of the data, which is to be analyzed. Generally, a necessary precondition to perform any kind of data analysis is an integrated and consistent database.

- Data Mining. The discipline of data mining develops computational methods to automatically Data mining: automatically extract valuable information from raw data by means of automatic analysis algorithms. In almost all data analysis algorithms, a variety of parameters needs to be specified: a problem which is usually not trivial and often needs supervision by a human expert. Interactive visualization can help with this, and can also be used in presenting the results of the automatic analysis, so called visual data mining.

- Spatio-temporal Data Analysis. The analysis of data with references both in space and in time, spatial-temporal data, has added complexities of scale and uncertainty. For instance, it is often necessary to scale maps to look for patterns over wide and also localized areas, and similarly for time, we may wish to look for trends that occur during a day and others that occurs on a yearly basis.

- Perception and Cognition. Perception and cognition represent the more human side of visual analytics. Visual perception is the means by which people interpret their surroundings and for that matter, images on a computer display. Cognition is the ability to understand this visual information, making inferences largely based on prior learning.

- Infrastructure. Infrastructure is concerned with linking together all the processes, functions and services required by visual analytic applications so they work in harmony, in order to allow the user to undertake their data exploration tasks in an efficient and effective manner.

- Evaluation. Evaluation is very difficult given the explorative nature of visual analytics, the wide range of user experience, the diversity of data sources and the actual tasks themselves.

To summarize, Visual analytics is considered as a workflow process. We describe, in the following, such a process.

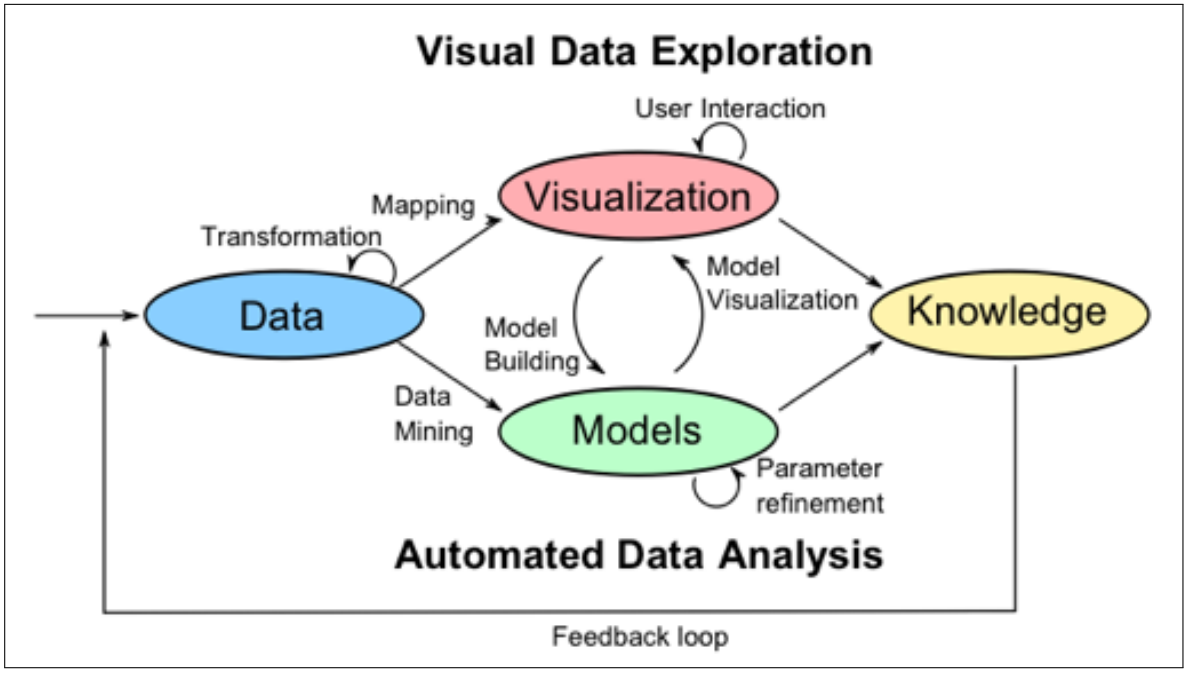

Figure 3.3: The visual analytics process.

3.3 Visual analytics as a process

The visual analytics process combines automatic and visual analysis methods with a tight coupling through human interaction in order to gain knowledge from data [55]. Figure 3.3 shows an overview of the different stages and their transitions in the visual analytics process.

After the transformation and the cleaning of data, the analyst may choose between applying visual or automatic analysis methods.

If the user chooses an automated analysis to be performed first, data mining algorithms are applied to generate models of the original data. Once a model is created the analyst has to evaluate and refine it. This task can best be done by interacting with the data. Visualizations allow the analysts to interact with the automatic methods by modifying parameters or selecting other analysis algorithms. Model visualization can then be used to evaluate the findings of the generated models. Alternating between visual and automatic methods is characteristic for the visual analytics process and leads to a continuous refinement and verification of preliminary results.

If the user chooses a visual data exploration to be performed first, he/she has to confirm the generated hypotheses by an automated analysis. User interaction with the visualization is needed to reveal insightful information, for instance by zooming in on different data areas or by considering different visual views on the data. Findings in the visualizations can be used to steer model building in the automatic analysis.

Most visual analytics platforms are limited in the volume of data handled. The next section focuses on the scalability challenge in visual analytics.

3.4 Scalability challenge in visual analytics

In [4], the authors define the scalability as follows: "Scalability, as a property of systems, is generally difficult to define [49] and in any particular case it is necessary to define the specific requirements for scalability on those dimensions that are deemed important. It is a highly significant issue in electronics systems, databases, routers, and networking. A system, whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system. An algorithm, design, networking protocol, program, or other system is said to scale, if it is suitably efficient and practical when applied to large situations (e.g., a large input data set or a large number of participating nodes in the case of a distributed system). If the design fails when the quantity increases, it does not scale".

Current technologies cannot support the scale and complexity of the growing analytical challenge. New techniques and underlying scientific foundations are needed to deal with the scale of the problems we are facing in threat analysis, emergency management, and border protection [77].

We present the following five scalability issues related to the visual analytics field.

- Information Scalability. Information scalability implies the capability to extract relevant information from massive data streams. Methods of information scalability include methods to filter and reduce the amount of data, techniques to represent the data in a multi-resolution manner, and methods to abstract the data sets.

- Visual Scalability. Visual scalability is the capability of visualization representation and visualization tools to effectively display massive data sets, in terms of either the number or the dimension of individual data elements [32]. Factors affecting visual scalability include the quality of visual displays, the visual metaphors used in the display of information, the techniques used to interact with the visual representations, and the perception capabilities of the human cognitive system [77].

- Display Scalability. Most published visualization techniques are designed for one size display. Applications that support display scalability include a variety of display to take advantage of whatever capabilities are available to support analysis and collaboration. These applications make effective use of everything from a wall-sized display in an emergency response situation room to a PDA or phone-sized display in the hands of a first responder in the field.

- Human Scalability. The number of humans involved in analytical problem-solving, border protection, and emergency preparedness and response activities scale. Most published techniques for supporting analysis are targeted for a single user at a time.

Applications that support human scalability scale from a single user to a collaborative (multi-user) environment.

- Software Scalability. Software scalability is the capability of a software system to interactively manipulate large data sets. Software scalability includes the generation of new algorithms that scale to the ever-increasing information sets.

3.5 Visual analytics systems

3.5.1 Requirements and functionalities of visual analytics systems

Following [77], we present a set of diverse analytical tasks that must be enabled by visual analytics tools:

- Understanding past and present situations quickly, as well as the trends and events that have produced current conditions;
- Identifying possible alternative futures and their warning signs;
- Monitoring current events for emergence of warning signs as well as unexpected events;
- Determining indicators of the intent of an action or an individual;
- Supporting the decision maker in times of crisis.

These tasks will be conducted through a combination of individual and collaborative analysis, often under extreme time pressure. Visual analytics must enable hypothesis-based and scenario-based analytical techniques, providing support for the analyst to reason based on the available evidence [77].

We present the following two classes of visual analytics tools. The first class describes the specialized tools. It focuses on the particular area such as astrophysics, genetics, etc. The second class has a more general aspect and includes visual analysis technique that can be used in several areas.

3.5.2 WikiReactive

WikiReactive [17] is an open architecture to compute incrementally and maintain several aggregated measures on the French Wikipedia. These measures, if available and visualized effectively, could spare a lot of monitoring time to Wikipedia users, allowing them to focus on quality and coverage of articles instead of spending their time navigating heavily to track vandals and copyright infringements. The WikiReactive infrastructure, shown in Figure 3.4 was experimented over the French Wikipedia provided as a compressed file (4.5 GB) from Wikipedia api. This file is about 10 times larger once uncompressed.

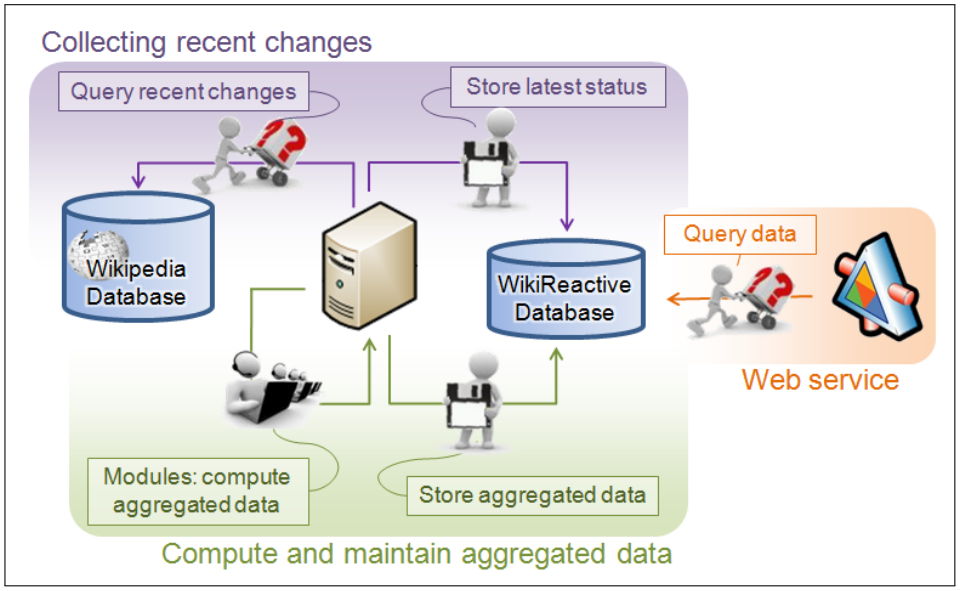

Figure 3.4: Overall scheme of the WikiReactive infrastructure.

The Wikipedia infrastructure uses five modules to compute and maintain aggregated information in multiple tables:

- Module 1. Diff computation: In this step, a minimal set of editing operations (insertions, deletions and moves) is computed at the character level to mimic the edition process. For each revision, the detailed list of editing operations is stored as well as the associated article and user in the detailed diff table.
- Module 2. Per-user contribution computation: This step consists of assigning each user an identifier and using it to sign each character in the revision. The contribution table of the latest revision of each article is stored.
- Module 3. Article activity computation: This module consists of computing the sum of characters touched by the different change operations in the aggregated diff table for each revision.
- Module 4. User contribution computation: In this module, the aggregated sum of users operations on individual articles for each user, is kept in the users table.
- Module 5. Character per user computation: The last module updates, for each user, the number of characters remaining in each article he or she has contributed to.

The computed aggregates are queried through a Web service based on the Google Visualization Data Source [1] providing a communication protocol and a simplified query language similar to SQL.

3.5.3 Vox Civitas

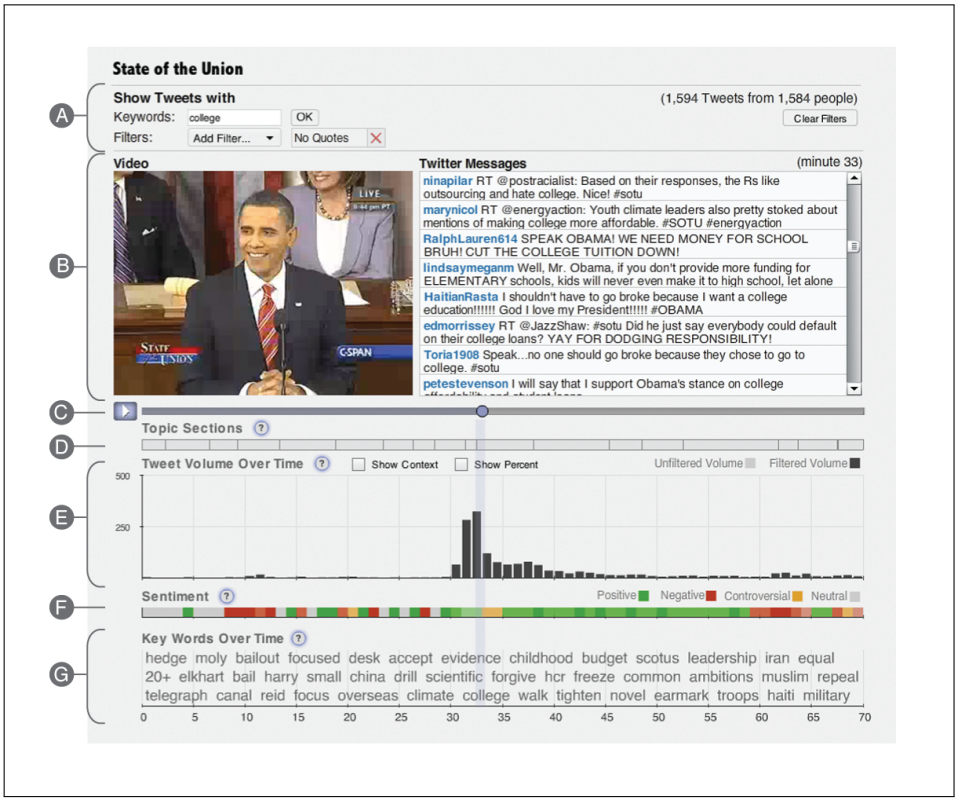

Figure 3.5: Vox Civitas user interface.

Vox Civitas [29] is a visual analytics tool designed to help journalists and media professionals extract news value from large scale aggregations of social media content around broadcast events. This tool is based on several text analysis algorithms that support the journalistic goals and enhance the use of Vox Civitas as a visual analytics tool for journalistic inquiry. The text analysis algorithms can be classified into four families:

- Relevance: The relevance of messages is measured by calculating term-vector similarity of messages to the moment in the event during which the messages were posted.
- Uniqueness: The uniqueness of a message is measured in relation to the other messages sent during the same time interval. It is computed as the difference between the term-vector space representations of the message to the centroid term-vector representation of all of the messages for that particular minute of the event.
- Sentiment: The sentiment analysis is used to inform an analyst's understanding of the polarity (i.e. positive versus negative) of the social media reaction to the event. This analysis is done following two steps. The first step, the messages are classified based on whether they were carrying subjective. The classifier used for this task is

based on a lexicon of words. The second step, a supervised learning algorithm is applied trained with 1900 manually tagged messages.

- Keyword extraction: The text analysis algorithms used in this step aim to identify keywords used in the social media stream that could be useful and interesting for guiding analysts. For each minute the top 10 keywords are extracted and ranked by their tf-idf score [61].

As shown in figure 3.5, the Vox Civitas interface integrates video from an event with the ability to visually assess the textual social media response to that event. Filtering messages is done via the module shown in the A part of the figure. Next to the video content, the B panel enables viewing the actual Twitter messages posted about the event. While the C panel shows the video timeline, the D panel presents the topic timeline. The E, F and G parts of the figure present three views (volume graph, sentiment timeline and keywords component) for aggregate response analysis.

3.5.4 Radio frequency fingerprinting-based localization system

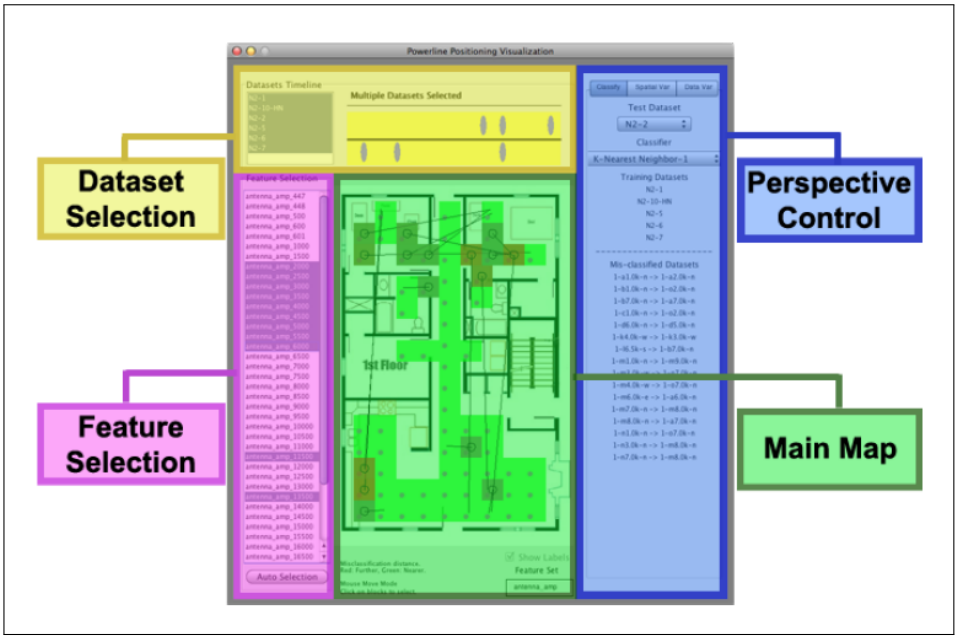

Figure 3.6: User interface of the radio frequency system.

In [44], authors present a visual analytics system enabling developers of Radio frequency (RF) fingerprinting-based techniques for localization to visually gain insight on whether their collected datasets and chosen fingerprint features have the necessary properties to enable a reliable RF fingerprinting-based localization system.

Figure 3.6 shows the user interface of the system. It consists of four main panels:

- Dataset selection: This panel is for selecting the datasets that will be used.
- Feature selection: This panel allows the system's users to select features for fingerprints.
- Main map: This panel is used to display area for the geospatial visualization.
- Perspective control: The perspective panel is used to control the viewing perspectives.

Each computation module is related to a perspective. We describe in the following three modules of calculation used by the system:

- Data variance perspective: It shows the raw data of all the datasets with their corresponding feature sets.
- Spatial variance perspective: This perspective shows the spatial variance between fingerprints in the high dimensional feature space using the selected features.
- Test classification perspective: It provides a geospatial representation to show the results of the location classification using the generated fingerprints.

Discussion

Visual analytics applications are currently built using ad-hoc architectures: no clear reference model exists to guide application designers [54]. Such applications also tend to use and extend the software models of the main architect's application domain. Since most visual analytics systems are built by information visualization practitioners, they usually follow the information visualization reference model [23; 27] for the visualization as well as the interaction part; the data analysis part being integrated or performed separately depending on the level of computation and data management needed.

In addition, visual analytics systems contain two essential parts: (i) a recurring part for analytical processing and reactive to data and (ii) an exploratory and interactive part allowing a reactivity to the user actions.

Scientific workflow systems allow the first part of treatment (analytical processing), however, they are not reactive. Similarly, visual analytics systems are exploratory but do not control the data analysis.

Thus, a director component is missing in visual analytics systems as well as scientific workflows systems. This component must have the role of management and control of dynamic change of data and user interactions.

3.6 Conclusion

In this chapter, we presented the necessary background on the visual analytics field to understand the rest of the dissertation. This background describes the different tools and approaches that will be used in the next part.

Most of existing interactive platforms for data visualization [35; 48] focus on the interaction between the human expert and a data set consisting of a completely known set of values. They do not ease the inclusion of data analysis programs on the data. Moreover, as previously explained, most of them do not support the definition of structured processes, nor (by absence of an underlying DBMS) do they support persistence and sharing. An exception is [36] which is a visualization tool combining database technology. However, there is no repair mechanism and the change propagation is not supported.

Chapter 4

Conclusion of the part

This chapter can help readers position themselves in relation to the state of the art and identify the contributions of our work.

It appears at the end of this state of the art that the fields of workflows systems and visual analytics are very active and continue to attract interest from many research communities (Database, Statistics, Software Engineering, etc.).

In this state of the art, we have described and analyzed various features and challenges related to these two areas. We present briefly in the following each of these challenges and the solutions we provide to solve them.

4.1 Challenges

Handling dynamic data The management of dynamic data is a common challenge between scientific workflows and visual analytics communities. Indeed, start from the assumption that a scientist starts a workflow process with long-term activities (ie, clustering program with images of the solar system). During the execution of this process, new data enter the system. Today, the only option available to the scientist to take into account the new data in the process is to stop the current process and restart it again. What is inconceivable when you have long-term activities.

As well as, 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; conversely, processes (or visualization) may update the data. The possible interactions between all these updates must be carefully thought out, in order to support efficient and flexible applications.

Scalability and interaction challenges Most visual analytics platforms are memory-based and are therefore limited in the volume of data handled. Moreover, the integration of each new algorithm (e.g., for clustering) requires integrating it by hand into the platform. Finally, they lack the capability to define and deploy well-structured processes where users

with different roles interact in a coordinated way sharing the same data and possibly the same visualizations.

Most current visual analytics tools rarely rely on persistent databases (with the exception of [36]). Instead, the data is loaded from files or databases and is manipulated directly in memory because smooth visual interaction requires redisplaying the manipulated data 10-25 times per second. Standard database technologies do not support continuous queries at this rate; at the same time, ad-hoc in-memory handling of classical database tasks (e.g., querying, sorting) has obvious limitations.

4.2 Our contribution: the EdiFlow platform

We have designed a generic architecture called EdiFlow for integrating a visual analytics tool and a DBMS. EdiFlow is a workflow platform for visual analytics applications. EdiFlow uses a simple structured process model and is backed by a persistent database, storing both process information and process instance data.

The main EdiFlow contributions are:

- *generic architecture for deploying a VA platform on top of a DBMS.* The integration is based on a core data model, providing support for (*i*) visualizations, (*ii*) declaratively-specified and automatically-deployed workflows based on a high-level specification.
- a mechanism to manage dynamic data efficiently, relying on standard features of persistent DBMS systems and incremental propagation of data updates through complex processes.
- isolation semantic built on top of DBMS to allow modules to run isolated from changes made by other modules or from the external environment until they are ready to handle them.
- an architecture that respects the Reference Model in visualization by allowing several views of visualization.
- the use of a database cache mechanism to achieve the speed required for visualization and computation.

The different advantages of EdiFlow for managing scientific workflows are presented in Chapter 5, while the advantages related to the visual analytics fields are described in Chapter 6.

Part III

Contributions

Chapter 5

Interactive changes in workflow systems

Contents

5.1 Introduction

This chapter presents the EdiFlow platform which is developed to address the questions raised by the integration of a DBMS in a visual analytics platform. The main EdiFlow contribution is a *generic architecture for deploying a VA platform on top of a DBMS*. The integration is based on a core data model, providing support for (*i*) visualizations, (*ii*) declaratively-specified, automatically-deployed workflows, and (*iii*) incremental propagation of data updates through complex processes, based on a high-level specification. This model draws from the existing experience in managing data-intensive workflows [6; 19; 24; 72]. Thanks to this design, our architecture can be adapted with modest effort to existing scientific workflow platforms [7; 22; 72], to endow them with the powerful interactive data analysis tools and de facto transform them into visual analytics platforms.

This chapter is organized as follows. Section 5.2 presents our proposed data model, while the process model is described in Section 5.3. Section 5.4 describes the different alternatives proposed by EdiFlow to manage dynamic change of data. We describe our integration architecture in Section 5.5, discuss some aspects of its implementation in our EdiFlow platform. Section 5.6 focuses on the isolation management in EdiFlow. Section 5.7 describes several applications encountered in different contexts, illustrating the problems addressed in this work. We then conclude in Section 5.8.

5.2 Data model

We illustrate in this section the data model of our system in two ways: a logical (conceptual) data model, in section 5.2.1, describing entities that appear in EdiFlow and a physical data model, 5.2.2, describing the concrete environment used to model EdiFlow.

5.2.1 Logical data model

The conceptual data model of visual analytics application is depicted in Figure 5.1. For the sake of readability, entities and relationships are organized in several groups.

The first group contains a set of entities capturing *process definitions*. A process consists of some activities. An activity must be performed by a different group of users (one can also see a group as a role to be played within the process). Process control flow is not expressed by the data model, rather, it is described in the process model (see Section 5.3). An activity instance has a start date and an end date, as well as a *status* flag ranging in the following set of values: {*not_started*, *running*, *completed*}. The flag *not_started* states that the activity instance is created by a user who assigns it to another for completion, but the activity's task has not started yet. The *running* flag indicates that the activity instance has started and has not yet finished. Finally, the flag *completed* means that the activity instance has terminated. Process instances will also take similar values.

Entities in the second group allow recording *process execution*. Individual users may belong to one or several groups. A user may perform some activity instances, and thus be involved in specific process instances. A *ConnectedUser* records the host and port from which a user connects at a given time. This information is needed to propagate updates, received while the process is running, to a potentially remote visualization component running on the remote user's desktop. This point will be further discussed in Section 5.5.3.

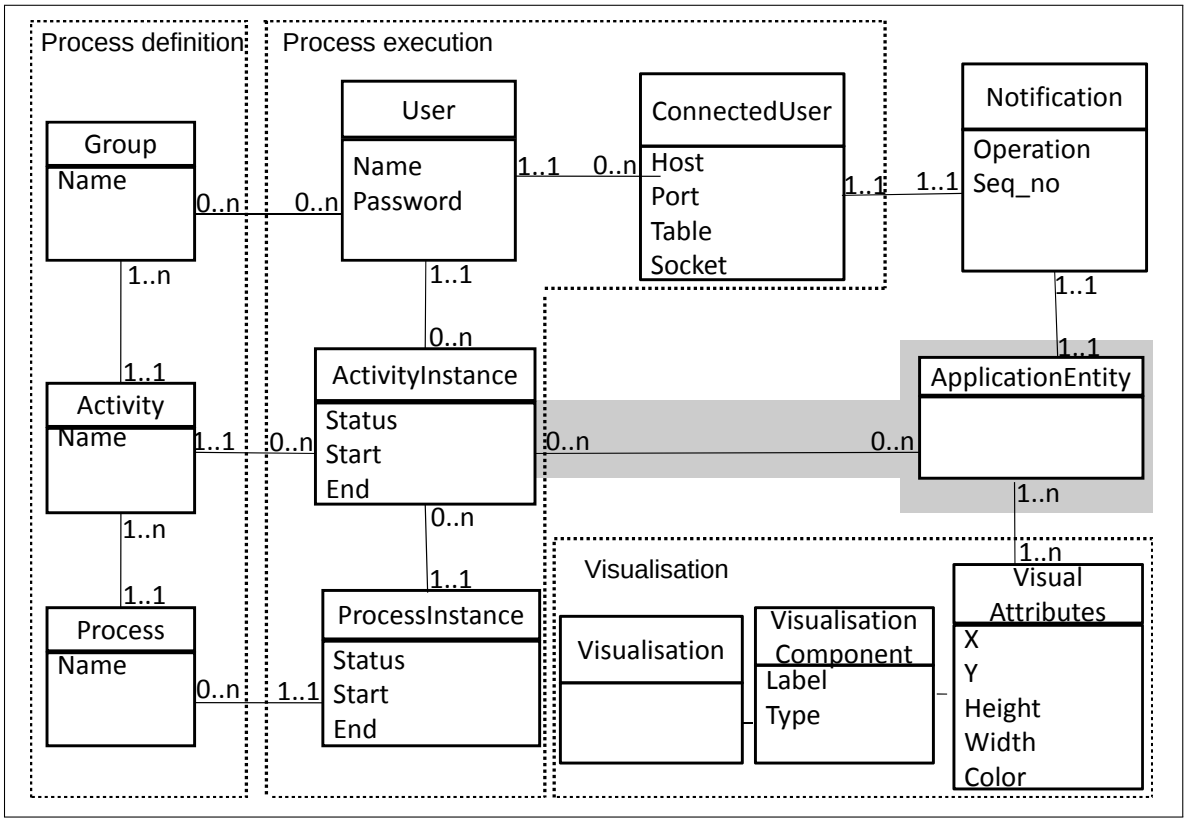

Figure 5.1: Entity-relationship data model for EdiFlow.

The gray area can be seen as a meta-model, which has to be instantiated for any concrete application with one or several entities and relationships modeling it. For instance, in the Wikipedia application, one would use the entities *Article*, *User*, and *Version*, with relationships stating that each version of an article is produced by one user's article update. Black-box functions, such as Wikipedia user clustering functions, must also be captured by this application-dependent part of the data model. Tracking workflow results requires at a simple level that for each data instance, one may identify which activity instance which created it, updated it etc. To that purpose, specific customized relationships of the form *createdBy*, *validatedBy* may be defined in the conceptual model. They are represented in Figure 5.1 by the gray background relationship between, *ApplicationEntity* and *ActivityIn-*

stance. Of course, many more complex data provenance models can be devised e.g., [8; 75]. This aspect is orthogonal to our work.

The third group of entities is used to model visualization. A *Visualization* consists of one or more *VisualisationComponent*s. Each component offers an individual perspective over a set of entity instances. For example, in Figure 5.9, three visualization components are shown in the bar at the left of the article, making up a given visualization associated with the article's edit history. Components of a same visualization correspond to different ways of rendering the same objects. In each visualization component, a specific set of *VisualAttributes* specifies how each object should be rendered. Common visual attributes include (x, y) coordinates, width, height, color, label (a string), whether the data instance is currently *selected* by a given visualization component (which typically triggers the re-computation of the other components to reflect the selection).

Finally, the *Notification* entity is used to speedily propagate updates to the application entities in various places within a running process. A notification is associated with one or more instances of a particular application entity. It refers to an update performed at a specific moment indicated by the *seq_no* timestamp, and indicates the kind of the update (insert/delete/modify). Its usage is detailed in Section 5.5.3.

5.2.2 Physical data model

We assume a simple *relational* enactment of this conceptual model. We have considered XML but settled for relations since performant visualization algorithms are already based on a tabular model [33]. 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. By issuing a query to the database, one can determine "which are the completed activity instances in process P", or "which is the R tuple currently selected by the user from the visualization component VC_1".

We distinguish two kinds of relations. DBMS-hosted relations are by definition persistent inside a database server and their content is still available after the completion of all processes. Such relations can be used in different instances, possibly of different processes. In contrast, temporary relations are memory-resident, local to a given process instance (their data are not visible and cannot be shared across process instances) and their lifespan is restricted to that of the process instance which uses them. If temporary relation data are to persist, they can be explicitly copied into persistent DBMS tables, as we shortly explain below.

5.3 Process model

We consider a process model inspired by the Workflow Management Coalition model [84]. Figure 5.2 outlines (in a regular expression notation) the syntax of our processes. We use

a set of variables, constants and attribute names N, a set of atomic values V, and a set of atomic data types T; terminal symbols used in the process structure are shown in boldface. The main innovative ingredient here is the treatment of *data dynamics*, i.e., the possibility to control which changes in the data are propagated to which part(s) of which process instances. To facilitate the understanding of the process model, we divide our model into three parts:

- Data part: This part includes elements related to data and queries.
- Computation part: This part focuses on computational elements related to managing dynamic changes on workflow process.
- Process part: This part includes elements related to the workflow components and patterns.

We now describe each part in detail.

Process	::=	Configuration Constant* Variable+ Relation+ Function* StructProcess
Configuration	::=	DBdriver DBuri DBuser DBpassword
Constant	::=	name value name $\in N$, value $\in V$
Variable	::=	name type name $\in N$, type $\in T$
Relation	::=	name primaryKey RelType
RelationType	::=	(attName attType)*, attName $\in N$, attType $\in T$
Function	:=	name classPath
StructuredProcess	:=	Activity | Sequence | AndSplitJoin | OrSplitJoin | ConditionalProcess
Sequence	::=	Activity , StructuredProcess
AndSplitJoin	::=	**AND-split** (StructuredProcess)+ **AND-join**
OrSplitJoin	::=	**OR-split** (StructuredProcess)+ **OR-join**
ConditionalProcess	::=	**IF** Condition StructuredProcess
Activity	::=	activityName Expression
Expression	::=	askUser | callFunction | runQuery

Figure 5.2: XML schema for the process model.

5.3.1 Data part

Relations and queries A process is built on top of a set of relations implementing the data model. Relations are denoted by capital letters such as R, S, T, possibly with subscripts. A query is a relational algebraic expression over the relations. We consider as operators: selection, projection, and Cartesian product. Queries are typically designated by the letter Q possibly with subscripts.

Variables A variable is a pair composed of a name, and of an (atomic) value. Variables come in handy for modeling useful constants, such as, for example, a numerical threshold for a clustering algorithm. Variables will be denoted by lower-case letters such as v, x, y.

Constants The constants are similar to variables. They are described by (name, value) pairs. However the values of Constants cannot be changed during the execution process. The constants are useful to define long strings. To reuse these strings, we must only call them by indicating their names.

5.3.2 Computation part

Procedures A procedure is a computation unit implemented by some external, black-box software. A typical example is the code computing values of the visual attributes to be used in a visualization component. Other examples include, e.g., clustering algorithms, statistical analysis tools.

A procedure takes as input l relations $R_1, R_2, \ldots, R_l$ which are read but not changed and m relations $T_1^w, T_2^w, \ldots, T_m^w$ which the procedure may read *and* change, and outputs data in n relations:

$$p : R_1, R_2, \ldots, R_l, T_1^w, T_2^w, \ldots, T_m^w \rightarrow S_1, S_2, \ldots, S_n$$

We consider p as a black box, corresponding to software developed outside the database engine, and outside of EdiFlow, by means of some program expressed, e.g., in C++, Java, MatLab. Functions are processes with no side effects ($m = 0$).

Input procedure A special case of procedure is the generic *data input procedure* $p_{in}(R^w)$, which does not return any result. This procedure is implemented by some user interface mechanism, and it allows the user to provide values for new tuple which is added to R.

Delta handlers Associated to a procedure may be *procedure delta handlers*. Given some update (or delta) to a procedure input relation, the delta handler associated to the procedure may be invoked to propagate the update to a process. Two cases can be envisioned:

1. Update propagation is needed while the procedure is being executed. Such is the case for instance of procedures which compute point coordinates on a screen, and must update the display to reflect the new data.

2. Updates must be propagated after the procedure has finished executing. This is the case for instance when the procedure performs some quantitative analysis of which only the final result matters, and such that it can be adjusted subsequently to take into account the deltas.

The designer can specify one or both of these handlers. Formally, each handler is a procedure in itself, with a table signature identical to the main procedure. The convention is that if there are deltas only for some of p's inputs, the handler will be invoked providing empty relations for the other inputs. With respect to notations, $p_{h,r}$ is the handler of p to be used while p is *r*unning, and $p_{h,f}$ is the handler to be used after p *f*inished. Just like other procedures, the implementation of handlers is opaque to the process execution framework. This framework, however, allows one to recuperate the result of a handler invocation and inject it further into the process, as we shall see.

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 ΔX be the set of tuples added to X. If p is distributive then:

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

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

Expressions We use a simple language for expressions, based on queries and procedures. More formally:

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

The simplest expressions are queries. More complex expressions can be obtained by calling a procedure p, and retaining only its j-th output table. If p changes some of its input table, evaluating the expression may have side effects. If the side effects are not desired, p can be invoked by giving it some new empty tables, which can be memory-resident, and will be silently discarded at the end of the process. Observe that the first n invocation parameters are expressions themselves. This allows nesting complex expressions.

5.3.3 Process part

Activities We are now ready to explain the building blocks of our processes, namely activities.

$$a ::= v \leftarrow \alpha \mid upd(R) \mid (S_1, S_2, \ldots, S_n) \leftarrow p(e_1, e_2, \ldots, e_n, T_1^w, T_2^w, \ldots, T_n^w)$$

Among the simplest activities are *variable assignments* of the form $v \leftarrow \alpha$. Another simple activity is a declarative update of a table R, denoted $upd(R)$. Unlike the table modifications that an opaque procedure may apply, these updates are specified by a declarative

SQL statement. Finally, an activity may consist of invoking a procedure p by providing appropriate input parameters, and retaining the outputs in a set of tables.

Visualization activities must be modeled as procedures, given that their code cannot be expressed by queries.

Processes A process description can be modeled by the following grammar:

$$P ::= \epsilon \mid a, P \mid P \| P \mid P \vee P \mid e?P$$

In the above, a stands for an activity. A process is either the empty process (ϵ), or a *sequence* of an activity followed by a process (,), or a *parallel (and) split-join* of two processes ($\|$), or an *or split-join* of two processes (with the semantics that once a branch is triggered, the other is invalidated and can no longer be triggered). Finally, a process can consist of a *conditional block* where an expression e (details below) is evaluated and if this yields true, the corresponding process is executed.

Reactive processes A *reactive process* can now be defined as a 5-tuple consisting of a set of relations, a set of variables, a set of procedures, a process and a set of *update propagations*. More formally:

$$RP ::= R^*, v^*, p^*, P, UP^*$$

5.4 Propagating changes on workflow process

An *update propagation* UP specifies what should be done when a set of tuples, denoted ΔR, are added to an application-dependent relation R, say, at $t_{\Delta R}$. Several options are possible. We discuss them in turn, and illustrate with examples:

1. Ignore ΔR 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 locking at process instance granularity, 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.

   Concrete example: an application instance would be to compute statistics over the current state of the database. The user has to launch its scientific workflow in an isolated environment (snapshot of the database). The updates must not be included in the computation process.

   Another example would be a social scientist who applies a sequence of semi-automated partitioning and clustering steps to a set of Wikipedia pages. Then, the scientist visualizes the outcome. In this case, propagating new items to the visualization would

be disruptive to the user, which would have to interrupt her current work to help apply the previous steps to the new data.

2. Ignore ΔR 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 starts after* $t_{\Delta R}$ may also use this data.

   The data visible to a given activity instance may depend on its starting time. To achieve this, we use the activity instance table that stores all information about activities such as their status, starting time and ending time. All activities that are already started before $t_{\Delta R}$ will not include the updated data. Updates will be only propagated for coming activities.

   Concrete example: the social scientist working on a Wikipedia fragment first has to confirm personal information, give some search criteria for the pages to be used in this process. Then, she must interact with a visualization of the chosen pages. For this activity, it is desirable to provide the user with the freshest possible snapshot, therefore additions between the beginning of the process instance, and the moment when the user starts the last activity, should be propagated.

3. As a macro over the previous option and the process structure, one could wish for ΔR to be propagated to *instances of all activities that are yet to be started in a running process.*

   Intuitively, data should not "disappear" during the execution of a process instance (unless explicitly deleted). In the previous use case, if we add an extra activity at the end of the process, that activity would typically expect to see the whole result of the previous one.

4. Propagate the update ΔR to *all the terminated instances of a given activity.* We can moreover specialize the behavior on whether we consider only activity instances whose *process instances have terminated*, only activity instances whose *process instances are still running*, or both.

   Concrete example: we consider a process whose first activities are automatic processing steps, e.g., computing *diff*s between the old and the new version of a Wikipedia page, updating a user's contribution, the page history etc. The last activity is a visualization activity where the scientist should be shown fresh data. Typically, the visualization activity will last for a while, and it may refresh itself at intervals, to reflect the new data. In this case, it makes sense to apply the automated processing activities to the new pages received while running the process instance, even after the respective activities have finished.

5. Propagate the update ΔR to all the running instances of a given activity, whether they had started before $t_{\Delta R}$ or not.

This may be used to propagate newly arrived tuples to all running instances of a visualization activity, to keep them up-to-date.

Formally then, an update propagation action can be described as:

$$UP ::= R, a, (('\text{ta}', ('\text{rp}'|'\text{tp}')) \mid '\text{ra}' \mid ('\text{fa}', '\text{rp}'))$$

where R is a relation and a is an activity. An update propagation action describes a set of instances of activity a, to which the update ΔR must be propagated. The possible combinations of terminal symbols designate:

ta rp: *t*erminated *a*ctivity instances part of *r*unning processes;

ta tp: *t*erminated *a*ctivity instances part of *t*erminated processes;

ra: *r*unning *a*ctivity instances (obviously, part of running processes);

fa rp: *f*uture *a*ctivity instances part of *r*unning *p*rocesses.

It is possible to specify more than one compensation action for a given R and a given activity a. For instance, one may write: $(R, a,$ 'ra'), $(R, a,$ 'fa', 'rp').

For simplicity, the syntax above does not model the macro possibility numbered 3 in our list of options. One can easily imagine a syntax which will then be compiled into UP's as above, based on the structure of P.

5.5 EdiFlow architecture

Our proposed architecture is depicted in Figure 5.3. This architecture is divided into 3 layers:

- The DBMS: The workflow management logic runs on top of the DBMS. The database ensures the relation between the layers. It contains all information about the process execution and data tables of several entities.
- The EdiFlow process: It corresponds to the XML specification of the process. Processes are specified in a high-level syntax following the structure described in Section 5.3.
- The modules: This is a set of procedures and functions invoked by the user through the process file. These modules may correspond to visualization softwares.

The enactment of a process thus specified consists of adding the necessary tuples to the Process and Activity relations. During process executions, the necessary data manipulation statements are issued 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 and (*iii*) record the connections between users and application instances, and application data.

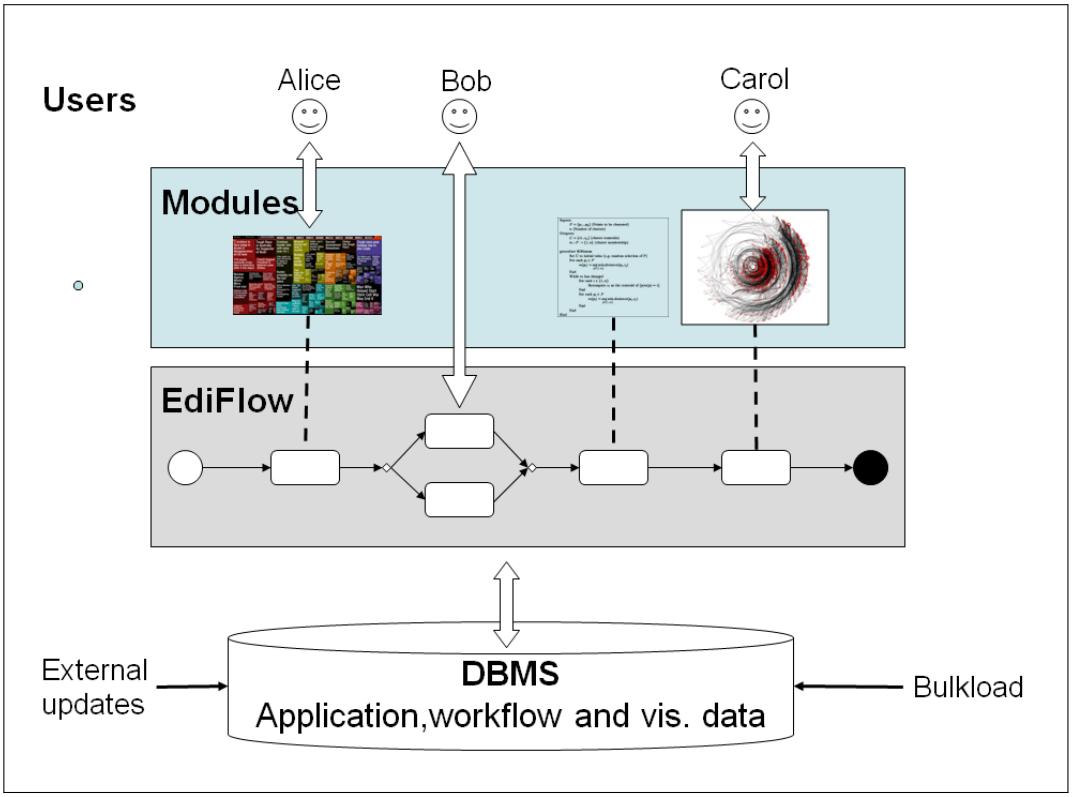

Figure 5.3: The high level architecture of EdiFlow.

5.5.1 Benefits of using a DBMS

Most current visual analytics tools have some conceptual drawbacks. Indeed, they rarely rely on persistent databases (with the exception of [36]). Instead, the data is loaded from files or databases and is manipulated directly in memory because smooth visual interaction requires redisplaying the manipulated data 10-25 times per second. Standard database technologies do not support continuous queries at this rate; at the same time, ad-hoc in-memory handling of classical database tasks (e.g., querying, sorting) has obvious limitations. Based on the architecture of several visualization tools [3; 33; 48], we argue connecting a visual analysis tool to a persistent database management system (DBMS) has many benefits:

- Scalability: larger data volumes can be handled based on a persistent DBMS.
- Persistence and distribution: several users (possibly on remote sites) can interact with a persistent database, whereas this is not easily achieved with memory-resident data structures. Observe that users may need to share not only raw data, but also visualizations built on top of this data. A visualization can be seen as an assignment of visual attributes (e.g., X and Y coordinates, color, size) to a given set of data items. Computing the value of the visual attributes may be expensive, and/or the choice

of the visualized items may encapsulate human expertise. Therefore, visualizations have *high added value* and it must be easy to store and share them, e.g., allowing one user to modify a visualization that another user has produced.

- Data management capabilities provided by the database: complex data processing tasks can be coded in SQL and/or some imperative scripting language. Observe that such data processing tasks can also include user-defined functions (UDFs) for computations implemented outside the database server. These functions are not stored procedures managed by the database (e.g., Java Stored Procedure). These are executable programs external to the database.

We now discuss the implementation of the update propagation actions. EdiFlow compiles the UP (update propagation) statements into statement-level triggers which it installs in the underlying DBMS. The trigger calls EdiFlow routines implementing the desired behavior, depending on the type of the activity (Section 5.3), as follows. Variable assignments are unaffected by updates. Propagating an update ΔR_i to relation R_i to a query expression leads to incrementally updating the query, using well-known incremental view maintenance algorithms [42]. Propagating an update to an activity involving a procedure call requires first, updating the input expressions, and then, calling the corresponding delta handler.

5.5.2 Synchronizing disk-resident and in-memory tables

Tracking dynamic changes in DBMS tables is an essential mechanism of EdiFlow for two purposes: internally, to manage the details to the *delta handlers* and, externally, to implement data caches in modules. We describe here the low-level implementation of the mechanism.

The main feature of EdiFlow is to react to dynamic data changes; reaction management complicates the logic of the system but also its implementation. To implement *delta handlers*, EdiFlow needs to keep track of changes occurring in the DBMS since the last time a module has been invoked because the *delta handlers* are invoked with the list of changed tuples.

The mechanism used to collect the updates for EdiFlow has been extended to manage cached tables that are kept consistent with tables in the DBMS. When a DBMS table is cached in an EdiFlow module, the in-memory table is kept consistent with the on-disk table in both directions: changing the in-memory table propagates to the on-disk table and vice-versa: changes done externally to the DBMS are propagated to the in-memory cache, potentially invalidating entries.

EdiFlow implements a protocol to efficiently propagate updates made to a disk-resident table, call it R_D, to its possibly partial memory image, call it R_M. Conversely, when the module modifies R_M, these changes must be propagated back to R_D. Observe that R_M exists on the client side and therefore may be on a different host than R_D.

To that end, we install CREATE, UPDATE and DELETE triggers monitoring changes to the persistent table R_D. Whenever one such change happens, the corresponding trigger adds to the Notification table stored in the database (recall the data model in Figure 5.1) one tuple of the form (seq_no, ts, tn, op), where seq_no is a sequential number, ts is the update timestamp, tn is the table name and op is the operation performed. Then, a notification is sent to R_M that "there is an update". Smooth interaction with a visualization component requires that notifications be processed very fast, therefore we keep them very compact and transmit no more information than the above. A notification is sent via a socket connected to the process instance holding R_M. Information about the host and port where this process runs can be found in the Client table (Figure 5.1). When the visualization software decides to process the updates, it reads them from the Notification table, starting from its last read seq_no value.

The table has a unique constraint on TABLE_Name and ROW_ID; so if two triggers are fired for the same row, only one entry remains. The operation kept in the table is computed as follows:

- CREATE + CREATE = CREATE (should not happen)
- CREATE + UPDATE = CREATE
- CREATE + DELETE = DELETE
- UPDATE + CREATE = CREATE (should not happen)
- UPDATE + UPDATE = UPDATE
- UPDATE + DELETE = DELETE
- DELETE + CREATE = UPDATE (weird)
- DELETE + UPDATE = DELETE (should not happen)
- DELETE + DELETE = DELETE (should not happen)

After the trigger has filled the NOTIFICATION table, it writes to a socket connected to the R_M process to notify it. This is done through a CLIENT table managing information for each DB client. The synchronization protocol between R_M and R_D can be summarized as:

1. A *memory object* is created in the memory of the Java process (R_M).
2. It asks the *connection manager* to create a connection with the database.
3. The connection manager creates a network port on the local machine and associates locally a quadruplet to R_M: $(db, R_D, ip, port)$.

4. The quadruplet is sent to the DBMS to create an entry in the ConnectedUser table.

5. The DBMS connects back to the client using at the $ip : port$ address, and expects a HELLO message to check that it is the right protocol.

6. The connection manager accepts the connection, sends the HELLO message and expects a REPLY message to check that it is the expected protocol too.

7. When the R_D is modified, the DBMS trigger sends a NOTIFY message with the table name as parameter to client at *ip*:*port*, which holds R_M.

8. The visualization software may decide what are the appropriate moments to refresh the display. When it decides to do so, it connects to the DBMS and queries the created/updated/deleted list of rows, and propagates the changes to R_M.

9. When R_M is modified, it propagates its changes to the R_D and processes the triggered notifications in a smart way to avoid redundant work.

10. When R_M is deleted, it sends a disconnect message to the database that closes the socket connection and removes the entry in the ConnectedUser table.

11. The Notification table can be purged of entries having *seq_no* lower than the lowest value in the Client table.

Note that some DBMS already provide an in-memory extension (e.g. Oracle In-Memory Database Cache). We do not rely on these systems for two reasons: 1) they are usually available for specific DBMS and we want EdiFlow to be able to work on any SQL DBMS and, 2) we encapsulate our in-memory data objects with a thin wrapper that can adapt to new implementations so nothing prevents an EdiFlow application from using a specific in-memory extension instead of the generic one.

At first glance, this mechanism may look similar to updates over views (a.k.a. *materialized views*). However, our architecture has two main differences compared to materialized views:

- Propagation process. The propagation process for materialized views is relatively simple. Indeed, when changes occur on relations, the corresponding relevant views are updated. The difficulty is to know "when" and "how" the view should be updated. Moreover, updates are generally limited to insert operations. However, in our architecture, a change that occurs on a relation may invoke many different update operations which generally correspond to external program's invocations. This is what we call *repair mechanism*.

- Two-way propagation. In the framework of materialized views, updates are usually done in one way (relation towards view). However, our architecture allows to manage

changes that occur on the database while the analysis process is running. Moreover, it allows to update the database when users perform visual interaction.

5.5.3 EdiFlow tool implementation

EdiFlow relies on a standard persistent DBMS for managing the workflow-related data and the application data. EdiFlow reads the specification of a reactive workflow from an XML file. It is then compiled into the corresponding Process and Activity tuples and the workflow is started. Users may control the EdiFlow process step by step or let it run automatically.

The execution is control-driven from the specification and also data driven in the sense that it reacts in a well-specified way to database updates.

To implement the data-driven behavior, with the collection of tuples constituting the delta, EdiFlow installs database triggers, automatically derived from the process specification. Each trigger invokes EdiFlow through a fast network-based notification protocol which performs the necessary actions.

EdiFlow procedures are implemented as Java modules using the Equinox implementation of the OSGi Service Platform [65]. A procedure instance is a concrete class implementing the `EdiFlowProcess` interface. This interface requires four methods: `initialize()`, `run(ProcessEnv env)`, `update(ProcessEnv env)` and `String getName()`. The class `ProcessEnv` represents a procedure environment, including all the useful information about the environment in which the processes are executed. An instance of `ProcessEnv` is passed as a parameter to a newly created instance of a procedure. Integrating a new processing algorithm into the platform requires only implementing one class, and serving the calls to the methods. All the dependencies in term of libraries (JAR files) are managed by the OSGi Platform.

The implementation is very robust, well documented, efficient in term of memory footprint and lightweight for programming modules and for deploying them, which is important for our goal of sharing modules.

The use of a DBMS provides *scalability*, *persistence* and *distribution*. Several users and applications, possibly on remote sites, can interact with a DBMS.

Applications may require different levels of sharing (or, conversely, of isolation) among concurrent activities and processes. EdiFlow implements various degrees of isolation between concurrent processes operating on top of the same database: (i)Process- and activity-based isolation and (ii) Timestamp-based isolation.

EdiFlow saves traces of process executions. These saved traces include all processes and activities specifications and states (started, running, finished). This information allows us to know who is working on what to perform the operation change propagation to the rest of the users and activities.

EdiFlow enforces user rights management: for the effective management of users, the EdiFlow's data model provides two tables (a connected users table and a table for managing groups). Individual users may belong to one or several groups. A user may perform some activity instances, and thus be involved in specific process instances. The ConnectedUser's table records the host and port from which a user connects at a given time. This information is needed to propagate updates, received while the process is running, to a potentially remote visualization component running on the remote user's desktop.

All these mechanisms are already useful to implement some visual analytics applications but the management of visualizations and interactions is left to the modules. Since relying on a remote DBMS connection does not provide the required speed for visualization and continuous interaction, each module would have to implement some mechanisms to load data in memory and keep it updated with the database.

5.6 Isolation management in EdiFlow

Applications may require different levels of sharing (or, conversely, of isolation) among concurrent activities and processes.

Process- and activity-based isolation Let a_1 be an instance of activity a, such that a_1 is part of a process instance p_1. By default, queries evaluated during the execution of p carry over the whole relations implementing the application-specific data model. Let R be such a relation.

It may be the case that a_1 should only see the R tuples created as part of executing p_1. For instance, when uploading an experimental data set, a scientist only sees the data concerned by that upload, not the data previously uploaded by her and others. Such isolation is easily enforced using relationships between the application relations and the *ActivityInstance* table (recall Figure 5.1 in Section 5.2). A query fetching data from R for a_1 should select only the R tuples created by p_1, the process to which a_1 belongs, etc. These mechanisms are fairly standard.

Time-based isolation As discussed in Section 5.3, the data visible to a given activity or process instance may depend on the starting time of that instance. To enable such comparisons, we associate to each application table R a *creation timestamp*, which is the moment when each R tuple entered the database (due to some process or external update). R tuples can then be filtered by their creation date.

Isolating process instances from tuple deletions requires a different solution. If the process instance p_3 erases some tuples from R, one may want to prevent the deleted tuples from suddenly disappearing from the view of another running process instance, say p_4. To prevent this, tuples are not actually deleted from R until the end of p_3's execution. We denote that moment by $p_3.end$. Rather, the tuples are added to a *deletion table* R_{-}. This table holds tuples of the form $(tid, t_{del}, pid, \perp)$, where tid is the deleted R tuple identifier,

t_{del} the deletion timestamp, *pid* the identifier of the process deleting the tuple. The fourth attribute will take the value $p_3.end$ at the end of p_3. To allow p_3 to run as if the deletion occurs, EdiFlow rewrites queries of the form select * from R implementing activities of p_3 with:

```
select * from R where tid not in
                    (select tid from R_ where pid=p3)
```

When p_3 terminates, if no other running process instance uses table R[1], then we delete from R and R_- the tuples $\sigma_{pid=p_3}(R_-)$. Otherwise, R and R_- are left unchanged, waiting for the other R users to finish. However, a process instance started after $t_0 > p_3.end$ should not see tuples in R_- deleted by p_3, nor by any other process whose end timestamp is smaller than t_0. In such a recently-started process, a query of the form select * from R is rewritten by EdiFlow as:

```
select * from R where tid not in
                    (select tid from R_ where processend < t0)
```

We still have to ensure that deleted tuples are indeed eventually deleted. After the check performed at the end of p_3, EdiFlow knows that some deletions are waiting, in R_-, for the end of a process instances started before $p_3.end$. We denote these process instances by $wait_{R,p_3}$. After $p_3.end$, whenever a process in $wait_{R,p_3}$ terminates, we eliminate it from $wait_{R,p_3}$. When the set is empty, the tuples $\sigma_{pid=p_3}(R_-)$ are deleted from R and R_-.

5.7 Experimental results

In this section, we report on the performance of the EdiFlow platform in real applications.

5.7.1 Experimental setup

Our measures used a client PC with Intel 2.66GHz Dual Core CPUs and 4GB memory running. Java heap size was set to 850MB. The Oracle database is mounted on a workstation with 8 CPUs equipped with 8GB RAM. The PC is connected to the database through the local area network.

The experiments are divided into two classes. While the first family of experiments has been developed to validate the use of EdiFlow in several real use cases, the second family of experiments has been involved to study the performance of EdiFlow for processing updates.

5.7.2 Datasets

We present in what follows a description of the datasets used in experiments.

[1]The definition of a process explicitly lists the tables it uses, and from the process, one may retrieve the process instances and check their status (Figure 5.1).

US Election dataset The US.Election dataset presents all the voting results for the U.S. elections of 2000, focusing on the scores for the Democratic and Republican parties. This data set was obtained from data sets provided in the platform IVTK. Table 5.1 presents a sample of this dataset that provides information on the votes of each region of the United States but said the votes of the years 1992, 1996 to study the trends of each region.

Party in 1996	Party in 1992	Electoral Votes	Bush Votes	Gore Votes	Population
Republican	Republican	515096	203053	1722850	Rockies Utah
Democrat	Democrat	1061949	981720	4877185	Central Tennessee
Democrat	Democrat	878502	1616487	6016425	North East Massachusetts
Republican	Republican	190700	118804	696004	Far Midwest South Dakota

Table 5.1: Sample of the US.Election dataset

INRIA activity report dataset We used a dataset of co-publications between INRIA researchers. The data are collected from Raweb (INRIA's legacy collection of activity reports available at http://ralyx.inria.fr). These data include information about INRIA teams, scientists, publications and research centers. Our goal was to build a self-maintained application which, once deployed, would automatically and incrementally recompute statistics, as needed. To that end, we first created a database out of all the reports for the years 2005 to 2008. Simple statistics were then computed by means of SQL queries: age, team, research center distribution of INRIA's employees, etc. The dataset includes several tables such as persons, institutions, affiliations, etc. Table 5.2 illustrates a sample from the person's table.

firstame	lastname	affiliation	categoryPro	research-centre	moreinfo	hdr
Serge	Abiteboul	INRIA	Chercheur	Saclay	Senior Researcher	oui
Frédéric	Cazals	INRIA	Chercheur	Sophia	Team leader, DR2 Inria	oui
Wael	Khemiri	CNRS	PhD	Saclay	Allocataire MENRT, Paris 11	non
Nathalie	Pernelle	UnivFr	Enseignant	Saclay	Associate Professor	

Table 5.2: Sample of the person table in INRIA activity reports dataset

Wikipedia dataset The third dataset presents the french version of Wikipedia. This version contains about 1 million articles. We store the current-meta-page dump containing only the metadata of the last revision of each article. The French Wikipedia compressed file of December 2010 is about 4 GB in size. After storing the dump, the next steps consist in getting the revision history of the articles from this large dump file and calculating the relevant aggregated information. The database contains 6 tables:

- aggregated_article: includes statistics information about the article;

- aggregated_diff: includes statistics information about the diffs between the new revisions of articles;
- aggregated_user: includes statistics information about the user;
- detailed_diff: includes the diffs between the new revisions of articles;
- user: includes information about the wikipedias' writers;
- wikipedia_state: includes the title and the last revision of the articles.

The table 5.3 illustrates a sample from the aggregated_diff table.

pageid	revid	revdatetime	userid	charadd	chardel	charmov	pagelength
3	5	2002-09-08 20:49:46	1	1960	0	0	1960
3	10031	2002-10-31 10:11:31	2	0	0	0	1960
3	79617	2003-06-10 09:53:02	1	6	6	0	2023
3	10031	2002-10-31 10:11:31	2	0	0	5409354	1960
5409351	64159522	2011-04-09 07:46:58	2756176	3481	0	0	3481

Table 5.3: Sample of the aggregated_diff table in Wikipedia dataset

5.7.3 Real-case applications

In this section we study various scenarios in which we have integrated the EdiFlow's architecture. We specify for each use case, the overall context and the advantage of integrating EdiFlow. The choice of these scenarios is not arbitrary. Indeed, this choice is guided by the features of each application. The first use case is the US.Election scenario representing a simple workflow for monitoring the results of the American presidential elections as the results arrive. The second use case is Wikireactive involving complex analysis procedures using EdiFlow. The third scenario is the INRIA activity reports scenario seeking to compute a global view of INRIA researchers by analyzing some statistics.

5.7.3.1 US Election scenario

This application aims at providing a dynamic visualization of elections outcome, varying as new election results become available. The database contains, for each state, information such as the party which won the State during the last three elections, the number of voters for the two candidates, the total population of the state. On the voting day, the database gradually fills with new data. This very simple example uses a process of two activities: computing some aggregates over the votes, and visualizing the results. Upon starting, a TreeMap visualization is computed over the database (distinguishing the areas where not enough data is available yet), as shown in Figure 5.5. The user can choose a party, then the

51 states are shown with varying color shades. The more the states vote for the respective party, the darker the color. When new vote results arrive, the corresponding aggregated values are recomputed, and the visualization is automatically updated.

This use case presents a simple workflow for monitoring the results of the American presidential elections as the results arrive. The results are presented using a Treemap visualization managed by a visualization module. It visualizes a data table computed by a data aggregation module that reads the detailed table of the election results and aggregates it into the tree structure. The election results can be updated by humans or be read by a robot from an official web-site.

As shown in the workflow presented in Figure 5.4, the application defines 3 tasks. Each task calls an external function:

- Vote reporting: This function allows retrieving the results of votes from an official site and updating the database as they arrive. The database contains the score of each candidate in county subdivision. Additional information is available such as the winning party during the last three elections, the number of voters for the two candidates and the total population of the county subdivision.

- Aggregate calculation: Following the updating of the counts in the database, the sum of votes of the party in the corresponding state is computed and stored in an aggregated table.

- Update views: The visualization and view are dynamically changed to reflect the results of the votes. Still, the user can change the configuration of the Treemap visualization, map one party or the other to the size of the Treemap or to the color and change the layout. The updates are still reported.

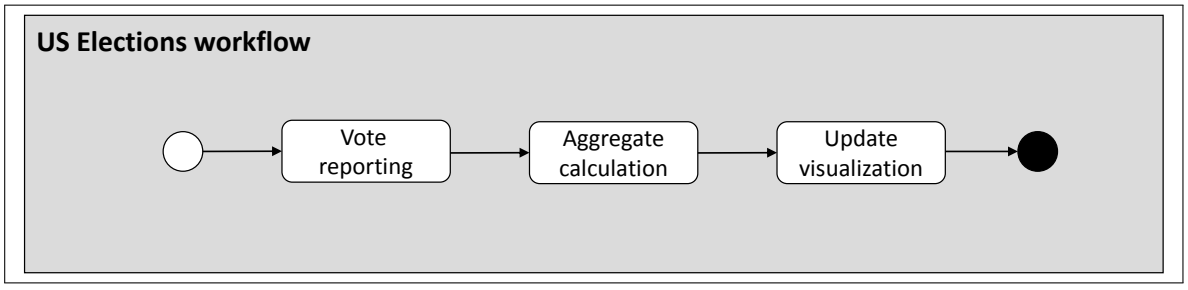

Figure 5.4: US Election workflow

This very simple example uses a process of two activities: computing some aggregates over the votes, and visualizing the results. Initially, a Treemap visualization is computed over the database, highlighting areas where data is still missing, as shown in Figure 5.5. The user can select to color by scores of a specific party; the 51 states are then shown with varying intensities of colors. When new vote results arrive, the corresponding aggregated

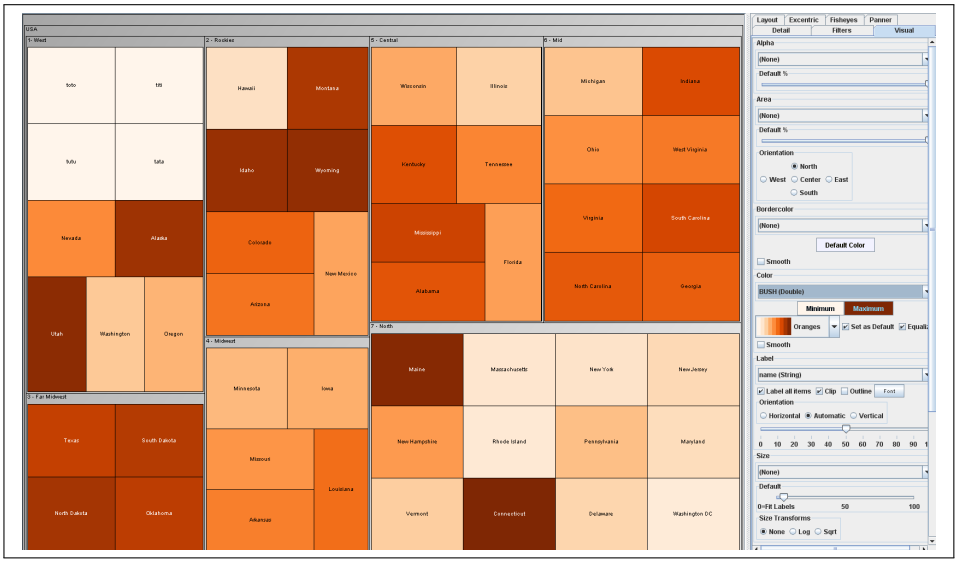

Figure 5.5: US Election screen shot.

values are automatically recomputed by EdiFlow and the visualization is automatically updated.

5.7.3.2 INRIA activity report scenario

We have been involved in the development of an application seeking to compute a global view of INRIA researchers by analyzing some statistics. The data are collected from Raweb (INRIA's legacy collection of activity reports available at http://ralyx.inria.fr). These data include information about INRIA teams, scientists, publications and research centers. Currently, the report of each team from each year is a separate XML file; new files are added as teams produce new annual reports. Our goal was to build a self-maintained application which, once deployed, would automatically and incrementally re-compute statistics, as needed. To that end, we first created a database out of all the reports for the years 2005 to 2008. Simple statistics were then computed by means of SQL queries: age, team, research center distribution of INRIA's employees. Other aggregates were computed relying on external code such as the similarity between two people referenced in the reports in order to determine whether an employee is already present in the database or needs to be added.

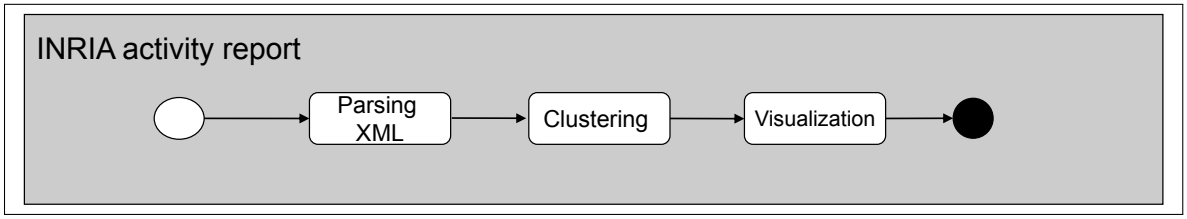

Figure 5.6: INRIA activity report workflow

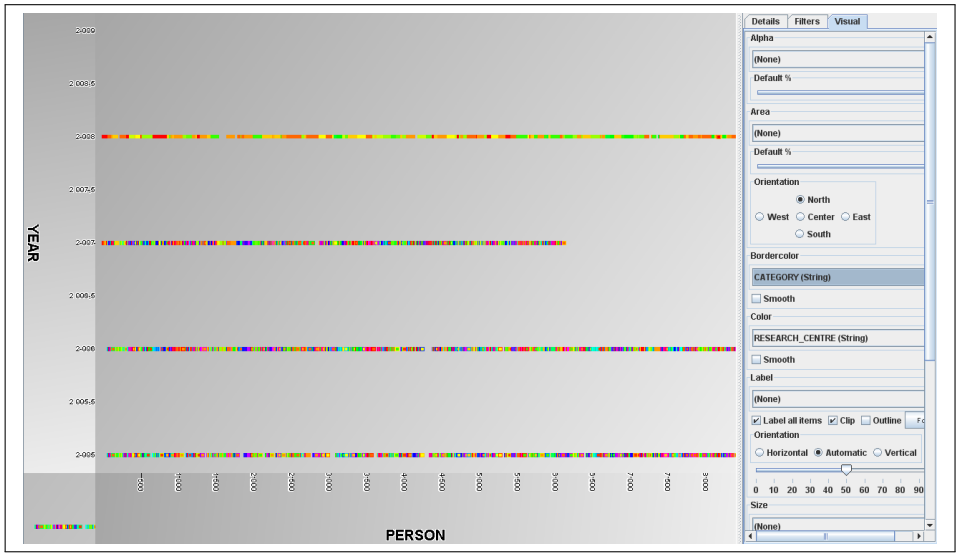

Figure 5.7: Scatter plot of person and the hiring year.

5.7.3.3 WikiReactive scenario

This use case is the re-implementation of the WikiReactive system [17] in which updating a table triggers a series of complex operations performed in cascade. The goal of the application is to provide to Wikipedia readers and contributors several measures related to the quality of an article: how many contributors participated to the article? How did the page evolve over time? How trustworthy are the contributors who wrote it? This last metric corresponding to a ratio: if i is the sum of the numbers of characters inserted by a user in every of her/his contribution and r is the number of characters she/he entered that remain in the latest version of Wikipedia, then the trustworthiness is r/i. The challenge is then to compute, store and maintain these metrics for the whole Wikipedia database. Once it is computed, it can be displayed to users with simple visualization as in the WikipediaViz system [26] or explored more thoroughly with the HistoryFlow system [80].

The metrics should be updated as Wikipedia is updated and, since computing all these metrics from scratch takes days, the system should maintain data structures to allow an incremental update.

WikiReactive can be designed, according to Figure 5.8, in five elementary tasks:

- For each article, compute and maintain the differences (diffs) between successive revisions;
- For each user, maintain the total number of characters added, deleted, and moved;
- For the last revision of each article, compute and maintain the contribution table that stores at each character index the identifier of the user who entered the character;
- For each article, compute the number of distinct contributors;

- For each user, maintain the total number of characters remaining in all the contribution tables.

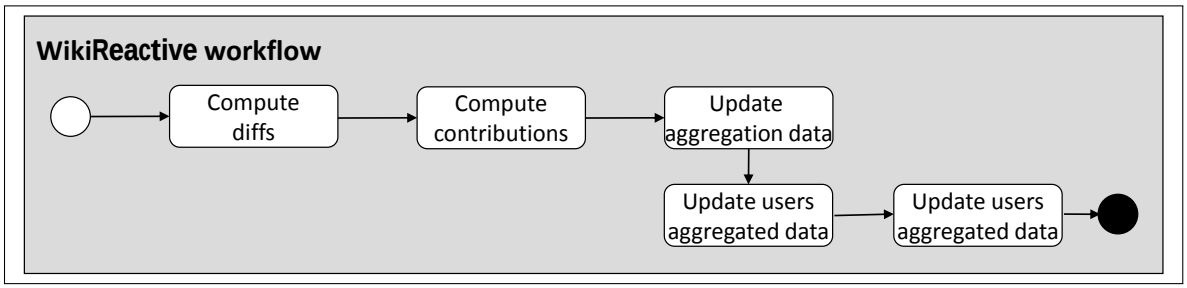

Figure 5.8: WikiReactive workflow

All these applications feature data- and computation-centric processes which must react to data changes while they are running and need visual data exploration. The Wikipedia application is the most challenging, by the size of the database, the complexity of its metrics, and the high frequency of updates requiring re-computations.

5.7.4 Layout procedure handlers

Our first goal was to validate the interest of procedure handlers in the context of data visualization. In our INRIA co-publication scenario, the procedure of interest is the one computing the positions of nodes in a network, commonly known as *layout*. We use the Edge LinLog algorithm of Noack [63] which is among the very best for social networks, and provides good results. What makes EdgeLinLog even more interesting in our context is that it allows for effective delta handlers (introduced as part of our process model in Section 5.3).

In our implementation, the initial computation assigns a random position to each node and runs the algorithm iteratively until it converges to a minimum energy and stabilizes. This computation can take several minutes to converge but, since the positions are computed continuously, we can store the positions in the database at any rate until the algorithm stops. Saving the positions every second or at every iteration if it takes more than one second allows the system to appear reactive instead of waiting for minutes before showing anything.

If the network database changes, for example when new publications are added to/removed from the database, the handler proceeds in a slightly different manner. First, it updates the in-memory co-publication graph, discards the nodes that have been removed and adds new nodes. To each new node it assigns a position that is close to their neighbors that have already been laid-out. This is to improve the time and quality of the final layout. If disconnected nodes are added, they are assigned a random position. Then, the

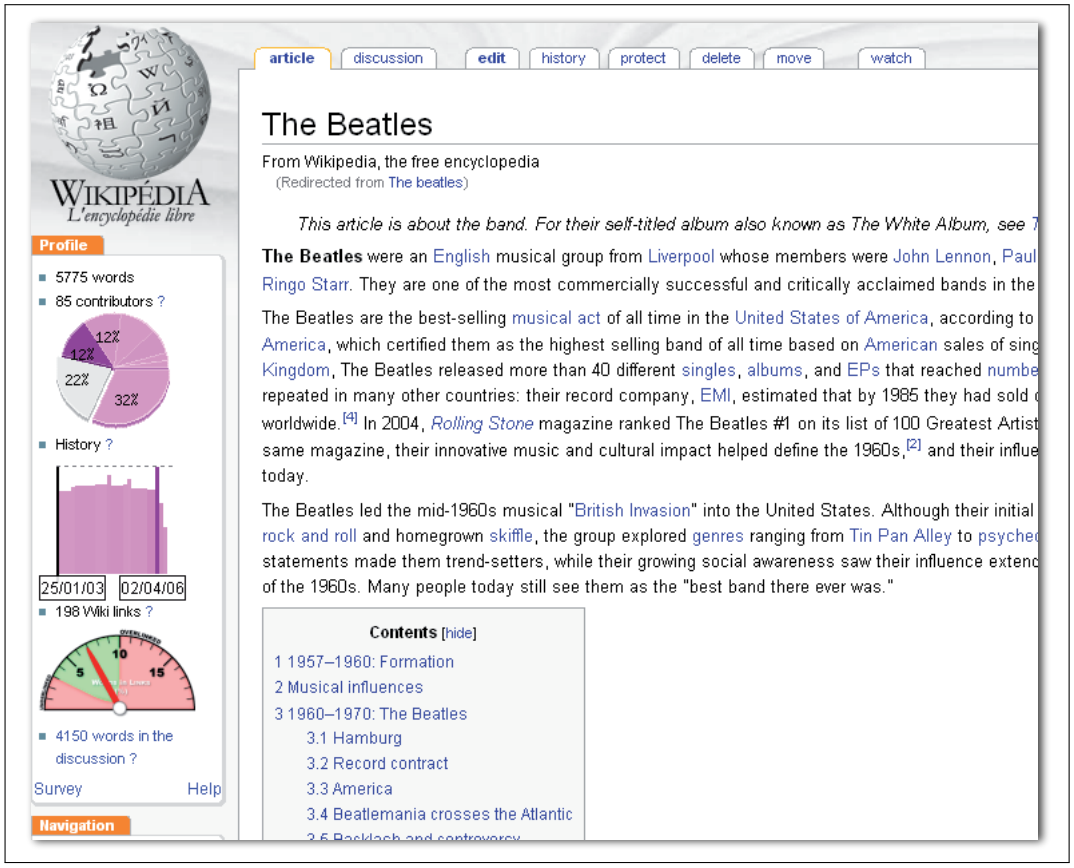

Figure 5.9: Wikipedia screen shot.

algorithm is run iteratively like for the initial computation, but it terminates much faster since most of the nodes will only move slightly: the convergence of the iterations will be much faster. Like before, we store in the DBMS the results of some of the iterations to allow the visualization views to show them.

Using this strategy, we have obtained an incremental layout computation, remarkably stable and fast.

5.7.5 Robustness evaluation

Our second experimental goal was to study how the EdiFlow event processing chain scales when confronted with changes in the data. For this experiment, the DBMS is connected via a 100 MHz Ethernet connection to two EdiFlow instances running on two machines. The first EdiFlow machine computes visual attributes (runs the layout procedure), while the second extracts nodes from VisualAttributes table and displays the graph. This second EdiFlow machine is a laptop.

We study the robustness of our architecture when adding increasing numbers of tuples to the database. Inserting tuples requires performing the sequence of steps below, out of

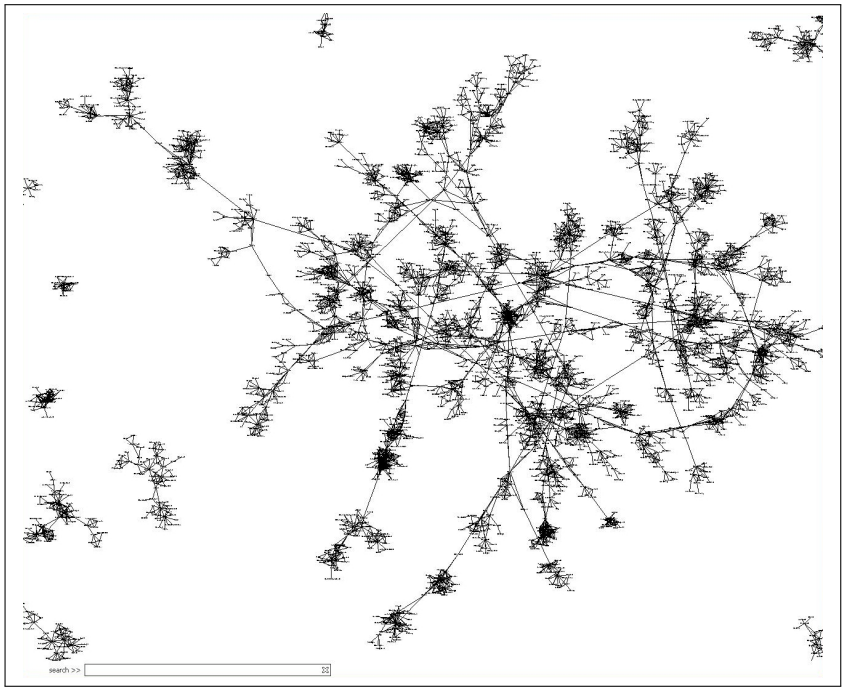

Figure 5.10: Part of the graph of INRIA co-publications.

which steps 1, 2 are performed on the first EdiFlow machine, while steps 3, 4 and 5 are performed on all machines displaying the graph.

1. Parsing the message involved after insertion in nodes table. It refers to step 7 in the protocol described in section 5.5.2.

2. Inserting the resulting tuples in the VisualAttributes table managed by EdiFlow in the DBMS.

3. Parsing the message involved after insertion in VisualAttributes table. After inserting tuples, in VisualAttributes, a message is sent to all machines displaying the graph. The message is parsed to extract the new tuple information. It refers to step 9 in the protocol described in section 5.5.2.

4. Extracting the visual attributes of the new nodes, from the VisualAttributes table, in order to know how to display them at the client.

5. Inserting new nodes into the display screen of the second machine.

The times we measured for these five steps are shown in Figure 5.11 for different numbers of inserted data tuples. Figure demonstrates that the times are compatible with the requirements of interaction, and grow linearly with the size of the inserted data. The dominating time is required to write in the VisualAttributes table. This is the price to pay

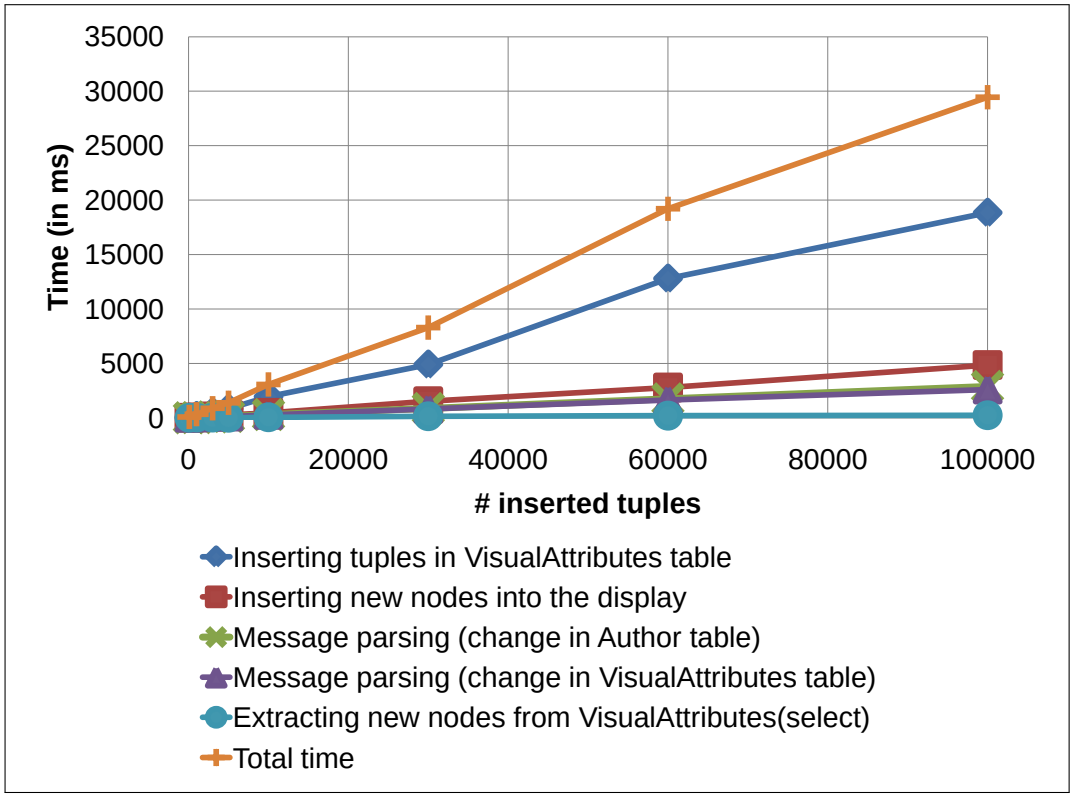

Figure 5.11: Time to perform insert operation.

for having these attributes stored in a place from where one can share them or distribute them across several displays.

5.8 Conclusion

In this chapter, we have described the design and implementation of EdiFlow, the first workflow platform aimed at capturing changes in data sources and launching a repair mechanism. EdiFlow unifies the data model used by all of its components: application data, process structure, process instance information and visualization data. It relies on a standard DBMS to realize that model in a sound and predictive way. EdiFlow supports standard data manipulations through procedures and introduces the management of changes in the data through *update propagation*. Each workflow process can express its behavior w.r.t data change in one of its input relations. Several options are offered to react to such a change in a flexible way.

EdiFlow reactivity to changes is necessary when a human is in the loop and needs to be informed of changes in the data in a timely fashion. Furthermore, when connected to an interactive application such as a monitoring visualization, the human can interactively

perform a command that will change the database and trigger an update propagation in the workflow, thus realizing an interactively driven workflow.

Chapter 6

EdiFlow for visual analytics

Contents

6.1 Introduction

Scalability is a major concern in visual analytics. Processing large amounts of data requires complex systems that are costly to design, implement and maintain. In contrast, managing large amounts of data is common in the domain of *scientific workflows*. A scientific workflow is defined in [25] as follow: "A scientific workflow is a formal specification of a scientific process, which represents, streamlines, and automates the analytical and computational steps that a scientist needs to go through from dataset selection and integration, computation and analysis, to final data product presentation and visualization."

Scientific Workflows systems share many characteristics of visual analytics systems: they combine storage management, complex and distributed analysis capable of handling very large datasets. However, Scientific Workflows are designed to carry automated analytical processes to completion, but not to meet the goals of visual analytics applications. They offer little or no visualization at any step except sometimes to monitor their activity, nor do they offer interaction to steer the computations or interact with the results or the process (with one notable exception [22]). Finally, they do not manage dynamic data.

In the previous chapter, we have described how EdiFlow implements the first two layers of the EdiFlow architecture. For the third layer, leaving the implementation of visualization and interaction to modules raises two questions: 1) is the DBMS architecture fast enough to support interactive visualization, and 2) are there data structures and operations that every visualization module will have to re-implement and that could be factorized by EdiFlow?

The chapter is organized as follows: the next section describes the relation between scientific workflows and visual analytics. The third section summarizes the different experiments made on in-memory and persistent databases. The fourth and the fifth sections describe the design of EdiFlow support for visualization and interaction in details. We then report on application we have built using EdiFlow before concluding.

6.2 Scientific workflows and visual analytics

The reason for integrating Scientific Workflows with visual analytics is to scale the Information Visualization Reference Model [23; 27] beyond the standard program-in-one-process model. This is required because some visual analytics systems need to run continuously to process large amounts of dynamic data in a distributed environment using the best available resources in term of data management, computing power, graphics and interaction. While there are several challenges to address for achieving an infrastructure that meets all the visual analytics goals [54], EdiFlow is a first attempt at implementing a software model of a run-time architecture that is able to perform continuous automatic processing of data as well as interactive visualization, and which can scale and manage dynamic data efficiently.

Translating the Reference Model at the distributed system level requires solving several problems: managing a distributed store, managing a distributed execution architecture and implementing a distributed interaction and visualization architecture. Our inspiring model for managing a distributed store is described by Boncz et al [15]: using a distributed database with several levels of smart caches. For managing a distributed system, we rely on a workflow architecture. For the interaction and visualization, this chapter describes a novel model based on the distributed store instead of adding another ad-hoc mechanism for communication.

On top of this reactive workflow system, EdiFlow also provides specialized mechanisms to implement visualizations and interactions. In this regard, its contributions are:

- the unification of mechanisms for dynamic data management and interaction management: interaction is implemented as data changes,
- the use of a database cache mechanism to achieve the speed required for visualization and computation,
- the definition and implementation of a polylithic visualization model [12] relying on a persistent table that allows for rendering on any display (from small screen to wall-sized display) and collaboration.

EdiFlow is designed to help modules implement the Information Visualization Reference Model. All the implementations of the reference model rely on an in-memory database to hold data structures and sometimes visual structures too (e.g., Prefuse [48]). EdiFlow being database oriented, we want the data structures and visual structures of a visualization module to be consistent with the rest of the EdiFlow architecture. Therefore, we measured the performance of database systems to assess how far DBMS are, compared to in-memory databases.

Interaction and visualization are modeled as *data changes* in EdiFlow. We detail how data changes are managed, then how EdiFlow modules use it to implement interaction and visualization.

We note that the visualization modules are directly related to an in-memory database. Such structure allow a quick refresh of the visualization and facilitates the establishment of a collaborative environment. We detail the purpose of the in-memory databases in section 6.6.

6.3 Experiments

We present the following a set of experiments performed to estimate the cost of using triggers in the dynamic data management protocol in EdiFlow.

6.3.1 Experimental setup

We used a client PC with an Intel 2.66GHz Dual Core processor and 4GB memory running Java on Windows7. The Java heap size was set to 850MB. The DBMS systems we tested are Oracle 11g and MySQL 5. The in-memory database systems we tested are Prefuse [48] and IVTK [33].

We measured the performance of persistent database systems and in-memory database systems.

The goal of the experiment was to measure the response time needed for each system to insert, update and delete data. These operations were performed in three modes: unit, batch and atomic modes (described below). Since the implementations of the Information Visualization Reference Model rely on notifications to manage dynamic data and that the standard way of implementing notifications in persistent database is through triggers, we also measure the operations with and without the call of triggers.

All experiments were carried out on a table containing initially 2 million tuples.

6.3.2 Performance over unit mode

Unit mode: In the so-called unit mode, the three operations (insert, update, and delete) are processed tuple by tuple. Conventional database checks such as functional dependencies, reference integrity and locking are done after each operation. If an operation in invalidated for one tuple during one of the checks, it does not interfere with the operations done before or after. Triggers are invoked in order, once after each operation.

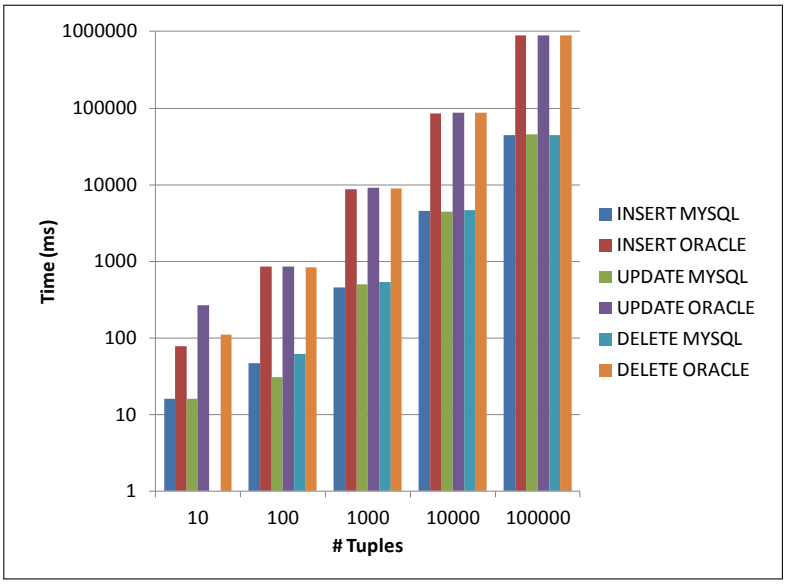

Figure 6.1: Experiments for unit mode without triggers for Oracle and MySQL (log-log scale).

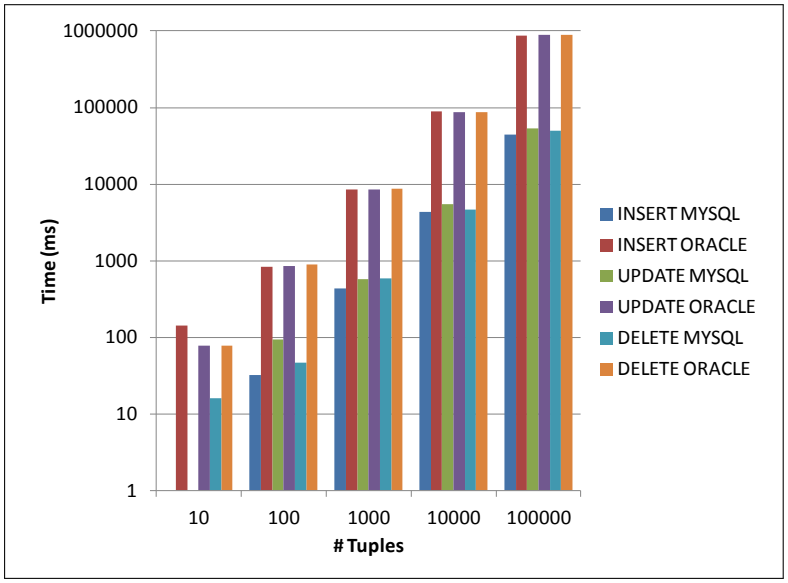

Figure 6.2: Experiments for unit mode with triggers for Oracle and MySQL (log-log scale).

6.3.3 Performance over batch mode

Batch mode: In the so-called batch mode, batched operations are done asynchronously, which improves the performance since the connection between the database server and the database client relies on a communication channel that is more efficient when it sends substantial chunks of data rather small amounts of data that one operation would require. Triggers are also invoked in order, just like in the Unit mode, but only at the end of the batch. The semantic is the same as for the unit mode, the only difference is the delay between the operation performed and the trigger invocation.

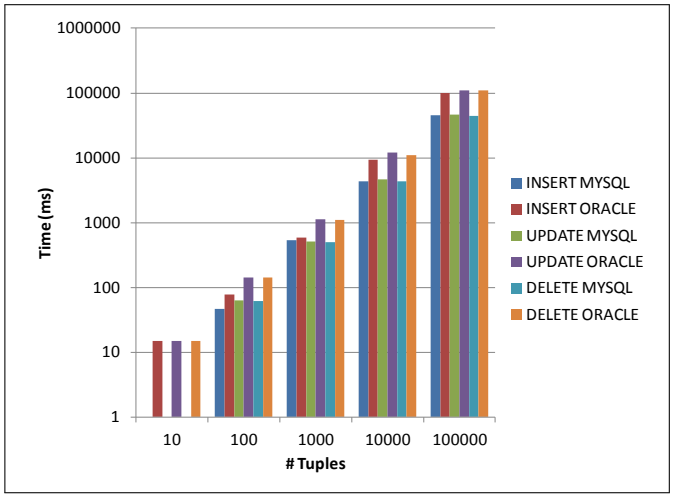

Figure 6.3: Experiments for batch mode without triggers for Oracle and MySQL (log-log scale).

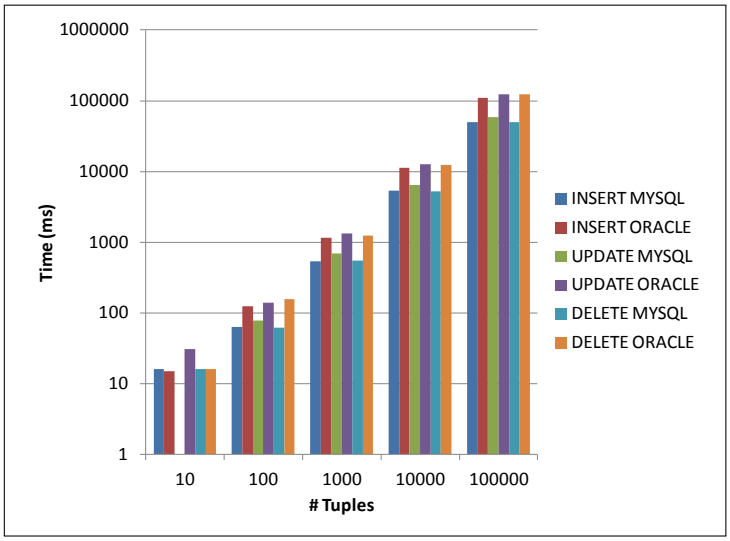

Figure 6.4: Experiments for batch mode with triggers for Oracle and MySQL (log-log scale).

6.3.4 Performance over atomic mode

Atomic mode: In the so-called atomic mode, every operation in the atomic group succeeds or none of them do. All the database checks are done at the end of the transaction and, if the transaction succeeds, triggers are called once for all the tuples concerning one operation (all the inserts, all the updates, all the deletes).

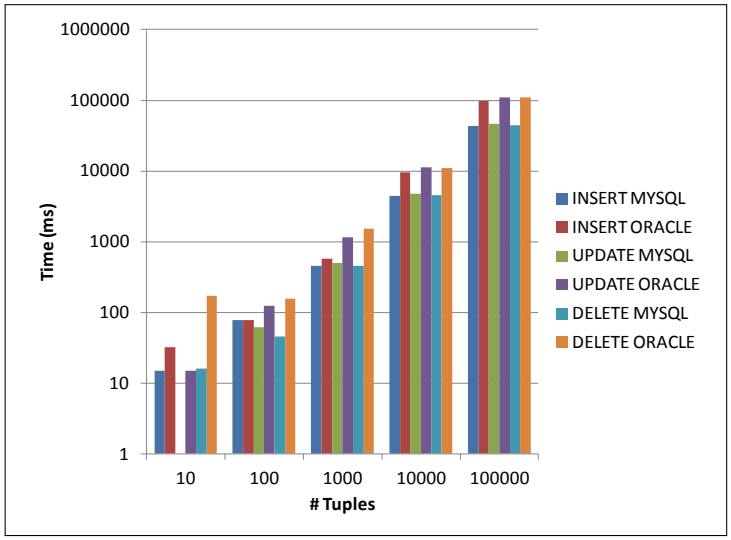

Figure 6.5: Experiments for atomic mode without triggers for Oracle and MySQL (log-log scale).

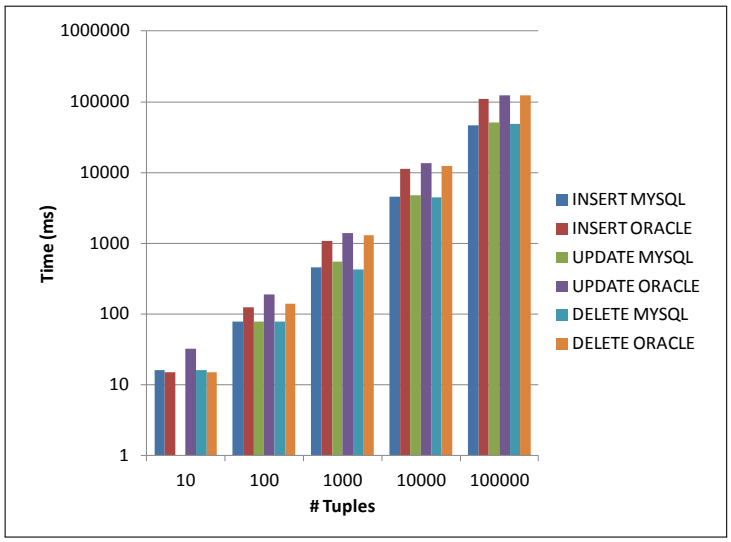

Figure 6.6: Experiments for atomic mode with triggers for Oracle and MySQL (log-log scale).

6.3.5 Performance over in-memory database systems

All of the existing popular toolkits for Information Visualization such as Prefuse [48], the InfoVis toolkit (IVTK) [33], Tulip [10] or Improvise [82], use an in-memory database to manage their data structures to offer the required speed for visualization and interaction. Keeping all the data in memory is out of question in VA applications. However, redisplay and continuous interaction require keeping some data in memory, not all: VA applications

do not visualize all the attributes of all the data structures all the time; therefore some part could be kept out of memory. This can be left to the virtual memory manager: when the data to load is larger than the amount of RAM on a machine, it moves the unused data to disk. However, to be managed by the virtual memory manager, data has to be loaded in memory first, which can take time, partly uselessly when only portions of the data are needed. Furthermore, the memory organization of the data has to be compatible with the strategy of the virtual memory manager. Column-oriented in-memory databases are better suited to the current strategies according to [33].

We present in the following the results of Prefuse and IVTK only for the unit mode. Indeed, the batch and atomic mode are not supported by these systems.

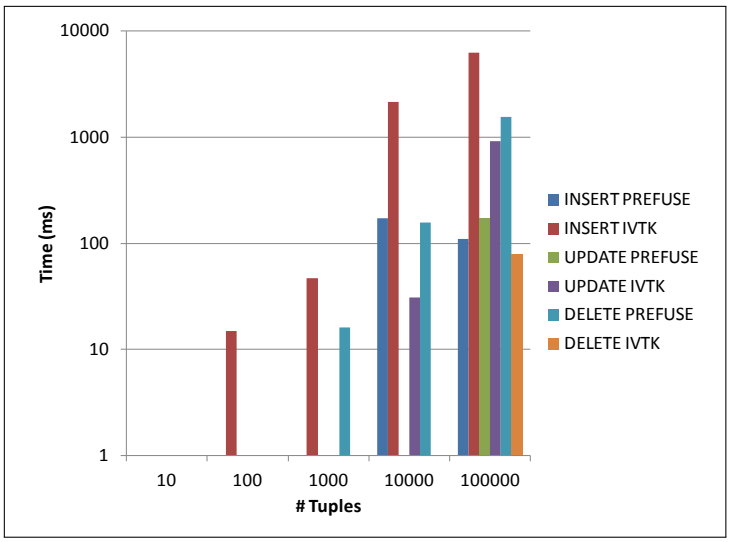

Figure 6.7: Experiments without triggers for Prefuse and IVTK (log-log scale).

6.4 Performance analysis

6.4.1 MySQL and Oracle results

According to the results of unit, batch and atomic modes, MySQL is 2-3 times faster than Oracle, while the general trends are the same for both systems. This can be explained by the additional security mechanisms implemented by Oracle (e.g., ACL) that are checked after each operation on the database. In the unit mode, the Oracle performances are much slower than MySQL, this is because the checks are applied after each operation. However, in batch and atomic modes, performances of Oracle and MySQL are close enough, probably because the checks are only applied once at the transactions end.

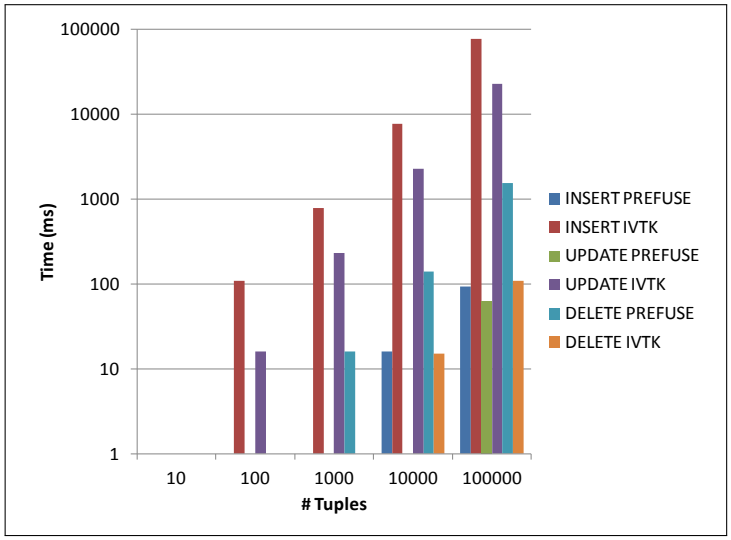

Figure 6.8: Experiments with triggers for Prefuse and IVTK (log-log scale).

To assess the trigger's cost, we performed the same experiments without calling a trigger procedure. The time for insert, update and delete operations increases by about 10% regardless of the DBMS.

Access time is much faster than modification time for both DBMS but Oracle has a constant delay of about 0.5s before it starts returning data from a select statement. MySQL is much faster, serving 100,000 items in 183ms, about 2 microseconds per item. Oracle takes about 5 times more, 10 microseconds per item. These results seem to be compatible with continuous redisplay but these times are averaged and can vary greatly from one call to another.

6.4.2 Prefuse and IVTK results

From the results shown in Figures 6.7 and 6.8, Prefuse and IVTK exhibit similar performance for the "delete" operation, with a slight advantage to Prefuse for the "Insert" and "Update" operations. The gap observed between the results of the two systems is due to the use of different memory allocations constants.

Prefuse and IVTK are 10 times faster than Oracle and MySQL for modification operations. This speed is due to several factors such as lack of controls on data integrity, the data are stored in a cache structure, and no security mechanism is triggered after each insert operation. Nevertheless, Prefuse and IVTK use table updates to manage dynamic queries and selection in interactive time when it would be completely impractical with a DBMS.

Access time for Prefuse and IVTK are about 1ms for 100,000 items, 100 times faster than MySQL.

6.4.3 Discussion

According to Nielsen [62], there are three important orders of magnitude to the perception of response times for a human interacting with a computer:

- 0.1 second: when data is updated on screen below this limit, the user perceives the changes as instantaneous or continuous. This is required for display updates during animation or continuous interaction.
- 1.0 second: limit for users feeling that they are freely navigating the command space without having to unduly wait for the computer.
- 10 seconds: limit for users keeping their focus on the task.

According to the results above, the exclusive use of a persistent database gives a response time not acceptable for continuous interaction but acceptable for the feedback of commands. So, we are faced with a problem: we want to rely on a persistent database for its benefits (scalability, data management, user management, conflict management, ACID properties, and maturity) but we need to use an in-memory database for continuous interaction and visualization.

Therefore, EdiFlow implements a caching mechanism to have the best of both worlds: visualization and interaction will be done on the in-memory database with the required speed. Our model of distributed data store based on a DBMS works, although with some additional complexity.

6.5 Visualization management

EdiFlow implements visualizations using a data table that contains graphic information, such as position (x, y), shape, color, and label. Each module could implement one data table to suit its needs but it is more efficient to share a common table and, furthermore, it offers several interesting benefits.

We describe in the following details about the visual table management in the EdiFlow platform and its visualization architecture allowing several views for the same visualization.

6.5.1 Visual table schema

A visualization module reads data tables, computes graphic attributes associated to these input data tables and populate the visual table of EdiFlow. The module can also display the visualization on a screen or delegate the display to another module or an external application that will read specific items from the database visualization table. Using the EdiFlow caching mechanism, visualization modules are invoked when the input data tables they manage are modified. They can then update the visual table by either recomputing all the attributes or only the ones that have been changed. For example, a graph drawing

module would probably recompute a layout, taking into account all the items in the input tables while a scatter-plot visualization module would only recompute the positions of changed items.

Conceptually and in the implementation, EdiFlow visualization modules are very similar to Prefuse visualizations: they read from data tables and write to data tables that can be rendered.

The concrete schema of the visualization table is:

Name	VisID	X	Y	ObjectID	Shape	Label	SrcTable
Type	String	Real	Real	String	String	String	String

- The *VisID* is a string that identifies the visualization that has set this entry (all the visualizations store their results in the shared visualization table). Visualizations are free to choose any identifier they like.
- X and Y are the 2D position of the item.
- *ObjectID* is a unique identifier to the visualized object in the source table and is used for picking and updating.
- *Shape* is the specification of a geometry; currently, it contains the Java Shape object serialized.
- *Label* is the label of the object or null.
- *SrcTable* is, as its name implies, the name of the table from where this graphic item has been computed. This schema is very simple on purpose and should be enriched when EdiFlow evolves.

6.5.2 An architecture for several views

EdiFlow allows several views to display the same visualization. As shown in Figure 6.9, the visual attributes can be shared by several views and by several users who may choose to visualize some or all of the data; for example on a smart phone showing only 10% of the items, on a laptop showing 30% and on a large display showing 100%.

EdiFlow can maintain several visualization views for one visualization. As shown in Figure 6.9, the visual attributes can be shared by several visualization views and by several users that may choose to visualize some or all of the data (e.g., on an iPhone showing 10% of the data, on a laptop showing 30% and on our WILD Wall-Sized display [83]) showing all of the data.

Moreover, in applications such as the INRIA co-publications example outlined in Section 5.7.3, a user may want to visualize a scatter plot displaying the number of publications per year on one machine and displaying the number of publication by author on another machine. The two are obtained from the same data but using two different views. To

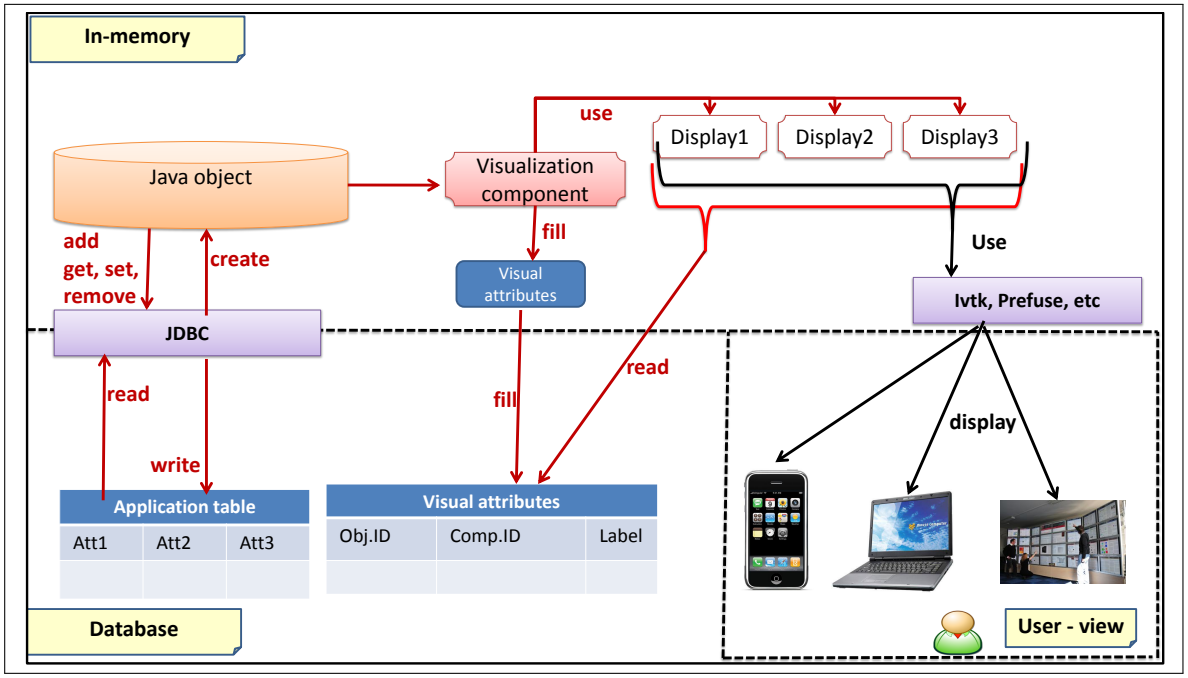

Figure 6.9: EdiFlow architecture for managing several visualization views.

this purpose, the visualization component computes and fills the visual attributes only once regardless of the number of generated views. For each view, a display component is activated to show the data on the associated machine using a visualization toolkit such as Prefuse [48] or the InfoVis Toolkit [33]. This architecture offers several advantages:

- It allows sharing visual attributes by different views and maintaining consistency between data and views.
- The computation of visual attributes is done only once. If an update occurs, the VisualAttributes table is updated and all associated views will be automatically updated.
- Such architecture can satisfy the principle of visualization: a visualization may have several views.

In practice, to display the co-publications graph on the WILD, we used a workstation running the visualization module and a cluster of 16 machines to display the graph over the 32 screens of the WILD (cf. Figure 6.10). Each machine controls two screens and runs an EdiFlow instance to launch visualization view modules. When the data is updated, the DBMS notifies the visualization module to compute new visual attributes and to insert them into the in-memory table to refresh the visualization. The VisualAttributes table is then synchronized with the in-memory table. The database notifies the running visualization view modules that they need to refresh all displays.

Figure 6.10: INRIA co-publications graph on a wall-sized display.

6.6 Interaction management

In an interactive system, interaction is managed in one or two phases. Commands are operations performed by the user using an interaction device such as a keyboard or a mouse (e.g., clicking on a button widget). They require an explicit validation and some feedback (e.g., pushing then releasing a physical mouse button with some highlight of the widget button on screen). When the command is executed, the state of the system is modified.

Commands are discrete actions and perceived as discrete by the user. There are also continuous actions that are important for dynamic queries and smooth interaction. For example, a graph visualization system can allow nodes to be moved interactively on screen using a mouse. Moving nodes should be perceived as continuous by users, even if the actual implementation uses discrete events. Depending on the semantic of the operation and on the performance of the system, continuous interactions can be implemented as multiple discrete commands or as a "chunk" where the command is actually executed at the end of the chunk, even if the system provides a continuous feedback.

For example, Window Explorer allows files to be dragged and dropped from one window to another or on top of an application to launch the application with the file as parameter, even to the trash can to discard the file. When a file is dragged, the icon is continuously moved with the mouse pointer but the actual command is only triggered when the mouse button is released. This is an example of continuous feedback that does not change the system's state until the end of the chunk.

On the contrary, a graph-drawing visualization can consider that when nodes are moved using a mouse-drag, the positions associated with the nodes are updated at each mouse

movement and the system's state is changed each time. In that case, it is a design choice since, in the end, the user will not see much difference between a deferred commands and a continuous action. At best, for a deferred action, only "ghost" nodes will be dragged interactively while the real nodes stay in place before the action is ended [34].

Using EdiFlow, modules have no choice for continuous interaction: they have to defer the command to the end of the interaction chunk and perform a local continuous feedback. Selecting 100 nodes and moving them interactively by directly changing the visualization table would send a number of updates to the DBMS, incompatible with the continuous interaction speed. We believe that this is a small price to pay to benefit from the whole EdiFlow environment.

According to the Information Visualization Reference Model, the interaction can impact the view, the visualization, the data and, in the case of visual analytics, it can also impact any analytical module. Since all of the modules read their input and parameters from DBMS tables, changing a parameter boils down to changing a value in a table and the propagation will be done by EdiFlow. If several parameters need to be changed at the same time, say a visualization module will use another algorithm and a set of new parameters related to this algorithm, then an atomic transaction should be used to set all the parameters at the same time to avoid the visualization module being reactivated several times.

If EdiFlow modules want to provide continuous interaction on large visualizations, the only solution is integration or sampling. Integration means that, for example, a visualization module will also implement a view so that changing interactively a color-mapping or a layout can result in a smooth animation being performed inside the module, even if only the final result is propagated to the DBMS. Sampling means that the rate at which the changes will be sent to the DBMS should be adapted to the speed of the feedback received. This strategy, while theoretically possible with EdiFlow, needs to be further experimented. So far, we have only implemented simple strategies.

There are several in-memory database systems (Protovis [16], Jung [64], IVTK [33], etc.). We used Prefuse and IVTK through EdiFlow. Include such components in the architecture of EdiFlow allows us to benefit from the power and functionalities of the persistent databases, but also get even faster response times. By bringing data closer to the application and processing queries in an in-memory database, applications are able to access and update data with much shorter response times.

Thus, during each refresh operation, the in-memory database is updated to achieve a response time of one second. A synchronization phase is still scheduled with the persistent database so other users can be informed when some objects are being updated.

One consequence of the unification of interaction with data updates is the possibility of collaboration between multiple users since objects stored in the DBMS, in particular the visual table, can be shared. We report on some experiments regarding collaboration and conflict avoidance in the example section 6.7.

While EdiFlow does not currently implement any high-level service for collaboration, we plan on designing one in the near future since sharing information across the network comes for free.

To avoid conflicts in the collaboration environment, we use the notification process that follows the same principle of dynamic management of data.

Take for example the co-publications graph as application. If a user is manipulating a set of objects, information regarding the handling is stored in a collaboration table. If other users are currently viewing all or part of objects viewed by the first user, all users will be notified that these objects are being manipulated and to avoid conflict they have to wait until the user release the corresponding objects.

The level of notification depends on the level of interaction. The notification information is sent in real time to all users (concerned by manipulated objects). This information can be modeled as a cloud encircling all objects manipulated in visualizations of other users.

Updating the views of the other users depends on the performance of the system since this update is related to updating the VisualAttributes table. Here are two possible scenarios:

- Updating the VisualAttributes table after that the user releases the objects;
- Updating the VisualAttributes table after each update on the in-memory table.

6.7 Use case: publication database cleaning scenario

Application context. In an international context of the dissemination and exploitation of research results, were born of new models of direct scientific communication between researchers such as open archives (OA).

Their goal is to provide free access to scientific publications for researchers around the world and to build interoperable reservoirs of knowledge and information. The valuation of publications and preprints is provided by a better visibility, durability and ease of access.

HAL-INRIA, INRIA Open Archive was launched April 27, 2005. It relies on software HAL (Hyper Article on Line) developed by the Center for Direct Scientific Communication (CCSD) of CNRS. It offers scientists an environment of deposition and consulting in the areas of STIC (Sciences and Technologies of Information and Communication). HAL-INRIA is one component of the multidisciplinary international archive HAL, set up by the CCSD (Centre for Direct Scientific Communication) of the CNRS. The multi-disciplinary open archive HAL, is for the filing and dissemination of scientific research level, whether published or not, and theses, from educational institutions and research French or foreign, public or private laboratories.

Publications submitted via HAL-INRIA Open Archive feed National HAL, which allows to deposit and to publish scientific papers in all disciplines.

The HAL-INRIA archive was created to:

- Maximize the visibility of the scientific and institutional laboratories;
- Ensure the sustainability of data stored in the archive;

For each deposit, a basic check of the contents of the metadata is performed. For a deposit with text, the audit also seeks to control if the files are readable and do not violate the rules set by publishers in open archives.

The current database of HAL-INRIA contains a significant amount of duplication. These duplicates can be of different types:

- Authors: one of the recurring causes the duplicates is given in writing name and surname of the author by using abbreviations. For example: F. Aster and Fred Aster.
- Publications: the same publication can be entered twice by two co-authors of the article.
- Institutions: the authors of the same team and the same institutions can enter informations differently. Here is an example of entries that can be found in the institutions' table institutions in the HAL database: INRIA, INRIA Saclay, Aviz/INRIA, Leo/INRIA, etc.

Duplicates are currently treated with a manually by browsing through the list of authors and publications. This task is cumbersome given the large number of authors and publications stored in the database.

It is in this context that the SAS team of INRIA asks us to develop an application that automates this task by helping as much as possible the end user to detect duplicates.

EdiDuplicate is an application designed to detect and help remove duplicated author entries in a large database of publications called HAL-INRIA [43]. Except for trivial cases, deciding it two similar author names refer to the same person requires a human decision.

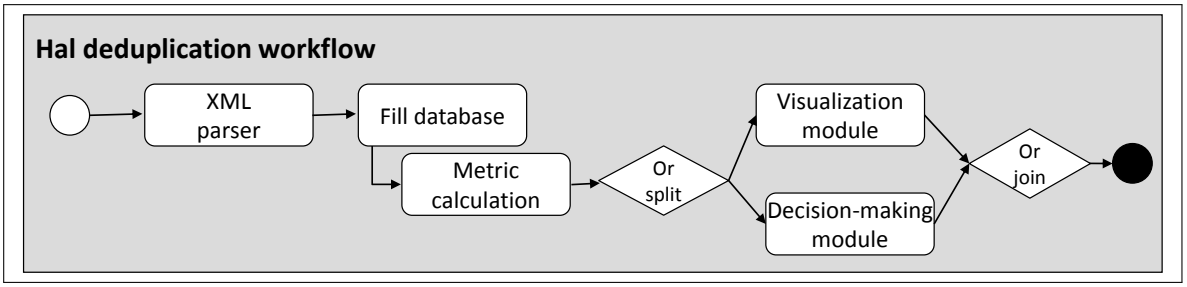

Figure 6.11: EdiDuplicate workflow

New data is collected from the HAL-INRIA database server every night. Figure 6.11 shows the workflow corresponding to this application. It consists of the following tasks:

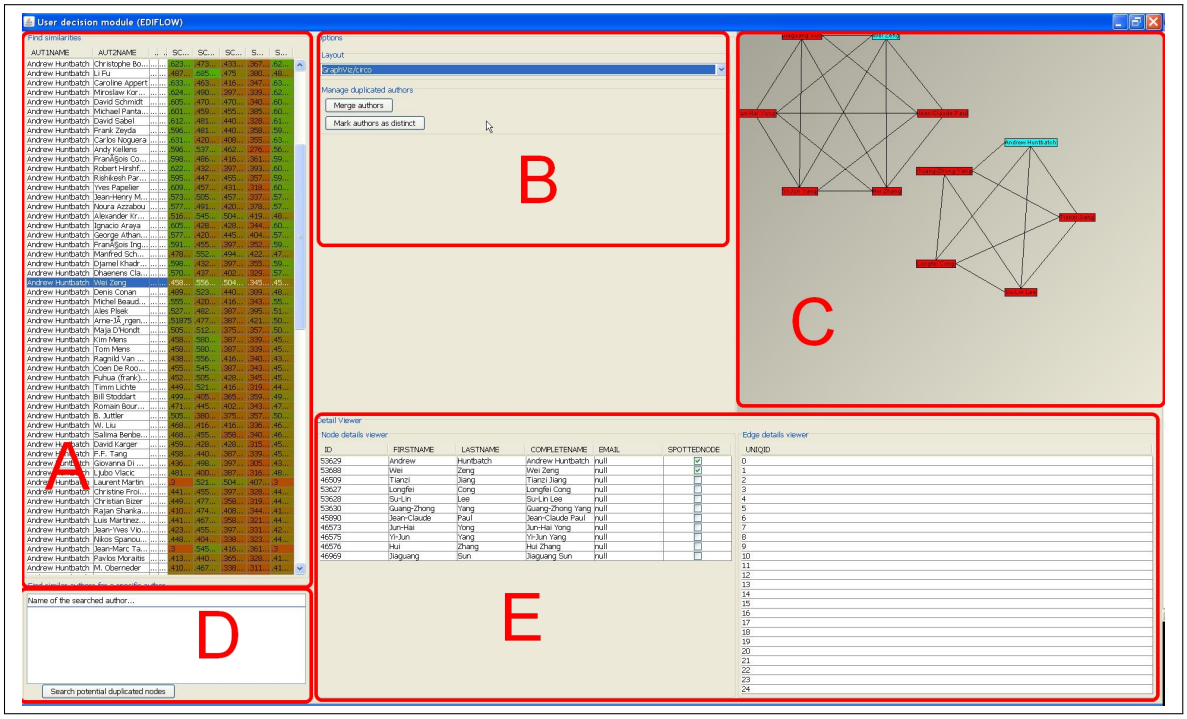

Figure 6.12: EdiDuplicate interface.

- Input: Data is imported in XML format, parsed and stored in the EdiFlow DBMS using multiple tables such as author and publication.

- Similarity computation: for each new author name inserted in the author table, we compute its similarities with all the other authors already in the table. We use several similarity metrics to capture many possible misspellings. The user can define a threshold below which two authors are considered different. All authors who have a higher rating are kept to be displayed to a user for decision.

- Visualization module: Figure 6.12 shows the interface from which users can decide whether two author names refer to the same person. This interface consists of five elements:

  1. the A pane contains a table with one row per candidate name. The columns contain the names to compare and the multiple similarities values between the names. The table is sorted by one of the similarity scores. The similarity columns are colored from green to red depending on the similarity score.

  2. the B pane allows to change visual attributes of the graphs that appear on the C panel. In this version of the application, we allow the change of the visualization layout. However, it is possible to integrate other features. This panel also contains two important buttons. A button that allows to assign the

same alias to the authors for whom the end user decides that they are identical. The second button allows to assign different aliases.

3. the C pane is used when the user selects a row; a co-publication network is then visualized in a way similar to [14]. This network helps the user decides it two names refer to the same author: if the neighborhood of authors is similar, sharing one or two co-authors, then the chance is high that the names refer to the same person. If the neighborhoods intersection is empty, then there is less support to merge the names unless the user can recognize a standard misspelling.
4. the D pane is used to refine the search at the table in panel A. This panel is especially useful when the number of entries in the table is very important.
5. the E panel is used to get more information on authors, such as email addresses, the institutions to which they belong, etc. This information is very useful to the end user of the application to decide whether two authors are identical or not.

- Decision-making module: this module allows the user to decide whether two authors are identical or not. To avoid forgetting decisions and to allow correcting errors later, the EdiDuplicate never removes authors from the table; it only updates an "alias" field in this table that points to the canonical author entry. If two authors have the same alias, they are considered to refer to the same person.

EdiDuplicate is meant to be connected to the HAL-INRIA database eventually. The input module would not be required any more but several other modules would be added to compute publication statistics from the data. De-duplicating authors would require a re-computation of the statistic, a typical visual analytics scenario when an interaction triggers more computations to update complex measures.

Furthermore, we have designed the visualization and decision-making modules to be able to work in isolation from the workflow, just in connection with the database. Using this module in standalone mode, many users can connect to HAL-INRIA and clean the data in parallel. The decision made by one user is immediately visible to the other users thanks to the cached tables. To better support collaboration, we have added feedback of other users watching specific authors so that all the users are aware that they share some common items in their visualizations. The implementation consists in managing a shared table containing the list of visual items displayed by users. The visualizations can add some feedback (e.g., a halo around the items). This feedback helps when users collaborate on cleaning the authors of a specific research group for example. It also prevents users from seeing their visualizations change unexpectedly if another user is cleaning it concurrently; at least, the user is warned.

EdiDuplicate features data-, computation- and user-centric processes that must react to changes while the workflow is running. Visual data exploration helps users make deci-

sions that will be fed back into the workflow. The EdiDuplicate application is the most challenging, by the size of the database, the complexity of its metrics, and consideration of the user interaction in the application.

6.8 Conclusion

In this chapter, we have described the EdiFlow reactive workflow system designed to address the problem of scalability in visual analytics. EdiFlow implements a novel mechanism for managing changes in data sources. Based on this mechanism, EdiFlow provides a unified model that is both control-driven and data-driven. The control is defined by the workflow specification file and triggers a standard run of the specification, as other Scientific workflows systems do. Contrary to the other workflow systems, when data changes in the DBMS, the dependencies are run again in a special mode to re-process the data: EdiFlow is *reactive*. Interactions are modeled as data changes. EdiFlow relies on a standard DBMS to realize its semantic model in a sound and predictive way. EdiFlow supports standard data manipulations through procedures and introduces the management of changes in the data through *update propagation*. Each workflow process can express its behavior w.r.t data change in one of its input relations. Several options are offered to react to such a change in a flexible way to optimize the reaction according to the semantic of the process.

We also presented in this chapter a real-life use case. This use case has several challenges such as complex analysis tasks and user interaction.

We strongly believe that formally specifying the services required for visual analytics in term of user requirements, data management and processing, and providing a robust implementation is the right path to develop the fields of visual analytics and scientific workflows together.

Part IV

Conclusion

Chapter 7

Conclusion and perspectives

We have studied in this dissertation the design and implementation of a platform aimed at capturing changes in data sources and launching a repair mechanism.

7.1 Summary

Visual analytics aims at combining interactive data visualization with data analysis tasks. Given the explosion in volume and complexity of scientific data, e.g., associated to biological or physical processes or social networks, visual analytics is called to play an important role in scientific data management.

Most visual analytics platforms, however, are memory-based, and are therefore limited in the volume of data handled. Moreover, the integration of each new algorithm (e.g., for clustering) requires integrating it by hand into the platform. Finally, they lack the capability to define and deploy well-structured processes where users with different roles interact in a coordinated way sharing the same data and possibly the same visualizations.

We have designed and implemented *EdiFlow*, a workflow platform for visual analytics applications. EdiFlow uses a simple structured process model, and is backed by a persistent database, storing both process information and process instance data. EdiFlow processes provide the usual process features (roles, structured control) and may integrate visual analytics tasks as activities. We presented its architecture, deployment on a sample application, and main technical challenges involved.

We present in the following a list of contributions related to our work.

A generic data model. We investigated the choice of a generic data model, yet amenable to efficient computations. The designed data model comprises three kinds of entities. Application-dependent entities model, Workflow-related entities and visualization-related entities.

A process model. An appropriate process model dialect has been identified, modeling user dataset interactions in scientific applications. The model allows for the usual process composition primitives (in the style of the Workflow Coalition Model) but extended to allow the user to inspect, interrupt, resume, and guide the process evolution for all or part of the active instances, with little effort, at any point. This interactivity is crucial for the success of real-life data processing applications. Therefore, the design of the process model strives to capture, to the extent possible, the various interactions that rich data manipulations interfaces allow.

EdiFlow processes are specified in a simple XML syntax, closely resembling the XML WfMC syntax XPDL [86]. The main innovative ingredient of the process model is the treatment of *data dynamics*, i.e. the possibility to control which changes in the data are propagated to which part(s) of which process instances.

Design and implementation of the architecture. We have designed an architecture and implemented its parts to validate it by running some of the scenarios. The designed architecture is composed of three layers: the DBMS, the EdiFlow process and the modules. The workflow management logic runs on top of the DBMS; visualization software is integrated under the form of modules. During process executions, the necessary data manipulation statements are issued to record the connections between users and application instances, and application data.

Validation of the architecture on real applications. We validated our architecture over three real applications. These scenarios start from data, transform them into clean tables, apply some workflow operators to them for analysis or transformation and allow navigation and visualization at interactive rate. We started from the cleanest data sets and problems and advanced towards more complex and demanding datasets and scenarios.

The implementation of these applications could not be achieved without the use of persistent database. Indeed, effective management of large data volumes such as HAL-INRIA or Wikipedia could not be done simply by using memory-based mechanism.

Identification of gaps in persistent databases In this dissertation, we have found several shortcomings related to the databases management systems for visualization systems (response time, notification management, etc.). Identifying these gaps can help for designing a new generation of DBMS with appropriate properties and features for visual analytics applications.

7.2 Research directions

Many roads go in a star of our work and each arouses our interest, some of these paths will be presented in the following.

7.2.1 Improve the provenance management process

Since EdiFlow keeps track of the dependencies between tables and modules, it maintains a rough level of provenance: each output table is the result of processes taking a set of input tables. However, at the EdiFlow level, nothing more precise can be known. Since our modules need to implement an abstract data type to be run by EdiFlow, we are considering improving this interface to support exporting mode accurate Provenance data. However, full provenance information can become very large and it is important to study what level of provenance is really required for real visual analytics applications.

One solution consists on using the Oracle Total Recall system. It is a multi-versioning mechanism that ensures read consistency while maintaining a high degree of concurrency. When DML (Data manipulation Language) operations such as INSERT, UPDATE, or DELETE are executed, Oracle writes data to an undo tablespace that is used for transaction rollbacks and for guaranteeing read consistency.

Total Recall contains a Flashback Data Archive component which creates an internal history table for every tracked table. The internal history table is initially a replica of the source table with additional metadata columns. When one or more columns in the tracked table are updated, a new row is inserted into the history table that is the before-image of the row before the transaction.

7.2.2 Improve the visual table schema

Currently, the visual table schema is minimal and we have to extend it to manage real visualizations. It is possible to extend the functionalities for the "Shape" field. Indeed, this field currently contains a string describing the shape of the object. An extension is to use the libraries used in geographic information systems. These libraries have the advantage of including different methods to perform complex geometrical operations between shapes such as the intersection between two objects. An example of these libraries is JTS (Java Topology Suite).

The JTS Topology Suite is a Java API that implements a core set of spatial data operations using an explicit precision model and robust geometric algorithms. JTS is intended to be used in the development of applications that support the validation, cleaning, integration and querying of spatial datasets. This document is the design specification for the classes, methods and algorithms implemented in the JTS Topology Suite.

7.2.3 Specify collaboration management mechanisms

A fundamental aspect of successful collaboration is an effective division of labor among participants. This involves both the segmentation of effort into proper units of work and the allocation of individuals to tasks in a manner that best matches their skills and disposition. Primary concerns are how to split work among multiple participants and meaningfully aggregate the results [47].

EdiFlow is distributed by design. We started to play with some collaboration mechanisms but we need to experiment more to provide guidance to programmers and a standard behavior for modules.

7.2.4 Integration with VisTrails

Exploration of workflows configurations, provenance and history management are three very important features we want to integrate with EdiFlow and that are very well supported by VisTrails. Conversely, VisTrails does not manage dynamic data and does not try to unify communication mechanisms between modules as EdiFlow does. We are interested in incorporating the mechanisms of EdiFlow into VisTrails. It would also allow us to benefit from the large set of modules VisTrails already has.

VisTrails is written in Python and uses Qt as its GUI toolkit (through PyQt Python bindings). EdiFlow is written in JAVA. It is therefore necessary to establish a Jython wrapper in order to call different visualizations and features used in the source code of VisTrails.

7.2.5 Management of dynamic workflows

Dynamic workflows need to evolve over time as execution data become available. To manage dynamic workflows, EdiFlow must capture provenance information and store the different versions of the workflows. By tracking detailed provenance information, we can ensure reproducibility and allow scientists to easily navigate through the different versions of the workflow and parameter settings used in a given exploration task. Unlike VisTrails, the current version of EdiFlow does not allow the management of dynamic workflows. Thus, to facilitate integration with VisTrails, it is important to develop this feature in EdiFlow.

Appendix A

Données en masse et workflows interactifs pour la visualisation analytique (Résumé étendu)

A.1 Contexte

La quantité des données électroniques de toute forme, produites par des utilisateurs (pages Web, blogs, des contenus structurés tels que Wikipedia) et des outils automatiques (capteurs, services Web, programmes scientifiques ou des outils d'analyse) conduisent à une prolifération sans précédent des données, ce qui complique l'extraction de nouvelles connaissances, des nouvelles corrélations, et l'interprétation des données.

La visualisation analytique est une nouvelle branche de la visualisation de l'information et de l'interaction homme-machine [77]. Son objectif est de permettre aux utilisateurs d'interagir étroitement avec de grandes quantités de données à l'aide des outils de visualisation. Grâce à ces outils, un utilisateur peut détecter des phénomènes ou de déclencher une analyse détaillée qui peut ne pas avoir été identifiée par les outils automatiques.

La plupart des outils de visualisation analytique ont quelques inconvénients conceptuels. En effet, ils sont rarement liés à des bases de données persistantes (à l'exception de [36]). Dans ces systèmes, les données sont chargées à partir de fichiers ou bases de données et sont manipulées directement dans la mémoire centrale car la fluidité de l'interaction visuelle nécessite une fréquence de réaffichage de 10 à 25 fois par seconde. Les bases de données standards ne supportent pas les requêtes continues à ce rythme. Cependant, malgré ces inconvénients, la connexion entre une base de données et un système de visualisation analytique présente plusieurs avantages :

- Passage à l'échelle : de gros volumes de données peuvent être traitées grâce aux bases de données persistantes.
- Persistance et distribution: plusieurs utilisateurs (éventuellement à partir de sites distants) peuvent interagir avec une base de données, alors que ce n'est pas facile à réaliser avec des structures de données résidants en mémoire centrale.
- Gestion des données : des tâches complexes de traitement de données peuvent être exprimées en SQL et/ou un langage de script impératif. Ces tâches peuvent également inclure des fonctions définis par l'utilisateur (UDF) pour les calculs mis en oeuvre en dehors du serveur de bases de données. Ces fonctions ne sont pas des procédures stockées gérées par le SGBD. Ce sont des programmes exécutables externes au serveur de base de données.

L'intégration d'un Système de Gestion de Base de Données (SGBD) dans une plate-forme de visualisation doit prendre en considération les aspects suivants répandus dans les applications de visualisation analytique :

- Convergence des visualisations analytiques et des workflows : les outils actuels de visualisation analytique ne sont pas basés sur les modèles de processus workflow. Cependant, plusieurs applications de visualisation analytique nécessitent toutefois un processus récurrent, bien soutenu par un système de workflow. Les tâches de traitement de données doivent être organisées en séquence ou en boucle. Les utilisateurs ayant des rôles différents peuvent collaborer dans certaines applications avant de continuer l'analyse. Il peut aussi être nécessaire de se connecter et de permettre l'inspection de l'exécution de chaque tâche. Les plates-formes de workflow (scientifiques) permettent l'automatisation de ces traitements par la combinaison des outils de bases de données avec l'invocation de fonctions externes.
- Gestion des données dynamiques et propagation du changement : une classe importante d'applications de visualisation analytique doit faire face aux données dynamiques, qui sont actualisées en permanence tandis que le processus d'analyse est encours d'exécution. Les interactions possibles entre toutes ces mises à jour doivent être soigneusement étudiées, afin de supporter des applications efficaces et flexibles.

Notre travail porte sur l'intégration d'un SGBD dans une plate-forme de visualisation analytique.

A.2 Etat de l'art

Les applications de visualisation analytique sont actuellement conÃ§ues à l'aide d'une architecture ad-hoc. In n'existe pas de modèle clair qui guiderait les designers à con-

cevoir ce type d'applications. Comme la plupart des systèmes de visualisation analytique sont construites par des praticiens de la visualisation de l'information, ils se basent, généralement, sur le modèle de référence de la visualisation de l'information [23; 27] pour la partie visualisation ainsi que pour la partie interaction. La partie analyse des données est intégrée ou gérée séparément selon le niveau de calcul.

Nous proposons dans ce qui suit un liste non exhaustives des principaux systèmes de visualisation analytiques basés sur le modèle référence de visualisation ainsi que les principaux systèmes de workflow scientifique. Nous évoquons leurs forces et faiblesses pour les outils de visualisation analytique.

A.2.1 Systèmes de visualisation analytique

Toutes les outils de visualisation de l'information tels que Prefuse [48], InfoVis (IVTK) [33] ou improvise [82], sont basés sur un système de mémoire centrale pour gérer leurs structures de données et offrir la vitesse requise pour la visualisation et l'interaction. Il est inenvisageable de garder toutes les données en mémoire centrale dans les applications de visualisation analytique. Toutefois, le réaffichage et les interactions continues nécessitent de garder certaines données en mémoire. Néanmoins, les applications de visualisation analytique ne visualisent pas tous les attributs de toutes les structures de données, donc une partie pourrait être gardée en dehors de la mémoire centrale. C'est ce que le gestionnaire de mémoire virtuelle doit effectuer lorsque la quantité de données est plus importante que la quantité de RAM sur une machine. Toutefois, pour qu'elles soient gérées par le gestionnaire de mémoire virtuelle, les données doivent être chargées en mémoire d'abord. Ce processus peut prendre du temps, en partie inutilement lorsque seules des portions de données sont nécessaires. En outre, l'organisation de la mémoire des données doit être compatible avec la stratégie du gestionnaire de mémoire virtuelle. Les bases de données orientées colonnes sont mieux adaptées aux stratégies actuelles [33].

A.2.2 Systèmes de workflow scientifique

A scientific workflow is a formal specification of a scientific process, which represents, streamlines, and automates the analytical and computational steps that a scientist needs to go through from dataset selection and integration, computation and analysis, to final data product presentation and visualization [25].

Nous présentons dans ce qui suit une liste des principaux systèmes de workflow scientifique :

Parmi les projets les plus récents et bien développé de workflow scientifique, Kepler [7] est conÃ§u pour aider les scientifiques, les analystes et les programmeurs informatiques à créer et exécuter les workflows à travers un large éventail de disciplines scientifiques. Kepler fournit une interface graphique qui permet aux utilisateurs de sélectionner, puis connecter les composants d'analyse aux sources de données pour créer un workflow scientifique.

Triana [76] est un système de workflow construit à l'origine pour fournir un outil d'analyse rapide de de données d'ondes gravitationnelles. À ses débuts, les procédures ont été modélisées et exécutées localement ou à distance. Plus récemment, Triana a été étendu à incorporer des composants distribués, orientés grille ou service Web.

Ces systèmes de workflow scientifique peuvent traiter de grandes quantités de données mais ils fournissent très peu visualisation et pas d'interaction: ils sont destinés à fonctionner dans des processus bien spécifiques, et non pas à des tâches exploratoires. Certains de ces systèmes offrent des mécanismes pour lancer des visualisations implémentées comme une application externe. Par ailleurs, même si la plupart de ces workflows scientifiques permettent des connexions aux bases de données pour charger et stocker des données, ils considèrent que ces données sont immuables pendant l'exécution du processus workflow. Ils ne peuvent pas réagir à tout changement sur les sources de données une fois le workflow démarré.

Tous ces systèmes de workflow scientifique ne sont donc pas adaptés pour la visualisation analytique : ils fournissent peu de visualisation, aucune interaction ni aucune gestion des données dynamiques.

A.2.3 Systèmes de workflow scientifique basés sur la visualisation

VisTrails [22; 73] est un système de workflow scientifique basé sur un système de visualisation qui permet de gérer efficacement les activités exploratoires. Il offre une interface graphique pour créer le processus workflow, exécuter et visualiser les résultats. La visualisation est implémentée en utilisant la librairie VTK [52; 81]. Ainsi, les scientifiques n'ont pas besoin d'utiliser des programmes externes pour visualiser les résultats de leurs expériences.

VisTrails offre plusieurs caractéristiques, telles que l'analyse exploratoire et la gestion efficace de la provenance. Toutefois, il n'offre aucun mécanisme permettant de réagir aux changements dans les sources de données.

A.3 Contributions

Nous avons conçu une architecture générique pour intégrer les outils de visualisation analytique avec les SGBDs. Il s'agit d'EdiFlow, une plateforme de workflow pour les applications de visualisation analytique. EdiFlow utilise un modèle de processus structuré et est soutenu par une base de données persistante permettant de stocker les informations relatives aux données et au contrôle des instances de processus. Nous présentons dans ce qui suit, les différents ingrédients d'EdiFlow, à savoir, son modèle de processus, son architecture et sa gestion de la dynamicité, de la visualisation et de l'interaction.

Process	:==	Configuration Constant* Variable+ Relation+ Function* StructuredProcess
Configuration	::=	DBdriver DBuri DBuser
Constant	::=	name value name $\in N$, value $\in V$
Variable	::=	name type name $\in N$, type $\in T$
Relation	::=	name primaryKey RelType
RelationType	::=	(attName attType)*, attName $\in N$, attType $\in T$
Function	:=	name classPath
StructuredProcess	:=	Activity | Sequence | AndSplitJoin | OrSplitJoin | ConditionalProcess
Sequence	::=	Activity **,** StructuredProcess
AndSplitJoin	::=	**AND-split** (StructuredProcess)+ **AND-join**
OrSplitJoin	::=	**OR-split** (StructuredProcess)+ **OR-join**
ConditionalProcess	::=	**IF** Condition StructuredProcess
Activity	::=	activityName Expression
Expression	::=	askUser | callFunction | runQuery

Figure A.1: Schéma XML du modèle de processus.

A.3.1 Modèle de processus

EdiFlow implémente un modèle de processus inspiré du Workflow Management Coalition model [84]. La Figure 5.2 décrit la syntaxe des processus. Nous utilisons un ensemble de variables, de constantes, de noms d'attribut N, de valeurs atomiques V, et un ensemble de types de données atomiques T. Le principal ingrédient innovant ici est le traitement des *données dynamiques* : la possibilité de contrôler les changements. Nous décrivons dans ce qui suit les principaux ingrédients de ce modèle.

- Un processus est construit sur un ensemble de *relations* (de manière informelle appelées tables) implémentant le modèle de données. Les relations sont désignées par des lettres capitales tels que R, S, T.

- Une *requête* est une expression algébrique sur les relations. Les requêtes sont généralement désignées par la lettre Q.

- Une *variable* est un couple composé d'un nom et d'une valeur. Les variables sont utilisées pour la modélisation des constantes, comme par exemple, un seuil numérique pour un algorithme de clustering. Les variables seront dénotées par des lettres minuscules tels que v, x, y.

- Une *procédure* est une unité de calcul implémentée par des logiciels externes. Un exemple typique est le code de calcul d'un clustering. Une procédure prend en entrée l relations $R_1, R_2, \ldots, R_l$ et envoie les données en sortie vers n relations :

Nous considérons une procédure p comme une boîte noire, correspondant à un programme développé à l'extérieur du moteur de bases de données et à l'extérieur du système EdiFlow, au moyen de certains programmes exprimés, par exemple, en C++, Java, Matlab. Les fonctions sont considérées comme des processus sans effets secondaires.

Une procédure peut être associée aux *gestionnaires de mise à jour*. Un gestionnaire de mise à jour est invoqué avec une liste de tuples qui ont été modifiés via la procédure précédente. Cette liste de tuples est appelée *delta*. EdiFlow implémente deux types de mises à jour: immédiate et différée. Les mises à jour immédiates sont propagées immédiatement même si la procédure est en cours d'exécution. Par exemple, quand un graph est en cours de construction et que de nouveaux noeuds arrivent, la disposition devrait tenir compte de la mise à jour dans le calcul. Les mises à jour différées sont propagées après que la procédure ait terminée son exécution. Par exemple, lorsque la procédure effectue une analyse quantitative dont seul le résultat final compte, elle peut être ajusté par la suite pour prendre en compte les deltas. Les gestionnaires de changements sont des mécanismes uniques à EdiFlow, ils sont censés gérer efficacement les données dynamiques.

- Les *activités* sont les blocs de nos processus. Les activités les plus simples sont les *affectations* de variables de la forme $v \leftarrow \alpha$. Un autre type d'activité serait une mise à jour d'une table R, notée $upd(R)$. Contrairement aux modifications de table faites à partir d'une procédure opaque, ces mises à jour sont spécifiées par un instruction SQL déclarative. Enfin, une activité peut invoquer une procédure de p en fournissant les paramètres d'entrée appropriés, et de retenir les sorties dans un ensemble de tables.

  Les activités de *visualisation* doivent être modélisées comme des procédures, étant donné que leur code ne peut être exprimé par des requêtes.

- Un *processus* est soit "vide", soit une *séquence* d'activités suivie d'un processus, soit un *parallel (and) split-join* de deux processus, soit un *(or) split-join* de deux processus (avec la sémantique qu'une fois une branche est déclenchée, l'autre est invalidé et ne peut plus être déclenchée). Enfin, un processus peut être considéré comme un *bloc conditionnel* où une expression est évaluée. Si le résultat est "vrai" le processus est exécuté.

- Un *processus réactif* est défini par un ensemble de relations, un ensemble de variables, un ensemble de procédures, un processus et un ensemble de *propagations de mise à jour*. Une *propagation de mise à jour* spécifie ce que doit être fait quand un ensemble de tuples sont ajoutés à une relation. Plusieurs options sont possibles pour gérer, propager, ou ignorer les modifications.

A.3.2 Architecture du système d'Ediflow

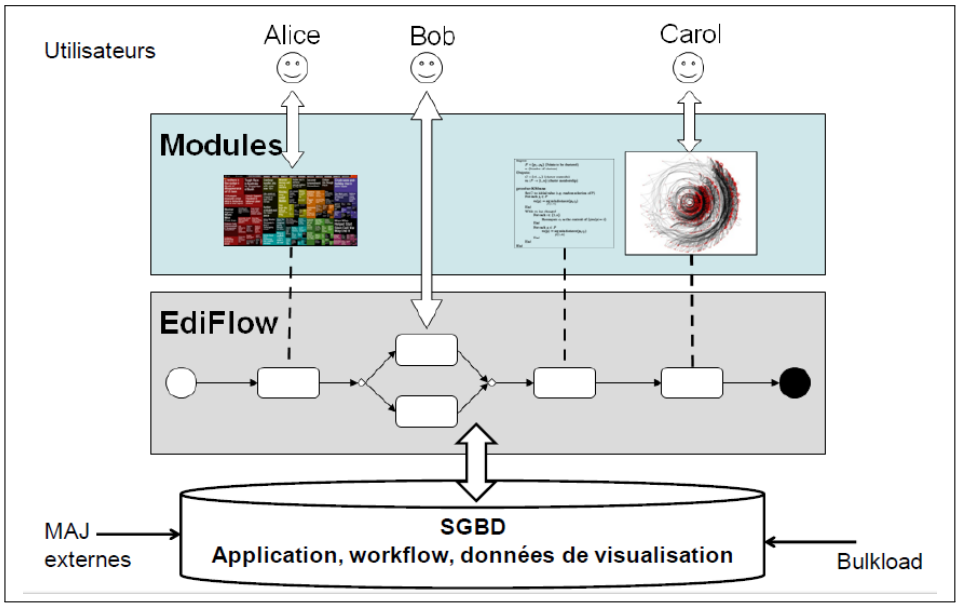

Figure A.2: Architecture du système EdiFlow.

L'architecture du système EdiFlow est décrite dans la Figure A.2; elle est divisée en trois couches :

- La couche SGBD gère les données utilisées à la fois pour le workflow et pour les modules. Pour le workflow, le SGBD conserve toutes les informations relatives à l'état des activités, des processus, etc. Concernons les modules, le SGBD mémorise toutes les informations internes sur l'état du modules, l'ordre de leur exécution, les dépendances entre les tables et les modules, etc. EdiFlow peut être considéré comme un système d'exploitation distribué dont le SGBD contient sa mémoire.

- La couche EdiFlow est en charge de l'exécution du processus dans un ordre déterminé par la spécification XML du workflow et également de la réaction aux changements dans les tables.

- La couche des modules est constituée de l'ensemble des modules (boîtes noires). Ils peuvent invoquer des visualisations et des interactions.

A.3.3 Gestion de la dynamicité

La capture des changements dynamiques dans les tables des SGBDs est un mécanisme essentiel d'EdiFlow pour deux raisons : en interne, pour traiter les détails du gestionnaire de changement et, en externe, à mettre en oeuvre les caches de données dans les modules. Nous décrivons ici l'implémentation bas niveau du mécanisme.

La principale caractéristique d'EdiFlow est de réagir à la dynamique des changements de données. La gestion de la réactivité complique la logique du système, mais aussi sa mise en oeuvre. Pour mettre en place *les gestionnaires de changement*, EdiFlow a besoin de garder une trace des changements intervenus dans le SGBD depuis la dernière invocation du module.

Le mécanisme utilisé pour recueillir les mises à jour pour EdiFlow a été étendu pour gérer les tables en cache qui sont conservés conformément aux tables dans le SGBD. Quand une table du SGBD est mise à jour dans un module EdiFlow, la table en mémoire est maintenue cohérente avec le table sur disque : l'évolution de la table en mémoire se propage à celle sur disque et vice-versa.

EdiFlow implémente un protocole visant à propager efficacement les mises à jour effectuées sur une table du disque, appelée R_D à son image de la mémoire, appelée R_M. Inversement, lorsque le module modifie R_M, ces changements doivent être propagées vers R_D. R_M existe dans la partie client et peut donc être sur un autre hôte que R_D.

EdiFlow met en place un système pour créer, met à jour et supprime les triggers qui surveillent les changements dans la table persistante R_D. Chaque fois qu'un changement arrive, le trigger correspondant ajoute à la table de notification un tuple de la forme (seq_no, ts, tn, op), où seq_no est un numéro séquentiel, ts est l'instant de la mise à jour, tn est le nom de la table et op est l'opération effectuée (insertion, modification ou suppression). Ensuite, une notification est envoyée à R_M. L'interaction fluide exige que la notification doit être transmise très rapidement. Une notification est envoyée via un socket connecté à l'instance de processus tenant R_M. Les informations relatives à l'adresse hôte et le port sont mémorisées dans la table Client. Lorsque le module décide de traiter les mises à jour afin de synchroniser les tables en mémoire centrale, il lit les notifications de la table à partir de son dernier numéro de séquence seq_no.

Le protocole de synchronisation entre R_M et R_D peut être résumé comme suit :

1. Un *objet de mémoire* (R_M) est créé dans le processus Java.
2. Il demande au *gestionnaire de connexion* de créer une connexion avec la base de données.
3. Le gestionnaire de connexion ouvre un port sur la machine locale et associe localement un quadruplet à R_M : $(db, R_D, ip, port)$.
4. Le quadruplet est envoyée au SGBD pour créer une entrée dans la table ConnectedUser.
5. Le SGBD se connecte au client en utilisant l'adresse $ip : port$, et s'attend à recevoir un message HELLO pour vérifier qu'il s'agit du bon protocole.
6. Le gestionnaire de connexion accepte la connexion, envoie le HELLO et s'attend à un message REPLY pour vérifier qu'il s'agit du protocole attendu.

7. Quand R_D est modifiée, le trigger du SGBD envoie au client ($ip : port$) qui détient R_M, un message de NOTIFICATION avec le nom de la table comme paramètre.

8. Le logiciel de table en mémoire peut décider quelle est le moment approprié pour rafraîchir son contenu et peut-être invoquer une notification qui mettra à jour une visualisation. Quand il décide de le faire, il se connecte au SGBD et extrait la liste des tuples créés/modifiés/supprimés, et propage les modifications apportées à R_M.

9. Lorsque R_M est modifiée, elle propage ses changements à la table R_D et traite les notifications déclenchées de manière intelligente afin d'éviter le travail redondant.

10. Lorsque R_M est supprimée, un message de déconnexion est envoyé à la base de données qui ferme la socket et supprime le tuple correspondant dans la table ConnectedUser.

11. La table des notifications peut être purgée de tuples ayant un *seq_no* inférieure à la valeur la plus basse dans la table Client.

Notons que certains SGBD offrent déjà une extension mémoire (par exemple Oracle In-Memory Database Cache). Nous n'avons pas connecté EdiFlow à ces systèmes pour deux raisons: 1) ils sont habituellement disponibles pour des SGBD spécifiques et nous voulons qu'EdiFlow soit agnostique par rapport aux SGBDs, 2) nous encapsulons nos objets en mémoire avec une fine surcouche qui peut s'adapter aux nouvelles implémentations donc rien n'empêche qu'EdiFlow puisse utiliser une extension mémoire.

A.3.4 Gestion de la visualisation

EdiFlow implémente les visualisations en utilisant une table contenant un ensemble d'informations graphiques, telles que la position (x, y), la forme, la couleur, l'étiquette, etc. Chaque module peut mettre en place une table pour répondre à ses besoins mais il est plus efficace de partager une table commune.

Par conséquent, un module de visualisation lit le contenu des tables, calcule les attributs graphiques associés à ces tables d'entrée et remplit la table de visualisation dans EdiFlow. Le module permet également d'afficher la visualisation sur un écran ou de déléguer l'affichage à un autre module ou à une application externe qui va lire les éléments spécifiques de la base de données. En utiliser le mécanisme de cache dans EdiFlow, les modules de visualisation sont invoqués lorsque les tables de données d'entrée ont été modifiées. Ils peuvent ensuite mettre à jour la table des attributs visuels soit en recalculant tous les attributs ou seulement ceux qui ont été changés.

Conceptuellement et dans notre implémentation, le module de visualisation d'EdiFlow est très similaire à celui de Prefuse Le schéma concret de la table des attributs visuels est le suivant :

Nom	VsID	X	Y	ObjectID	Forme	Etiquette	SrcTable
Type	String	Real	Real	String	String	String	String

- *VisID* est une chaîne de caractère qui identifie la visualisation qui a été à l'origine de cet élément (toutes les visualisations stockent leurs résultats dans la table des attributs visuels).
- *X et Y* sont la position de l'élément 2D.
- *ObjecID* est un identificateur unique de l'objet visualisé dans la table source et est utilisé pour la sélection et la mise à jour
- *Forme* est la spécification d'une géométrie; actuellement, on maintient un objet java sérialisé.
- *Etiquette* permet d'attribuer un alias à un objet.
- *SrcTable* est, comme son nom l'indique, le nom de la table à partir de laquelle cet élément graphique a été calculé.

Ce schéma est très simple et doit être enrichi quand EdiFlow évolue.

EdiFlow permet de gérer plusieurs vues pour une même visualisation. Les attributs visuels peuvent être partagés par plusieurs vues et par plusieurs utilisateurs qui peuvent choisir de visualiser tout ou une partie des données. Par exemple sur un téléphone, nous montrons que 10% des éléments, sur un ordinateur portable, nous montrons 30% et sur un grand écran nous montrons 100% des éléments.

A.3.5 Gestion de l'interaction

Dans un système interactif, l'interaction est gérée dans une ou deux phases. Les commandes sont des opérations effectuées par l'utilisateur en utilisant un appareil d'interaction tel qu'un clavier ou une souris (par exemple en cliquant sur un bouton). Lorsque la commande est exécutée, l'état du système est modifié. Les commandes sont des actions discrètes et perÃ§ues comme discrètes par l'utilisateur. Il y a aussi des actions continues qui sont importantes pour les requêtes dynamiques et les interactions fluides.

Dans EdiFlow, les modules n'ont pas le choix d'une interaction continue. Ils doivent reporter la commande à la fin du morceau d'interaction et effectuer une rétroaction continue locale. Sélectionner 100 noeuds et les déplacer d'une manière interactive en changeant directement la table des attributs visuels génère un nombre de mises à jour à la base de données, incompatible avec la vitesse de l'interaction. Nous croyons que ceci est un petit prix à payer pour bénéficier de l'ensemble des fonctionnalités de l'environnement EdiFlow.

Il existe plusieurs systèmes basés sur une mémoire centrale (Protovis [16], Jung [64], IVTK [33], etc.). Nous avons utilisé Prefuse et IVTK. Inclure les bases de données basées

sur la mémoire centrale dans l'architecture d'EdiFlow, permet de bénéficier des fonctionnalités des bases de données persistantes, mais aussi obtenir des temps de réponses temps de réponse intéressants grâce aux bases de données mémoires.

Ainsi, lors de chaque opération d'actualisation, la base de données en mémoire est mise à jour afin d'atteindre un temps de réponse d'une seconde. Une phase de synchronisation est toujours prévue avec la base de données persistante pour que les autres utilisateurs puissent être informés lorsque certains objets sont mis à jour.

A.4 Conclusion et perspectives

Nous avons étudié dans cette thèse la conception et la mise en place d'une plateforme visant à capturer les changements dans les sources de données ainsi qu'à lancer un mécanisme de calcul et de réparation.

La visualisation analytique vise à combiner la visualisation interactive des données avec des tâches d'analyse et de fouille. Compte tenu de l'explosion du volume et la complexité des données scientifiques (données associées à des processus biologiques ou physiques ou de réseaux sociaux), la visualisation analytique est appelée à jouer un rôle important dans la gestion des données scientifiques.

La plupart des plates-formes de visualisation analytique, cependant, sont basées sur des structures mémoire, et sont donc limités dans le volume des données traitées. Par ailleurs, l'intégration de chaque nouvel algorithme (par exemple pour le clustering) nécessite de l'intégrer à la main dans la plate-forme. Enfin, il leur manque la capacité à définir et déployer des processus bien structurés où les utilisateurs interagissent avec différents rôles d'une manière coordonnée en partageant les mêmes données et éventuellement les mêmes visualisations.

Nous avons conÃ§u et mis en place EdiFlow, une plateforme de workflow pour les applications de visualisation analytique. EdiFlow utilise un modèle structuré de processus, et est connecté à une base de données persistante. Un processus dans EdiFlow peut intégrer des tâches de visualisation analytique dans ses activités. Nous avons présenté l'architecture, le déploiement sur un ensemble d'application, ainsi que les principaux défis techniques rencontrés.

Nous présentons dans ce qui suit une liste de contributions liées à notre travail.

Un modèle générique de données. Nous avons étudié le choix d'un modèle de données générique. Le modèle conÃ§u comprend trois types d'entités. Les entités liées aux applications et aux tables de données, les entités liées aux contrôles du processus workflows et les entités liées aux visualisations.

Un modèle de processus. Nous avons mis en place un modèle de processus faisant intervenir les données des utilisateurs, ainsi que la modélisation des interactions dans les

applications scientifiques. Le modèle permet les opérations standards (dans le style de la Coalition workflow Model), mais aussi, étendu pour permettre à l'utilisateur d'inspecter, d'interrompre, reprendre et guider l'évolution du processus à tout moment. Cette interactivité est essentielle pour la gestion des applications faisant appel à une masse importante de données.

Conception et implémentation de l'architecture. Nous avons spécifié et mis en place un architecture permettant de faire face aux différents défis, à savoir, la gestion de la dynamicité, le passage à l'échelle et l'interaction. L'architecture conÃ§ue est composé de trois couches: le SGBD, le processus EdiFlow ainsi que les modules.

Validation de l'architecture sur des applications réelles. Nous avons validé notre architecture par trois applications réelles. Ces applications commencent par transformer et nettoyer les données, appliquer certains opérateurs workflow pour des fins d'analyse ou de transformation et permettre l'exploration visuelle du résultat.

Identification des lacunes des bases de données persistantes. Dans cette thèse, nous avons relevé plusieurs inconvénients liés à l'utilisation des bases de données persistantes pour la visualisation analytique (temps de réponse, gestion de la notification, etc.). Identifier ces lacunes peut aider à la conception d'une nouvelle génération de SGBD avec des propriétés et fonctionnalités adéquates aux applications de visualisation analytique.

A.4.1 Perspectives

Nous présentons dans ce qui suit l'ensemble des perspectives liées aux contributions décrites dans ce manuscrit.

Amélioration du processus de gestion de la provenance. EdiFlow garde une trace des dépendances entre les tables et les modules en maintenant un niveau approximatif de provenance : chaque table de sortie est le résultat de processus faisant appel à un ensemble de tables d'entrées. Nous envisageons d'améliorer cette interface pour améliorer le niveau de précision de la provenance au niveau d'EdiFlow. Toutefois, les informations de provenance peuvent devenir très volumineuses et il est ainsi important d'étudier quel est le niveau de la provenance nécessaire pour les applications de visualisation analytique.

Amélioration de la structure de la table des attributs visuels. Actuellement, le schéma de la table des attributs visuels visuelle est minime et nous devons l'étendre afin de gérer plusieurs types de visualisations. Il est possible, par exemple, d'étendre les caractéristiques du champ "Forme". En effet, ce champ contient actuellement une chaîne de caractère décrivant la forme de l'objet. Une extension est d'utiliser les bibliothèques des systèmes d'information géographique. Ces bibliothèques ont l'avantage d'inclure différentes

méthodes pour effectuer des opérations géométriques complexes entre les formes telles que l'intersection entre deux objets.

Mise en place d'un mécanisme de gestion de collaboration. Un aspect fondamental de la collaboration réussie est une véritable division du travail entre les participants. Cela implique à la fois de la segmentation des efforts dans des unités appropriées de travail et la répartition des individus à des tâches d'une manière qui correspond le mieux à leurs compétences et leur aliénation.

EdiFlow est distribué par son design. Nous avons commencé à implémenter certains mécanismes de collaboration, mais plus d'expérimentations sont nécessaires afin que ce mécanisme soit au point.

Intégration avec VisTrails. L'exploration des workflows, la provenance et la gestion de l'historique sont trois éléments essentiels que nous voulons intégrer dans EdiFlow et qui sont déjà pris en charge par VisTrails. Cependant, VisTrails ne gère pas la dynamicité des données et ne possède pas un mécanisme d'unification entre ses différents modules. Il est ainsi intéressant d'intégrer le mécanisme de gestion des données dynamiques dans VisTrails.

Gestion des workflows dynamiques. Les workflows dynamiques ont besoin d'évoluer au fil du temps que les données d'exécution sont disponibles. Pour gérer les workflows dynamiques, EdiFlow doit capturer les informations de provenance et mémoriser les différentes versions du workflow.

Bibliography

[1] Google Visualization Data Source. http://code.google.com/p/google-visualization-java/.

[2] Orchestra: Managing the collaborative sharing of evolving data. http://www.cis.upenn.edu/~zives/orchestra/.

[3] Protovis. http://vis.stanford.edu/protovis/.

[4] Scalability. http://en.wikipedia.org/wiki/Scalability.

[5] The workflow management coalition specification. *Workflow Management Coalition Terminology & Glossary*, February 1999.

[6] A. Ailamaki, Y. E. Ioannidis, and M. Livny. Scientific workflow management by database management. In *SSDBM*, pages 190–199, 1998.

[7] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. *Proceedings 16th International Conference on Scientific and Statistical Database Management 2004*, pages 423–424, 2004.

[8] E. Anderson, S. Callahan, D. Koop, E. Santos, C. Scheidegger, H. Vo, J. Freire, and C. Silva. VisTrails: Using Provenance to Streamline Data Exploration. In *Poster Proceedings of the International Workshop on Data Integration in the Life Sciences (DILS) 2007*, 2007. Poster presentation.

[9] C. R. Aragon, S. S. Poon, G. S. Aldering, R. C. Thomas, and R. Quimby. Using Visual Analytics to Maintain Situation Awareness in Astrophysics. *Nature*, 9:27–34, 2008.

[10] D. Auber, Y. Chiricota, M. Delest, J.-P. Domenger, P. Mary, and G. Melançon. Visualisation de graphes avec Tulip : exploration interactive de grandes masses de données en appui à la fouille de données et à l'extraction de connaissances. In M. Noirhomme-Fraiture and G. Venturini, editors, *EGC*, Revue des Nouvelles Technologies de l'Information, pages 147–156. Cépaduès-Éditions, 2007.

[11] R. S. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, K. Grochow, and E. Lazowska. Trident: Scientific Workflow Workbench for Oceanography. *2008 IEEE Congress on Services*, pages 465–466, 2008.

[12] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, 2004.

[13] L. Bernardinello and F. De Cindio. A Survey of Basic Net Models and Modular Net Classes. *Advances in Petri Nets 1992*, pages 304–351, 1992.

[14] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-Dupe: An Interactive Tool for Entity Resolution in Social Networks. *2006 IEEE Symposium On Visual Analytics And Technology*, 3843:43–50, 2006.

[15] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in MonetDB. *Communications of the ACM*, 51(12):77–85, 2008.

[16] M. Bostock and J. Heer. Protovis : A Graphical Toolkit for Visualization. *Computer*, 15(6):1121–1128, 2009.

[17] N. Boukhelifa, F. Chevalier, and J.-D. Fekete. Real-time Aggregation of Wikipedia Data for Visual Analytics. In *Proceedings of Visual Analytics Science and Technology 2010*, Los Alamitos, CA, USA, 2010. IEEE Computer Society. to appear.

[18] S. Bowers and B. Ludäscher. Actor-Oriented Design of Scientific Workflows. In *Proc. of the International Conference on Conceptual Modeling (ER)*, pages 369–384, 2005.

[19] M. Brambilla, S. Ceri, P. Fraternali, and I. Manolescu. Process modeling in web applications. *ACM Trans. Softw. Eng. Methodol.*, pages 360–409, 2006.

[20] Y. Breitbart, A. Deacon, H. J. Schek, A. Sheth, and G. Weikum. Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. *SIGMOD Records*, 22(3):23–30, 1993.

[21] P. Buneman, S. Khanna, and W.-c. Tan. *Why and Where: A Characterization of Data Provenance*, volume 1973, pages 316–330. Springer, 2001.

[22] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. *VisTrails: visualization meets data management*, volume 1, pages 745–747. ACM, 2006.

[23] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999.

[24] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kauffmann, 2003.

[25] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi. Storing and Querying Scientific Workflow Provenance Metadata Using an RDBMS. *Third IEEE International Conference on eScience and Grid Computing eScience 2007*, pages 611–618, 2007.

[26] F. Chevalier, S. Huot, and J.-D. Fekete. WikipediaViz: Conveying Article Quality for Casual Wikipedia Readers. *Pacific Visualization Symposium PacificVis 2010 IEEE*, pages 49–56, 2010.

[27] E. H.-H. Chi and J. Riedl. An operator interaction framework for visualization systems. In *Proceedings of the 1998 IEEE Symposium on Information Visualization*, pages 63–70, Washington, DC, USA, 1998. IEEE Computer Society.

[28] C. A.-N. Consortium. The active database management system manifesto: a rulebase of ADBMS features. *Sigmod Record*, 25(3):40–49, 1996.

[29] N. Diakopoulos, M. Naaman, and F. Kivran-swaine. Diamonds in the Rough : Social Media Visual Analytics for Journalistic Inquiry. *Interfaces*, pages 115–122, 2009.

[30] J. Eder, H. Groiss, and W. Liebhart. Workflow Management and Databases. In *2ème Forum Int dInformatique Appliquée Tunis*, 1996.

[31] J. Eder and W. Liebhart. *The Workflow Activity Model WAMO*, pages 87–98. 1995.

[32] S. G. Eick and A. F. Karr. Visual Scalability. *Journal Of Computational And Graphical Statistics*, 11(1):22–43, 2002.

[33] J. D. Fekete. The InfoVis Toolkit. *IEEE Symposium on Information Visualization*, pages 167–174, 2004.

[34] J.-D. Fekete and M. Beaudouin-Lafon. Using the multi-layer model for building interactive graphical applications. *Proceedings of the 9th annual ACM symposium on User interface software and technology UIST 96*, pages 109–118, 1996.

[35] J. D. Fekete and C. Plaisant. Interactive information visualization of a million items. *IEEE Symposium on Information Visualization 2002*, 2002:117–124.

[36] J. Gerth and P. Hanrahan. Maintaining interactivity while exploring massive time series. *2008 IEEE Symposium on Visual Analytics Science and Technology*, pages 59–66, 2008.

[37] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. S. Szalay, D. DeWitt, and G. Heber. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):34–41, 2005.

[38] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-Science Experiments - experience from Bioinformatics. *OST eScience Second All Hands Meeting 2003 AHM03*, 4:223–226, 2003.

[39] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, and M. Reiter. Conventional Workflow Technology for Scientific Simulation. *Guide to eScience*, pages 0–31, 2011.

[40] R. Grønmo and I. Solheim. *Towards modeling web service composition in uml*, pages 72–86. Citeseer, 2004.

[41] A. Gupta, B. Ludäscher, and L. Raschid. Report on the 2nd International Workshop on Data Integration in the Life Sciences: (DILS'05). *SIGMOD Record*, 35(2):56–58, 2006.

[42] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. *Maintaining Views Incrementally*, volume 22, pages 157–166. ACM, 1993.

[43] The HAL INRIA publication server. http://hal.inria.fr.

[44] Y. Han, E. P. Stuntebeck, J. T. Stasko, and G. D. Abowd. A visual analytics system for radio frequency fingerprinting-based localization. *2009 IEEE Symposium on Visual Analytics Science and Technology*, pages 35–42, 2009.

[45] M. C. Hao, R. K. Sharma, D. A. Keim, U. Dayal, C. Patel, and R. Vennelakanti. Application of Visual Analytics for Thermal State Management in Large Data Centres. *Computer Graphics Forum*, 29(6):1895–1904, 2010.

[46] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[47] J. Heer and M. Agrawala. Design considerations for collaborative visual analytics. *Information Visualization*, 7(1):49–62, 2008.

[48] J. Heer, S. K. Card, and J. A. Landay. *Prefuse: a toolkit for interactive information visualization*, volume 05pp, pages 421–430. ACM, 2005.

[49] M. D. Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990.

[50] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[51] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):W729–W732, 2006.

[52] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf, second edition, 2005.

[53] Z. G. Ives, T. J. Green, G. Karvounarakis, N. E. Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira. The ORCHESTRA Collaborative Data Sharing System. *Sigmod Record*, 37(3):26–32, 2008.

[54] D. A. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering The Information Age - Solving Problems with Visual Analytics*. Eurographics, November 2010.

[55] D. A. Keim, F. Mansmann, A. Stoffel, and H. Ziegler. Visual analytics. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 3341–3346. Springer US, 2009.

[56] F. Leymann. *Supporting Business Transactions Via Partial Backward Recovery In Workflow Management Systems*, pages 51–70. Springer-Verlag, 1995.

[57] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation Practice and Experience*, 18(10):1039–1065, 2006.

[58] B. Ludäscher, M. Weske, T. M. McPhillips, and S. Bowers. Scientific Workflows: Business as Usual? *BPM*, 5701:31–47, 2009.

[59] G. V. M. Weske. *Handbook on Architectures of Information Systems. (International Handbooks on Information Systems)*, chapter Workflow Languages, pages 359–379. Springer, 1998.

[60] N. Mandal, E. Deelman, G. Mehta, M.-h. Su, K. Vahi, and M. D. Rey. Integrating Existing Scientific Workflow Systems : The Kepler / Pegasus Example. *Information Sciences*, pages 21–28, 2007.

[61] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2008.

[62] J. Nielsen. *Usability Engineering*, volume 44. Morgan Kaufmann, 1993.

[63] A. Noack. *An Energy Model for Visual Graph Clustering*, volume 2912, pages 425–436. Springer, 2004.

[64] J. O'Madadhain, D. Fisher, S. White, and Y. Boey. The JUNG (Java Universal Network/Graph) Framework. *University of California Irvine*, (UCI-ICS 03-17):3–17, 2003.

[65] OSGi service platform, core specification, release 4, version 4.2. http://www.osgi.org/Release4/Download, 2009.

[66] S. G. Parker and C. R. Johnson. *SCIRun: A Scientific Programming Environment for Computational Steering*, volume 2, page 52. IEEE Press, 1995.

[67] N. W. Paton and O. Díaz. Active database systems. *ACM Computing Surveys*, 31(1):63–103, 1999.

[68] T. Risch and M. Sköld. Monitoring Complex Rule Conditions. In *Active Rules in Database Systems*, pages 81–102. Springer, New York, 1999.

[69] N. Russell, A. H. M. Ter Hofstede, W. M. P. V. D. Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. *Business*, 2:06–22, 2006.

[70] A. Rygg, P. Roe, and O. Wong. GPFlow: An Intuitive Environment for Web Based Scientific Workflow. *2006 Fifth International Conference on Grid and Cooperative Computing Workshops*, pages 204–211, 2006.

[71] A. Savikhin, R. Maciejewski, and D. S. Ebert. Applied visual analytics for economic decision-making. *2008 IEEE Symposium on Visual Analytics Science and Technology*, pages 107–114, 2008.

[72] S. Shankar, A. Kini, D. J. DeWitt, and J. Naughton. Integrating databases and workflow systems. *ACM SIGMOD Record*, 34(3):5, 2005.

[73] C. T. Silva, E. Anderson, E. Santos, and J. Freire. Using VisTrails and Provenance for Teaching Scientific Visualization. *Most*, 30(1):75–84, 2010.

[74] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *ACM SIGMOD Record*, 34(3):31–36, 2005.

[75] V. Tannen. Provenance for database transformations. *Proceedings of the 13th International Conference on Extending Database Technology EDBT 10*, page 1, 2010.

[76] I. Taylor, I. Wang, M. Shields, and S. Majithia. Distributed computing with Triana on the Grid. *Concurrency and Computation Practice and Experience*, 17(9):1197–1214, 2005.

[77] J. J. Thomas and K. A. Cook. *Illuminating the path: The research and development agenda for visual analytics*, volume 54. IEEE, 2005.

[78] W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[79] J. J. van Wijk. Guest Editor's Introduction: Special Section on the IEEE Symposium on Visual Analytics Science and Technology (VAST). *IEEE Trans. Vis. Comput. Graph.*, 17(5):555–556, 2011.

[80] F. B. Viégas, M. Wattenberg, and K. Dave. *Studying cooperation and conflict between authors with history flow visualizations*, volume 6, pages 575–582. ACM Press, 2004.

[81] The visualisation toolkit. http://www.vtk.org/.

[82] C. Weaver. Building Highly-Coordinated Visualizations in Improvise. *IEEE Symposium on Information Visualization*, pages 159–166, 2004.

[83] WILD: Wall-sized interaction with large datasets. http://insitu.lri.fr/Projects/WILD.

[84] The workflow management coalition reference model. http://www.wfmc.org/reference-model.html.

[85] Project Trident: A scientific workflow workbench. http://research.microsoft.com/en-us/collaboration/tools/trident.aspx.

[86] XML Process Definition Language. http://www.wfmc.org/xpdl.html.