



HAL
open science

Efficient, continuous and reliable Data Management by Coordinating Services

Geneveva Vargas-Solar

► **To cite this version:**

Geneveva Vargas-Solar. Efficient, continuous and reliable Data Management by Coordinating Services. Databases [cs.DB]. Institut National Polytechnique de Grenoble - INPG, 2014. tel-01006214

HAL Id: tel-01006214

<https://theses.hal.science/tel-01006214>

Submitted on 1 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITY OF GRENOBLE

Doctoral School

Mathematics, Sciences and Information Technologies, Informatics

2014

”Habilitation à diriger des recherches”

Discipline: Informatics

Presented and defended in public by

Genoveva Vargas-Solar

22 - 05 - 2014

Maison de Sciences de l’Homme, Campus Saint Martin d’Hères

**EFFICIENT, CONTINUOUS AND RELIABLE DATA
MANAGEMENT BY COORDINATING SERVICES**

Jury

President	:	Pr.	University
Reviewers	:	Ricardo BAEZA-YATES, Professor	University of Chile, Yahoo! Labs, Spain
		Barbara PERNICI, Professor	Politecnico di Milano, Italy
		Nicole BIDOIT, Professor	University Paris Sud 11, France
Examiners	:	Christine COLLET, Professor	Grenoble INP, France
		Didier GEORGES, Professor	Grenoble INP, France
		Andreas GEPPERT, Dr.	Crédit Suisse AG, Switzerland
		Yassine LAKNECH, Professor	University Joseph Fourier, France
		Marc H. SCHOLL, Professor	University of Konstanz, Germany

Genoveva VARGAS-SOLAR

Efficient, continuous and reliable Data Management by Coordinating Services

100 pages.

I dedicate these twelve years of research adventure to the hours I did not spend with the people I love: family, relatives, friends and especially couple. As a real mercenary I stole our time for devoting it to my very own pleasure. I do not have any alibi, yet I am just thankful to my beloved ones for the unconditional support and patience, and for having let me do what I desired, without regret.

Aknowledgements

I want to thank the reviewers of my work, Prof. Dr. Ricardo Baeza Yates, Prof. Nicole Bidoit-Tollu, and Prof. Barbara Pernici, for the time they devoted for reading this document, for their valuable comments that encourage me to start a new cycle in my profesional life. I thank equally, Dr. Andreas Geppert, Prof. Dr. Didier Georges, Prof. Dr. Yassine Laknech, Prof. Dr. Marc H. Scholl for accepting to participate as examiners in the evaluation board and for the interest to my work.

Through these lines I particularly thank Prof. Dr. Christine Collet, who has been my advisor, my colleague, my promoter and above all my friend. We have shared around eighteen years of professional and personal complicity that have positively influenced my work which is, in a good part, hers too.

I am and I will always be thankful to Prof. Dr. Michel Adiba, for being the one who supported my project of coming to France eighteen years ago. Through time, he has been a reference, his words have been a clever advice that put objective order to my thoughts and doubts. I am grateful for his friendship, and for the interest he has always shared with me about (Mexican) culture. Finally, I also thank him for his help and useful discussions during the preparation of this "HDR".

I would like to thank the members of the group HADAS, for the work that we have done together. Their comments and their critical eye has been helpful and encouraging for giving direction and questioning my research.

I have done quite a bit of my research collaborating with colleagues in different countries. Of course, I am thankful to my Mexican colleagues from different institutions (LANIA, CNVESTAV, CICESE, Tecnológico de Monterrey, INAOE) particularly from UDLAP in San Andrés Cholula. They have hosted me as invited scientist and several research results have been achieved by collaborating with them.

I also thank my colleagues from Universidad de la República, Uruguay, Universidad Federal Rio, Grande do Norte, Brazil, Universidad Rey Juan Carlos de Madrid, and Universidad Politécnica de Madrid, Università di Sant'Ana, Pisa, Universities Lyon I, II and III, University of Valenciennes for their confidence and availability always ready to share knowledge and time for doing exciting projects.

To my current and previous students for their confidence shown by sharing, even for a short period, the responsibility of modelling great expectations for their profesional lives. I have, of course, special thoughts for my ten PhD graduated students who are today brilliant colleagues: Xiangru Jonas Yuan, Khalid Belhajjame, Gennaro Bruno, Thi Huong Giang Vu, Hanh Tan, Alberto Portilla-Flores, Víctor Cuevas-Vicenttín, Placido Antonio Souza-Neto, Javier A. Espinosa Oviedo, and Mohamad Othman Abdallah. Devotion, humility and tenacity are some of the lessons learned by working with them. Thanks also to the postdoctoral fellows that worked with me in recent time. I am specially grateful to Dr. Paolo Buccioli for his remarkable help in innovation activities. Special thanks to Khalid for having helped me reading this document and for lending me his attentive ears while rehearsing for the viva.

I thank my current students Carlos Manuel López Enríquez, Juan Carlos Castrejón, and Orléant Njamen Epal, that give hope to some of my open research interests.

I still promise that one day I will learn my PhD students mother languages...

I thank also Laboratory of Informatics of Grenoble (LIG), and the French Mexican Laboratory of Informatics and Automatic Control (LAFMIA) for providing me an encouraging context for working in very good conditions.

This very special thank you to J for being oxygen and fresh air to day by day life. I am grateful for solidarity and for encouraging me to dare projects I thought I had aborted, for example putting me on a bike, introducing me to online console games.. May be there is a way of catching up old springs, after all ...

Great thank you to my energetic Mom who has empowered me my whole life. Mother, friend, and trainer in the early days of my existence, she showed me the value of work, the importance of discipline and balance, and to understand that any action in life is worth as long as it leads to happiness.

A nostalgic thanks to my best friend ever, Maga. Owner of my sweet childhood days, of my exciting youth, and my graceful adult life. She is also owner of the bitter days of an inevitable good by. I can walk life with sure steps thanks to your unconditional complicity.

With emotion I also thank my beloved aunts Chata and Bicha for their effort of creating memories about an immaterial "me" in our family. I also thank my Chipilo family for systematically receiving me with open arms in this special and unique Republic, and not denying my double nationality despite my long absence. Thank you also to the intense characters that form my "padrinos band" for discretely following my nomad life.

José Luis my companion, my friend, my colleague who knows the best and worse aspects of myself. Thank you from the deepest part of my heart for being, as a well known rap song says: "*Un mec qui donne des ailes!*"... Always recall that: "*El mundo cambia cuando dos se miran y se reconocen*".

Grenoble, 22 May 2014

Geneveva

Contents

Table of contents	7
1 Introduction	1
1.1 Professional timeline	1
1.2 <i>Scientific project 2002-2007</i> : Services based distributed database applications	2
1.2.1 <i>Contribution 1</i> : Adaptable service coordination	2
1.2.2 <i>Contribution 2</i> : Data integration	3
1.3 <i>Scientific project 2007-</i> : Towards efficient, reliable and, continuous service based data management	4
1.3.1 <i>Contribution 1</i> : Evaluating and optimizing hybrid queries by coordinating services . .	5
1.3.2 <i>Contribution 2</i> : Non functional properties for making services coordinations reliable .	6
1.4 Valorisation of results	6
1.5 Organization of the document	7
2 Data management evolution	9
2.1 From monolithic to customizable DBMS architectures	12
2.1.1 Classic functional architecture	13
2.1.2 Component oriented DBMS	14
2.1.3 Service oriented DBMS	16
2.2 Breaking scalable and extensible	17
2.2.1 Relaxing data management and program independence	18
2.2.2 Configuring and unbundling data management	19
2.2.3 NoSQL data store managers	20
2.3 Discussion	22
3 Coordinating services for implementing hybrid queries	25
3.1 Hybrid query evaluation	26
3.1.1 Query workflow	27
3.1.2 Generating a query workflow	28
3.1.3 Computing a query workflow cost	31
3.2 Optimizing hybrid queries using SLA contracts	32
3.2.1 Generating potential query workflow space	33
3.2.2 Computing an optimization objective	34
3.2.3 Choosing an optimum plan family	35
3.3 HYPATIA: Service coordination query evaluator	36
3.3.1 Validation	36
3.3.2 Experimental results	39
3.4 Related work	40
4 Non functional requirements for service coordinations	43
4.1 A-Policy model	44
4.1.1 Service Type	47
4.1.2 Activity Type	48
4.1.3 Execution Unit Type	51
4.1.4 A-Policy Type	53
4.2 AP Language	54

4.2.1	Defining A-policies	55
4.2.2	OAuth A-Policy	55
4.2.3	HTTP-Auth A-Policy	56
4.2.4	Associating policies to a services' coordination	57
4.3	A-Policy execution	57
4.3.1	Executing one rule	58
4.3.2	Executing several rules	59
4.3.3	Executing Actions for Reinforcing non-functional properties	60
4.4	VIOLET: an Active Policy Based Workflow Engine	63
4.4.1	Workflow Engine	64
4.4.2	Active Policy Engine	65
4.4.3	Experimental validation	67
4.5	Discussion	67
5	Towards economy oriented data management and processing	71
5.1	Intensive big data management	72
5.1.1	Observing QoS measures	73
5.1.2	Dealing with multiple storage spaces	74
5.1.3	Parallel model for implementing data processing functions	75
5.2	Data market places: economic guided data management	76
5.2.1	Economic cost oriented data brokering	77
5.2.2	Defining economic business models for curating, managing, and brokering data in data market places	78
5.3	Discussion	78
6	Data management perspectives	81
6.1	Adaptable datasets sharding and storage	82
6.2	Efficiently querying datasets	83
6.3	Big datasets management	84
7	Publications	85
	Bibliography	89

Résumé

Les nouvelles architectures matérielles mettent à disposition des services à des différentes empreintes: services réduits pour les systèmes embarqués et illimité pour le cloud, par exemple ; certaines ajoutent aussi des contraintes pour accéder aux données, qui portent sur le contrôle d'accès et la réservation/affectation de ressources par priorités (e.g., dans la grille) et sur le coût économique (e.g., dans le cloud). Les applications qui utilisent ces architectures établissent aussi des préférences de qualité de service (service level agreement SLA) qui portent sur le temps de traitement, la pertinence et la provenance de données ; le coût économique des données et le coût énergétique qui implique leur traitement. Ainsi, la gestion de données doit être revisitée pour concevoir des stratégies qui respectent à la fois les caractéristiques des architectures et les préférences des utilisateurs (service level agreements SLA).

Notre recherche contribue à la construction de systèmes de gestion de données à base de services. L'objectif est de concevoir des services de gestion de données guidée par des contrats SLA. Il s'agit d'offrir des méthodologies et des outils pour l'intégration, le déploiement, et l'exécution d'un assemblage de services pour programmer des fonctions de gestion de données. La composition de services de données particulièrement lorsqu'il s'agit des services bases de données, doit respecter des critères de qualité de service (e.g., sécurité, fiabilité, tolérance aux fautes, évolution et adaptabilité dynamique) et des propriétés de comportement (par ex., exécution transactionnelle) adaptées aux besoins des applications.

Abstract

The emergence of new architectures like the cloud open new challenges for data management. It is no longer pertinent to reason with respect a to set of computing, storage and memory resources, instead it is necessary to conceive algorithms and processes considering an unlimited set of resources usable via a "pay as U go model", energy consumption or services reputation and provenance models. Instead of designing processes and algorithms considering as threshold the resources availability, the cloud imposes to take into consideration the economic cost of the processes vs. resources use, results presentation through access subscription, and the parallel exploitation of available resources.

Our research contributes to the construction of service based data management systems. The objective is to design data management services guided by SLA contracts. We proposed methodologies, algorithms and tools for querying, deploying and executing service coordinations for programming data management functions. These functions, must respect QoS properties (security, reliability, fault tolerance, dynamic evolution and adaptability) and behavior properties (e.g., transactional execution) adapted to application requirements. Our work proposes models and mechanisms for adding these properties to new service based data management functions.

CHAPTER 1

Introduction

What we call the beginning is often the end. And to make an end is to make a beginning. The end is where we start from.

— T.S. Elliot.

This document presents a synthesis of the main results of the research I conducted in the French Council of Scientific Research (CNRS) as Research Scientist at the Laboratory *Logiciels Systèmes et Réseaux* (LSR) and Laboratory of Informatics of Grenoble (LIG UMR¹ 5217). Since 2002 I have worked in the domain of distributed database systems in the database groups Networked Open Database Services (NODS), and Heterogeneous Autonomous Distributed Data Services (HADAS). This work certainly results from the collaboration with postdoctoral fellows, graduate and, undergraduate students and, with colleagues in Europe and Latin America, particularly in Mexico at the *Laboratoire Franco Mexicain d'Informatique* (LAFMI) and, *Laboratoire Franco - Mexicain d'Informatique et Automatique* (LAFMIA UMI² 3175).

1.1 Profesional timeline

Background. Between 1996 - 2000 my research work during my graduate studies addressed problems in the domains of database management systems and distributed systems. My work was done at the LSR laboratory, in the database group STORM under the direction of Prof. Dr. Christine COLLET. I worked on the integration of distributed data base applications through an adaptable and extensible event service applying meta-programming and, reflexion techniques; and, on the specification and construction of the event service ADEES (Adaptable and Extensible Event Service) [16] and, the system ODAS (Open Distributed Active System) [17]³.

Between 2000 - 2002 I was a postdoctoral fellow at the University of Zurich, in the Database Management Group under the supervision of Prof. Dr. Klaus DITTRICH. We addressed data integration problems within database federations, particularly integration of integrity constraints from heterogeneous databases. We proposed meta-models for facilitating the integration of data expressed using different models (relational, object oriented, XML). Meta-models included integrity constraints for enabling databases integration respecting their semantics. These meta-models were validated in the context of the European project E-Parking and the project MIGI financed by the Swiss National Science Agency. We also validated of our proposals in the context of the system CALMECAC for integrating learning objects, and the system COMICS for the integration of data meta-models.

1. Mixt Research Unit of the CNRS.

2. Mixt International Unit of the CNRS.

3. See Chapter 7 that gives a list of my major publications numbered according to these references.

First five years of professional service. In 2002 I applied for and, obtained a research scientist position at the French Council of Scientific Research (CNRS). I therefore defined a scientific project around the construction of services and systems for programming distributed databases applications based on heterogeneous components. The activities of this project were organized in three axes:

1. Data integration and schemata definition used for describing heterogeneous data produced by services and DBMS.
2. Specification of adaptable and extensible systems through meta-modeling techniques (specification of systems using parametric models).
3. Process synchronization based on reactivity (event-condition-action rules) applied to services coordinations.

This scientific project was developed between 2002-2007. It led to two main contributions in the fields of *adaptable service coordination* for managing data and *data integration*. These contributions are described in the following sections.

1.2 **Scientific project 2002-2007: Services based distributed database applications**

Modern applications are, in general, distributed, heterogeneous and built out of distributed, autonomous components that are loosely coupled. The evolution of these applications underlines the need to develop configurable infrastructures for guaranteeing certain common functions like communication, persistence, duplications and querying.

During the last twenty years, industrials and academics have proposed infrastructures for building distributed applications. Middlewares are examples of these infrastructures and they are a fundamental backbone of distributed applications. The rigidity of some middlewares is contradictory to the aim of flexibility. In general, these infrastructures are black boxes providing close models that do not ease their use for building applications.

Event services based on push/pull technologies are part of middlewares and inherit of their characteristics and associated limitations. Reflexive solutions based on meta-programming emerged as a response with flexible and adaptable solutions. In this context, database technology evolved towards cooperation and integration. Databases moved from monolithic approaches to component oriented systems.

The construction of database applications using autonomous services implies the deployment of components or services necessary for (i) managing data and resources (e.g., persistency, duplication, distributed execution control, transactions services); and (ii) making application servers interact, like events and reaction services. Composing these services leads to the construction of data servers as evolutive legos, under the condition of mastering a good knowledge of the dependencies among internal and external DBMS functions implemented by such services.

The objective of my scientific project between 2002-2007 was to propose models and mechanisms for coordinating services and building database servers used for integrating heterogeneous databases.

1.2.1 **Contribution 1: Adaptable service coordination**

Thanks to the standardization of the notion of service used for modeling software, hardware and network resources, there are languages for defining service interfaces (e.g., WSDL in the Web world), protocols

(e.g., SOAP) and, architectures (e.g., REST) for enabling the invocation of methods. There are directories (name services) for looking up services and facilitating their access. These directories are implemented using simple discovery approaches (e.g. white pages) but also using semantic and non-functional properties (NFP) expressed by measures like availability, reliability and, response time. Finally, service oriented middleware that facilitates their location, access and, invocation.

We proposed approaches⁴ [6] guided by semantics for discovering services using formal approaches and logic programming methods. Our objective in these proposals was to understand the role of the discovery process in the evaluation of queries based on services coordination. The construction of service oriented systems consist of coordinating method (or operation) calls exported by services (e.g., discovered thanks to a directory) for implementing an application logic (i.e., functional aspect).

Standard languages were proposed (e.g., BEPL) for expressing invocations coordination; engines for ensuring their execution and languages for expressing protocols for managing certain non-functional properties (security, atomicity, exception handling, adaptability). These protocol specification languages (W3C WS* protocols) generate coordinations weaving the code of functional and non-functional aspects, which makes them difficult to maintain.

We proposed the framework PYROS [15] for building coordination mechanisms necessary for the construction of service based systems. PYROS implemented a coordination model that enabled the personalization of communication protocols between a service and a coordination mechanism (i.e., push/pull, synchronous/asynchronous). We proposed algorithms for verifying termination and deadlock free properties on a coordination. Service management and coordination mechanisms built using PYROS implement functions to support dynamic adaptation of coordinations. For ensuring the consistency of modifications, PYROS coordination mechanisms provide verification modules based on the proposed algorithms. We developed two applications based on the W3C standards and the construction of Web portals (TERRA-ACQUA [13,14] and PyROS [15]) developed on Java and .NET platforms.

1.2.2 *Contribution 2: Data integration*

We proposed ADEMS [18]⁵, a knowledge based service for configuring adaptable mediators based on (i) meta-data (knowledge) modeled by ontologies expressed using description logics, and (ii) reasoning mechanisms for configuring mediators in an "intelligent" way. A mediator was based on reasoning for processing queries. Queries were expressed intuitively by choosing nodes in an ontology. The verification and rewriting processes are based on deduction. A mediator resulting from this process is adaptable to different application domains and requirements (data sources and other mediators). Part of the implementation of ADEMS was validated in an automatic learning environment called SKIMA⁶.

Our research results were validated on Web based applications, bio-informatics and geographic data management (projects MEDIAGRID financed by the ANR⁷; and, BIOLAVand, SPIDHERS financed by the CUDI⁸); the evolution of data warehouses, the implementation of virtual enterprises (project DAWISinc collaboration with University Rey Juan Carlos of Madrid, Spain).

This scientific project evolved towards a project on data management following the evolution of the database domain, technology and data management requirements of modern applications. We give an

4. See Chapter 7 that gives a list of my major publications numbered according to these references.

5. See Chapter 7 that gives a list of my major publications numbered according to these references.

6. SKIMA was developed by H. Pérez Urbina in the context of his final project of his engineering studies at University of Las Américas in Puebla Mexico

7. French Agency of Research, <http://www.agence-nationale-recherche.fr>

8. Academic Corporation for the Development of Internet in Mexico <http://www.cudi.mx>

overview of this project in the following lines.

1.3 *Scientific project 2007- : Towards efficient, reliable and, continuous service based data management*

The ANSI/SPARC architecture deployed on client - server architecture models that characterized classic data management systems (relational, object and XML DBMS) evolves in parallel with the requirements of new mobile, ubiquitous and dynamic applications. This evolution yield new ways of delivering data management functions - internal and external: as components (at the end of the 90's), as peer to peer networks (in the beginning of the 2000), and as services the last years. Today, service based data management infrastructures coordinate and adapt services for implementing ad hoc functions like storage, fragmentation, replication, analysis, decision making and, data mining for managing huge distributed multimedia and multiform data collections. Services are deployed on different hardware and, software architectures like the grid, P2P and sensors networks, embedded systems, the Web 2.0 and lately the cloud.

Consider an e-health scenario where an ambulance, in an accident, uses an electronic device for obtaining hospital addresses - images and maps- available surgery rooms, correlated with telephone numbers and professional information of surgeons available. Hospitals less than 3 KM from the position of the ambulance which is moving at 80 Km/h average speed, must be located.

This query can be processed by accessing information provided by services: "Google street" for the hospitals address images; location services for tracking the ambulance moving " position"; proprietary services exported by hospitals for obtaining the occupancy of surgery rooms and the names of the surgeons in charge; and, finally professional services like "LinkedIn" for retrieving information about surgeons professional skills. In order to discover and locate services on Internet, there are directory services. First it should be possible to find and choose services, then coordinate them for retrieving data and then, correlate these data for building results. My research has addressed services description and discovery and, their coordination for building data querying. The following lines briefly sketch the most important results of my work.

Novel architectures provide services at different granularities: light services for embedded systems, unlimited services for the cloud, for example. Some of these services add constraints for accessing data addressing access control and resources reservation/assignment based on priorities (e.g., in Grid architectures) and, on economic cost (e.g., in the cloud). Applications using these architectures define also quality of service preferences specifying Service Level Agreements (SLA⁹) that deal with data processing time, pertinence, provenance; data transportation and, processing economic and energy costs.

For example, in our scenario, the paramedical staff in the ambulance can have preferences concerning the query. The evaluation must minimize the energy cost in order to avoid consuming the battery of the device used for posting it; the answer must arrive as fast as possible and, it should contain pertinent data; the economic cost must be minimized (i.e., reduce the transferred mega octets for avoiding to exhaust octets specified in the data transfer subscription contract with a telecommunication company). Thus, data management must be revisited for conceiving strategies that respect the architectural characteristics and, users preferences (SLA). In this context we identify three scientific challenges:

9. A service-level agreement (SLA) is a negotiated agreement between two parties, where one is the customer and the other is the service provider. This can be a legally binding formal or an informal "contract" (for example, internal department relationships). Contracts between the service provider and other third parties are often (incorrectly) called SLAs because the level of service has been set by the (principal) customer, there can be no "agreement" between third parties; these agreements are simply a "contract."

- SLA guided access and processing of data (flows), where data are produced by services and, the devices that host them are connected to heterogeneous networks.
- Data access and processing energy, economic and temporal cost estimation: use cost for expressing preferences in SLA contracts.
- Optimization strategies and algorithms for minimizing data access and processing time, energy, economic costs while maximizing the use of computing resources provided under "pay as U go models".

Our research contributes to the construction of service based data management systems that provide solutions to the previous challenges. The objective is to design data management services guided by SLA contracts. We aim at proposing strategies, algorithms and, tools for querying, deploying and, executing a service assembly that implement data management functions. Service composition, must respect QoS properties (security, reliability, fault tolerance, dynamic evolution and, adaptability) and, behavior properties (e.g., transactional execution) adapted to application requirements (expressed in SLA contracts).

1.3.1 *Contribution 1: Evaluating and optimizing hybrid queries by coordinating services*

Once data management is delivered as service coordinations, it can have associated non-functional properties (NFP): exception handling and recovery. We proposed mechanisms for optimally accessing data by coordinating services respecting SLA contracts. We assume that services give access to data through Application Programming Interfaces (API's), produce spatio-temporal data flows periodically and, in batch. We also make the assumption that there is no full-fledged DBMS available providing data management functions for processing queries. This research led to the following original contributions¹⁰:

- Query evaluation (continuous, recurrent or batch) through reliable service coordinations guided by SLAs; data services can be mobile and static, and data can be spatio-temporal and, produced continuously (streams).
- Service coordination optimization for reducing economic, energy and, time cost.

We introduced the notion of hybrid query, analyzed in a taxonomy that includes the types of queries that industrial and, academic systems can evaluate [1,4]:

1. The query language HSQL (Hybrid Services Query Languages) for expressing hybrid queries based on service coordination [1], and, the language MQLiST (Mashup Query Spatio Temporal Language, project CLEVER) for studying the way hybrid query results can be integrated in a mashup.
2. Specification of a query model based on data service coordinations using abstract state machines (ASM) [2,4]. The model is based on the notion of query workflow.
3. The algorithm BP-GYO [1] that is based on the Graham-Yu-Ozsoyoglu (GYO) algorithm in [Yan81], for generating the query workflow that implements a query expressed in HSQL.
4. An algorithm for computing the query workflows search space that implement an hybrid query and, that respect an associated SLA expressed as an aggregation of measures (economic, temporal and, energetic costs).
5. Implementation of the hybrid query evaluation engine HYPATIA[1].

These results were obtained in the context of the project ANR-ARPEGE program OPTIMACS¹¹.

10. See Chapter 7 that gives a list of my major publications numbered according to these references.

11. <http://optimacs.imag.fr>

1.3.2 *Contribution 2: Non functional properties for making services coordinations reliable*

We proposed models and systems for representing non-functional properties of service coordinations in an orthogonal way and, strategies for reinforcing them dynamically¹²:

- MEOBI[10] a security framework for building authentication, non repudiation, message encryption services and, dependable service coordinations.
- SEBASa framework for managing evolution and mechanisms for dynamic adaptation of service coordinations. The framework is based on an event ,reaction services for specifying and executing evolution operations. Finally, evolution operations are guided by the semantic of services and coordination.
- VIOLET[11] an execution system for ensuring atomic execution of service coordinations. A coordination engine interacts with a policy engine through an interface that exports methods for stoping, aborting and, re-executing the activities of a service coordination. The engine evaluates policies by synchronizing them with the execution of a service coordination.

These results were done respectively in the context of the projects ECOS-ANUIES ORCHESTRA¹³ and WebContent [2]. These models were consolidated by the active policy model A-Policy Model [11] and, the methodology with its associated environment π -SODM [8] . This work was done in the context of the projects ORCHESTRA, E-CLOUDSS¹⁴ and, CLEVER¹⁵. We also addressed the problem of adding properties to service coordinations using Aspect Oriented Programming (AOP) methods and, Abstract State Machines (ASM). We proposed the system MexADL for managing maintainability (non functional property) of systems¹⁶.

1.4 Valorisation of results

Our research results have been valorized by developing applications in the following domains:

- e-science
 - data mediation applied to biologic data (the system SISELS developed in the project BIOLAV of the CUDI program);
 - construction of the mexican astronomic virtual observatory (system ANDROMEDA proposed in the project E-GROV);
 - construction of a climatology observatory in Latin America (projects RED-SHINE, <http://red-shine.imag.fr>), CLEVER, <http://clever.imag.fr>);
- data querying for transport applications particularly on vehicular networks [3], and e-government applications¹⁷.
- data management on sensor and inter-vehicular networks¹⁸.

These activities were done in the context of cloud architectures in projects addressing the optimization of queries implemented by service compositions¹⁹ and reliable climatology data service compositions (RED-SHINE <http://red-shine.imag.fr>, CLEVER <http://clever.imag.fr>), and industrial processes data management (CAISES <http://www.fp7cases.eu/>).

12. See Chapter 7 that gives a list of my major publications numbered according to these references.

13. <http://orchestra.imag.fr>

14. <http://e-cloudss.imag.fr>

15. <http://clever.imag.fr>

16. <http://code.google.com/p/mexadl/>

17. Project E-CLOUDSS <http://e-cloudss.imag.fr>

18. Projects DELFOS supported by the French Mexican Laboratory of Informactis (LAFMI), and S2EUNET of the FP7 People IRSES program <http://s2eunet.org>

19. Project OPTIMACS <http://optimacs.imag.fr>

1.5 Organization of the document

The remainder of this document is organized as follows:

- Chapter 2 describes our vision of the evolution of data management systems, their evolution towards the notion of data management in order to describe the context in which we specified our research project: service coordination for implementing data management functions.
- Chapter 3 introduces our approach for evaluating hybrid queries using service coordinations: rewriting, and optimization. The chapter introduces the notion of query workflow that implements a service coordination used for implementing an hybrid query. It shows how to generate a query workflow given a declarative hybrid query expression, how to compute a query workflow cost and how to optimize an hybrid query. It also introduces HYPATIA the hybrid query evaluator that we proposed.
- Chapter 4 describes our policy based approach for providing non-functional properties to service coordinations. It introduces the notion of policy based service coordination. It shows how to define policies and associate them with services coordinations and thereby make them reliable.
- Chapter 5 describes our current work concerning the association of economic models to data management. It introduces the notion of data market and an economic cost oriented approach for delivering data.
- Chapter 6 underlines the contributions presented in this document and discusses research perspectives.

CHAPTER 2

Data management evolution

It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change.

— Charles Darwin.

Database management systems (DBMSs) emerged as a flexible and cost-effective solution to information organization, maintenance, and access problems found in organizations (business, academia, and government). DBMSs addressed these problems with (i) modeling and long - term reliable data storage capabilities; as well as with (ii) retrieval and manipulation facilities for persistent data by multiple concurrent users or transactions [DG00]. The concept of data model (most notably the relational models, Codd 1970; and the object-oriented data models, Atkinson et al. 1989; Cattell & Barry 1997), the Structured Query Language (SQL, Melton & Simon 1996), and the concept of transaction (Gray & Reuter 1992) are crucial ingredients of successful data management in current enterprises.

Today, the data management market is dominated by major Object-Relational Database Management Systems (OR-DBMSs) like Oracle¹, Universal DB2², or SQLServer³. These systems arose from decades of corporate and academic research pioneered by the creators of System R [ABC⁺76] and Ingres [SHWK76]. Since their inception, innovations and extensions have been proposed to enhance DBMSs in power, usability, and spectrum of applications (see Figure 2.1).

The introduction of the relational model [Cod70], prevalent today, enabled to address the shortcomings of earlier data models. Subsequent data models, in turn, were relegated or became complementary to the relational model. Further developments focused on transaction processing, and on extending DBMSs to support new types of data (e.g. spatial, multimedia, etc.); data analysis techniques and systems (e.g., data warehouses and OLAP systems, and data mining). The evolution of data models and the consolidation of distributed systems made it possible to develop mediation infrastructures [Wie92] that enable transparent access to multiple data sources through querying, navigation, and management facilities. Examples of such systems are multi-databases, data warehouses, Web portals deployed on Internet or Intranets. Common issues tackled by such systems are (i) how to handle diversity of data representations and semantics? (ii) how to provide a global view of the structure of the information system while respecting access and management constraints? (iii) how to ensure data quality (i.e., freshness, consistency, completeness, correctness)?

Besides, the Web of data has led to Web-based DBMS and XML data management systems serving as pivot models for integrating data and documents (e.g., active XML). The emergence of the Web marked a

1. <http://www.oracle.com>
2. <http://www-01.ibm.com/software/data/db2/>
3. <http://www.microsoft.com/sqlserver/2008/en/us/>

History of DBs & DBMS

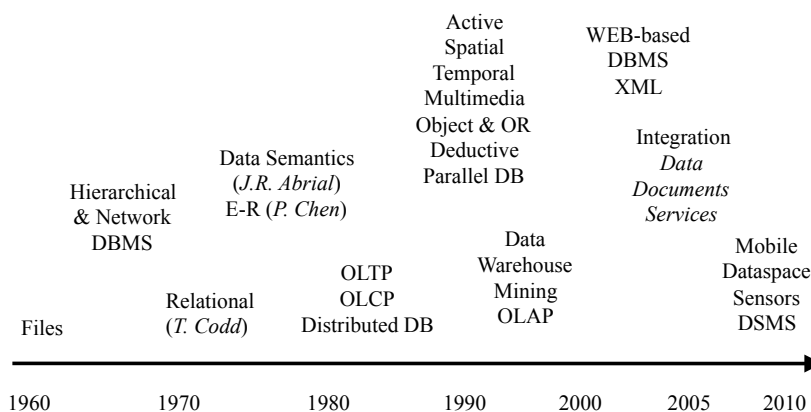


Figure 2.1: Historical outline of DBMSs [Adi07]

turning point, since the attention turned to vast amounts of new data outside of the control of a single DBMS. This resulted in an increased use of data integration techniques and exchange data formats such as XML. However, despite its recent development, the Web itself has experienced significant evolution, resulting in a feedback loop between database and Web technologies, whereby both depart from their traditional dwellings into new application domains.

Figure 2.2 depicts the major shifts in the use of the Web. A first phase saw the Web as a means to facilitate communication between people, in the spirit of traditional media and mainly through email. Afterwards, the WWW made available vast amounts of information in the form of HTML documents, which can be regarded as people-to-machines communication. Advances in mobile communication technologies extended this notion to mobile devices and a much larger number of users. A more recent trend exemplified by Web Services, and later Semantic Web Services, as well as Cloud computing⁴, consists of machine-to-machine communication. Thus, the Web has also become a means for applications and a plethora of devices to interoperate, share data and resources.

Most recent milestones (see Figure 2.1) in data management address data streams leading to Data Stream Management Systems (DSMS), mobile data providers and consumers that have also led to data management systems dealing with mobile queries and mobile objects. Finally, the last five years concern challenges introduced by the XXL phenomenon including the volume of data to be managed (i.e. big data) that makes research turn back the eyes towards DBMS architectures (clouded DBMS), data collection construction⁵ and parallel data processing. In this context, it seems that big datasets processing must profit from available computing resources by applying parallel execution models, thereby achieving results in "acceptable" times.

4. Cloud computing enables on-demand network access to computing resources managed by external parties.

5. See <http://www.datascienceinstitute.org>

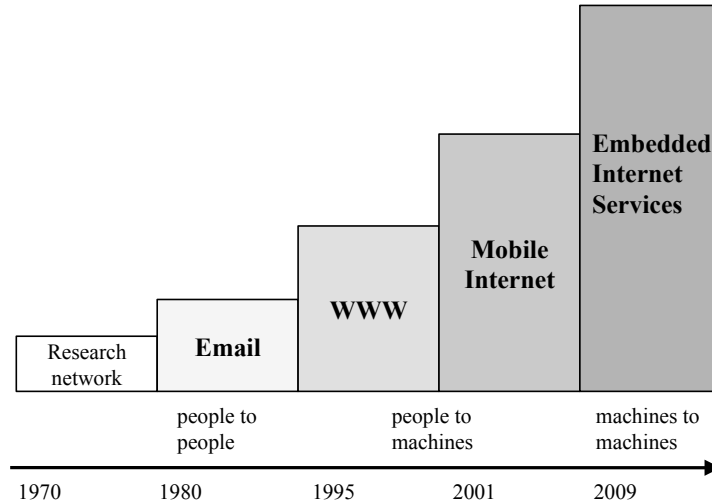


Figure 2.2: Shifts in the use of the Web [Mat01]

Relational queries are ideally suited to parallel execution because they consist of uniform operations applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. By streaming the output of one operator into the input of another operator, the two operators can work in series giving pipelined parallelism [DG92]. By partitioning the input data among multiple processors and memories, an operator can often be split into many independent operators each working on a part of the data. This partitioned data and execution gives partitioned parallelism.

In this context, we can identify three major aspects that involve database and Web technologies, and that are crucial in satisfying the new information access requirements of users.

- First, a large number of heterogeneous data sources accessible via standardized interfaces, which we refer to as *data services* (e.g., Google service).
- Second, computational resources supported by various platforms that are also publicly available through standardized interfaces, which we call *computation services* (e.g. hash Amazon E3C service).
- Third, mobile devices that can both generate data and be used to process and display data on behalf of the user.

The new DBMS aims at fulfilling ambient applications, data curation and warehousing, scientific applications, online games, among others [HTT09]. Therefore, future DBMS must address the following data management requirements:

- Data storage, persistence for managing distributed storage spaces delivered by different providers (e.g., Dropbox, Skydrive, and Google store); and efficiently and continuously ensuring data availability using data sharding, and duplication; data curation and maintenance in order to ensure their usability, and their migration into new hardware storage supports.
- Efficient and continuous querying, and mining of data (flows). These are complex processes requiring

huge amounts of computing resources. They must be designed⁶, implemented and deployed on well adapted architectures such as the grid, the cloud but also sensor networks, mobile devices with different physic capacities (i.e., computing and storage capacity).

- Querying services that can implement evaluation strategies able to:
 - process continuous and one-shot queries that include spatio-temporal elements, and nomad sources;
 - deal with exhaustive, partial, and approximate answers;
 - use execution models that consider accessing services as data providers, and that include as a source the wisdom of the crowd;
 - integrate "cross-layer" optimization that includes the network, and the enabling infrastructure as part of the evaluation process, and that can use dynamic cost models based on execution, economic, and energy costs.

The DBMS of the future must also enable the execution of algorithms, and of complex processes (scientific experiments) that use huge data collections: multimedia documents, complex graphs with thousands of nodes. This calls for a thorough revision of the hypotheses underlying the algorithms, and protocols developed for classic data management approaches, and the architecture of the DBMS [CAB⁺13]. In the following sections we discuss some of these issues focussing mainly on the way data management services of different granularities are delivered according to the DBMS architectures. Section 2.1 analyses DBMS architectures evolution, from monolithic to customizable systems. Section 2.2 discusses how scalability and extensibility properties, important across all DBMS ages, have been ensured; the section also discusses associated implications on the systems performance. Section 2.3 discusses open perspectives on DBMS architectures and how they deliver their functions for fulfilling application requirements.

2.1 From monolithic to customizable DBMS architectures

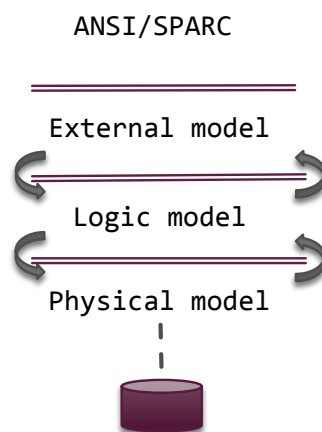


Figure 2.3: ANSI-SPARC DBMS architecture

Different kinds of architectures serve different purposes. The ANSI/SPARC [Adi07] architecture (Figure 2.3) that characterizes classic database management systems (relational, object oriented, XML) deployed on client-server architectures has evolved in parallel to the advances resulting from new application require-

6. See MapReduce models for implementing relational operators [ASM⁺12].

ments, data volumes, and data models. The three-level-schema architecture reflects the different levels of abstraction of data in a database system distinguishing (i) the external schemas that users work with, (ii) the internal integrated schema of the entire database, and (iii) the physical schema determining the storage, and organization of databases on secondary storage.

The evolution of devices with different physical capacities (i.e, storage, computing, memory), and systems requiring data management functions started to show that adding more and more functions to the monolithic DBMS does not work. Instead, it seems attractive to consider the alternative of extending DBMSs allowing functionality to be added or replaced in a modular manner, as needed. The structure of a monolithic DBMS shown in Figure 2.4 shows three key components of the system: the storage manager, the transaction manager, and the schema manager.

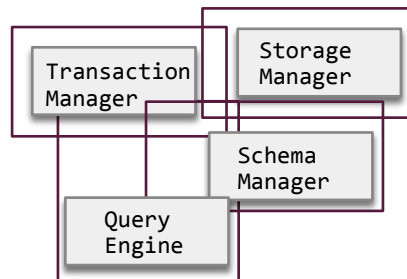


Figure 2.4: Classic DBMS functions [DG00]

While such a closely woven implementation provides good performance/efficiency, customization is an expensive and difficult task because of the dependencies among the different components. For example, changing the indexing or clustering technique employed by the storage manager, changing the instance adaptation approach employed by the schema manager or the transaction model can have a large ripple effect on the whole system [DG00].

During the last twenty years, in order to better match the evolution of user and application needs, many extensions have been proposed to enhance the DBMS functions. In order to meet all new requirements, DBMSs were extended to include new functionalities. Extensible and personalizable database systems [CH90] were an attempt to ease the construction of DBMSs by exploiting software reusability [GD94b], and proposing a general core that can be customized or extended, or even used to generate some DBMS parts. Trade-offs between modularity and efficiency, and granularity of services, and the number of inter-service relationships result in DBMS designs which lack customizability⁷. A study of the standard task oriented architecture of DBMSs can be useful to determine their viability in new environments, and for new applications. The following sections give an overview of the main elements for showing how DBMS have and should evolve in order to address the scalability and performance requirements for data management.

2.1.1 Classic functional architecture

The classic DBMS architecture consists of a number of layers [HR01, HR83, DFKR99] each supporting a set of data types and operations at its interface. It consists of several components (modules or managers of concrete or abstract resource). The data types and operations defined for the modules of one layer are

7. "Although a task oriented architecture is much more suitable for reasoning about extensibility, and DBMS construction, reference architectures rarely exist (with the strawman architecture developed by the Computer Corporation of America, CCA 1982, as a notable exception)" [DG00].

implemented using the concepts (data types and operations) of the next-lower level. Therefore, the layered architecture can also be considered as a stack of abstract machines. The layered architecture model as introduced by Härder and Reuter (1983) is composed of five layers described in [DG00]:

1. The uppermost layer supports logical data structures such as relations, tuples, and views. Typical tasks of this layer include query processing and optimization, access control, and integrity enforcement.
2. The next layer implements a record-oriented interface. Typical entities are records and sets⁸ as well as logical access paths. Typical components are the data dictionary, transaction management, and cursor management.
3. The middle layer manages storage structures (internal records), physical access paths, locking, logging, and recovery. Therefore, relevant modules include the record manager, physical access path managers (e.g., a hash table manager), and modules for lock management, logging, and recovery.
4. The next layer implements (page) buffer management and implements the page replacement strategy. Typical entities are pages and segments.
5. The lowest layer implements the management of secondary storage (i.e., maps segments, and pages to blocks and files).

Due to performance considerations, no concrete DBMS has fully obeyed the layered architecture [DG00]. Note that different layered architectures and different numbers of layers are proposed, depending on the desired interfaces at the top layer. If, for instance, only a set-oriented interface is needed, it is useful to merge the upper two layers. In practice, most DBMS architectures have been influenced by System R [ABC⁺76], which consists of two layers:

- The relational data system (RDS), providing for the relational data interface (RDI) it implements SQL (including query optimization, access control, triggers, etc.);
- The relational storage system (RSS), supporting the relational storage interface (RSI), it provides access to single tuples of base relations at its interface.

Layered architectures [HR83] were designed to address customizability, but they provide partial solutions at a coarse granularity. In the layered architecture proposed by [HR83], for example, the concurrency control components are spread across two different layers. Customization of the lock management or recovery mechanisms (residing in the lower layer) have a knock-on effect on the transaction management component (residing in the higher layer) [DG00].

As we will discuss in the next sections, layered architectures are used by existing DBMS, and they remain used despite the different generations of these systems. In each generation, layers and modules were implemented according to different paradigms (object, component, and service oriented) changing their granularity and the transparency degree adopted for encapsulating the functions implemented by each layer.

2.1.2 Component oriented DBMS

Component aware [Fro98, DW98, KA98, ND95, NM95, OHE96, Szy97] was a paradigm to address reusability, separation of concerns (i.e., separation of functional from non-functional concerns) and ease of construction⁹. Component based systems are built by putting components together to form new software

8. As found in the Committee on Data Systems Languages, CODASYL data model.

9. A (software) component, then, is a software artifact modeling and implementing a coherent and well-defined set of functions. It consists of a component interface and a component implementation. Components are black boxes, which means that clients can use them properly without knowing their implementation. Component interface and implementation should be separated such that multiple implementations can exist for one interface and implementations can be exchanged.

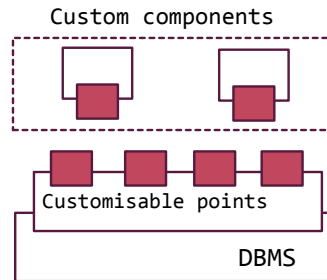


Figure 2.5: Component DBMS [DG00]

systems. Systems constructed by composition can be modified or extended by replacing or adding new components.

Approaches to extend and customize DBMS adopted the component oriented paradigm for designing at least the customizable modules of the architecture, as components. Plug-in components are added to functionally complete DBMS and fulfil specialized needs. The components of CDBMS (Component Database Management Systems) are families of base and abstract data types or implementations of some DBMS function, such as new index structures. To date, all systems in this category are based on the relational data model and existing relational DBMSs, and all of them offer some object-oriented extensions. Example systems include IBM's DB2 UDB (IBM 1995), Informix Universal Server (Informix 1998), Oracle8 (Oracle 1999), and Predator [Ses98]. Descriptions of sample component developments can be found in [BSSJ99, DM97].

Furthermore, the customization approach employed by most commercial DBMSs are still largely monolithic (to improve performance). Special points similar to hot spots in OO frameworks [FS97] allow to custom components to be incorporated into the DBMS. Examples of such components include Informix DataBlades¹⁰, Oracle Data Cartridges [BKMM00] and DB2 Relational Extenders [DCC⁺01]. However, customization in these systems is limited to the introduction of user-defined types, functions, triggers, constraints, indexing mechanisms, and predicates, etc.

Another way of addressing DBMS componentization was to provide database middlewares. Such middlewares leave data items under the control of their original (external) management systems while integrating them into a common DBMS-style framework. External systems exhibit, in many cases, different capabilities, such as query languages with varying power or no querying facilities at all. The different data stores might also have different data models (i.e., different data definition and structuring means), or no explicit data model at all. The goal of graceful integration is achieved through componentization.

The architecture introduces a common (intermediate) format into which the local data formats can be translated. Introduced components perform this kind of translation. Second, common interfaces and protocols define how the database middleware system and the components interact (e.g., in order to retrieve data from a data store). These components (called wrappers) are also able to transform requests issued via these interfaces (e.g., queries) into requests understandable by the external system. In other words, these components implement the functionality needed to access data managed by the external data store. Examples of this approach include Disco [TRV98], Garlic [RS97], OLE DB [Bla96a, Bla96b, BP98], Tsimmis [GMPQ⁺97], Harmony [RB99] (which implements the CORBA query service), and Sybase Adaptive Server Enterprise [OPSY98]. Sybase allows access to external data stores, in Sybase called specialty data stores, and other types of database systems. ADEMS [CBB⁺04, BCVS06] proposes mediation cooperative components

10. <http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.dbdk.doc/dbdk26.htm>

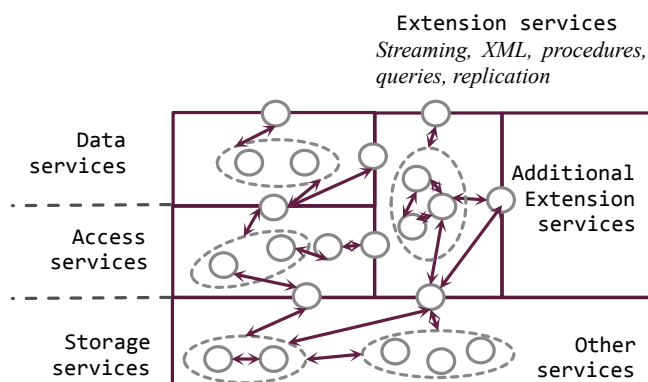


Figure 2.6: Service oriented DBMS [SZD07]

or services that can broker and integrate data coming from heterogeneous sources. The cooperative brokers allow to build an extensible data mediation system.

CDBMS were successful because of the adoption of the notion of cartridge or blade by commercial DBMS. Other academic solutions were applied in some concrete validations. It is true that they enabled the configuration of the DBMS, but they still provide monolithic, complex, resources consuming systems ("kernels") that need to be tuned and carefully managed for fulfilling the management of huge data volumes. These systems continued to encourage classic conception of information systems, with clear and complete knowledge of the data they manage, with global constraints, and homogeneous management with well identified needs. Yet, the evolution of technology, and the production of data stemming from different devices and services, the access to non-curated continuous data collections, the democratized access to continuous information (for example in social networks) calls for light weight data management services delivered in ad hoc personalized manners and not only in full-fledged one fits all systems like the (C)DBMS. Together with this evolution, emerged the notion of service aiming to ease the construction of loosely coupled systems. DBMS then started to move towards this new paradigm, and were redefined as data management service providers.

2.1.3 Service oriented DBMS

Today the DBMS architecture has evolved to the notion of service based infrastructure where services¹¹ are adapted and coordinated for implementing ad hoc data management functions (storage, fragmentation, replication, analysis, decision making, data mining). These functions are adapted and tuned for managing huge distributed multiform multimedia data collections, known as the big data phenomenon. Applications can extend the functionality of DBMS through specific tasks that have to be provided by the data management systems, these tasks are called services, and allow interoperability between DBMS and other applications [GSD97].

[SZD07] proposes a database architecture on the principles of Service Oriented Architecture (SOA) as a system capable of handling different data types, being able to provide methods for adding new database

11. Services that are accessible through a well defined and described interface enable any application to extend and reuse existing services without affecting other services. Services in the SOA approach are accessed only by means of a well defined interface, without requiring any knowledge on their implementation. SOAs can be implemented through a wide range of technologies like RPC, RMI, CORBA, COM, and web services, not making any restrictions on the implementation protocols. In general, services can communicate using an arbitrary protocol, for example, they can use a file system to send data between their interfaces.

features (see Figure 2.6). The SBDMS (Service Based Data Management System) architecture borrows the architectural levels from Haerder [Hae05], and includes new features and advantages introduced by SOA into the field of database architecture. It is organized into functional layers that each with specialized services for specific tasks.

Storage services work on the byte level, in very close collaboration with file management functions of the operating system. These services have to handle the physical specifications of each non-volatile device. In addition, they provide services for updating existing data and finding stored data, propagating information from the Access Services Layer to the physical level. Since different data types require different storage optimizations, special services are created to supply their particular functional needs. This layer is equivalent to the first and second layer of the five layer architecture presented by Haerder and Reuter [HR85, Hae05].

Access services is in charge of the physical data representations of data records and provides access path structures like B-trees. It provides more complex access paths, mappings, particular extensions for special data models, that are represented in the Storage Services Layer. Moreover, it is responsible for sorting record sets, navigating through logical record structures, making joins, and similar higher-level operations. This layer represents a key factor to database performance. The Access Services Layer has functions that are comparable to those in the third and fourth layer as presented by Härder and Reuter [HR85, Hae05].

Data services provide data represented in logical structures like tables or views. These are data structures without any procedural interface to the underlying database. The Data Service Layer can be mapped to the Non-Procedural and Algebraic Access level in the architecture by Haerder and Reuter [HR85, Hae05].

Extension services users can design tailored extensions for example, creating new services or reusing existing ones from any available service from the other layers. These extensions help to manage different data types like XML files or streaming data. In this layer, users can integrate application specific services in order to provide specific data types or specific functionalities needed by their applications (e.g., for optimization purposes).

A service based DBMS externalizes the functions of the different systems layers, and enables the programming of personalized data management as a service systems. They make it possible to couple the data model characteristics with well adapted management functions that can themselves be programmed in an ad-hoc manner. The DBMS remains a general purpose system that can be personalized, thanks to service composition, to provide ad-hoc data management. It is then possible to have services deployed in architectures that make them available to applications in a simple way.

2.2 Breaking scalable and extensible

As discussed before the evolution of the DBMS architecture responds to the evolution of applications requirements in regard to efficient management. With the emergence of the notion of service, the DBMS architecture has been 'fragmented' into components and services that are deployed in distributed platforms such as the Web 2.0. Applications use different kinds of data that must be managed according to different purposes: some data collections are read oriented with few writes; other data is modified continuously, and it is exploited by non concurrent read operations. Some collections are shared, and they can support low

consistency levels as long as they are available. Furthermore, such data is multiform, and more and more multimedia, they are modeled or at least exchanged as documents particularly if they stem from the Web.

Requirements concerning data management performance vs. volume, and the effort of constructing data collections themselves has determined the evolution of DBMS towards efficiency. The three level architecture that encouraged program-data independence based on series of transformations among layers seems inappropriate to answer to performance requirements. The architectures are making levels separations thin. The principle being that the less transformations among data are required the more performant are data management functions particularly querying, accessing, and processing. It seems that the very principle of independence between programs and data management is a very expensive quality that is not worth paying in certain situations.

2.2.1 Relaxing data management and program independence

The first evolution of DBMS when the object oriented paradigm emerged, and the logic and physical level started to approach in order to provide efficient ways of dealing with persistent objects. Together with the OO paradigm emerged applications requiring databases that can handle very complex data, that can evolve gracefully, and that can provide the high-performance dictated by interactive systems. Database applications could be programmed with an OO language, and then object persistency was managed by an OO DBMS. The OO DBMS manifesto [ABD⁺89] stated that persistence should be orthogonal, i.e., each object, independent of its type, is allowed to become persistent as such (i.e., without explicit translation). Persistency should also be implicit: the user should not have to explicitly move or copy data to make it persistent. This implied also that transparency was enforced regarding secondary storage management (index management, data clustering, data buffering, access path selection, and query optimization).

Extensible database systems [CH90] allowed new parts such as abstract data types or index structures to be added to the system. Enhancing DBS with new Abstract Data Type (ADT) or index structures was pioneered in the Ingres/Postgres systems [SRG83, SR86, LS88]. Ingres supports the definition of new ADTs, including operators. References to other tuples can be expressed through queries (i.e., the data type postquel), but otherwise ADTs, and their associated relations still had to be in first normal form. This restriction was relaxed in systems that have a more powerful type system (e.g., an object-oriented data model) [BDK92, DKA⁺86, DGL86, LKD⁺88, SPSW90]. Another area in which extensions have been extensively considered are index structures. In Ingres/Postgres, existing indexes (such as B-trees) can be extended to also support new types (or support existing types in a better way). To extend an index mechanism, new implementations of type-specific operators of indexes have to be provided by the user. In this way, existing index structures were tailored to fit new purposes, and thus have been called extended secondary indexes (see DB2 UDB object-relational DBMS¹²).

This evolution responded to the need of providing flexibility to the logic level adapting the physical level in consequence. The idea was to approach the three levels by offering ad hoc query facilities, and let applications define the way they could navigate the objects collections, for instance, a graphical browser could be sufficient to fulfill this functionality [ABD⁺89]. This facility could be supported by the data manipulation language or a subset of it.

As the architecture of the DBMS evolved according to the emergence of new programming paradigms like components and services, and to "new" data models like documents (XML) the frontiers among the three levels started to be thinner; and transparency concerning persistency and transaction management was less important. Component oriented middleware started to provide persistence services and transaction monitors

12. <http://www-01.ibm.com/software/data/db2/>

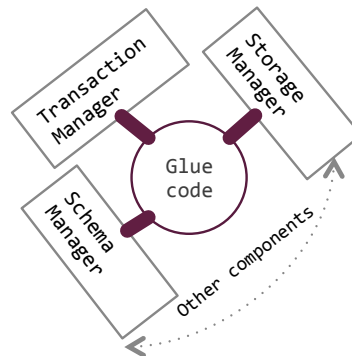


Figure 2.7: Extensible DBMS [DG00]

as services that required programmers to configure and integrate these properties within the applications. Data and program independence was broken but the ad-hoc configuration of data management components or services seemed to be easier to configure since it was more performant to personalize functions according to application needs.

2.2.2 Configuring and unbundling data management

Configurable DBMSs rely on unbundled DBMS tasks that can be mixed and matched to obtain database support (see Figure 2.7). The difference lies in the possibility of adapting service implementations to new requirements or in defining new services whenever needed. Configurable DBMSs also consider services as unbundled representations of DBMS tasks. However, the models underlying the various services, and defining the semantics of the corresponding DBMS parts can now, in addition, be customized. Components for the same DBMS task can vary not only in their implementations for the same standardized interface, but also in their interfaces for the same task. DBMS implementors select (or construct new) components implementing the desired functionality, and obtain a DBMS by assembling the selected components. There are different approaches for configuring and composing unbundled DBMS services: kernel systems, customizable systems, transformational systems, toolkits, generators and frameworks [DG00].

In principle, (internal) DBMS components are programmed and exchanged to achieve specific functionality in a different way than in the original system. A crucial element is the underlying architecture of the kernel, and the proper definition of points where exchanges can be performed. Examples of this kind of DBMS are Starburst [HCL⁺90, LMP87]. Its query language can be extended by new operators on relations [HFLP89]. Various phases of query processing in Starburst are also customizable. Functions are implemented using the interfaces of a lower layer (kernel) sometimes using a dedicated language. GENESIS [BBG⁺88, Bat88] is a transformational approach that supports the implementation of data models as a sequence of layers. The interface of each layer defines its notions of files, records, and links between files. Transformations themselves are collected in libraries, so that they can be reused for future layer implementations. Another transformational approach that uses specification constructs similar to those of Acta [CR94] has been described by [GHKM94]. EXODUS [CDF⁺91] applies the idea of a toolkit for specific parts of the DBMS. A library is provided for access methods. While the library initially contains type-independent access methods such as B-trees, grid files, and linear hashing, it can also be extended with new methods. Other examples are the Open OODB (Open Object-Oriented Database) approach [Bla94, WBT92], Trent

[KSWU92] for the construction of transaction managers (mainly, transaction structures and concurrency control) and *A la carte* [DKH92] for the construction of heterogeneous DBMSs.

One problem in any toolkit approach is the consistency (or compatibility) of reused components. Generation approaches instead, support the specification of (parts of) a DBMS functionality and the generation of DBMS components based on those specifications. A programmer defines a model (e.g., an optimizer, a data model, or a transaction model), which is input to a generator. The generator then automatically creates a software component that implements the specified model based on some implementation base (e.g., a storage manager or kernel in the case of data-model software generation). An example of a generator system is the EXODUS query-optimizer generator [GD87]. Volcano [GM93], the successor of the EXODUS optimizer generator, also falls into the group of generator systems. Volcano has been used to build the optimizer for Open OODB [Bla94].

Systems like KIDS [GSD97], Navajo, and Objectivity [Guz00] provide a modular, component-based implementation. For example, the transaction manager (or any other component) can be exchanged with the ripple effect mainly limited to the glue code. However, in order to strike the right balance between modularity and efficiency the design of the individual components is not highly modular. In fact, the modularity of the individual components is compromised to preserve both modularity and efficiency of the DBMS. Approaches like NODS [Col00] proposed services oriented networked systems at various granularities that cooperated at the middleware level. The NODS services could be customized on a per-application basis at a fine grained level. For example, persistency could be configured at different levels [GBDC03] memory, cache or disk and it could cooperate with fault tolerance protocols for providing, for example, different levels of atomic persistent data management. Other frameworks for building query optimizers are the ones described in [ÖMS95], Cascades [Gra95], and EROC (Extensible Reusable Optimization Components) [MBHT96] and [CV04, VC04]. Framboise [FGD98] and ODAS [CVSGR00, VSCGR00] are frameworks for layering active database functionality on top of passive DBMSs.

However, customizability at a finer granularity (i.e., the components forming the DBMS) is expensive. Such customization is cost-effective if changes at the fine granularity were localized without compromising the system performance obtained through closely woven components, i.e., both modularity and efficiency need to be preserved.

2.2.3 NoSQL data store managers

New kinds of data with specific structures (e.g., documents, graphs) produced by sensors, Global Positioning Systems (GPS), automated trackers and monitoring systems has to be manipulated, analyzed, and archived [Tiw11]. These large volumes of data sets impose new challenges and opportunities around storage, analysis, and archival. NoSQL stores seem to be appropriate models that claim to be simpler, faster, and more reliable. This means the traditional data management techniques around upfront schema definition and relational references is being questioned.

Even if there is no standard definition of what NoSQL means¹³ there are common characteristics of these systems: (i) they do not rely on the relational model, and do not use the SQL language; (ii) they tend to run on cluster architectures; (iii) they do not have a fixed schema, allowing to store data in any record. The systems that fall under the NoSQL umbrella are quite varied, each with their unique sets of features and value propositions. Examples include MongoDB, CouchDB, Cassandra, Hbase, and also BigTable and SimpleDB which are tied to cloud services of their providers, they fit in general operating characteristics.

13. The notion was introduced in a workshop in 2009 according to [MP12].

Despite the profound differences among the different NoSQL systems, the common characteristic with respect to the architecture is that the external and logic levels of RDBMS disappear. This means that the applications are close to the physical level with very few independence program and data. Data processing functions like querying, aggregating, analyzing are conceived for ensuring efficiency. For example, Google's Bigtable adopts a a column-oriented data model avoiding consuming space when storing nulls by simply not storing a column when a value does not exist for that column. Columns are capable of storing any data types as far as the data can be persisted in the form of an array of bytes. The sorted ordered structure makes data seek by row-key extremely efficient. Data access is less random and ad-hoc, and lookup is as simple as finding the node in the sequence that holds the data. Data is inserted at the end of the list.

Another evidence of the proximity to the physical model exploited by NoSQL systems are key-value stores that exploit hashMap (associative array) for holding key-value pairs. The structure is popular because thereby stores provide a very efficient $O(1)$ average algorithm running time for accessing data. The key of a key-value pair is a unique value in the set and can be easily looked up to access the data. Key-value pairs are of varied types: some keep the data in memory, and some provide the capability to persist the data to disk. The underlying data storage architecture is in general a cluster, and the execution model of data processing functions is Map-Reduce. Thus data are in general managed in cache the memcached protocol¹⁴ being a popular for example in key-value stores. A cache provides an in-memory snapshot of the most-used data in an application. The purpose of cache is to reduce disk I/O.

In some situations, availability cannot be compromised, and the system is so distributed that partition tolerance is required. In such cases, it may be possible to compromise strong consistency. The counterpart of strong consistency is weak consistency. Inconsistent data is probably not a choice for any serious system that allows any form of data updates but eventual consistency could be an option. Eventual consistency alludes to the fact that after an update all nodes in the cluster see the same state eventually. If the eventuality can be defined within certain limits, then the eventual consistency model could work. The term BASE (Basically Available Soft-state Eventually) [Vog09] denotes the case of eventual consistency¹⁵.

NoSQL systems promote performance, scalable, clustered oriented data management and schemaless data design focussing on data distribution, duplication and on demand persistency. The logic and external levels of the classic DBMS architecture are erased exposing the physical level to applications with certain transparency. The application describes its data structures that can be persistent, and that can be retrieved using indexing mechanisms well adapted to these structures. Assuming good amounts of available memory resources, they promote parallel data querying including data processing tasks. Data availability is ensured through replication techniques and persistency on second memory is done on demand. For the time being, given that data management is done at the physical level, and that there is few data - program independence, there is a lot of programming burden to be undertaken by application programmers. Most NoSQL systems do not support high level query languages with built-in query optimization. Instead, they expect the application programmer to worry about optimizing the execution of their data access calls with joins or similar operations having to be implemented in the application [Moh13]. Another drawback mainly associated to the historical moment is that application programming interfaces (APIs) are not yet standardized, thus standardized

14. According to the Wikipedia memcached is a general-purpose distributed memory caching approach that was originally developed by Danga Interactive <http://www.memcached.org>. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read.

15. Eventual consistency is a consistency model used in distributed computing that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value. Eventual consistency is purely a liveness guarantee (reads eventually return the same value) and does not make safety guarantees: an eventually consistent system can return any value before it converges (Wikipedia).

bindings are missing, and they have to be programmed and maintained.

NoSQL overcome some of the shortcomings of the relational systems but leave aside good principles of the RDBMS, which go beyond the relational model and the SQL language. The schemaless approach seems to respond to a schema evolution requirement stemming from applications dealing with data in a very simple way (read/write operations). As discussed in [Moh13], Web data like logs of activities in an e-commerce site or data managed by social media applications like Facebook are examples of cases needing schema evolutions because data is not very structured and, even when it is structured, the structure changes a lot over time.

2.3 Discussion

In order to face challenges introduced by applications requirements evolution the database community came up with new ways of delivering the system's internal and external functions to applications: as components (by the end of the 90's), as peer to peer networks (in the beginning of the 2000's), and as services based database management systems the last 3 or 5 years. These new ways of delivering data management functions is done under different architectures: centralized, distributed, parallel and on cloud. Services are deployed on different system/hardware architectures: client - server (on classic and embedded systems), distributed on the grid, on P2P networks, on the W2.0, and recently on the clouds.

From our high-level view of the architecture of DBMSs we can conclude that although they make efficient use of the resources of their underlying environment, they are fixed to that environment as well. In a service-oriented environment where various hardware and software platforms are hidden behind service interfaces, that kind of control is unreachable. Furthermore, for several applications in dynamic environments ACID transactions may not be feasible or required. In addition, DBMSs are not easily portable and often impose a large footprint. A related problem is that they are difficult to evolve and maintain. For these reasons, several researchers have concluded that they in fact exhibit underperformance [SC05] or are even inappropriate [Kos08] for a variety of new applications.

Consequently, the core functionality of the DBMS must be adapted for new settings and applications, among which we are particularly interested in dynamic and service-oriented environments. Services allow dynamic binding in order to accomplish complex tasks for a particular client. Moreover, services are reusable, and have high autonomy due to the fact that they are accessed only through well defined interfaces. Organizing services on layers is a solution to composing a large numbers of services, and will help in making decisions about their granularity. This allows to reuse optimized functionality that is shared by several applications instead of having to invest efforts on the implementation of the same functionality again. Another way of extending the system is by invoking internal services through calls from external web services or web servers. Users can thereby have their own tailored services on their personal computers to replace or extend existing SBDBM services according to their needs.

Developers of new applications can particularly benefit from service reuse by taking one or more services that run on the SBDBMS, from any available layer, and integrate these services into their application to provide optimized access to their application-specific data. For example, assume that an application needs access to the storage level of the DBMS in order to obtain statistical information, such as available storage space or data fragmentation. In this case, the developer of this application can add the necessary information services to the storage level. This way, she provides the information source required for her application. Then, the application has to just invoke these services to retrieve the data. Furthermore, other services from other layers can be used together with this kind of extension services if required. Services can be distributed, and be made redundant by using several computers connected through a network. Therefore, a SBDBMS

can be customized to use services from other specific locations to optimize particular tasks. This approach introduces a high degree of adaptability into the database system. Priorities can be assigned to services that demand a considerable amount of resources, thereby enabling Quality of Service agreements (QoS) for special data types like multimedia and streaming data.

These challenges imply the construction of services based middleware with two open problems:

1. Exploit available resources making a compromise between QoS properties and Service Level Agreements (SLA) considering all the levels of the stack (i.e., from the network (infrastructure) to the application level).
2. Optimally coordinate services considering applications'/users' characteristics for fulfilling their requirements.

CHAPTER 3

Coordinating services for implementing hybrid queries

Every science consists in the coordination of facts; if the different observations were entirely isolated, there would be no science.

— Auguste Comte.

Pervasive denotes something "spreading throughout", thus a pervasive computing environment is the one that is spread throughout anytime anywhere and at any moment. From the computing science point of view what is interesting to analyze is how computing and software resources are available and provide services that can be accessed by different devices. For facilitating availability to these resources, they are wrapped under the same representation called service. A service is a resource handled by a provider and that exports an application programming interface (API) that defines a set of method headers using an interface definition language. Consider a scenario where multiple users find themselves in an urban area carrying GPS-enabled mobile devices that periodically transmit their location. For instance, the users location is made available by a stream data service with the (simplified) interface:

```
subscribe() → {location:<nickname, coor>}
```

consisting of a subscription operation that, after invocation, will produce a stream of `location` tuples, each with a nickname that identifies the user and her coordinates. A stream is a continuous (and possibly infinite) sequence of tuples ordered in time.

The rest of the data is produced by the next two on-demand data services, each represented by a single operation:

```
profile(nickname) → {person:<age, gender, email>}  
interests(nickname) → {s_tag:<tag, score>}
```

The first provides a single `person` tuple denoting a profile of the user, once given a request represented by her nickname. The second produces, given the nickname as well, a list of `s_tag` tuples, each with a tag or keyword denoting a particular interest of the user (e.g. music, sports, fashion, etc.) and a score indicating the corresponding degree of interest.

Users access available services for answering some requirement expressed as a query. For instance, assume that Bob needs to find friends to make decisions whether he can meet somebody downtown to attend an art exposition. The query can be the following:

Find friends who are no more than 3 km away from me,
who are over 21 years old and that are interested in art

But issuing the query from a mobile device, is not enough for evaluating it, some Service Level Agreements (SLA) need to also be expressed. For example, Bob wants the query to be executed as soon as possible, minimizing the battery consumption and preferring free data services. Of course, the query cannot be solved by one service, some information will come for Google maps and Google location, other by Bob's personal directory, the availability of Bob's friend in their public agendas. Thus, the invocation to the different services must be coordinated by a program or script that will then be executed by an execution service that can be deployed locally on the user device or not. In order to do so, other key infrastructure services play an important role particularly for fulfilling SLA requirements. The communication service is maybe the most important one because it will make decisions on the data and invocation transmission strategies that will impact SLA.

Focussing on the infrastructure that makes it possible to execute the services coordination by making decisions on the best way to execute it according to given SLAs, we identify two main challenges:

- Enable the reliable coordination of services (infrastructure, platform and, data management) for answering queries.
- Deliver request results in an inexpensive, reliable, and efficient manner despite the devices, resources availability and the volume of data transmitted and processed.

Research on query processing is still promising given the explosion of huge amounts of data largely distributed and produced by different means (sensors, devices, networks, analysis processes), and the requirements to query them to have the right information, at the right place, at the right moment. This challenge implies composing services available in dynamic environments and integrating this notion into query processing techniques. Existing techniques do not tackle at the same time classic, mobile and continuous queries by composing services that are (push/pull, static and nomad) data providers.

Our research addresses novel challenges on data/services querying that go beyond existing results for efficiently exploiting data stemming from many different sources in dynamic environments. Coupling together services, data and streams with query processing considering dynamic environments and SLA issues is an important challenge in the database community that is partially addressed by some works. Having studied the problem in a general perspective led to the identification of theoretical and technical problems and to important and original contributions described as follows. Section 3.1 describes the phases of hybrid query evaluation, highlighting an algorithm that we propose for generating query workflows that implement hybrid queries expressed in the language HSQL that we proposed. Section 3.2 describes the optimization of hybrid queries based on service level agreement (SLA) contracts. Section 3.3 introduced the hybrid query evaluator HYPATIA, its general architecture, implementation issues and validation. Section 3.4 discusses related work and puts in perspective our work with existing approaches.

3.1 Hybrid query evaluation

We consider queries issued against data services, i.e., services that make it possible to access different kinds of data. Several kinds of useful information can be obtained by evaluating queries over data services. In turn, the evaluation of these queries depends on our ability to perform various data processing tasks. For example, data correlation (e.g. relate the profile of a user with his/her interests) or data filtering (e.g. select users above a certain age). We must also take into consideration restrictions on the data, such as temporality (e.g. users logged-in within the last 60 minutes).

We denote by "hybrid queries" our vision of queries over dynamic environments, i.e. queries that can be mobile, continuous and evaluated on top of push/pull static or nomad services. For example, in a mobile application scenario, a user, say Mike, may want to *Find friends who are no more than 3 km away from him, who are over 21 years old and that are interested in art*. This query involves three data services methods defined above. It is highly desirable that such query can be expressed formally through a declarative language named Hybrid Service Query Language (HSQL)¹. With this goal in mind we adopt a SQL-like query language which is similar to CQL[ABW06], to express the query as follows:

Example 3.1.

```
SELECT p.nickname, p.age, p.sex, p.email
FROM profile p, location l [range 10 min], interests i
WHERE p.age >= 21 AND l.nickname = p.nickname AND
i.nickname = p.nickname AND 'art' in i.s_tag.tag
AND dist(l.coor, mycoor) <= 3000
```

The conditions in the `WHERE` clause enable to correlate `profile`, `location`, and `interests` of the users by their nickname, effectively specifying join operations between them. Additional conditions are specified to filter the data. Thus, users who are older than 21 and whose list of interests includes the tag `'art'`, and whose location lies within the specified limit are selected. For the location condition, we rely on a special function `dist` to evaluate the distance between two geographic points corresponding to the location of users, the current location of the user issuing the query is specified as `mycoor`. Since a list of scored tags is used to represent the interests of an user, we use a special `in` operator to determine if the tag `'art'` is contained in the list, while the list in question is accessed via a path expression.

Since the location of the users is subject to change and delivered as a continuous stream, it is neither feasible nor desirable to process all of the location data, therefore temporal constraints must be added. Consequently, the `location` stream in the `FROM` clause is bounded by a time-based window which will consider only the data received within the last 10 minutes. Given that the query is continuous, this result will be updated as the users' location changes and new data arrives. This is facilitated by a special `sign` attribute added to each tuple of the result stream, which denotes whether the tuple is added to the result (positive sign) or removed from it (negative sign).

In order to evaluate a declarative hybrid query like the one presented in Example 3.1 we need to derive an executable representation of it. Such executable representation in our approach is a query workflow.

3.1.1 Query workflow

A workflow fundamentally enforces a certain order among various activities as required to carry out a particular task. The activities required to evaluate a hybrid query fall into two basic categories: data access and data processing. Both of these types of activities are organized in a workflow following a logical order determined by the query. The execution of each of the activities, in turn, is supported by a corresponding service; data access activities by data services and data processing activities by computation services.

Following our service-based approach, the workflow used to evaluate a hybrid query consists of the parallel and sequential coordination of data and computation services. For example, the workflow representation for the query in Example 3.1 is depicted in Figure 3.1.

1. This language was proposed in the PhD dissertation of V. Cuevas Vicenttin of University of Grenoble.

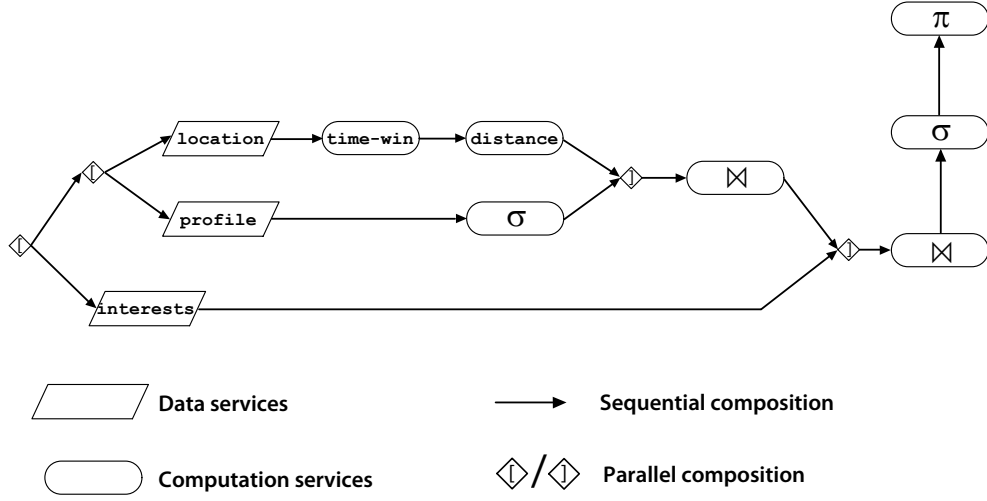


Figure 3.1: Query coordination for the query in Example 3.1

The data services are represented by parallelograms, whereas computation services are represented as rounded rectangles and correspond to traditional query operators such as join or selection. The arrows indicating sequential composition not only imply order dependencies among the activities but also data dependencies; in particular, tuples that need to be transmitted between the different activities that produce and consume them.

Since the workflow enabling the evaluation of a given hybrid query acts in fact as a service coordination (comprising data and computation services), we refer to it as *query workflow*. We discuss the generation of a query workflow from a given declarative query in the following section.

Evaluating a hybrid query from a given query coordination depends first on finding the adequate (data and computation) services, second on their invocation, and finally on their communication and interoperability. We deal with these aspects next.

3.1.2 Generating a query workflow

Given an HSQL (Hybrid Service Query Language) query, it is necessary to generate a corresponding workflow specifying a service coordination to enable its evaluation. Two important characteristics of a query workflow are that it reflects the logic of the query and that it is executable, (i.e., *feasibility*). The feasibility property implies that the data dependencies of on-demand data services are satisfied, i.e., it is possible to obtain the required input parameters to invoke the data services methods. The input parameters can only originate from constant values in the query, data stream tuples, or tuples retrieved from other on-demand services. When constants are not involved we essentially perform an operation known as bind-join described in [FLMS99], which is analogous to a normal join but involves retrieving tuples as required by a *binding pattern*, which annotates the attributes of a data source as either *bound* (for input attributes) or *free* (for output attributes).

Thus a primary task to carry out in order to build a query workflow is to determine the appropriate joins. We propose an algorithm for this purpose that is based on the Graham-Yu-Ozsoyoglu (GYO) algorithm in [Yan81] used in database theory to determine if a relational query is acyclic. We extended this algorithm to take into consideration the binding patterns associated with on-demand data services. Our algorithm

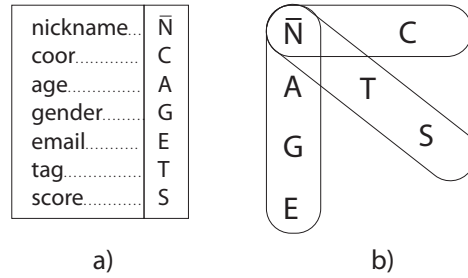


Figure 3.2: Attribute symbols and Hypergraph

consists of three main phases:

1. Represent the join dependencies by a hypergraph
2. Process the hypergraph to yield a parse tree denoting the valid join orders
3. Derive a join order by traversing the tree and then add the remaining operators

Join dependencies as a hypergraph Consider the following representation of the data services interfaces involved in our example query using the notion of binding patterns. By default all of the attributes of a stream data service like the location service are considered free.

```
location:⟨nicknamef, coordf⟩
profile:⟨nicknameb, agef, genderf, emailf⟩
interests:⟨nicknameb, tagf, scoref⟩
```

These services are represented in the query by the aliases **l**, **p**, and **i**, respectively.

Let us now consider the equality conditions in the **WHERE** clause of the query between the attributes of the different data operations.

```
l.nickname = p.nickname AND
i.nickname = p.nickname
```

Each of these two conditions implies a bidirectional join dependency between two data operations, and which further propagate transitively (in this case via **p** at the right) as occurs in traditional queries. Thus a join dependency among the three data services arises, in particular, based on the common **nickname** attribute.

In the general case, the join dependencies between several data service operations can be represented by a *hypergraph*, formed by nodes and hyperedges; the latter which unlike ordinary graph edges, can be sets of any number of nodes.

To construct the hypergraph, the attributes from the data operations are first given a symbol that corresponds to a node in the hypergraph. The attributes that take part in a chain of joins are associated with symbols distinguished by a bar (e.g. \bar{N}), the same symbol being used for all of the attributes in the join chain. The symbols given to the attributes of the data service methods from the example are presented in Figure 3.2 a), which when applied to the binding patterns associated with the same data service methods yield

```
location:⟨ $\bar{N}$ f, Cf⟩
profile:⟨ $\bar{N}$ b, Af, Gf, Ef⟩
```


interests: $\langle \bar{N}^b, T^f, S^f \rangle$

The resulting hypergraph is depicted in Figure 3.2 b. As the hypergraph shows, a join dependency exists on the **nickname** attribute of the methods' interfaces.

Parse tree construction Once a hypergraph has been built, it can be used in step 2 to obtain, by a reduction process, a parse tree which denotes the valid join orders. Performing a reduction of a query hypergraph essentially corresponds to removing nodes and hyperedges down to a single hyperedge, which is a sufficient and necessary condition for the query to be acyclic as discussed in [Yan81]. The reduction of a hybrid query hypergraph and the parse tree construction proceed as follows.

We identify an *ear* H in the hypergraph, which represents a hyperedge whose nodes can be divided into two sets: *isolated* nodes that appear in H and no other hyperedge, and *shared* nodes that are contained in another hyperedge G . If multiple ears exist, one is selected arbitrarily. For example, in the hypergraph of Figure 3.2 b), we identify $\bar{N}C$ as an ear, whose attributes are divided into the sets $\{C\}$ and $\{\bar{N}\}$ of isolated and shared nodes, respectively.

The isolated nodes are entirely removed, while the shared nodes remain in the hypergraph but not as part of the ear, which is removed. The hyperedge G that shares a subset of nodes with the ear H is said to *consume* the ear. For example, the $\bar{N}C$ ear denoted by a dashed line in Figure 3.3 a) is consumed by the hyperedge $\bar{N}AGE$, resulting in the hypergraph presented in b) of the same figure. Note that node C is removed but node \bar{N} is kept.

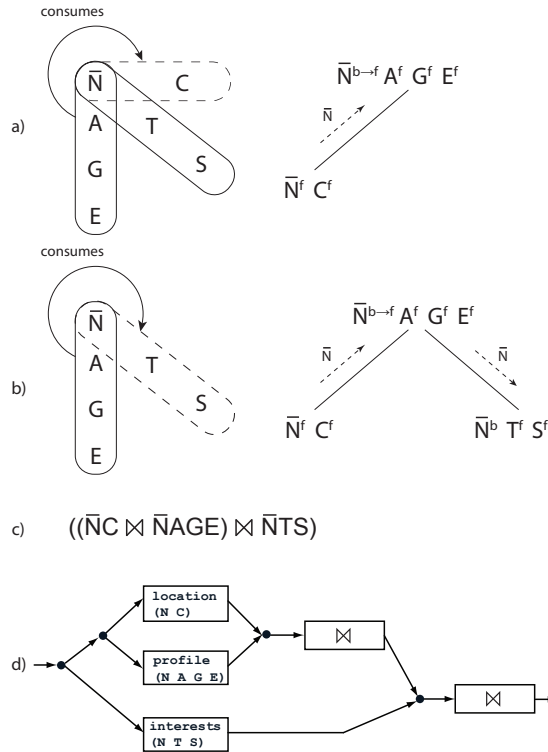


Figure 3.3: Hypergraph reduction and parse tree construction

Whenever a hyperedge G consumes an ear H we add a subtree to the parse tree of the hypergraph, with

the consumer G ($\bar{N}AGE$ in the example) as parent of the ear H ($\bar{N}C$ in the example), as shown in Figure 3.3 a). The reduction process proceeds in this manner until a single hyperedge remains, and a full parse tree is built, as shown in Figure 3.3 b) after $\bar{N}AGE$ also consumes $\bar{N}TS$.

An additional condition for a given hyperedge to consume an ear is that by consuming the ear the consumer hyperedge can obtain *all* of the bound attributes needed (if any) to access the data operation the consumer hyperedge represents.

Parse tree traversal The last step consists in deriving a join order by traversing the parse tree and then adding the remaining operators. A query workflow which satisfies the binding patterns of the data operations is generated by traversing the parse tree in a bottom up manner. Considering the parse tree in Figure 3.3 b), we first visit the $\bar{N}C$ node, then the $\bar{N}AGE$ node, and finally the $\bar{N}TS$ node. When we visit $\bar{N}C$ we obtain \bar{N} and can therefore use it to access the **profile** operation represented by $\bar{N}AGE$. For this reason, \bar{N} changes from bound to free in $\bar{N}AGE$ and can thus be used to access the **interests** operation represented by $\bar{N}TS$. Such dependencies determine if a simple equi-join or a bind-join should be performed. The first applies when all of the attributes are available, while the latter, to cases in which an operation needs to be invoked with input parameters as specified by the binding pattern.

The result of the parse tree traversal is an infix expression consisting of binary joins, as illustrated in Figure 3.3 c) for our example. Based on this infix expression, we can build a join workflow that satisfies all of the join dependencies present in the query. For instance, the join workflow for our example is presented in Figure 3.3 d).

Regarding the remaining operators, windows that bound the streams always come after the stream data services and are easily identifiable in the **FROM** clause of the query. Selections and projections, on the other hand, are added to the query workflow following optimization heuristics that determine their appropriate place, which however must also respect the binding patterns. For example, the selection for a particular tag value can only be applied after the respective data is obtained from a bind join.

3.1.3 Computing a query workflow cost

In order to use SLA's to guide query workflows evaluation, it is necessary to propose a cost model that can be used to evaluate a query workflow cost. The query workflow cost is given by a combination of QoS measures associated to the service methods it calls², infrastructure services such as the network and the hosting device it uses. The cost model considered for query workflows is defined by a combination of three costs: execution time, monetary cost and battery consumption. These costs are computed by calculating the cost of activities of a query workflow.

Query workflow cost is computed by aggregating its activities costs based on its structure. The aggregation is done by following a systematic reduction of the query workflow such as in [Jae96, WCSO08]. For each sequential or parallel coordination, the reduction aggregates the activities costs. For the nested coordinations, the algorithm is applied recursively. The resulting cost is computed by a pondered average function of the three values.

$$\text{Cost} = \alpha (\text{temporal}_c) + \beta (\text{economic}_c) + \gamma (\text{energy}_c) / 3$$

². QoS measures are a set of quantitative measures that describe the possible conditions in which a service method invocation is executed.

We assume that the activity costs are estimated according to the way data are produced by the service (i.e., batch for on-demand services, continuous by continuous services).

Cost of an activity calling an on-demand service For data produced in batch by on-demand services, the global activity cost is defined by the combination of three costs:

- *Temporal cost* given by (i) the speed of the network that yields transfer time consumption, determined by the data size and the network's conditions (i.e., latency and throughput) both for sending the invocation with its input data and receiving results; (ii) the execution of the invoked method has an associated approximated method response time which depends on the method throughput³.
- *Economic cost* given by (i) the type of network: indeed, transmitting data can add a monetary cost (e.g., 3G cost for mega octets transfer); (ii) the cost of receiving results from a service method invocation sent to a specific service provider, for example getting the scheduled activities from the public agenda of my Friends can have a cost related to a subscription fee.
- *Energy cost* produced as a result of using the network and computing resources in the device hosting the service provider that will execute the method called. These operations consume battery entailing an energy cost.

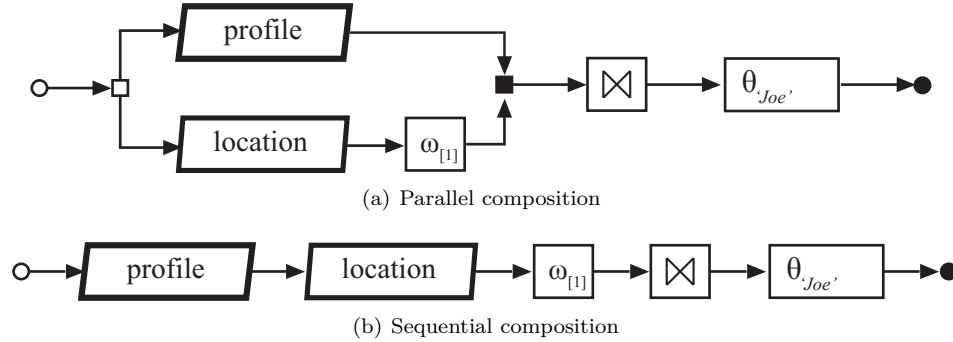
Cost of an activity calling a continuous service cost For data produced by continuous services the global activity cost is defined by the combination of three costs that depend on the data production rate resulting from the invocation of a method. The costs are multiplied by the number of times data must be pulled and transferred. The economic cost can be associated to a subscription model where the cost is determined by the production rate. For example receiving data frequently (e.g., give my current position every five minutes, where five minutes is the expected production rate) can be more expensive than receiving data in specific moments (e.g., the number of times Bob went to the supermarket during a month). The temporal and energy costs are also determined by the frequency in which data are processed (processing rate): data can be processed immediately, after a threshold defined by the number of tuples received, or the elapsed time, or a buffer capacity. Both production rate and processing rate impact the execution time cost, execution economic cost, and battery consumption cost.

3.2 Optimizing hybrid queries using SLA contracts

Given a hybrid query and a set of services that can be used for answering it, several query workflows can be used for implementing it. For example, consider the query workflow in Figure 3.4a that is a version of the friend finder example. The figure shows a query workflow coordinating activities in parallel for retrieving the profile and the location datasets. Then, the filtering activity that implements a window operator is placed just after the activity that retrieves the location. This activity reduces the input dataset size. Then, both datasets are correlated and finally the last activity filters the dataset to get data related only to 'Joe'. Placing the last activity just after of the retrieval of the profile dataset can reduce the processing time.

Now consider the query workflow in Figure 3.4b that coordinates activities sequentially. Each activity in the control flow consumes and produces data that at the end result in a dataset which is equivalent with the first one.

3. The method throughput is given by the amount of requests in a period of time (e.g., each minute) and the state of the device such as memory or CPU.

Figure 3.4: Query workflows of the *Search space*.

Optimizing a hybrid query implies choosing the query workflow that best implements it with respect to a given SLA. Similar to classic query optimization techniques, we propose an optimization process that consists in two phases: (i) generating its search space consisting in "all" the query workflows that implement a hybrid query and (ii) choosing the top-k query workflows with costs that are the closest to the SLA.

3.2.1 Generating potential query workflow space

We use rewriting operations (e.g. split, aggregate, parallelize, etc.) for generating a set of "semantically" equivalent query workflows. The rewriting process is based on two notions: function and data dependency relationships.

- Function: represents a data processing operation. We consider the following functions:
 1. **fetch** for retrieving a dataset from a data provision service (e.g., get Bob's friends);
 2. **projection** of some of the attributes of each item (e.g., tuple) of a dataset (e.g., get name and location of Bob's friends assuming that there each friend has other attributes);
 3. **filter** the items of a dataset according to some criterion (e.g. Alice's friends located 3 Km from her current position); and,
 4. **correlation** of the items of two datasets according to some criterion (e.g., get friends shared by Bob and Alice that like "Art").
- Data dependency relationships between functions. Intuitively, given two functions with input parameters and an output of specific types, they are
 1. F_1 **independent** F_2 if they do not share input datasets;
 2. F_1 **concurrent** F_2 if they share common input datasets;
 3. F_1 **dependent** F_2 if they use common input datasets.

We propose rewriting rules and algorithms for generating a representation of an HSQL expression as a composite function and a data dependency tree. This intermediate representation is used for finally generating a query workflow search space. This generation is based on composition patterns that we propose for specifying how to compose the activities of a query workflow. Let F_1 and F_2 be functions of any type according to their dependency relationship they can give rise to two activities A_1 and A_2 related according to the composition patterns shown in Figure 3.5.

1. F_1 **independent** F_2 leads to three possible composition patterns: A_1 **sequence** A_2 or A_1 **sequence** A_2 or A_1 **parallel** A_2 .

Relation	Notation	Composition pattern
A Independent of B	$A B$	
A Concurrent with B	$A \bowtie B$	
A Dependent on B	$A \triangleright B$	

Figure 3.5: Composition patterns

2. F_1 concurrent F_2 leads to the same sequential patterns of the previous case. In the case of the parallel pattern, it works only if and only if F_1 and F_2 are filtering functions or one of them is a filtering function and the other a correlation function.
3. F_1 dependent F_2 leads to a sequential composition pattern A_1 sequence A_2 .

The search space generation algorithm ensures that the resulting query workflows are all deadlock free and that they terminate⁴. Once the search space has been generated, the query workflows are tagged with their associated three dimensional cost. Then, this space can be pruned in order to find the query workflows that best comply with the SLA expressing the preferences of the user. This is done applying a top-k algorithm as discussed in the following section.

3.2.2 Computing an optimization objective

In order to determine which are the query workflows that answer the query respecting the SLA contract, we compute an optimization objective taking as input the SLA preferences and assuming that we know all the potential data and computing services that can be used for computing a query workflow. The SLA expressed as a combination of pondered measures, namely, execution time, monetary cost and energy. Therefore we propose an equation to compute a threshold value that represents the lowest cost that a given query can have given a set of services available and required for executing it and independently of the form of the query workflow (see Equation 3.1).

$$Opt(Q, R) = \min\left(\sum_j (f(A_j, \Omega_j) - \gamma(Q, R_j))\right) \quad (3.1)$$

The objective is to find the combination of resources (R i.e., services) that satisfies a set of requirements (Q, i.e., the preferences expressed by the user and associated to a query). Every service participating in the execution of a query exports information about its available resources and used resources. For example the number of requests that a service can handle and the number of requests that are currently being processes.

4. Details on the algorithm can be found in the dissertation of Carlos Manuel López Enríquez of the University of Grenoble.

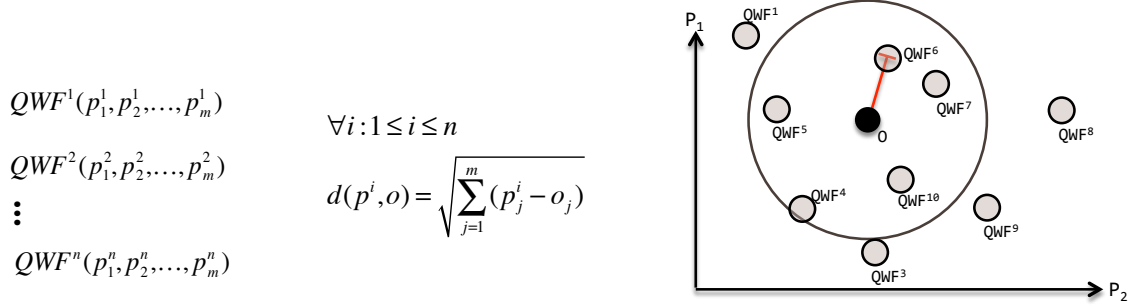


Figure 3.6: Optimization approach

The principle of the strategy is described as follows: determine to which extent the required resources by Q can be fulfilled by the resources provided by each service. The total result represents the combination of resources provided by available services that minimize the use of the global available resources (A) and the resources currently being used (Ω).

As shown in Figure 3.6 this value can be represented as a point in an n dimensional space where each dimension represents a SLA measure. Similarly, as shown in Figure 3.6, the query workflows cost which is also defined as a function of these dimensions, can be represented as a point in such n -dimensional space. The optimization process looks for points that are closest to the objective point by computing the Euclidean distance.

3.2.3 Choosing an optimum plan family

We adopt a top- k algorithm in order to decide which of the k query workflows that represent the best alternative to implement a hybrid query for a given SLA. We adopt the Fagin's algorithm [FLN03]. The top- k algorithm assumes m inverted lists L_1, \dots, L_m each of which is of the form $[\dots, (qw_i, c_{i,j}), \dots]$ with size $|\mathcal{S}|$ where $i \in [1 \dots |\mathcal{S}|]$, and $j \in [1 \dots m]$. The order of the inverted lists depends on the algorithm.

The Fagin algorithm assumes that each list L_j is ordered by $c_{i,j}$ in ascending order. The principle is that the k query workflows are close to the top of the m lists. In the worst case, the $c_{k,j}$ is the last item for some $j \in [1, \dots, m]$. The algorithm traverses in parallel the m lists by performing sequential access. Once k items have been visited in all the m lists, it performs random access over the m lists by looking for the already visited items and computes their scores. The scores are arranged in a sorted list in ascending order and thus the first k items are on the top of the score list. The main steps of the algorithm are:

1. Access in parallel the m lists by performing sequential access.
2. Stop once k query workflows have been seen in the m lists.
3. Perform random access over the m lists to obtain the m scaled attributes of each query workflow that has been processed and compute its *score*.
4. Sort in ascending order the scores.
5. Return the k query workflows on the top.

It is applied for obtaining an ordered family of query workflows that compose the optimum plan family. According to a descending order, the first query workflow will be executed. The rest of the queries can be

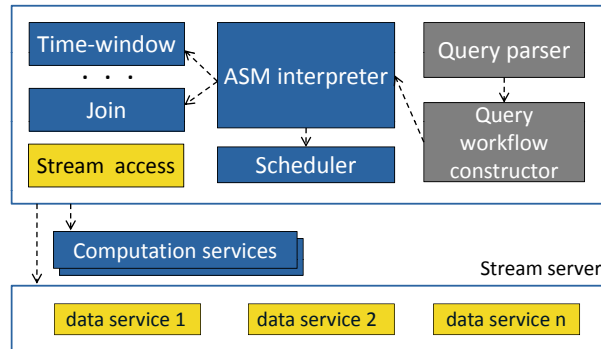


Figure 3.7: Architecture of HYPATIA

stored as knowledge and they can be used for further optimizations. We do not consider learning based optimization [MCBD12] but we believe that such a technique can be applied in this case.

3.3 HYPATIA: Service coordination query evaluator

We developed a proof of concept of our approach by implementing a service-based hybrid query processor named HYPATIA for enacting query workflows. Figure 3.7 presents its architecture. The system is based on the Java platform. Queries in HYPATIA are entered via a GUI (see Figure 3.8) and specified in our HSQL query language. Once a query is provided to the system it is parsed and then its corresponding query workflow is generated according to the algorithm that we described in Section 3.1.2. The query parser and the query workflow constructor components perform these tasks. The parser was developed using the ANTLR⁵ parser generator. The GUI also enables the user to visualize the query workflow, as well as the sub-workflows corresponding to composite computation services, which is facilitated by the use of the JGraph⁶ library.

To implement stream data services we developed a special-purpose stream server framework, which can be extended to create stream data services from sources ranging from text files to devices and network sources. This framework employs Web Service standards to create subscription services for the streams. Stream data access operators in query workflows subscribe to these services and also receive the stream via special gateway services.

The evaluation of a query is enabled by two main components that support the computation services corresponding to data processing operations. A scheduler determines which service is executed at a given time according to a predefined policy. Composite computation services communicate via asynchronous queues and are executed by an ASM interpreter that implements our workflow model.

3.3.1 Validation

We implemented two test scenarios and their corresponding data services to validate our approach. The first one, mainly a demonstration scenario, is the location-based application. In order to implement the Friend Finder scenario described in the Introduction we developed a test dataset using GPS tracks obtained from everytrail.com. Concretely, we downloaded 58 tracks corresponding to travel (either by walking, cycling, or driving) within the city of Paris. We converted the data from GPX to JSON and integrated it to our stream

5. <http://www.antlr.org/>

6. <http://www.jgraph.com/>

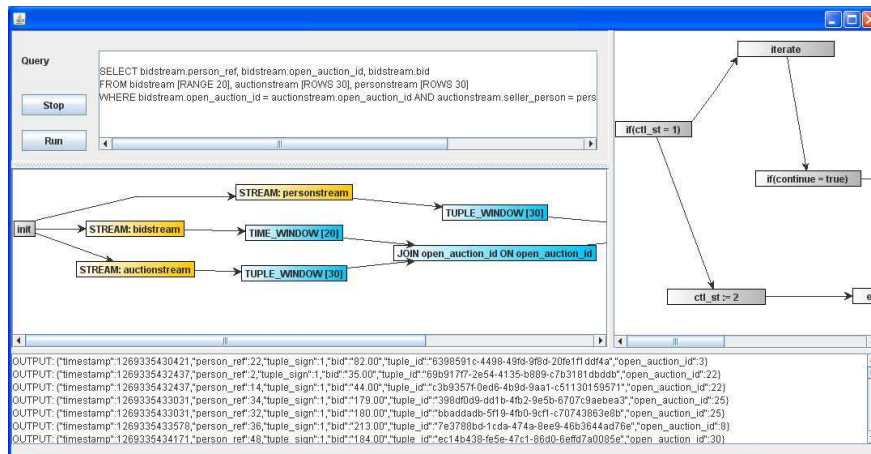


Figure 3.8: HYPATIA GUI

server to create the location service. For the profile and interests services we created a MySQL database accessible via JAX-WS Web Services running on Tomcat. The profile data is artificial and the interests were assigned and scored randomly using the most popular tags used in Flickr and Amazon. For the nearest-neighbor (NN) points of interest we converted a KML file⁷ containing the major tourist destinations in Paris into JSON, this data is employed by the corresponding NN service in conjunction with the R-tree spatial indexation service. Finally, we implemented an interface based on Google Maps that enables to visualize the query result, which is presented in Figure 3.9.

The second scenario was developed to measure the efficiency of our current implementation in a more precise manner; it is based on the NEXMark benchmark⁸. Our main goal was to measure the overhead of using services, so we measured the total latency (i.e. the total time required to process a tuple present in the result) for a set of six queries (see Table 3.1); first for our service-based system and then for an equivalent system that had at its disposal the same functionality offered by the computation services, but supported by objects inside the same Java Virtual Machine.

NEXMark proposes an auctions scenario consisting of three stream data services, **person**, **action** and, **bid**, that export the following interfaces:

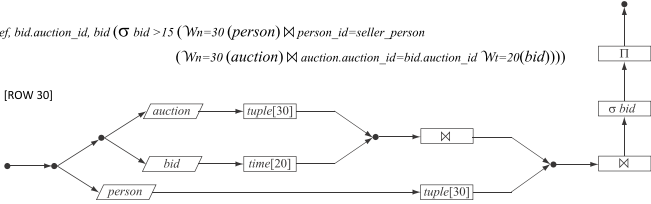
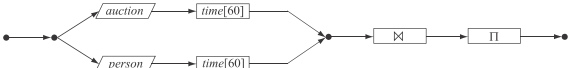
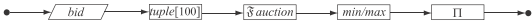
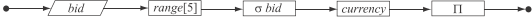
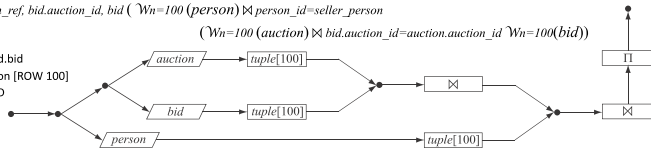
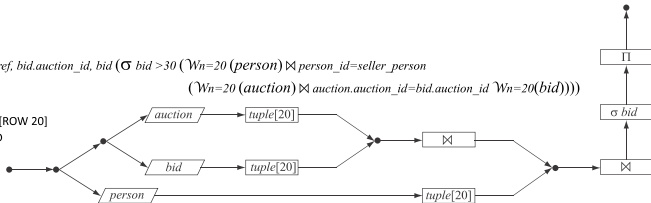
```
person: <person_idf, namef, phonef, emailf, incomef>
action: <open_auction_idf, seller_personf, categoryf, quantityf>
bid: <person_reff, open_auction_idf, bidf>
```

Auctions work as follows. People can propose and bid for products. Auctions and bids are produced continuously. Table 3.1 shows the six queries that we evaluated in our experiment; they are stated in our HSQL language and for each we provide the associated query workflow and equivalent operator expression that implements them (generated by HYPATIA). Queries $Q_1 - Q_2$ mainly exploit temporal filtering using window operators, filtering and correlation with and/split-join like control flows. Q_3 involves grouping and aggregation functions. Q_4 adds a service call to a sequence of data processing activities with filtering and projection operations. Finally, $Q_5 - Q_6$ address several correlations organized in and/split-join control flows.

7. Keyhole Markup Language <https://developers.google.com/kml/documentation/>

8. <http://datalab.cs.pdx.edu/niagara/NEXMark/>

Table 3.1: NEXMark queries

<p>- For the last 30 persons and 30 products offered, retrieve the bids of the last 20 seconds greater than 15 euros</p> <p>1)</p> <p>$\pi_{person_ref, bid, auction_id, bid} (\sigma_{bid > 15} (W_n=30(person) \bowtie person_id=seller_person$ $(W_n=30(auction) \bowtie auction.auction_id=bid.auction_id \ W_t=20(bid))))$</p> <pre>SELECT bids.person_ref, bid.auction_id, bid.bid FROM bid [RANGE 20], auction [ROW 30], person [ROW 30] WHERE bid.auction_id = auction.auction_id AND auction.seller_person = person.person_id AND bid.bid > 15;</pre> 
<p>- For the persons joining and the products offered during the last minute, generate the name and email of the person along with the id of the product he/she offers</p> <p>2)</p> <pre>SELECT P.name, P.email, A.auction_id FROM auction [RANGE 60] as A, person [RANGE 60] as P WHERE A.seller_person = P.person_id;</pre> <p>$\pi_{name, email, auction_id} (W_t=60(person) \bowtie person_id=seller_person \ W_t=60(auction))$</p> 
<p>- For the last 100 bids, find the maximum and minimum bid for each product</p> <p>3)</p> <pre>SELECT bid.person_ref, bid.auction_id, bid.bid, max(bid.bid), min(bid.bid) FROM bid [ROWS 100] GROUP BY bid.auction_id;</pre> <p>$\pi_{person_ref, auction_id, bid, max(bid), min(bid)} (\exists auction_id (W_t=60(auction)))$</p> 
<p>- Among the bids made in the last 5 seconds, find those whose amount is between 50 and 100 euros and their dollar equivalent</p> <p>4)</p> <pre>SELECT bid.person_ref, bid.auction_id, bid.bid, currency('EUR', 'USD', bid.bid) FROM bid [RANGE 5] WHERE bid.bid > 50.0 and bid.bid < 100.0;</pre> <p>$\pi_{person_ref, auction_id, bid, currency('EUR', 'USD', bid)} (\sigma_{bid > 50 \text{ and } bid < 100} (W_t=5(bid)))$</p> 
<p>- For the last 100 persons, products and bids; give the id of the seller person, the id of the product, and the amount of the bid</p> <p>5)</p> <p>$person_ref, bid.auction_id, bid (W_n=100(person) \bowtie person_id=seller_person$ $(W_n=100(auction) \bowtie bid.auction_id=auction.auction_id \ W_n=100(bid)))$</p> <pre>SELECT bid.person_ref, bid.open_auction_id, bid.bid FROM bid [ROW 100], auction [ROW 100], person [ROW 100] WHERE bid.auction_id = auction.auction_id AND auction.seller_person = person.person_id;</pre> 
<p>- For the last 20 persons, products and bids; give the id of the seller person, the id of the product, and the amount of the bid, whenever that amount is greater than 30</p> <p>6)</p> <p>$person_ref, bid.auction_id, bid (\sigma_{bid > 30} (W_n=20(person) \bowtie person_id=seller_person$ $(W_n=20(auction) \bowtie auction.auction_id=bid.auction_id \ W_n=20(bid))))$</p> <pre>SELECT bid.person_ref, bid.auction_id, bid.bid FROM bid [ROW 20], auction [ROW 20], person [ROW 20] WHERE bid.auction_id = auction.auction_id AND auction.seller_person = person.person_id AND bid.bid > 30;</pre> 

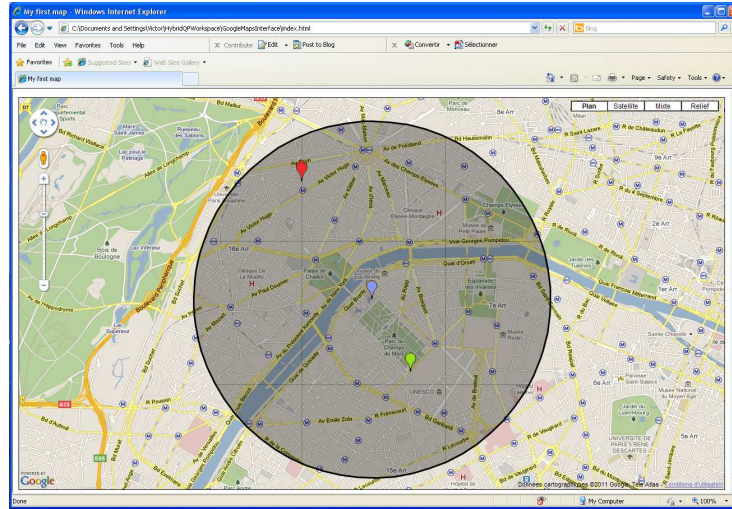


Figure 3.9: Friend Finder visualization GUI

3.3.2 Experimental results

For our experiments we used as a local machine a Dell D830 laptop with an Intel Core 2 Duo (2.10 GHz) processor and equipped with 2 GB of RAM. We also employed as a remote machine a Dell Desktop PC with a Pentium 4 (1.8 GHz) processor and 1 GB of RAM. In both cases running JSE 1.6.0_17, the local machine under Windows XP SP3 and the remote under Windows Server 2008.

As said before, to validate our approach we established a testbed of six queries based on our adaptation of the NEXMark benchmark. These queries include operators such as time and tuple based windows as well as joins. We measured tuple latency, i.e. the time elapsed from the arrival of a tuple to the instant it becomes part of the result, for three different settings. The first setting corresponds to a query processor using the same functionality of our computation services, but as plain java objects in the same virtual machine. In the second we used our computation services, which are based on the JAX-WS reference implementation, by making them run on a Tomcat container in the same machine as the query processor. For the third setting we ran the Tomcat container with the computation services on a different machine connected via intranet to the machine running the query processor.

The results are shown in Figure 3.10, and from them we derive two main conclusions. First, the use of services instead of shared memory resulted in about twice the latency. Second, the main overhead is due to the middleware and not to the network connection, since the results for the local Tomcat container and the remote Tomcat container are very similar. We believe that in this case the network costs are balanced-out by resource contingency on the query processor machine, when that machine also runs the container. We consider the overhead to be important but not invalidating for our approach, especially since in some cases we may be obliged to use services to acquire the required functionality.

From our experimental validation we learned that it is possible to implement query evaluation entirely relying on services without necessarily using a full-fledged DBMS or a DSMS (Data Stream Management Systems). Thereby, hybrid queries that retrieve on demand and stream data are processed by the same evaluator using well adapted operators according to their characteristics given our composition approach. The approach can seem costly because of the absence of a single DBMS, the use of a message based approach for implementing the workflow execution, and because there is no extensive optimization in the current version

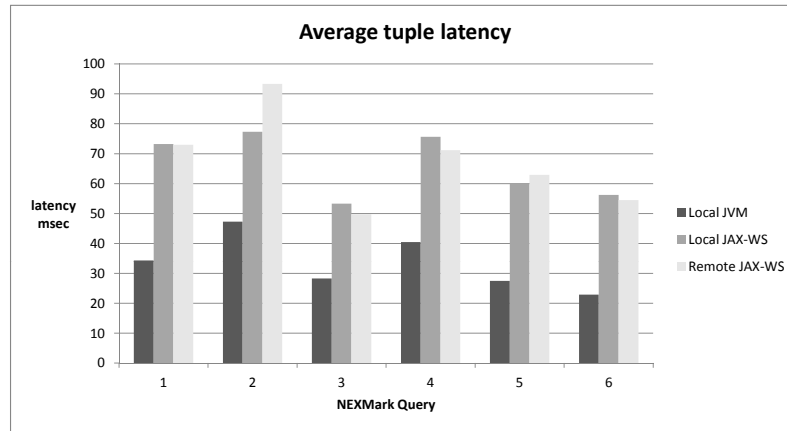


Figure 3.10: Tuple latency for a services-based vs. a single Java application query processor

of HYPATIA. Now that we have a successful implementation of our approach, we can address performance issues further in order to reduce cost and overhead.

3.4 Related work

In dynamic environments, query processing has to be continuously executed as services collect periodically new information (e.g., traffic service providing information at given intervals about the current state of the road) and the execution context may change (e.g., variability in the connection). Query processing should take into account not only new data events but also data providers (services) which may change from one location to another.

Existing techniques for handling continuous spatio-temporal queries in location-aware environments (e.g., see [BJKS06, LPM02, SR01, TXC07, ZZP⁺03, ZL01]) focus on developing specific high-level algorithms that use traditional database servers [MXA04]. Most existing query processing techniques focus on solving special cases of continuous spatio-temporal queries: some like [SR01, ZZP⁺03, ZL01, WSCY99] are valid only for moving queries on stationary objects, others like [HKGT03, PXK⁺02] (Carney et al. 2002) are valid only for stationary range queries. A challenging perspective is to provide a complete approach that integrates flexibility into the existing continuous, stream, snapshot, spatio-temporal queries for accessing data in pervasive environments.

The emergence of data services has introduced a new interest in dealing with these "new" providers for expressing and evaluating queries. Languages as Pig and LinQ combine declarative expressions with imperative ones for programming queries, where data can be provided by services. In general, query rewriting, optimization and execution are the evaluation phases that need to be revisited when data are provided by services and they participate in queries that are executed in dynamic environments. Query rewriting must take into consideration the data service interfaces, since some data may need to be supplied to these in order to retrieve the rest of the data. The existence of a large number of heterogeneous data services may also necessitate the use of data integration techniques. In addition, new types of queries will require the definition of new query operators.

Traditional query optimization techniques are not applicable in this new setting, since the statistics used in cost models are generally not available. Furthermore, resources will be dynamically allocated via

computation services, rather than being fixed and easy to monitor and control. Finally, query execution must also be reconsidered. First, the means to access the data is via services rather than scanning or employing index structures. Second, to process the data we depend on computation services, instead of a rigid DBMS.

[SMWM06] proposes a service coordination approach, where coordinations can be optimized by ordering the service calls in a pipelined fashion and by tuning the data size. The control over data size (i.e., data chunks) and selectivity statistics are key assumptions adopted by the approach. Another aspect to consider during the optimization is the selection of services, which can have an impact on the service coordination cost. [CAH05, WCSO08] optimize service coordinations by proposing a strategy to select services according to multidimensional cost. Service selection is done by solving a multi objective assignment problem given a set of abstract services defined by the coordination. Services implementing the coordination can change but the control flow of the coordination remains the same.

The emergence of the map-reduce model, has introduced again parallelization techniques. Queries expressed in languages such as Pig⁹, and SCOPE [CJL08a] can be translated into map/reduce [DG08] workflows that can be optimized. The optimization is done by intra-operator parallelization of map/reduce tasks. [Lim12] applies safe transformations to workflows for factorizing the map/reduce functions, partitioning of data, and reconfiguring functions. Transformations hold preconditions and postconditions associated to the functions in order to keep the data flow consistency. The functional programming model PACT [BEHK10] extends the map/reduce model to add expressiveness that are black boxes within a workflow. In [HPSR12] the black boxes are analyzed at build-time to get properties and to apply conservative reorderings to enhance the run-time cost. The map/reduce workflows satisfy the need to process large-scale data efficiently w.r.t. execution time. Although we do not address query optimization under such context in the present work, we provide a discussion of its related issues and possible solutions in [CVVSCB09].

Eventhough these problems have been already addressed, there are still some challenges that remain open:

- How to minimize the number of queries issued to evaluate continuous queries?
- How to capture changes arising in services, the execution environment and user preferences?
- How to propagate these changes to the query workflow ?
- How an atomic change impacts the quality of the query workflow and to what extent a set of atomic changes lead to a modification of the whole optimization process and impose the computation of a new query workflow?

Main contributions

Our fundamental research activities concern the specification and design of mechanisms implementing an optimized access to distributed data and computing services according to SLA contracts. Services hide data behind their APIs, they produce spatio-temporal data flows in batch or periodically and there is no full fledged DBMS providing data management functions.

Our original contribution consist of¹⁰:

- Query evaluation (continuous, recurrent or in batch) by reliably coordinating services guided by SLA contracts; data services can be mobile and static, and data can be spatio-temporal and produced in flows.
- Optimization of service coordinations for reducing their temporal, economic and energy cost.

9. <http://pig.apache.org>

10. See Chapter 7 that gives a list of my major publications numbered according to these references.

We introduced the notion of hybrid query, which was analyzed in a taxonomy of query types can be evaluated by academic and industrial approaches and systems[VSIC⁺11]. We proposed [1,4]:

1. The query language HSQL for expressing hybrid queries based on service coordinations and the language MQLiST (project CLEVER) for expressing the way the results of such a query can be mashed up.
2. The specification of a query model based on the coordination of data services using abstract state machines (ASM). The model is based on the notion of query workflow.
3. The algorithm BP GYO for generating the query workflow that implements an HSQL query.
4. The implementation of the hybrid query engine HYPATIA.
5. An algorithm for computing the space of query workflows that implement a hybrid query respecting an SLA contract expressed as the aggregation of measures: execution time, economic and energy cost.

Projects

Results on hybrid query processing in pervasive environments were mainly produced in the context of the projects OPTIMACS¹¹; E-CLOUDSS and CLEVER¹².

PhD students

- CUEVAS-VICENTTIN Victor, *Evaluation adaptable de requêtes dans des contextes largement distribués*, Advisors : Ch. Collet, G. Vargas-Solar, INPG, defended on July 2011.
- LOPEZ-ENRIQUEZ Carlos Manuel, *Services based query optimization*, double diploma program: Grenoble INP, France; U. de las Américas, Puebla, Mexico, 3rd year, expected date of defense: April 2014 (partially financed by the project OPTIMACS, and the CONACyT in Mexico) Advisors : Ch. Collet, J.L. Zechinelli-Martini, G. Vargas-Solar.

Prototypes

1. HYPATIA, Service based hybrid queries evaluator, <http://optimacs.imag.fr>

11. Financed by the ANR-ARPEGE program, <http://optimacs.imag.fr>

12. Financed respectively by financed by Microsoft through LACCIR project program <http://eclouds.imag.fr> and the STIC-AMSUD CNRS program <http://clever.imag.fr>

CHAPTER 4

Non functional requirements for service coordinations

Music is given to us with the sole purpose of establishing an order in things, including, and particularly, the coordination between man and time.

— Igor Stravinsky.

Expressing and enforcing non functional properties (NFP) for systems is a well-known problem addressed by models and protocols in distributed, and database systems for dealing with security, fault tolerance, and persistence (e.g, transaction monitors, synchronization protocols like 2PC). Data management functions are traditionally built-in functions in classic DBMS. These functions must be implemented by coordinating services for building Service Oriented Data Management Systems (SODMS). SODMS need to provide certain levels of ACID properties (i.e., NFP) according to (i) the type of data they manage (e.g., documents, key-value, graph), and (ii) the application requirements that may need specific storage oriented, aggregation oriented, partially consistent solutions.

Lately, applications implemented under "new" data management approaches, call for techniques that can include NFP in a flexible and adaptable manner. Some solutions leave this task to the middleware infrastructure (e.g., the service bus petals¹ offers some non-functional services), and to the engines that weave them within their execution model (e.g., workflow engine). Other approaches like [DDGJ01] include the code that implements NFP, as protocols within the service coordination code; or within the communication protocol used for calling service methods (e.g. data encryption of SOAP messages). For example, works stemming from the Business Process domain (BPM [OMG11]) and also to WS-* standards. Current NoSQL approaches already propose different levels of consistency, eventual persistency of data (relaxed durability), and relaxed atomicity protocols. Existing systems provide different protocols, and some of them provide interfaces so that the programmers can tune their own protocols [Cat11].

Our research has addressed strategies to provide reliable service coordination based data management (i.e., data consistency, exception handling, and atomicity). We adopted a policy based approach for modeling, defining, and ensuring NFP orthogonally to a service coordination that implements a data management function (e.g., data querying). We assumed that not respecting a NFP expressed in a specification in the design phase, implies an exceptional behavior of the application at execution time. Exceptions are dealt with, according to predefined actions expressed during the design phase devoted to tolerate these

1. <http://petals.ow2.org>

exceptions. Actions implement NFP protocols. The implementation of NFP in our work is based on the notion of active policy (A-Policy) that groups (i) sets of constraints and reactive recovery actions associated to service methods (parameter values); (ii) constraints evaluated at execution time; (iii) recovery actions triggered by exceptions. These exceptions are signaled when constraints are not satisfied.

The remainder of the chapter is organized as follows. Sections 4.1 and 4.2 describe the main concepts of the AP Model that we proposed and its associated AP Language for specifying policies and associating them to target service coordinations. Section 4.3 explains the strategies adopted for executing active policies by synchronizing their execution with that of a service coordination. Section 4.4 introduces the system Violet that provides an execution environment for active policy based workflows. It describes the validation scenarios of Violet and associated results. Section 4.5 discusses the position of our contributions with respect to existing NFP protocols and approaches.

4.1 A-Policy model

Consider the following scenario. A developer wants to implement a services coordination called Status Updater that:

- Reads information about the song a user is currently listening to (e.g., song title and artist name) using the service LastFM.
- Computes the mood of the user using the history of played songs, and a custom algorithm.
- Posts song information and user mood into the user’s Facebook and Twitter accounts (e.g., the string “Sting Fields of Gold”) using the services Facebook and Twitter. Figure 4.1 illustrates a workflow implementing the Status Updater coordination by ordering calls to the operations exported by the services LastFM, Facebook, and Twitter. In the diagram, services are represented as rectangles containing the name of the operation used by the workflow. Activities are represented as rounded rectangles connected to a service operation. Order among activities (i.e., control flow) is represented as solid arrows connecting activities.

The Status Updater workflow is composed of 4 activities:

- Get Song calling the operation `getLastSong` from the service LastFM. This activity receives a user `id` as input and outputs the song information.
- Compute Mood represents the algorithm that computes the user mood based on the music listened by a user during some period of time. This information is saved in a log.
- Update Twitter and Update Facebook calling the operations `updateStatus` of the services Facebook and Twitter. Both activities receive as input a user `id`, and some text (in this case the song information), and produce no result as output.

Now let us assume that the developer wants to extend the Status Updater workflow for addressing the following non-functional requirements:

- Authentication. Due to privacy protection Twitter and Facebook implement authentication protocols that control access to user information (e.g., only authorized applications can read and/or update users information). For instance, Twitter uses a basic authentication protocol based on username and password for preventing applications of updating the user status. In contrast, Facebook uses the OAuth protocol, a protocol based on third-party authentication for the same purpose.
- Network unavailability. Due to network unreliability services can be unavailable at certain periods of time. This can cause exceptions that must be handled by the workflow (e.g., by retrying the calls to the services or by compensating the activities).

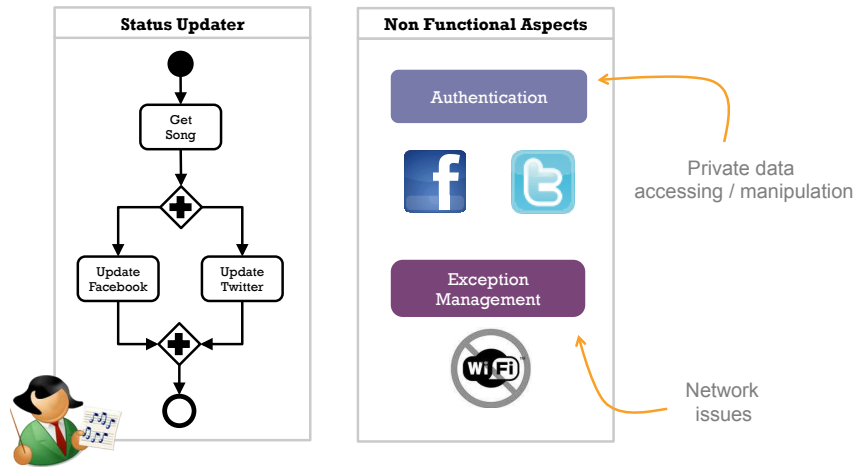


Figure 4.1: Status Updater workflow

The A-Policy Model provides concepts as a collection of types for representing service coordinations with non-functional properties. In our model, a service coordination is represented as a workflow composed of an ordered set of activities, each activity in charge of implementing a call to a service operation. We use the type **Activity** for representing a workflow and its components (i.e., the workflow activities and the order among them). A NFP is represented as one or several **Active Policy** types, each policy instance is composed of a set of event-condition-action rules in charge of implementing an aspect of the property. The instances of active policy and activity types are considered in the model as entities that can be executed. We use the **Execution Unit** type for representing them as entities that go through a series of states at runtime. When an active policy is associated to one or several execution units, its rules verify whether each unit respects the implemented non-functional property by evaluating their conditions over their execution unit state. When the property is not verified, the rules execute their actions for enforcing the property at runtime.

Figure 4.2 shows the types composing the AP Model as classes. They are grouped according to the concept they describe:

- The classes used for representing a services coordination (i.e., **Activity**, **Atomic Activity**, **Workflow Activity** and **Control Flow Activity**). The main class in this group is the class **Activity**. It represents the type **Activity**.
- The classes representing a non-functional property (i.e., **Active Policy**, **Rule**, and **Event**). The main class in this group is the class **Active Policy**. It represents the type **Active Policy**.
- The classes used for associating non-functional properties to services coordinations (i.e., **Execution Unit** and **Execution State**). The main class in this group is the class **Execution Unit**. It represents the type **Execution Unit**.

All possible objects considered in the model can be represented as an instance of one of these types. Figure 4.3 illustrates some examples of active policy instances. Instances **ap₁** and **ap₂** are examples of objects belonging to the **Basic** policy type (i.e., they conform to the properties, behavior and semantics described by the type). Instance **ap₃** is an example of an object belonging to the type **OAuth** policy. Instance **ap₄** is an example of an object of type **Exception Management Policy**.

Figure 4.3 also shows that the AP Model types can be specialized in subtypes. The type **Active Policy** represents all policy instances that can exist in the universe, no matter what non-functional property they deal

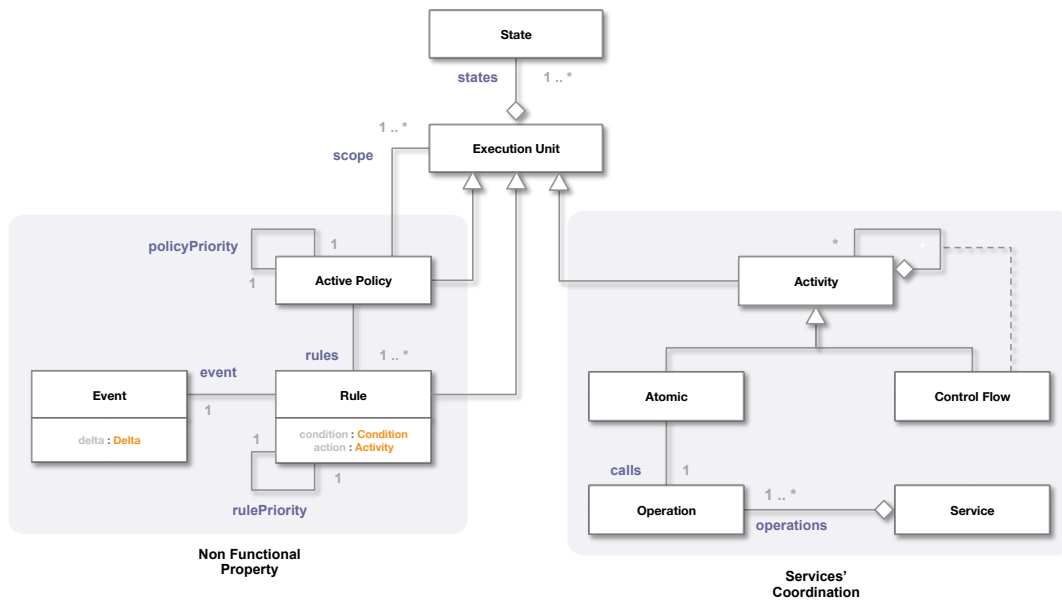


Figure 4.2: UML class diagram of the A-Policy Model

with. In contrast, the type **Authentication Policy** represents policy instances that deal with authentication aspects (i.e. a specific non-functional property). When a type T_1 is a subtype of T_2 all the instances of T_1 are also instances of T_2 (i.e., an instance can belong to more than one type). For instance, since the types **Basic** and **OAuth Policy** are subtypes of the type **Authentication** (see figure 4.2), instances ap_1 , ap_2 , and ap_3 are also instances of the type **Authentication Policy**. In the same way, even if ap_3 and ap_4 belong to different types, they are examples of instances of the type **Active Policy**.

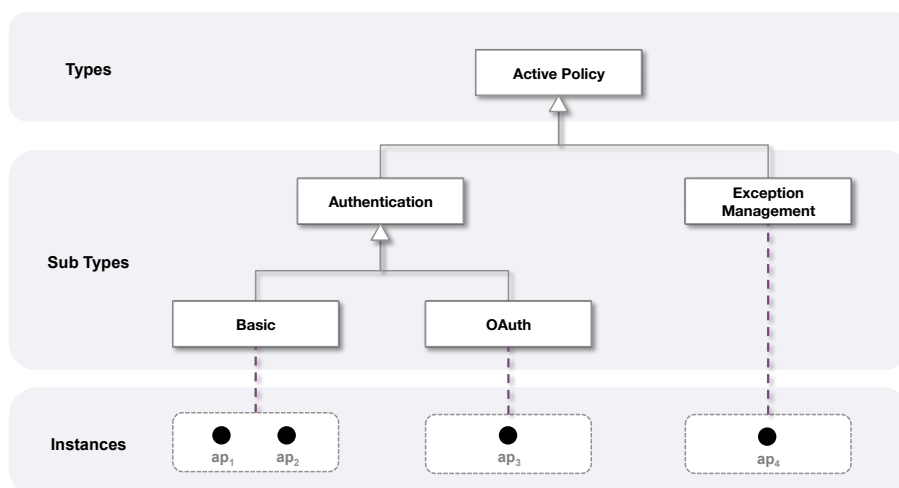


Figure 4.3: A-Policy Model levels of abstraction

AP Model types are built on top of the data types shown in figure 4.4. As shown in the figure, any type in the model can be classified as an Atomic or Composite type. Atomic types represent instances that

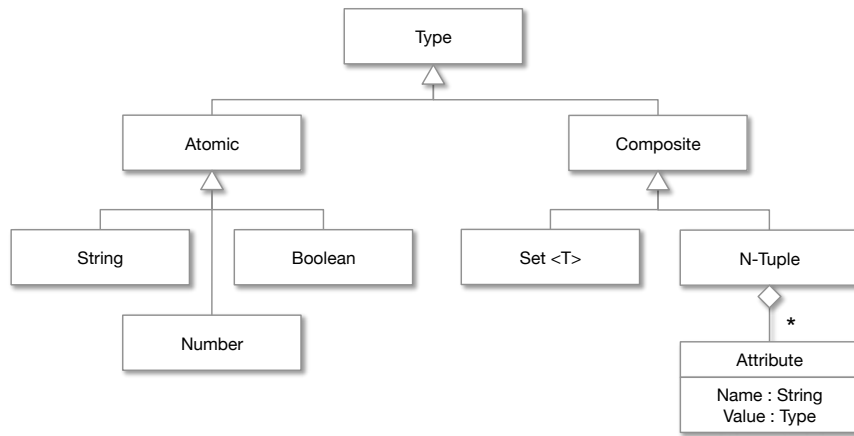


Figure 4.4: AP model data types

cannot be further divided in other values. The atomic types considered in the model are **String**, **Number**, and **Boolean** (see figure 4.4). The type **String** represents any chain of characters (e.g., "www", "James Bond", "pwd123"). The type **Number** represents any real or natural numbers (e.g., -1, 0, 1, 3.1416). The type **Boolean** represents values true and false (i.e., truth values). Composite types represent instances that are composed of other values (i.e. instances of other types). The composite types considered in the model are the types **Set**, **N-Tuple**, and **Attribute**. These types conform to the following rules:

- If $A_1 \dots A_n$ are different attribute names (i.e., $A_i \dots A_j, \forall_{i,j} \in [1 \dots n]$), and $T_1 \dots T_n$ are type names, the expression $\langle A_1 : T_1, \dots, A_n : T_n \rangle$ represents a type **N-Tuple**. For example, the expression:

$\langle \text{name} : \text{String}, \text{value} : \text{Type} \rangle$

represents the type **Variable** as a binary tuple composed of (i) a name attribute of type **String** representing the variable name, and (ii) a value attribute (of any type) representing the variable values type.

- If T is a type name then the expression **Set** $\langle T \rangle$ represents a typed collection where all the elements in the collections are of type T . For example, **Set** $\langle \text{String} \rangle$ represents a collection containing only elements of type **String**.

4.1.1 Service Type

The AP Model is also based on the notion of service. As shown in figure 4.5, we use the type **Service** for representing a service as an autonomous entity that exports a collection of operations via a network.



Figure 4.5: Service Type UML diagram

The type **Service** is defined as follows:

```
< address: String, operations: Set <Operation> >
```

Its attributes are interpreted as follows:

- **address** represents the URL where the service can be located.
- **operations** represents the set of operations exported by a service, each operation represented by the type `Operation`.

The type `Operation` has the following form:

```
< name: String, input: Set<Variable>, output: Type>
```

Its attributes are interpreted as follows:

- **name** represents the name of the operation;
- **input** represents the set of variables that an operation receives;
- **output** represents the type of the value produced by the execution of the operation.

4.1.2 Activity Type

We use the type `Activity`² for representing the concepts abstracting a services coordination (i.e., workflow, workflow activities, and the order relationship among the workflow activities). As shown in figure 4.6, the type `Activity` has the following structure:

```
< name: String, input: Set<Variable>, local: Set<Variable>, output: Type>
```

Its attributes are interpreted as follows:

- **name** uniquely identifies an activity type participating in a workflow;
- **input** contains the variables received as input of the activity;
- **local** contains the set of local variables used for computing the activity output;
- **output** represents the value produced by the execution of the activity.

Figure 4.6 also shows the activity types used for representing a services coordination:

- **Workflow activity** the type that represents a workflow;
- **Atomic activity** the type that represents a workflow activity;
- **Control Flow activity** the type that represents the order relationship among the workflow activities.

Since these types are subtypes of the type `Activity`, they all share the same structure i.e., every possible activity considered in the model (i) receives a set of variables as input (possible empty), (ii) produces a single variable as output, and (iii) have a set of local variables used internally for computing its output.

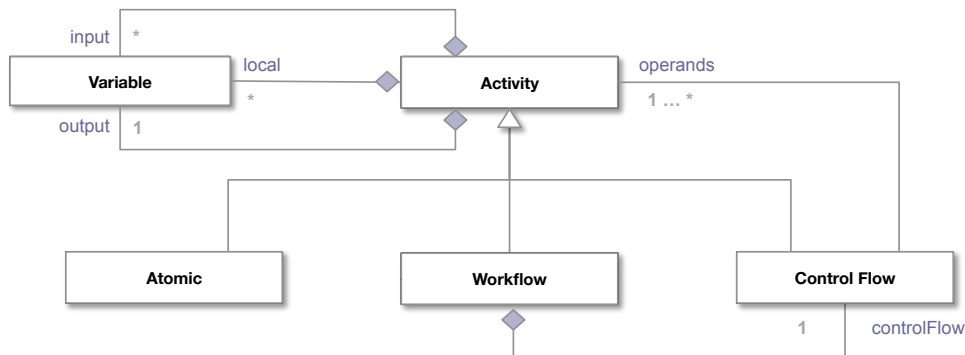


Figure 4.6: Activity Type UML diagram

2. The use of activity types for representing a services coordination is not proper to our model. It was first used in the Business Process Execution Language (BPEL) as a building block for expressing executable workflows.

4.1.2.1 Atomic Activity Type

The type **Atomic Activity** represents the logic handling a call to a service operation (see Figure 4.7). The type has the following structure :

```
< calls: Operation, operationResult: Type >
```

Its attributes are interpreted as follows:

- **calls** represents a reference to a service operation. An atomic activity can be associated (i.e., **calls**) to maximum one service operation.
- **operationResult** contains the result of a service operation call. Note that attribute can contain a value of any type.

When executed, an instance of the type **Atomic Activity** uses the service address, the operation name and the activity input variables for calling the service operation. Then, once the operation terminates, the activity stores the operation result in the attribute **operationResult**. Finally, the activity uses the **operationResult** value for computing the activity output.

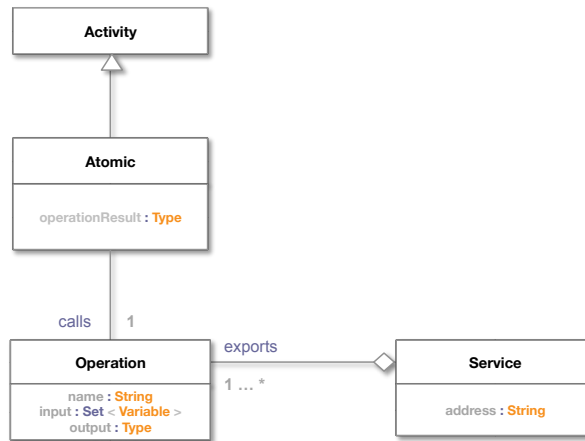


Figure 4.7: Atomic Activity Type UML diagram

4.1.2.2 Control Flow Activity Type

The type **Control Flow Activity** represents the logic implementing an order relationship among a set of activities called operands. The AP Model considers 3 types of control flow activities: **Sequence**, **IF**, and **While**.

- The activity type **Sequence** represents the logic implementing a sequential execution. For instance, let us assume that **Sequence (A, B)** represents an activity of type sequence that relates activities of types A and B. Now assume that **seq (a₁, b₁)** represents an instance of **Sequence (A, B)** where a₁ and b₁ are instances of A and B, respectively. This relationship implies that, when executed, instance **seq** will execute a₁ until completion before executing b₁. More formally, if a₁.ends is an integer representing the time when a₁ ends its execution, and b₁.starts is an integer representing the time when b₁ starts its execution, instance **seq** will respect the relationship a₁.ends < b₁.starts at runtime.

As an example consider the activities **Get Song** and **Compute Mood** of the **Status Updater** scenario. According to the scenario these activities have to be executed in sequence. The activities **Get Song**

and **Compute Mood** are represented as Atomic Activity types. When executed, an instance of the type **Sequence** will execute first an instance of the type **Get Song** (i.e., the lefside operand) and then, an instance of the type **Compute Mood** (i.e., the rightside operand).

- The Activity type **IF** represents the logic implementing a conditional execution. For instance, let us assume that **IF** (θ , **A**, **B**) represents activities of type **IF** that relate condition expression θ , and activity types **A** and **B**. Now assume that **if** (θ_1 , a_1 , b_1) represents an instance of **IF** (θ , **A**, **B**) where θ_1 , a_1 and b_1 are instances of **String**, **A** and **B** (respectively). This relationship implies that, when executed, instance **if** will execute a_1 if and only if θ_1 is evaluated to true. Otherwise instance **if** will execute b_1 .

As an example consider an extended version of the Status Updater scenario where the songs listened by a user can be retrieved from two different services: the services LastFM and Deezer. Also consider that the Status Updater coordination decides which service to call by evaluating the condition "useLastFM == true". The Activity type **IF** can be used for implementing this decision by relating the activity types **Get Song From LastFM** and **Get Song From Deezer**. When executed, an instance of the type **IF** will evaluate whether the value associated to useLastFM variable is true. In that case it will execute an instance of the activity **Get Song From LastFM** (i.e., the true operand). Otherwise it will execute an instance of the activity **Get Song From Deezer** (i.e., the false operand).

- The Activity type **While** represents the logic implementing an iterative execution. For instance, let us assume that **While** (θ , **A**) represents a while activity that relates condition expression θ and activity type **A**. Now assume that **while** (θ_1 , a_1) represents an instance of **While** (θ , **A**) where θ_1 and a_1 are instances of **String** and **A**, respectively. The relationship **while** (θ_1 , a_1) implies that, when executed, an instance while will execute a_1 if and only if θ_1 is evaluated to true. Otherwise it will execute nothing. If θ_1 is evaluated to true, the instance "while" will re-instantiate the activity type **A** for preparing a new iteration. This implies obtaining the instance represented by the expression **while** (θ_1 , a_2), and then it will re-evaluate θ_1 for deciding whether a_2 has to be executed or not. Note that an instance of the Activity type **While** may interact with multiple instances of other activity types at runtime. For instance, in the example **While** (θ , **A**), an instance while can interact with a_1 , a_2 , ..., a_n where a_i represents the i_{th} iteration.

As an example consider the activity **Get Song** of the Status Updater scenario. Recall that this activity is represented using the Atomic Activity type **Get Song** and, that it has to be executed several times for obtaining the last song played by a user. The Activity type **While** can be used for implementing the required iterative behavior over the activity **Get Song**. As shown in the figure, the expression "isUserListeningMusic()==true" is used for deciding whether a new iteration has to be executed (i.e., the condition). If the user is listening to music, an instance of the while activity will evaluate this expression to true, and then it will execute an instance of the activity type **Get Song** (i.e. the do operand). Otherwise the activity instance "while" will complete its execution.

4.1.2.3 Workflow Activity Type

The type **Workflow Activity** represents the logic implementing a services coordination as an ordered set of activities. The type **Workflow Activity** has the following structure:

```
< controlFlow: ControlFlowActivity >
```

Besides having a **name**, an **input**, an **output**, and **local variables** (inherited from the type **Activity**), an Activity type **Workflow** has a **controlFlow** that represents the order in which the activities composing a workflow have to be executed.

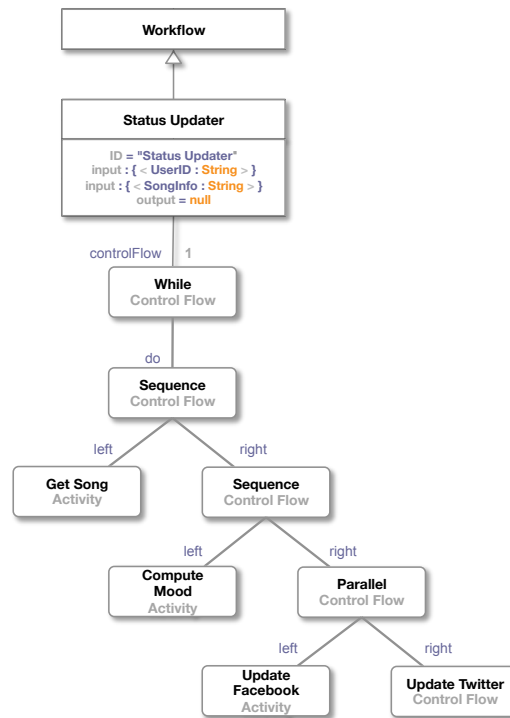


Figure 4.8: UML class diagram illustrating a specific workflow as a Workflow Activity type

For instance, figure 4.8 shows the Activity type **Workflow** representing the Status Updater services coordination. As shown in the figure, the Activity Workflow type **Status Updater** receives a `userID` as input, produces no output, and defines a local variable called `songInfo`. The figure also shows the Activity type **Control Flow** representing the control flow. Note that we represent it as an activity tree where:

- The leaf nodes represent the Atomic Activity types composing the workflow (i.e., activities `Get Song`, `Compute Mood`, `Update Facebook`, and `Update Twitter`).
- The internal nodes represent the Control Flow Activity types representing the order among activities (e.g., the fact that the activity `Get Song` must be executed before the activity `Compute Mood`).
- The root node represents the Activity type **Control Flow** containing a workflow implementation (e.g., the activity `While` containing the logic that specifies what to do for each song listened by a user).

4.1.3 Execution Unit Type

We use the type **Execution Unit** for representing an entity that goes through a series of states at runtime and notifies its progression to the execution environment by producing events. As shown in figure 4.9, the type **Execution Unit** has the following structure:

```
< states: Set<State>, notifies: Event >
```

Its attributes are described as follows:

- `states` represents the set of ordered states through which goes an execution unit when executed (e.g., an activity instance goes first to the initial state and later it arrives to the final state). We use the type `State` for representing execution unit states.
- `notifies` represents the type of events produced by an execution unit (e.g., the execution of an

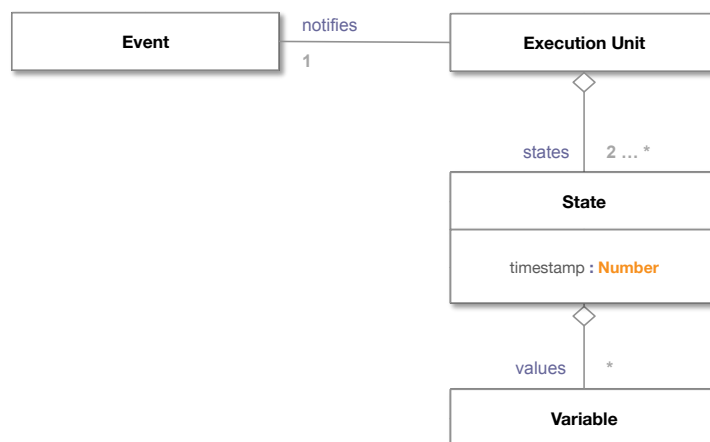


Figure 4.9: Execution Unit Type UML diagram

activity produces instances of type `ActivityEvent`). We use the type `Event` for representing events. The AP Model classifies execution units into activities, active policies, and policy rules. This implies that the behavior of instances of the types `Activity`, `Active Policy` and `Rule` can be described as a series of states and transitions among the states. This will be further described in the next section that addresses the execution of active policies.

The type `State` represents a momentum during the execution of an execution unit. This type has the following form:

```
< timestamp: Number, values: Set<Variable> >
```

Its attributes are described as follows:

- `timestamp`³ represents the time associated to a state.
- `values` represents the set of values that contained in the variables of an execution unit at time `timestamp`.

The type `Event` represents a happening of interest produced during the execution of an execution unit. For instance, an activity has started or ended its execution. The type `Event` has the following structure:

```
< timeStamp: Number, producerID: Number, delta: Set<Variable> >
```

Its attributes are described as follows:

- `timestamp` represents the time at which an event was produced.
- `producerID` represents a string uniquely identifying the execution unit producing an event. We assume that this identifier is assigned by the system at the moment of instantiating a type.
- `delta` contains information about the conditions in which an event is produced (i.e., the state of the execution unit producing the event). We represent the `delta` as a set of pairs of the form `variable = value`.

In the AP Model we assume that an execution unit produces an event every time it enters a state (e.g., when an activity is initialized, it produces an event instance of type `ActivityInitialized`). Thus, in order to describe the behavior of all the AP Model execution units, we have specialized the type `Event` into several subtypes (see figure 4.10):

3. The representation of time can be of different granularities (e.g., hour, day, minute, etc.). We consider that this granularity is determined by the system executing the workflows. We also consider that, independently of its representation, time can be transformed into a positive integer (i.e., a number $i \in [0, \infty]$).

- Activity Life Cycle Event types. They represent the events describing the lifecycle of every activity instance. For instance, the activity has been initialized, completed or failed.
- Atomic Activity Event types. They represent the events produced by an atomic activity instance during a service operation call. For instance, the activity is prepared for invoking the operation or the operation has been invoked.
- Control Flow Activity Event types. They represent the events produced during the execution of control flow activity instances. For instance, an activity of type IF has evaluated its associated condition or a sequence activity has executed one of its operands.
- Active Policy Event types. They represent the events describing the lifecycle of an active policy instance. For instance, a policy has been (de)activated or completed.
- Rule Event types. They represent the events describing the lifecycle of a rule instance. For instance, a rule has been triggered or its action has been executed.

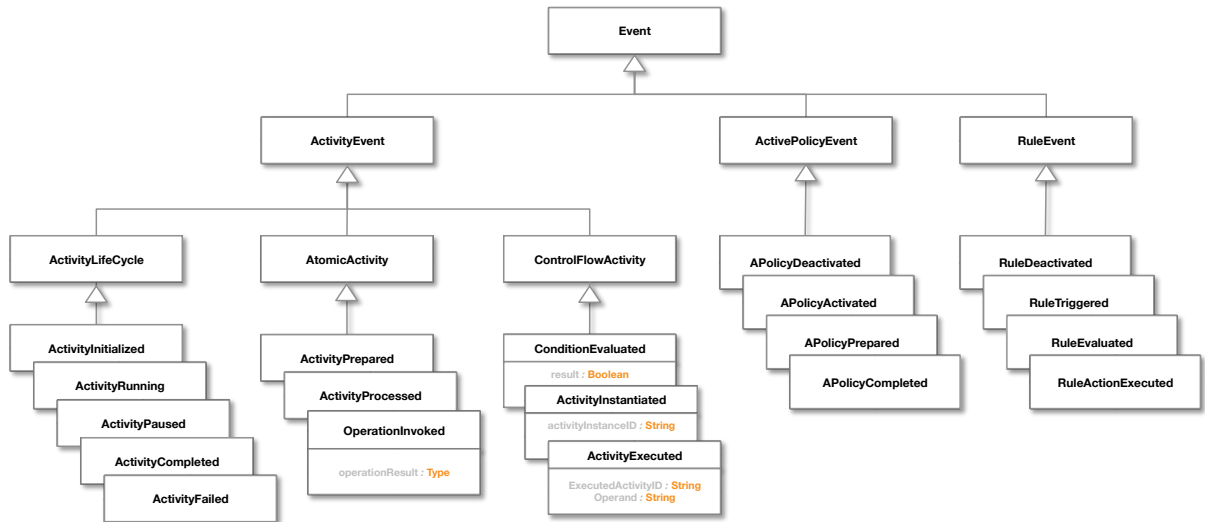


Figure 4.10: Event types UML class diagram

4.1.4 A-Policy Type

We use the type `Active Policy` for representing the logic implementing a non-functional property. The type `Active Policy` (see figure 4.11) has the following structure:

```
< rules: Set<Rule>, scope: Set<ExecutionUnit >, variables: Set<Variable> >
```

Its attributes are described as follows:

- `rules` represent a non-empty set of event-condition-action (ECA) rules that verify and enforce a non-functional property at runtime. We use the type `Rule` for representing ECA rules.
- `scope` represents the association of an active policy with one or several workflow execution units. When an execution unit `eu` is associated to a policy `ap` it is said that `eu` belongs to the scope of `ap` or that "ap applies to eu".
- `variables` represent the set of local `variables` composing an active policy, and which are accessible to the policy rules (i.e., `rules` can read, and modify variables values).

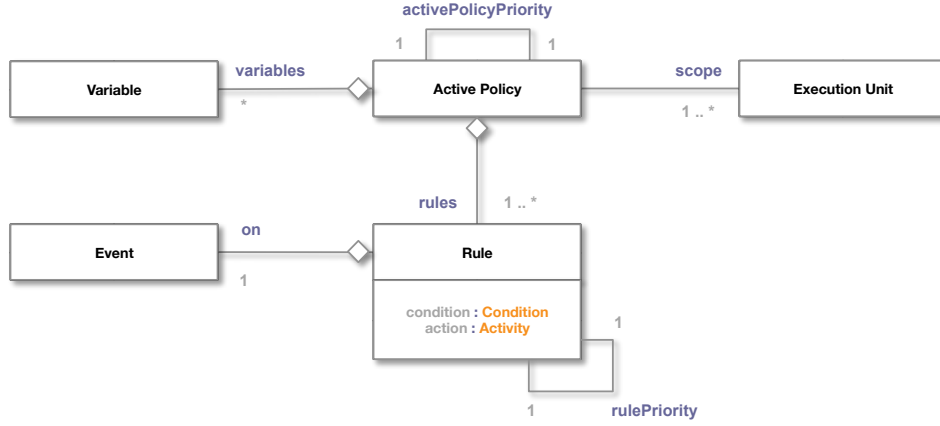


Figure 4.11: Active Policy Type UML diagram

The type `Rule` represents an ECA rule with the classical semantics: "on the notification of an event of type `E`, if a condition `C` is verified, execute an action `A`". The `Rule` type has the following structure:

`< event: Event, condition: Condition, action: Activity >`

Its attributes are described as follows:

- `event` represents a significant happening occurring during the execution of an execution unit at time `t`.
- `condition` represents a boolean predicate that is evaluated over the state of a policy (i.e., the values of the policy variables and the state of the scope execution units).
- `action` represents an activity that can (i) act on the execution of a workflow (e.g., stop, resume, update), (ii) signal an event, and (iii) activate (deactivate) the rules of a policy.

Active policy rules are related to each other using the notion of priority. At execution time, priorities are used for scheduling the execution of rules that are triggered at the same time (i.e., triggered by the same event instance). We consider two kinds of priorities:

- Priorities among rules. They specify the execution order among the rules of a same policy. For instance, if r_1 and r_2 are rules belonging to policy ap , and r_1 has lower priority than r_2 , when both rules are triggered r_1 has to be executed before r_2 .
- Priorities among policies. They specify the execution order among the rules belonging to different policies. For instance, if r_x and r_{x+1} are rules belonging to policy ap_x , r_y is a rule belonging to policy ap_y and ap_x has lower priority than ap_y , then rules r_x and r_{x+1} have to be executed before r_y when triggered at the same time.

4.2 AP Language

We propose the AP Language, for defining active policy types, an extension to the $C\sharp$ language with constructors for (i) defining Active Policy types, and (ii) associating these types to a Workflow. Since the AP Language is based on the $C\sharp$ language, defining a type implies defining a class (i.e., use of the class keyword, and definition of the variables and methods composing the class): (i) use the policy keyword for declaring a policy type (a class), (ii) specify the rules composing the policy, and (iii) specify the scope of the policy. Figure 4.12 illustrates the informal definition of the grammar of the AP Language.

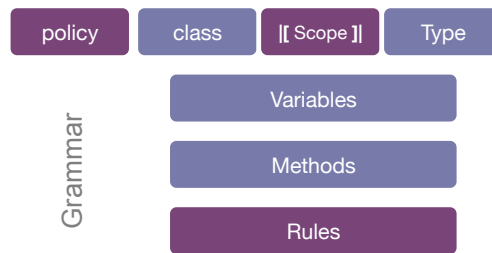


Figure 4.12: AP Language grammar

An active policy type defined using the AP Language specializes the types of the AP Model. A new active policy type can represent concrete non-functional properties (e.g., security, exception handling, atomicity, persistence, etc.). The following sections illustrate how to define new Authentication Active Policy types using the AP language for dealing with the authentication requirements of the Status Updater coordination (i.e., the implementation of the OAuth and Basic authentication protocols).

4.2.1 Defining A-policies

An authentication A-Policy represents the situation where an invocation in an activity occurs until its sender and/or its recipient have been identified. Typically, authentication A-Policies ensure that the invocation of the activity will be done within an authentication protocol.

In the AP approach these non-functional requirements are addressed by defining A-Policy types and by associating these types to a target workflow. For example, a developer can define an Authentication A-Policy type for addressing the authentication requirements of workflows. Then, as shown in figure 4.3 the developer can specialize this type for defining the policy types HTTP and OAuth protocols. These new types can be associated to the activities Update Twitter and Update Facebook of the Status Updater workflow. These policy types will be used for adding the HTTP and OAuth protocols (respectively) required by each service associated to the workflow activities.

Specialized classes **OAuth Policy** and **HTTPOAuth Policy** are then used for specifying concrete policies, and associating them to one or several activities of a services' coordination.

4.2.2 OAuth A-Policy

Open authentication (OAuth) allows a user to grant a third-party Web site or application access to its resources, without necessarily revealing their credentials, or even their identity. For example, the Twitter site that supports OAuth would allow its users to use a third-party authentication token provider to access a user account, without gaining full control of the user account. OAuth consists of a (i) mechanism for a user to authorize issuance of credentials which a third party can use to access resources on their behalf; and a (ii) mechanism for using the issued credential to authenticate HTTP requests (called "signatures" in current OAuth).

Figure 4.13 shows an implementation of the policy type **OAuth** using the AP Language. As you can see, the type **OAuth policy** is a kind of **Authentication policy type** that defines:

- An activity **getToken** that implements the logic for (i) retrieving the access token, and (ii) setting this value into the execution unit (i.e., the activity).
- Activity **renewToken** that implements the logic for renewing the access token.

```

policy class OAuth [[ Activity activity ]] : Authentication {
    Token token ;

    rule TokenRetrieval
    on ActivityPrepared
    if event.producedBy ( activity ) && token == null
    do token = GetToken ()

    rule TokenRenewal
    on ActivityPrepared
    if event.producedBy ( activity ) && token != null && token.isExpired
    do token = RenewToken ()
}

```

Figure 4.13: OAuth A-Policy definition

- Variable `token` used for containing the access token.
- Rule `tokenretrieval` stating that, if the activity in the policy scope is prepared for execution, and the token variable does not contain a value (i.e. the variable is null), the variable `token` will take the value retrieved by the activity `getToken`.
- Rule `tokenrenewal` stating that, if the activity in the policy scope is prepared for execution, and the token variable has expired, the variable `token` will take the value re-trieved by the activity `renewtoken`.

Note that the policy type imports the types defined in the namespace `OAuth`. The type `Token` belongs to this namespace.

4.2.3 HTTP-Auth A-Policy

```

policy class BasicAuth [[ Activity activity ]] : Authentication {
    String username, password;
    rule R1
    on ActivityPrepared
    if event.producedBy ( activity )
    do {
        activity.Request.Username = username;
        activity.Request.Password = password;
    }
}

```

Figure 4.14: HTTP-Auth A-Policy definition

In the context of an HTTP transaction, the basic authenticated access is a method designed to allow a Web browser, or other client program, to provide credentials - in the form of a user name and password - when making a request. Before transmission, the user name is appended with a colon and concatenated with the password. The resulting string is encoded with the Base64 algorithm. For example, given the user name 'Jane Doe' and password 'abc123', the string 'Aladdin:open sesame' is Base64 encoded, resulting in 'QWxhZGRpbjpvYVUyIHNlc2FtZQ=='. The Base64-encoded string is transmitted and decoded by the receiver, resulting in the colon-separated user name and password string.

Figure 4.14 gives the HTTP-Auth A-Policy type defined for implementing the HTTP-Auth protocol. This

type represents a policy that implements a basic authentication protocol based on a user username and password (e.g., the one used by HTTP protocol). As shown in the figure, the policy type (i) is composed of one rule and two variables, and (ii) can be associated to an activity of any type. The variables denote the values for username and password. The rule implements the authentication protocol by using the variables. Assuming that the policy is associated to an activity type, the policy works as follows:

- When the policy is instantiated the values for username and password are passed to the policy instance. Then the policy waits for the notification of triggering events (i.e., events of type `ActivityPrepared`).
- During its execution the activity will get prepared before calling its associated service operation. This causes the notification of an event of type `ActivityPrepared` that triggers rule `r1`.
- When `r1` is triggered, the rule uses the values in variables `username` and `password` for configuring the variable `activity request`. Since rule `r1` is marked as `sync`, the rule is executed synchronously with respect to the activity i.e., `r1` first pauses the execution of the activity, then `r1` configures the variable `activity request`, and finally `r1` resumes the activity execution.
- During execution the activity uses (internally) this request variable for calling the service operation. Since the variable now contains the user username and password, the operation call can be identified by the service as an authorized call.

4.2.4 Associating policies to a services' coordination

```
Map BasicAuth [| Root.N1.N2.N4.N5 |] as P1 with
  username = "jane.doe"
  password = "abc.123"
```

Figure 4.15: Association of a BasicAuth policy type to the Status Updater workflow

Figure 4.15 illustrates the use of the AP Language for associating the policy type `HttpAuth` to the Status Updater workflow. In the example the scope of the policy type is associated to activity `N5`, which represents the activity `Update Twitter`. Notice that the values `"jane.doe"` and `"abc.123"` are passed to the `username` and `password` policy variables, respectively.

In order to illustrate the use of the authentication policy types in the Status Updater scenario, Figure 4.16 shows once again the expression tree of the Status Updater workflow. However, this time the nodes of the tree are annotated with unique identifiers.

4.3 A-Policy execution

At execution time, policies can execute some code before and after the execution of an activity in order to implement an authentication protocol. This can be equivalent to adding activities to the Status Updater workflow before and after the activities `Update Twitter` and `Update Facebook` of the workflow. Yet, thanks to policies the authentication issues are maintained separated to the workflow. Indeed, in the AP approach A-Policy types are defined independently of concrete workflows. Thus, it is possible to keep separated the logic implementing a service coordination from the implementation of its non-functional requirements, simplifying the maintenance process. For example, if one of the services changes its authentication protocol,

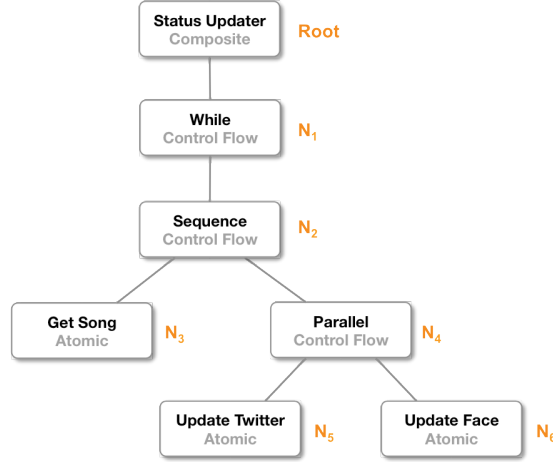


Figure 4.16: AAnnotated version of the status updater activity tree

the change remains transparent to the coordination because the A-Policy can be modified without touching the coordination implementation.

This Section describes how active policy-based workflows behave at runtime (i.e., how instances of the types `Active Policy`, `Rule`, and `Action` interact among each other for adding NFPs to a services coordination implemented as a `Workflow`). In particular this section describes (i) how events produced by execution units (i.e., active policies and activities) trigger active policy rules, and (ii) how action workflows (i.e., the action part of policy rules) interact with coordination workflows for enforcing NFP at runtime.

The evaluation process of a policy is presented in the UML state machine diagram of figure 4.17. As shown in the figure, a policy is `deactivated` until the notification of the beginning of the scope (`scope started`) associated to the policy, causing the policy to be `activated`, and the transition to the `triggerable` state. This state represents the fact that the rules belonging to the policy can be triggered. If one rule (or several) is triggered, the policy enters the `triggered` state where the triggered rules are executed before entering the `triggerable` state again. If more than one rule is triggered, the rules execution is scheduled during this state. A policy leaves the `activated` state when the execution of the scope terminates and an event `scope ended` is notified.

For instance for an instance of the type `0-Auth A-Policy`, an event e_a of type `ActivityPrepared` will trigger both of its rules. Triggered rules are ordered according to their priorities, then they are selected and executed, i.e., their condition is evaluated, and their action executed. In our example, rules are not explicitly ordered because they are commutative. In this case, the manager considers the definition order. Thus, $R_1 > R_2$ meaning that the manager will test first whether there is an existing token, and if not, then it will obtain one (R_1), otherwise the current one will be renewed (R_2). Once all triggered rules have been processed the policy returns to its `triggerable` state.

4.3.1 Executing one rule

Figure 4.18 presents the UML state machine diagram of a rule execution. It shows: i) when to execute a rule with respect to the notification of events, and (ii) how to synchronize its execution with the execution of the scope of its associated policy.

Events are considered as triggering events during their validity time interval (the execution of the scope

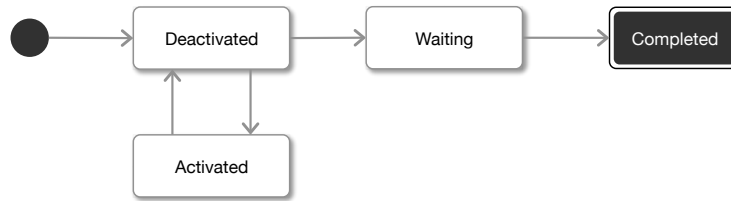


Figure 4.17: UML state machine diagram of an A-Policy execution

of the policy, i.e., an activity, a subworkflow, or of the whole coordination). In our example, the validity time interval of events is the execution of the activity that produces them, meaning from the moment in which the execution of the activity is in its prepared state, and it reaches its terminated state. For example, the event e_a of type `ActivityPrepared` produced within the execution of an activity of type `UpdateTwitter`, triggers the rules R_1 and R_2 of the policy `Twitter HTTP-Auth`. The event e_a is only valid while the activity is executed.

Event consumption policies specify how to handle an event that has triggered a rule. An event can be taken into account either for one execution of a rule or for several executions after its notification until the end of its validity time interval. For instance, suppose that the service `Twitter` is not available during the execution of the activity `UpdateTwitter`. The call can be re-executed instead of signaling an exception. In order to do that, the policy implementing the authentication protocol, must be triggered again. But the activity `UpdateTwitter` will not go back to its preparation state. Thus, the event e_a will be consumed again, and it will be valid because the execution of the activity is not terminated.

The rule’s execution starts either immediately after the notification of its triggering event or it can be deferred, for example to the end of the execution of an activity, a subworkflow or the whole coordination. For example, the rules of the authentication policies of our example are executed immediately in order to authenticate the user before calling the service method for reading/writing her status. Clearly, coupling aspects requires that the policy manager has both sufficient access to the execution status of the execution of the coordination, and influence to abort, block, and restart it according to rule behavior properties.

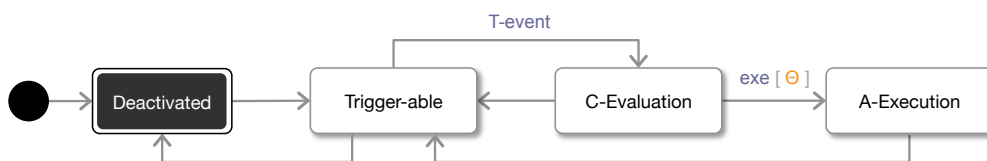


Figure 4.18: UML state diagram of a rule execution

4.3.2 Executing several rules

Finally, several rules can be triggered by the same event (see Figure 4.19). When this situation comes up, one has to determine when and how these rules have to be executed. This is done by building an execution plan based on execution strategies specified according to different criteria. In general, criteria combine sequential and parallel execution based upon rule, and policies dependencies. Sequential execution is achieved using rule ordering strategies that consider rule priorities, triggering, and definition order, and execution cycles.

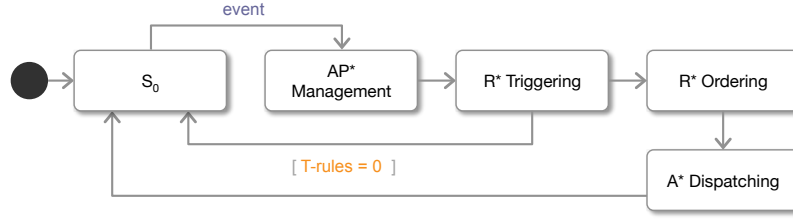


Figure 4.19: UML state diagram of the execution of several rules

The diagram of Figure 4.20 illustrates the general process used for scheduling the execution of several rules. The diagram is interpreted as follows: at certain point in time no rules are triggered (see state S_0). Then, once an event is notified, the scheduling process determines (i) whether the event activates/deactivates a policy, and thus, the rules of the policy (see AP^* Management state), and (ii) whether the event is a triggering event of one or more of the activated rules (see R^* Triggering state). If the event triggered more than one rule, the rules are ordered based on some criteria (see R^* Ordering state). After that, the process executes each of the rule actions one following the established order (A^* Dispatching). In this work, we consider that triggered rules can be ordered according to the following criteria:

- Priorities. Rules are ordered based on the priorities specified when defining the policies (i.e., using the priorities among rules, and the priorities among policies).
- FIFO. Rules are ordered according to the order in which they were triggered.
- Parallel. Rules can be executed simultaneously (i.e., there is no order dependency among the execution of the rules).
- Mixed. Rules are ordered using Priorities/FIFO/Parallel criteria (e.g., rules can be ordered first by priorities and then, all the rules having the same priority can be executed in parallel).

Note that it is also possible that the execution of a rule triggers a second rule, which in turns triggers a third rule, and so on (i.e., the execution of the action workflow associated to a rule may produce a cascade of triggering events). In this scenario the scheduling process determines the order among the rules based on the following strategies:

- Depth-first execution. Prioritizes the execution of newly triggered rules.
- Execution cycles. Prioritizes the execution of rules based on the cycle where they were triggered.

As an example consider the triggering event tree shown in figure 4.20. The figure is interpreted as follows: e_1 represents an event that triggers rules R_1 and R_2 ; e_2 represents an event produced by R_1 that triggers rules R_{1a} , and R_{2b} ; e_4 represents an event produced by R_2 that triggers rule and R_{2a} . When the scheduler process follows the depth-first strategy, the scheduler dispatches the rules as follows:

$$R_1 \rightarrow R_{1a} \rightarrow R_{2b} \rightarrow R_2 \rightarrow R_{2a}$$

In contrast, when the scheduler follows the execution cycle strategy, the scheduler dispatches the rules as follows:

$$R_1 \rightarrow R_2 \text{ (cycle 1)} \rightarrow R_{1b} \rightarrow R_{1a} \rightarrow R_{1b} \rightarrow R_{2a} \text{ (cycle 2)}$$

4.3.3 Executing Actions for Reinforcing non-functional properties

In order to ensure a NFP using active policies, we have to determine what kind of relationship exists between (i) the memory space of a workflow implementing a services coordination, and (ii) the operations

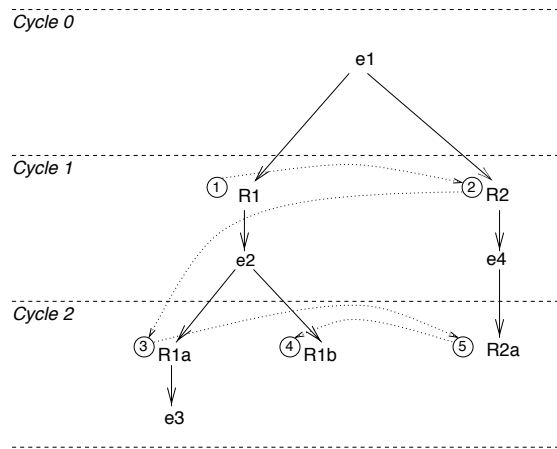


Figure 4.20: Example illustrating the scheduling of rules triggered by cascade triggering events

executed by the action implementing a NFP⁴ (i.e., whether they read and/or write the workflow memory space). From a general point of view a workflow can be seen as a process that executes read/write operations (i.e., activities) into a memory space. For example, figure 4.21 illustrates a workflow that operates on its own memory space (i.e., activities A, B, and C read/write workflow variables v_1 , v_2 , and v_3).

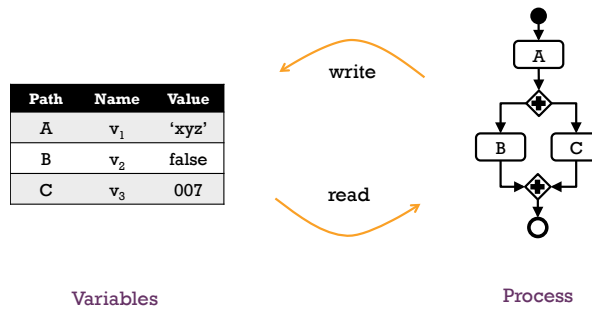


Figure 4.21: Workflow operating over its own memory space

In this work we consider that workflows can operate over two kinds of memory spaces: (i) the space composed of the set of variables belonging to a workflow, and (ii) the space composed of the data structures representing the activities of a workflow, and their order relationship (i.e., the workflow activity tree). We identify 3 kinds of relationship among workflows and a rule action: non-intrusive, memory intrusive, and control-flow and memory intrusive.

4.3.3.1 Non-intrusive relationship

In the non-intrusive relationship between a FR workflow (i.e., a functional requirement workflow that implements an application logic) and an NFR workflow (i.e., the one that implements a rule action), and action workflow only reads the variables of a workflow for implementing a non-functional property. In the Status Updater workflow, assume that the activity **Update Facebook** has an associated state management

4. In our approach an action is implemented by a workflow.

NFP for logging its state in a log (see figure 4.22). The activity starts its execution: it receives the value "xyz" associated to the variable `newStatus`, and assigns the value "abc" to the variable `oldStatus`. The Logging NFP workflow starts its execution reading these values from the execution state of the activity Update Facebook, and stores them in the log. Meanwhile, the activity Update Facebook continues its execution.

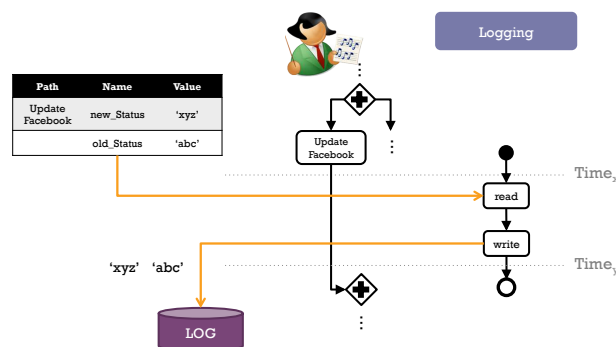


Figure 4.22: Non intrusive relationship example

4.3.3.2 Memory intrusive relationship

In the memory intrusive relationship the FR workflow shares its execution state with the NFR workflow that reads and writes on it when executed. The NFR workflow, instead, does not share its execution state with the workflow. The synchronization of both workflows is done giving access the their respective execution state. In the Status Updater example (see figure 4.23), the activity Update Facebook has an associated authentication NFP, where the activity first sends a new request. Then, it prepares an URI with the variable `newStatus` as output parameter. Before executing the request, the Authentication workflow that implements the Open Authentication Protocol, reads the user login and the password from the execution state of the activity Update Facebook shared with the workflow Authentication. Then, it produces a token stored in the variable `newStatus` of the state of the activity Update Facebook. The activity Update Facebook can call the service Facebook requesting a status update.

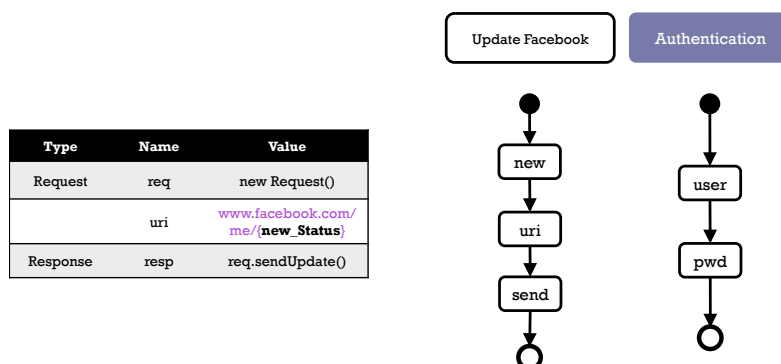


Figure 4.23: Memory intrusive relationship example

4.3.3.3 Control flow and memory intrusive relationship

In the control flow and memory intrusive relationship the workflows FR and NFR are synchronized by sharing their execution states in shared memory spaces, and by eventually modifying the NFR control flow. Both workflows read and write on the shared memory spaces.

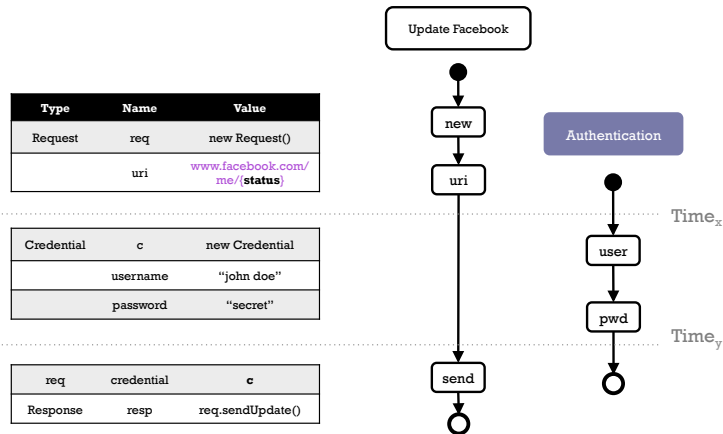


Figure 4.24: Memory intrusive relationship example

For example, assume now that the service Facebook updates its authentication protocol, and implements now an HTTP protocol, where a login and password are required (see figure 4.24). The Updater Status FR and NFR workflows are now synchronized as follows. The activity `Update Facebook` of the FR workflow starts creating a new request, then, provides an URI with the variable `status`. Doing this the control flow is now deviated for executing the authentication protocol by the workflow `Authentication`. It executes the activity `user` that requests the user login (e.g., variable `username` with value "john doe"), written on the execution state, and then the activity `pwd` for requesting the password that is also written on the execution state. Once executed, the control of the activity `Update Facebook` is resumed, and it executes the last activity `send`.

Adding exception handling to the Status Updater workflow is also an example of memory and control flow intrusive NFP, where the execution of the FR workflow must be observed. Then, the NFP must modify the control flow for catching the exception produced, and updating the control flow for recovering from the exception: signaling the exception, reexecuting the activity that produced the exception, rolling back the execution to a state previous to the production of the exception.

4.4 VIOLET: an Active Policy Based Workflow Engine

In order to execute active policy based workflows (i.e., workflow with associated active policies) we designed and implemented Violet, an active policy based workflow engine. The diagram of figure 4.25 shows VIOLET general architecture composed of 3 components:

- Workflow Engine for executing workflow instances and controlling their execution.
- Active Policy Engine for managing and executing active policy and rule instances.
- Event Service for monitoring and notifying the events produced by workflow and active policy instances.

Note that VIOLET has an external component called Definition Tool. This component is responsible of (i) receiving workflows and active policy definitions, and (ii) parsing them for generating the corresponding intermediate representations used by the Workflow Engine and Active Policy Engine. Also note that the Workflow Engine and Active Policy Engine components are the main components of Violet since they are responsible of executing the AP Model execution units. The following lines describe the architecture of these components.

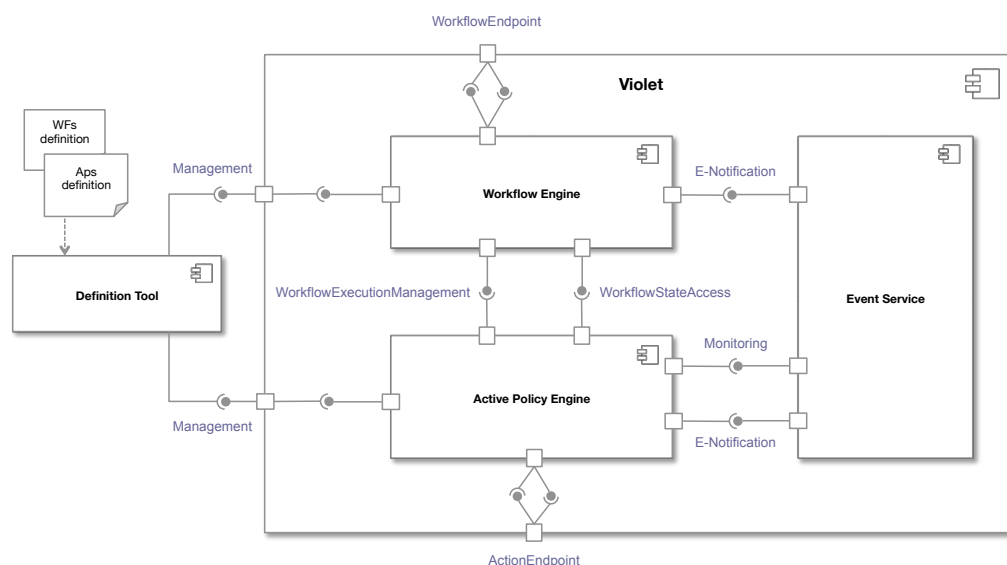


Figure 4.25: Execution environment global architecture

4.4.1 Workflow Engine

The Workflow Engine is composed of three main components (see Figure 4.26):

- Manager for controlling the lifecycle of workflow instances (e.g., it can pause and resume a workflow and/or its associated activities). The Manager is also responsible of controlling the execution of the Scheduler and Tracker components (e.g., it can interrupt their execution).
- Scheduler for executing workflows activity instances in the appropriate order. In particular, the Manager decides when to prepare workflow atomic activities, and when to call their associated services operations.
- Tracker for monitoring the execution of workflow and activity instances (e.g., when a workflow produces an event the Tracker detects the event and then, it forwards the event to the Event Service).

In this work we consider that any existing Workflow Engine respecting the architecture described above (the components and the interfaces) can interact with the Active Policy Engine of VIOLET as long as it complies with the following characteristics:

- Preemption right that enables the interruption of the execution a workflow instance at different points in time. Thus, it is possible to synchronize the execution of an active policy (and its rules) with the execution of the workflow activity instances.
- Activity atomic execution (i.e., the activity either commits or fails).
- Mutable state so that the execution state of the workflow can be modified arbitrarily. This is necessary

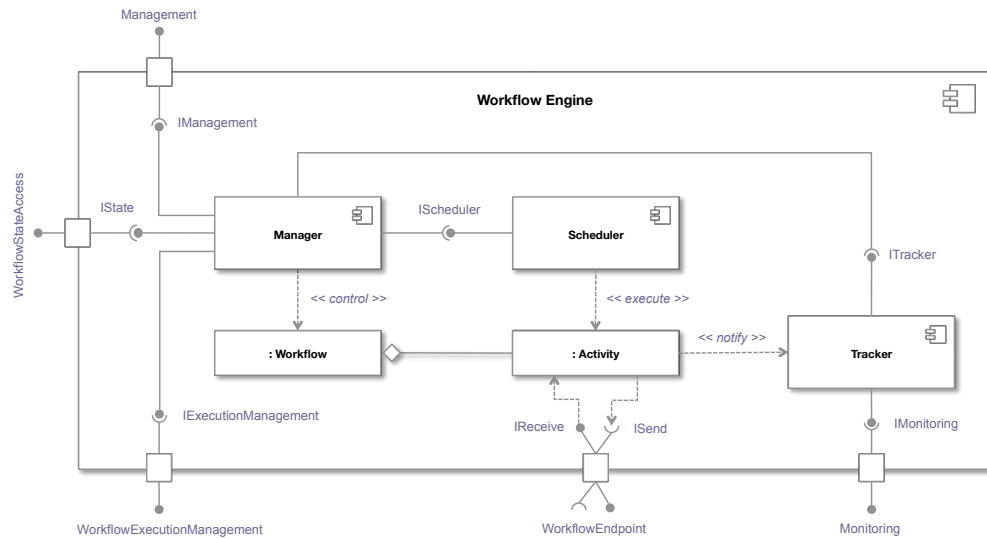


Figure 4.26: Workflow engine architecture

because after the execution of some rule actions the workflow execution state may change (e.g., when an activity is successfully retried with different input values).

4.4.2 Active Policy Engine

As shown in the diagram of figure 4.27, the Active Policy Engine is composed of three main components:

- Manager. Responsible of creating active policy instances, and managing their lifecycle (e.g., it activate and deactivate the policies).
- Scheduler for processing event notifications for determining (i) whether a policy rule has to be triggered, and (ii) the order of execution among the triggered rules. This component is also responsible of synchronizing the execution of a rule with the execution of the entity producing the event (e.g., if a workflow produced an event that triggered a rule, the Scheduler uses the WorkflowExecutionManagement interface for pausing the workflow execution, and resuming it after the rule completes).
- Action Engine which is a Workflow engine for executing the action of a policy rule (i.e., it executes the workflow implementing a rule action).

Note that the Active Policy Engine offers several interfaces. The following list enumerates the most important ones:

- Manager Interface. Offers operations for controlling the Active Policy Engine. In particular, this interface is used for controlling the Manager (e.g., using this interface it is possible to deactivate all the policies associated to a workflow).
- E-Notification Interface. Offers operations for notifying the events produced by active policies, policy rules and workflow action instances to the Event Service component.
- Monitoring Interface. Offers operations for receiving the events coming from the Event Service. These events are then sent to the Scheduler for processing.

In order to execute active policies the Active Policy Engine considers the following aspects:

- When to evaluate the rule conditions with respect to the notification of an event.
- How policies triggered at the same time are ordered.

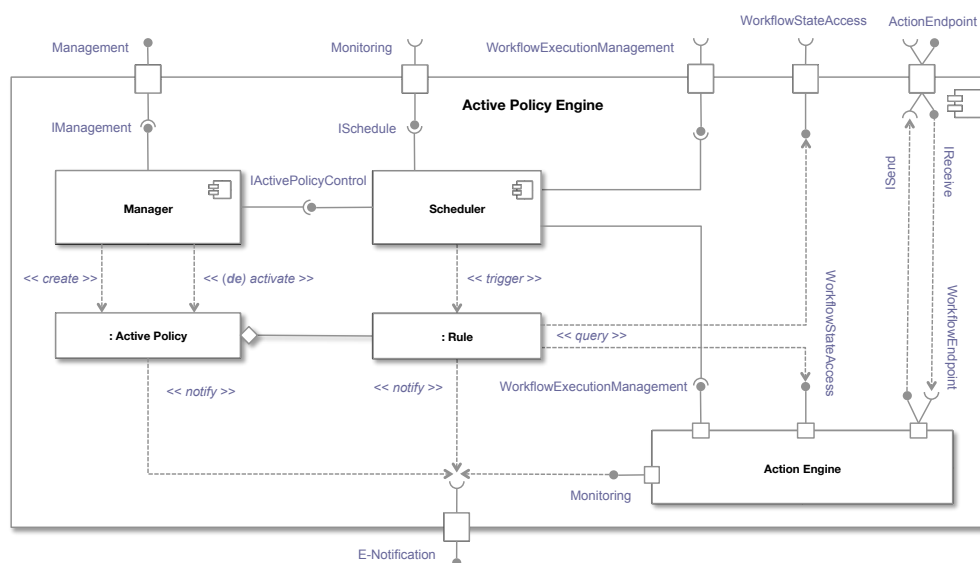


Figure 4.27: Active Policy Engine Architecture

— How to interact with the execution of a workflow for executing its associated policy rules (preemption).

Based on these aspects the Active Policy Engine works as follows: given a workflow implementing a services coordination and a set of active policies representing a non-functional property, the Active Policy Engine generates the code for evaluating every policy as well as the synchronization interface that interacts with the Workflow Engine. The code generation process is done by a policy compiler, which validates whether type declarations and policy expressions are well formed. The compiler also implements the transformation rules of AP Language expressions into AP Model types. The Active Policy Engine imports these types, and implements evaluation strategies for executing the policies. Then it uses the Event types defined in the policy types for constructing the Monitoring and E-Notification interfaces since event detection and event notification are specified on these interfaces.

Active policies express non-functional properties that must be evaluated at execution time, and within the execution of a workflow. We represent the execution of a workflow by a plan consisting of a set of execution units (i.e. activities and policies), and an order function. This plan is generated and used by the Scheduler when policies are triggered. The evaluation of a policy is done within two processes: events detection and execution strategies.

Events detection is used to observe the execution of an active policy based workflow and, to detect, notify and, store the events produced by this execution in a log. The log stores ordered instances of events produced during the execution of a workflow. The log also represents the execution state of a workflow, and is used to decide how to evaluate policies.

While a workflow is being executed, the Scheduler consumes the events that can trigger the policies associated to the workflow. Using the properties specified in these policies, the Scheduler uses the execution strategies for filtering the events and deciding which policies are triggered. Recovery actions are seen as activities that must be inserted into the execution of a workflow.

The evaluation of policies within the execution of a workflow introduces a best effort execution strategy. Indeed, the active policy evaluation is a process executed before the execution of the activities of a workflow, and can lead to the cancellation, re-execution or compensation of an activity. If for any reason an event

representing an exception is detected, the set of recovery actions will be triggered to treat the exception.

4.4.3 Experimental validation

Our results were experimentally validated in use cases where services coordinations are used for querying (project OPTIMACS), processing and mashing up data (projects E-CLOUDSS and CLEVER). Our experiment focused on a web application where the operations were executed ensuring fault tolerance (exception handling and recovery properties that lead to atomicity models), and security properties. We developed a second scenario focusing on data processing ensured by coordinating services. The objective was to focus on QoS properties to be ensured to data collections produced by services, and processed through workflows. For example ensuring their freshness and their validity. We showed how to use policies for ensuring these properties.

In both use cases we validated the use of Active Policy types related to fault tolerance, authentication, adaptability, and QoS properties associated to the data they process (freshness, validity, durability). In our experience, using the AP Model and its associated language for specifying AP types that implement an NFP can be easy as long as the programmer knows the protocol to express: (i) the events that must be observed for triggering an AP, (ii) the conditions to be verified in the execution state of the coordination, and (iii) the recovery actions to enforce the NFP. Policies provide independence of application logic and NFP, and this eases the maintenance of the service coordination. This is important because services can change their interfaces and their requirements (e.g., they can change the authentication protocol used for executing operations). Having AP modularizes these aspects, and help to maintain a service coordination only by associating new policies to it.

In order to provide guidelines for defining Active Policy Types, we proposed π -SODM [4]⁵, an MDD based methodology for building service compositions with non-functional requirements. π -SODM provides concepts for modelling NFP at the early stages of the development. π -SODM provides a conceptual structure to: first, capture the system requirements, and specification in high-level abstraction models (computation independent models, CIMs); next, starting from such models build platform independent models (PIMs) specifying the system details; next transform such models into platform specific models (PSMs) that bundles the specification of the system with the details of the targeted platform; and finally, serialize such model into the working-code that implements the system.

4.5 Discussion

Current standards in service composition implement functional, non-functional constraints and communication aspects by combining different languages and protocols. WSDL and SOAP among others are languages used respectively for describing services interfaces and message exchange protocols for calling methods exported by such services. For adding a transactional behaviour to a services coordination it is necessary to implement WS-Coordination, WS-Transaction, WS-BusinessActivity and WS-AtomicTransaction. The selection of the adequate protocols for adding a specific non-functional constraints to a service coordination (e.g., security, transactional behaviour, and adaptability) is responsibility of a programmer. As a consequence, the development of an application based on a services coordination is a complex and a time-consuming process. This is opposed to the philosophy of services that aims at facilitating the integration of distributed applications. Other works, like [FDDDB05], introduce a model for transactional services compo-

5. See Chapter 7 that gives a list of my major publications numbered according to these references.

sition based on an advanced transactional model. [BGP05] proposes an approach that consists of a set of algorithms and rules to assist designers to compose transactional services. In [VV04] the model introduced in [SABS02] is extended to web services for addressing atomicity.

As WS-* and similar approaches, our work enables the specification and programming of crosscutting aspects (i.e., atomicity, security, exception handling, persistence). In contrast to these approaches, our work specifies policies for a services coordination in an orthogonal way. Besides, these approaches suppose that NFPs are implemented according to the knowledge that a programmer has of a specific application requirements but they are not derived in a methodological way, leading to ad-hoc solutions that can be difficult to reuse. In our approach, once defined *A-Policies* for a given application they can be reused and/or specialized for another one with the same requirements or that uses services that impose the same constraints.

Therefore, in contrast to these approaches, our work is inspired by the philosophy of separation of concerns adopted in the construction of middleware for (i) specifying non functional aspects to a service coordination in an orthogonal way and, (ii) generating mechanisms for ensuring these properties at service coordination execution time.

Main contributions

Our research addressed challenges concerning the description and discovery of services, and their coordination for building data querying systems⁶. We proposed approaches [6] guided by semantics for discovering services by using formal approaches and logic programming methods, particularly in the PhD. work of Han TAN in 2009, and the master project of Gabriela MONTIEL-MORENO in 2008-2009. Our objective with these works was to understand the role of the services discovery process in the service based query evaluation process. Service discovery and matching were no longer in the centre of our scientific research and we mainly focussed on NFP for service compositions.

We proposed models and systems for representing non-functional properties of service coordinations and strategies for dynamically reinforcing them:

- We proposed a workflow engine based on Abstract State Machines (ASM) [21] in the context of the WebContent project. This engine served as the basis for the development of a service based query evaluation system as shown in the previous chapter.
- The A-Policy Model [11] for representing NFP and recovery actions, and associating them to service coordinations, and its associated methodology and implementation environment π -SODM [8].
- We implemented the prototype ROSE and VIOLET for adding atomicity properties to service coordinations. They interact with an existing workflow engine that exports methods for stopping, resuming, and rolling back the execution of a service coordination.
- We proposed an approach and associated system MEOBI [10] for controlling the authentication, non-repudiation, and message ciphering of service coordinations.
- We proposed alternative solutions for adding non functional properties to Web applications using AOP methods. We proposed MexADL for managing systems' maintainability (a non-functional aspect) in the master project of Juan Carlos CASTREJÓN (<http://code.google.com/p/mexadl/>).

Projects

- ECOS-ANUIES ORCHESTRA(<http://orchestra.imag.fr>)

6. See Chapter 7 that gives a list of my major publications numbered according to these references.

- WebContent (<http://www.webcontent.fr>)
- E-CLOUDSS(<http://e-cloudss.imag.fr>)
- CLEVER(<http://clever.imag.fr>).

PhD students

- BELHAJJAME Khalid, *Définition et orchestration des services ouverts pour la construction de systèmes d'information répartis*, INPG, France, Advisors: Ch. Collet et G. Vargas-Solar, June 2004
- VU Thi-Huong-Giang, *Composition sécurisée de services*. Advisors: Ch. Collet et G. Vargas-Solar, ED MSTII, INPG, October 2008
- HANH Tan, *Infrastructures à base d'événements pour la coordination évolutive de services*. Advisors: Ch. Collet et G. Vargas-Solar, ED MSTII, INPG, June 2009.
- PORTILLA-FLORES Alberto, *Coordination de services avec des propriétés transactionnelles*. Advisors: Ch. Collet, G. Vargas-Solar and J.L. Zechinelli- Martini, ED MSTII, INPG, October 2010.
- DE SOUZA NETO Placido Antonio, *A Methodology for Building Reliable Service Based Applications*, Universidade Federal do Rio Grande do Norte, December 2012 (associated to the projects CLEVER, fellowship CAPES), Advisors: M. Musicante, G. Vargas-Solar
- ESPINOSA-OVIEDO Javier A., *Contract based services coordination language*, double advisor framework: Grenoble INP, France; U. de las Américas, Puebla, Mexico, October 2013 (associated to the project ORCHESTRA and CLEVER, fellowship CONACyT Mexico), Advisors: Ch. Collet, J.L. Zechinelli-Martini, G. Vargas-Solar

Open source licenced prototypes

1. π -SODM environment, <http://clever.imag.fr>
2. MexADL, <http://code.google.com/p/mexadl/>
3. Web2MexADL, <http://code.google.com/p/web2mexadl/>
4. Model2Roo, <http://code.google.com/p/model2roo/>

CHAPTER 5

Towards economy oriented data management and processing

A businessman is a hybrid of a dancer and a calculator.

— Paul Valery.

In the information era users have to make decisions and generate knowledge by manipulating huge datasets. Organizations invest energy and time collecting, curating, and organizing data of different provenances and qualities. Curated datasets often remain private, and they are not always shared and published due to legal and economic reasons.

For example, in social networks like Facebook's timeline, people spend time organizing their important events on time, by defining sorts of checkpoints, tagging photos and other content. Then, other organizations access these datasets and exploit this information to their benefit. Data "owners" do not have any return of investment (ROI), and there is no explicit official "ownership", owners pass their rights to social networks. There is no clear business model (and contract) between data producers (e.g. Facebook users) and data holders (e.g. Facebook as a company).

Consider instead a scenario where data tagged with comments on consumers experiences and opinions about their quality and usefulness are exported as a data market with an associated cost model. For example, the cost model can specify a price according to:

1. the number of free accesses that can be executed on them;
2. a poker like "pay to see" model where the market exposes its catalogue but shares data of different qualities according to specific fees or memberships;
3. a cost per recurrent access to be executed on RSS depending on a data/loss rate;
4. the costs depending on whether the results are made persistent or not, open or not, continuously available or not.

Research on data processing and brokering are promising given the explosion of big datasets produced by different means and the requirements to consume them to have the right information, at the right place, at the right moment, with the right cost.

We consider that there are three main aspects to be considered for addressing this challenge. First, dealing with intensive big data management (collect, discover, and process big datasets according to available computing and storage resources). Second, making it possible to measure the implied execution cost (access data at the right moment) provided that several execution units are required for processing big data sets even if the operation is not computationally complex and economically costly (the right cost). Third, determining how to store big datasets provided that they require disk parks, and that they have to be available (at the right moment and place) and consistent (the right information). The following sections develop these three aspects. Section 5.1 discusses challenges introduced by the need for managing big data sets to enable data analytics processes. Section 5.2 discusses the implications of associating an economic model to data querying through the notion of data market. Section 5.3 discusses challenges and open problems of today's and future big data market places.

5.1 Intensive big data management

Cloud computing is emerging as a relatively new approach for dealing with and facilitating unlimited access to computing and storage resources for building applications. The underlying infrastructure manages such resources transparently without including code in the application for managing and reserving more resources than those really required. The difference with classic approaches is that the application can have an ad hoc execution context, and that the resources it consumes are not necessarily located in one machine. Thanks to the cloud properties, applications can have ad hoc execution contexts. Following the same approach, database management systems functions can be delivered as services that must be tuned and composed for efficiently and unexpensively managing, querying and exploiting huge data sets.

Cloud architectures provide services at different scales and add constraints for accessing data for instance, access control, resources reservation, and assignment using priorities (e.g., in grid architectures) and economic cost (e.g., in the cloud). Applications deployed in these architectures specify QoS preferences (SLA contracts) that include execution and processing time, data pertinence and provenance, economic cost, and data processing energy consumption cost.

Thus data management must be revisited for designing strategies that couple the characteristics of novel architectures with users' preferences. In this context we identify two key scientific challenges:

- Data (flows) access and processing guided by SLA contracts, where data are produced by services and devices connected on heterogeneous networks.
- Estimation and reduction of temporal, economic and energy consumption cost for accessing and processing data.
- Optimization of data processing guided by SLA contracts expressed using cost models as reference.

Our research contributes to the construction of service based data management systems. The objective is to design and implement data management services guided by SLA contracts for providing methodologies and tools for integrating, deploying, and executing value added services for programming data management functions. Value added services coordinate services taking into account QoS criteria like security, reliability, fault tolerance, and dynamic adaptability and behavior properties like transactions and consistency adapted to applications requirements (i.e., SLA contracts).

We propose approaches based on the coordination of services for programming data querying and retrieval on demand on the cloud. These approaches are inspired from existing techniques, and consider the characteristics of the cloud, and provide software engineering techniques for designing data management solutions. We focus on three cloud characteristics for addressing data processing and management (e.g.,

querying and retrieval):

1. The notion of service as construction and access unit to persistent or transient data.
2. SLA contracts and QoS measures associated with services for guiding data processing and resources consumption.
3. Map-Reduce model for parallel query execution and data retrieval processes.

We address the observation of services QoS measures in the context of a commercial cloud providers (i.e., Windows Azure) and data persistence on public cloud providers (i.e., Cloudfoundry, Openshift, and Xerund). We have deployed data management services on a multi-cloud architecture for processing big datasets using different NoSQL systems according to the type of operations applied on these collections: read/write oriented, data sharing, consistency level, among others. We validate our approach for processing energy big data collections in the context of the project SoGrid¹. We combine views management with aggregation, filtering, and big data correlations implemented using a Map-Reduce model. The following lines discuss issues to be addressed for computing measures.

5.1.1 Observing QoS measures

Data or event flows generated by producers such as wireless sensors and RFID readers, are raw data difficult to exploit by consumers. They must be combined or aggregated. Events from multiple producers, or from a single one during a given period of time can be composed and aggregated to produce more meaningful information, and notified to consumers according to different communication protocols (reliable, causal, atomic). Event composition is the process of detecting composite events from the occurrence of other events. A composite event is detected whenever a certain pattern of events is observed. The pattern can represent aggregation of previously detected events using temporal or causal relationships among events.

Several academic research and industrial systems have tackled the problem of event composition. Techniques such as complex pattern detection [GJS92, GD94a, CC96, PSB04], event correlation [Dou96, YB05], event aggregation [Luc02], event mining [AS95, GTB02], and stream processing [WDR06, DGP⁺07, BKKL07], have been used for composing events. In some cases event composition is done on event histories (e.g. event mining), and in other cases it is done dynamically as events are produced (e.g. event aggregation and stream processing). Nevertheless, to the best of our knowledge, there is no approach that integrates different composition techniques. Yet, pervasive and ubiquitous computing, require the observation of behavior patterns that can be obtained by aggregating and mining statically and dynamically huge event histories.

Our work proposes the distributed and continuous composition of event flows. Events can be stored in event histories or logs. An event history is a finite set of events ordered by their occurrence time. Because the number of produced events can reach thousands of events per second or higher [Luc02], the size of an event history can be huge, increasing the difficulty of its analysis for composing events. We propose a distributed event composition approach, done with respect to subscriptions managed as continuous queries, where results can also be used for further event compositions. According to the type of event composition strategy (i.e., aggregation, mining, pattern look up or discovery), event composition results can be notified as data flows or as discrete results. Event composition is done with respect to events stemming from distributed producers, and ordered with respect to a timestamp computed with respect to a global time line. Provided that our event approach aims at combining different strategies that enable the dynamic and postmortem event composition, we assume that different and distributed event histories can be used for detecting composite event patterns. For example, in order to calculate the overall load of a clustered system, the histories of the

1. <http://www.so-grid.com>

events representing memory and CPU consumption of each computer in the cluster have to be combined, and maybe integrated with events on streams. Thus, histories must be analyzed for relating and combining events to obtain the memory and CPU consumption of the cluster.

Event composition must handle complex subscriptions that integrate stream processing and database lookup to retrieve additional information. In order to do this kind of event composition, a number of significant challenges must be addressed. Despite the increasing sizes of event histories, event composition needs to be fast. Filtering, pattern matching, correlation and aggregation must all be performed with low latency. Our work proposes an event service that implements event composition by querying distributed histories ensuring scalability and low latency. Indeed, the need to detect and notify complex events from basic events is sometimes correlated with some quality of service requirements like latency, memory consumption, network occupancy, event priority, notification latency, etc. Those QoS requirements generally constrain the way the event processing must be achieved. In addition, they are not independent. For example, the reduction of network occupancy generally decreases the notification latency. Therefore, there exists trade-offs among these QoS metrics that need to be judiciously balanced by the event processing systems based on application needs. In order to achieve a QoS based complex event processing and notification, we model the dependencies among applications QoS requirements, and the characteristics of the event processing runtime.

In our event service applications subscribe to composite events by issuing complex event patterns to the system, with associated QoS requirements. The system determines the functionalities needed at each layer of the architecture, taking into account QoS, and dynamically deploys a set of corresponding event processing agents. The composite events generated by the event processing nodes are notified to consumers. Such service can act as a middleware on which utility applications can rely for detecting interesting or critical situations (sensors errors, alarms, etc.) with some QoS guarantees (e.g. priority, notification latency, etc.).

Finally, composition algorithms must deal with the processing of continuous event flows, and the discovery of systems behavior models using event histories in order to estimate their resources consumption and their cost. This information guides data management strategies for respecting SLA contracts. Efficient event flow processing can be costly, and it can require storage support, and the implementation of parallel solutions that can increase composition performance.

5.1.2 Dealing with multiple storage spaces

The use of heterogeneous data stores within a single system is gradually becoming a common practice in application development. Modern applications tend to rely on a polyglot approach to persistence, where traditional databases, non-relational data stores, and scalable systems associated with the emerging NewSQL movement, are used simultaneously.

As part of the emerging polyglot persistence movement [MP12], the simultaneous use of multiple scalable SQL, NoSQL, and NewSQL data stores within a single system is gradually becoming a common practice in modern application development [HHE⁺09, Mul12, Cat10]. Nonetheless, the combination of these heterogeneous databases, flexible schemas, and non-standard APIs represent an added complexity for application developers. For example, considering that the schemas used by these applications are spread across multiple data stores, each of which possibly relies on distinct data models (such as key-value, document, graph, etc.), developers must be familiar with a high number of implementation details, in order to effectively work with, and maintain the overall database model.

Due to the nature of schema-less data stores, developers also need to provide an adequate maintenance of the implicit schemas that these applications rely upon. This is due to the fact that the source code generally contains assumptions about the data structures used by the application (such as field names, types, etc.),

even if the data stores themselves do not enforce any particular schema [MP12]. Essentially, we consider that the schemas are shifted from the database to the application source code. However, having the data schemas as part of the application code can lead to maintenance and performance issues. For instance, developers have to manually analyze the full source code in order to effectively understand the data model used by these applications. This can be an error-prone activity, due to the combination of different programming styles, APIs, and development environments.

We propose an approach and tool named ExSchema² that enables the automatic discovery of schemas from polyglot persistence applications. The discovery mechanism is based on source code analysis techniques, particularly on API usage and on the analysis of standard data layer patterns. The tool receives application source code as input, containing invocations to the APIs of one or more data stores (graph, key-value, relational, and column). The ExSchema analyzers examine this application source code, and share their analysis results between each other. For example, in order to identify schema update methods, we first identify the declaration of variables. The schema fragments recovered by the code analyzers are grouped together, according to the data models supported by our tool, and by extending the translation mechanisms detailed in [ABR12], with the identification of relationships between graph entities. The discovered schemas are represented using a set of meta-layer constructs, and finally, this meta-layer representation is transformed into a PDF image, and a set of Spring Roo scripts [LM11]. This means that if, for example, the analyzed application relies on graph and document data stores, our tool will generate two schemas, one for each data model. Both schemas will be depicted in a unique PDF image and two Spring Roo scripts will be generated, one for each schema.

5.1.3 Parallel model for implementing data processing functions

A consensus on parallel and distributed database system architecture emerged in the 1990's. This architecture was based on a shared-nothing hardware design [SKPO88] in which processors communicate with one another only by sending messages via an interconnection network. In such systems, tuples of each relation in the database were partitioned (declustered) across disk storage units attached directly to each processor. Partitioning allowed multiple processors to scan large relations in parallel without the need for any exotic I/O devices. Such architectures were pioneered by Teradata in the late seventies, and by several research projects. This design is used by Teradata, Tandem, NCR, Oracle-nCUBE, and several other products. The research community adopted this shared-nothing dataflow architecture in systems like Arbre, Bubba, and Gamma.

The shared-nothing design moves only questions and answers through the network. Raw memory accesses and raw disk accesses are performed locally in a processor, and only the filtered (reduced) data is passed to the client program. This allows a more scaleable design by minimizing traffic on the interconnection network. The main advantage of shared-nothing multi-processors is that they can be scaled up to hundreds and probably thousands of processors that do not interfere with one another [DG92]. Twenty years later, Google's technical response to the challenges of Web-scale data management and analysis was the Google File System (GFS) [BCL12]. To handle the challenge of processing the data in such large files, Google pioneered its MapReduce programming model and platform [GGL03]. This model enabled Google's developers to process large collections of data by writing two user-defined functions, map and reduce, that the MapReduce framework applies to the instances (map) and sorted groups of instances that share a common key (reduce) similar to the sort of partitioned parallelism utilized in shared-nothing parallel query processing [BCL12].

2. <http://code.google.com/p/exschema/>

Yahoo!, Facebook, and other large Web companies followed suit. Taking Google's GFS and MapReduce papers as rough technical specifications, open-source equivalents were developed, and the Apache Hadoop MapReduce platform, and its underlying file system (HDFS, the Hadoop Distributed File System) emerged³. Microsoft's technologies include a parallel runtime system called Dryad [IBY⁺07], and two higher-level programming models, DryadLINQ [YIF⁺08] and the SQL-like SCOPE language [CJL⁺08b]. The Hadoop community developed a set of higher-level declarative languages for writing queries, and data analysis pipelines that are compiled into MapReduce jobs, and then executed on the Hadoop MapReduce platform. Popular languages include Pig from Yahoo! [ORS⁺08], Jaql from IBM⁴, and Hive from Facebook⁵. Pig is relational-algebra-like in nature, and is reportedly used for over 60% of Yahoo!'s MapReduce use cases; Hive is SQL-inspired and reported to be used for over 90% of the Facebook MapReduce use cases [BCL12].

Recent works agree on the need to study the Map-Reduce model for identifying its limitations and pertinence for implementing data processing algorithms like relational operators (i.e. join). New research opportunities are open in the database domain for studying different Map-Reduce models and proposing parallel programming strategies for accessing data that will consider the characteristics of the cloud, its economic model and the QoS requirements of applications. Our research studies declarative and procedural data processing languages like LinQ (Language Integrated Query⁶) and PigLatin⁷; and, propose compiling strategies for generating parallel programs for executing service coordination based queries. In our opinion, research must be done on the construction of querying and data retrieval services on the cloud (IaaS and PaaS layers) based on:

1. Services' coordinations.
2. Parallel programming models for ensuring the optimum use of resources on the cloud.
3. Declarative languages for expressing queries and data retrieval requirements; and associated compilers for generating optimized programs for querying and accessing huge volumes of data.

Our approach is to define a language that enables the expression of coordination of data processing operations implemented under a parallel model and test these mechanisms in big data analytics scenarios. According to [BCL12] a consistent challenge is that real Big Data clusters involve multi-user, multi-class (i.e., mixed job size) workloads not just one analysis job at a time or a few analysis jobs at a time; but a mix of jobs and queries of widely varying importance and sizes. The next generation of big data management services will have to propose approaches for dealing with such workloads effectively: *A long-standing open problem in the parallel database world, and in the Hadoop world* as stated in [BCL12].

In our research we intend to address this issue coupling annotations to query expressions declaring statically some requirements. We will be able to rewrite query expressions by using map - reduce programming patterns for structurally optimizing them. Then, at execution time, we use monitoring systems to maintain a continuous global view of the resources consumption of query execution. This global view will be used for guiding the assignment of computing resources of the execution environment.

5.2 Data market places: economic guided data management

The sale of data has existed since the middle of the 19th century, when Paul Reuter began providing telegraphed stock exchange prices between Paris and London, and New York newspapers founded the Asso-

3. <http://hadoop.apache.org>

4. <http://code.google.com/p/jaql/>

5. <http://hive.apache.org>.

6. <http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx>

7. <http://pig.apache.org/docs/r0.10.0/basic.html>

ciated Press [Dum12]. As the ability to discover and exchange data improves (thanks to the Web), the need to rely on aggregators such as Bloomberg or Thomson Reuters is declining [Dum12]. Some of the problems that consumers face with data are not just the question of storage, but also the question of access. Indeed, the business models of large aggregators do not scale to web consumers (e.g., start-ups), or casual use of data in analytics. Instead, data is increasingly offered through online marketplaces: platforms that host data from publishers and offer it to consumers. A data marketplace is a place where data can be programmatically searched, licensed, accessed, and integrated directly into a consumer application. One might call it the eBay of data or the iTunes of data. iTunes might be the better metaphor because it is not just the content that is valuable, but also the convenience of the distribution channel and the ability to pay for only what you will consume [Wal11]. Consumers will be able to build value-added services on top of data, rather than having to worry about gathering and storing the data themselves. What is required is a key framework that will catalyze data sharing, licensing, and consumption.

Our vision is that it is necessary to see data management that goes beyond accessing timely and costly ready to use data sources. It should be seen as an effort for collecting datasets of different qualities, and stemming from different processes, curating and delivering them. We shall call this environment a data market because we will assume that data brokers have an associated cost model. These brokers can be then contacted for accessing to data that can be processed for building new data collections that can be then made available in the data market. The key issues here are:

- being able to associate a cost model for the data market, i.e., associate a cost to raw data, and to processed data according on the amount of data and processing resources used for treating it, for instance;
- then being able to combine these cost models and the consumer expectations (service level agreement) with processing resources cost required by data processing;
- providing data management and brokering processing mechanisms under ad hoc business models.

5.2.1 Economic cost oriented data brokering

We will tackle economy-oriented data brokering process of large tagged data collections exported by data markets. We aim to propose a novel data brokering process that can coordinate the access to data markets guided by an economical model that can provide strategies guaranteeing data access, processing and delivery based on ad hoc business models.

Economy oriented data brokering will be tuned by the cost of accessing data markets, and the cost of using resources for brokering data. The challenge will be to manage the computation of results versus the cost of accessing completely or partially such results according to their completeness, data delivery frequency (continuous or one shot), their duration, and their quality:

- Completeness: a sample of resulting data that can be completed according to an economic model. This can be guided, for instance, by predefined fees (1M - 10euros, 10M - 15 euros), and the user can specify whether to buy data to complete results.
- Data delivery frequency: new data can be delivered while a data market proposes new data. It is up to the user to subscribe according to different fees.
- Duration: volatile/persistent results produced out of series of queries can be used for building a new data market. In this case the owner can require accessing and buying storage services for dealing with her data markets, and exporting them as paying services. The associated cost of such service will depend on the QoS, and the kind of Service level agreements that the service can honour.
- Content quality: data provenance, data freshness and degree of aggregation.

5.2.2 Defining economic business models for curating, managing, and brokering data in data market places

The greedy consumption of data by modern organizations opens up a new market of pre-processed data that can be curated and sold by brokers. Since data can be used for different purposes and in different scales, brokers should be able to adapt the way data are processed and delivered. These provision costs time and effort. Therefore, the product must be associated to business models that guide the data market. These models should take into consideration the technical, theoretical effort as well as the cost of resources used for pre-processing data collections and make them available with certain qualities.

For the time being four major data market providers offer solutions with subscription based models, as those used also by music providers and other current services today. For example DataMarket provides a data marketplace that is immediately useful for researchers and analysts with a strong emphasis on country data and economic indicators. Much of the data is available for free, with premium data paid at the point of use. DataMarket has recently made a significant play for data publishers, with the emphasis on publishing, not just selling data. Through a variety of plans, customers can use DataMarket's platform to publish and sell their data and embed charts in their own pages. Azure Data Marketplace has a strong emphasis on connecting data consumers to publishers and most closely approximates the popular concept of an iTunes for Data. Other data market places can be cited, such as Social data Gnip and Datasift specialize in offering social media data streams, in particular Twitter; Linked data Kasabi, currently in beta, is a marketplace that is distinctive for hosting all its data as Linked Data, accessible via web standard query languages such as SPARQL and RDF. Wolfram Alpha recently added a Pro subscription level that permits the end user to download the data resulting from a computation

Thanks to the cloud properties, applications can have ad hoc execution contexts. Our research relies on service oriented infrastructures for deploying a data brokering service for running tests that can evaluate the economic cost of the data processing and brokering process according to the consumed resources: data quality according to the data market that serves as provider, data volume, number, and frequency of access connections to data markets, storage consumption.

5.3 Discussion

An important observation to make is that in today's perspectives introduced by architectures like the cloud, and by movements like big data, there is an underlying economic model that guides directly the way they are addressed. This has not been a common practice in previous eras, but today the "pay as U go", the iTunes economic models have become an important variable of (big) data production, consumption, and processing.

Which is the value to obtain from big data? Big data is not a "thing" but instead a dynamic/activity that crosses many IT borders. Big data is not only about the original content stored or being consumed but also about the information around its consumption. Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis⁸. Even if technology has helped by driving the cost of creating, capturing, managing, and storing information the prime interest is economic: the trick is to generate value by extracting the right information from the digital universe.

The key is how quickly data can be turned into a currency by analyzing patterns and spotting relation-

8. <http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf>

ships/trends that enable decisions to be made faster with more precision and confidence; identifying actions and bits of information that are out of compliance with company policies can avoid millions in fines; proactively reducing the amount of data you pay (18,750 USD/gigabyte to review in eDiscovery) by identifying only the relevant pieces of information; optimizing storage by deleting or offloading non-critical assets to cheaper cloud storage thus saving millions in archive solutions⁹.

Current technology is not adequate to cope with such large amounts of data (requiring massively parallel software running on tens, hundreds, or even thousands of servers). Expertise is required in a wide range of topics including: machine architectures (HPC), service-oriented-architecture distributed/cloud computing, operating and database systems software engineering, algorithmic techniques and networking. This expertise must be put together with economic business models into the next generation of database management services conceived to address the big data challenge.

Main contributions

Our work has contributed to event flows composition in the context of the External Research Contract with France Telecom¹⁰. The objective was to study the distributed architecture for detecting and composing events. In order to profit from a distributed architecture it is also necessary to propose distributed event composition algorithms and notification protocols. This research is done in the projects CNRS-PEPS AIWS (<http://aiws.imag.fr>) and ADEME SoGrid.

As the volumes of data to be processed becomes large and heterogeneous with specific requirements, it is necessary to have well adapted storage strategies. We are currently working on the proposal of an efficient storage model on the cloud that can be guided by different constraints: execution time, economic cost, and energy consumption. The model will be validated on a cloud service oriented architecture, and will support scientific applications. this work is related to the PhD projects of Mohamad OTHMAN ABDALLAH and Juan Carlos CASTREJÓN [20, 21].

Some data processing operations can be greedy in terms of computing resources, and they require to be implemented in distributed or parallel infrastructures providing such resources. Data processing operations must be revisited so that they can exploit such resources, and provide results in reasonable time. The Map-Reduce model provides a theoretic framework for redesigning data processing operations. We are addressing this challenge of the master thesis of Matías Hernández of the Universidad de la República, Uruguay, and in the context of an associate professor internship assigned to Dr. Martin Musicante (associate professor of Universidade Federal Rio Grande do Norte) financed by the CNRS visits program.

Projects

- SoGrid, Development of innovative components for a communication system providing real - time control of Intelligent Public Electricity Distribution Networks, ADEME program Réseaux Electriques Intelligents.
- KeyStone (semantic keyword-based search on structured data sources). The objective is to provide tools for easily accessing structured repositories as they currently do with documents. Financed by the COST ICT program of the european union (contact: Francesco Guerra, University of Modena and Reggio Emilia - Italy).

9. *ibid.*

10. See Chapter 7 that gives a list of my major publications numbered according to these references.

- Swans (Semantic Web Analytics: Processing Big Data on Energy Consumption) proposes a big data processing service-based framework for supporting experts and non-experts analysis on energy consumption. Financed by the STICAMSUD CNRS program.

PhD students

- OTHMANN-ABDALLAH Mohamad, *Reliably mashing up data on the web*, Grenoble INP, France, February 2013 (financed by the project RED-SHINE BQR Grenoble INP and associated to the project CLEVER), Advisors: Ch. Collet et G. Vargas-Solar
- CASTREJÓN Juan-Carlos, *Efficient data storage on Cloud*, INPG, France, Advisors: Ch. Collet et G. Vargas-Solar (3rd year). Financed by a bourse d'excellence of the Doctoral School of University of Grenoble
- EPAL-NJAMEN Orléant, *QoS guided event streams composition and processing*, INPG, France, Advisors: Ch. Collet et G. Vargas-Solar (2nd year). Financed by the project SoGrid financed by the French Environment and Energy Management Agency (ADEME).

Licensed prototypes

1. ExSchema, <http://code.google.com/p/exschema/>

CHAPTER 6

Data management perspectives

Imagination is the beginning of creation. You imagine what you desire, you will what you imagine and at last you create what you will.

— George Bernard Shaw.

The challenge is to build an infinitely fast processor out of infinitely many processors of finite speed, and to build an infinitely large memory with infinite memory bandwidth from infinitely many storage units of finite speed [DG92].

This quote extracted from a famous paper of De Witt and Grey in 1992 seemed to be a premonition of the forthcoming emergence of the cloud. Some decades afterwards, the cloud opens new challenges for data querying and retrieval. It is no longer pertinent to reason with respect a to set of computing, storage, and memory resources. Today it is necessary to conceive algorithms and processes considering an unlimited set of resources usable via a "pay as U go model", energy consumption or services reputation and provenance models. Instead of designing processes and algorithms considering as threshold the resources availability, the cloud imposes to take into consideration the economic cost of the processes vs. resources use, results presentation through access subscription, and the parallel exploitation of available resources.

Terabyte online databases, at the same time, consisting of billions of records, are becoming common as the price of online storage decreases. These databases are not always represented and manipulated using the relational model. Parallel databases and today Map-Reduce oriented models show that partitioned execution offers good opportunities for speed and scalability [DG92]. By taking processing operations (filtering, correlation) and partitioning their inputs and outputs, it is possible to use a divide-and-conquer strategy to turn one big job into many independent little ones. This is an ideal situation for speedup and scaleup. Partitioned data is thus the key to partitioned execution.

Therefore, future data management challenges implies datasets partitioning including "smart" organization and indexing on file systems, data stores or memories. In general, partitioning strategies pertinence depends on analysis requirements. Current data analysis solutions, for instance promoted by the Map-Reduce (i.e., hadoop) scenarios, devote effort on "preparing" datasets (guiding their partitioning and indexing) for ensuring performance. This methodology leads to performant ad-hoc per problem solutions. The database approaches, in contrast, promote general "one-fits all" solutions where partitioning strategies can be tuned. In both cases, datasets preparation or DBMS tuning require effort, expertise, and a lot of testing for finding the best balance to achieve good data analysis performance. We believe that there is need for having optimized data management solutions and particularly, adaptable sharding and storage, that can support datasets preparation in order to reduce effort and ensure performant exploitation (retrieval, aggregation, analysis). In the following lines we discuss these aspects.

6.1 Adaptable datasets sharding and storage

Data sharding has its origins in centralized systems that had to partition files, either because the file was too big for one disk, or because the file access rate could not be supported by a single disk. Relational distributed databases use data partitioning when they place relation fragments at different network sites. Each partition forms part of a shard, which may in turn be located on a separate database server or physical location. Since the dataset is divided and distributed into multiple servers, the size of the dataset to be processed in each server is reduced. This reduces index size, which generally improves search performance. In addition, if the database shard is based on some real-world segmentation of the data (e.g. European customers vs. American customers) then it may be possible to infer the appropriate shard membership easily and automatically, and query only the relevant shard. Although it has been done for a long time by hand-coding this is often not elastic and not feasible when the number of servers exploits and the dataset grows.

Data sharding allows parallel database systems to exploit the I/O bandwidth of multiple disks by reading and writing them in parallel. Relational DBMS implement different strategies for distributing data (i.e., tuples) across fragments: round robin seems appropriate for processes accessing the relation by sequentially scanning all of it on each query, hash-partitioning seems suited for sequential and associative access to data avoiding the overhead of starting queries on multiple disks.

While partitioning is a simple concept that is easy to implement, it raises several physical database design issues. Each dataset must have a partitioning strategy and a set of disk fragments. Increasing the degree of partitioning usually reduces the response time for an individual query and increases the overall throughput of the system. In parallel DBMS for sequential scans, the response time decreases because more processors and disks are used to execute the query; for associative scans, the response time improves because fewer tuples are stored at each node, and hence the size of the index that must be searched decreases.

There is a need to support adaptable sharding automatically, both in terms of adding code support for it, and for identifying candidates to be sharded separately. Consistent hashing is one form of automatic sharding, used by web-scale companies to spread large loads across multiple smaller services and servers. According to the sharding criterion, it is possible that some data will be replicated and as new information is added about their profile the application has to ensure the consistency of the copies. Therefore, it is necessary to adopt synchronization criteria for enabling the tuning of data consistency.

NoSQL stores are putting this requirement back in the core challenges of today's data management. These stores are dealing with sharding and replication techniques in order to ensure data availability and fault tolerance. They rely on automatic replication mechanisms that sometimes imply data sharding. Reads and data querying implemented at the application level are then executed by applying map-reduce execution models. Research on map-reduce, and particularly on the hadoop platform are also developing approaches for obtaining better scaleup and speedup measures with data analytics cases that imply big datasets. Datasets analysis are required by social networks, the Web, content managers, e-science, economy, online gaming (e.g. viral marketing, profiling, recommendation systems, content providers, retail applications). The database community is also providing solutions through Big Data Management Systems¹, that use parallel DBMS know-how and integrate plug-ins to execute data analytics programs developed for other platforms (e.g., hadoop programs).

The NoSQL momentum introduces new datasets storage possibilities. One of the characteristics with big datasets is that curation and preparation are costly processes that can rely in data models that can

1. <http://asterix.ics.uci.edu>

cope with the original structure of the data. Graph, key-value, multidimensional records stores can provide storage solutions with efficient read/write operations possibilities. Partitioning can be then also guided by the structure of data, and it can be coped to ad-hoc stores that can ensure persistency and retrieval. This implies also a tuning effort of different NoSQL stores that should then provide efficient that reads/writes with specific degrees of availability, fault tolerance, and consistency. Existing work on hadoop, for example, couples the HDFS (Hadoop Distributed File System) with Vertica and MySQL cluster systems. Having different NoSQL stores, providing sharding solutions for a given data analytics layer seems to be a promising perspective for data management.

6.2 Efficiently querying datasets

Important work has been done around data querying in the database domain and on data analytics community. In our work we contributed with an approach that combines classic data querying (spatio-temporal, mobile/static, continuous, recurrent) with information search guided by quality of service criteria (precision, pertinence, cost, energy consumption). Yet there are still challenges to be addressed.

Our service oriented query evaluation approach based on the notion of query workflow, opened interesting opportunities for addressing efficient data querying and analysis. For the evaluation of a given hybrid query, it is highly desirable to examine equivalent alternative query workflows, since operator reorderings can lead to significantly better performance. The BP-GYO algorithm that we proposed could potentially facilitate the exploration of the search space of query workflows, because different choices of the hyperedges that are consumed (which are now arbitrary) lead to different query workflows. It is important to consider, however, that in a dynamic environment the statistics commonly used for optimization will not be readily available, and QoS criteria become predominant.

In our approach, query operators can be evaluated via composite computation services that implement operator algorithms. In a dynamic environment, different basic computation services can be available that could serve to implement, via different composite computation services, different algorithms for the same operator. The choice of different computation services thus represents an optimization opportunity, in which the optimal choice is not only linked to the algorithm *per se*, but also to the basic computation services that implement it (i.e., their cost, access time, etc.).

Once a query workflow has been created it needs to be executed on one or several machines by means of multiple processes. The operators scheduling determines when and how long for each of the various operators assigned to a particular machine or process is executed. An adequate scheduling policy represents an opportunity for optimization. Operator localization, on the other hand, implies determining which operators will be assigned to each machine available. In this case the problem is more complex than in traditional settings, since not only query operators but the various computation services involved. We believe that both scheduling and localization could benefit from a cost model specified using queueing theory. Then the various policies and algorithms available in the literature could be tested for scheduling. For operator localization, constraint-based logic programming could be employed to determine an optimal solution.

In some cases, the data obtained from data or computation services can be cached, thus avoiding potentially expensive operation calls. A highly ambitious alternative involves optimization over a global view of a query workflow. This implies considering all the interdependencies between the activities of the various operator workflows that participate in a query workflow. Optimization techniques could then be applied at this lower-level granularity. In this case the techniques used for the optimization of programming languages could prove useful.

Further performance improvements in our approach could be obtained by the adoption of intra-operator parallelism. In principle, our approach is compatible with intra-operator parallelism, since we specify query operators at a finer granularity by employing composite computation services. With the appropriate parallel algorithms handling data partitioning, a greater number of computation services could be exploited to evaluate query operators.

6.3 Big datasets management

Although relatively recent as a topic, researchers, practitioners, and the general public are aware of Big Data, which denotes large amounts of variable datasets eventually generated at high pace. This data overload is partly due to falls in the prices of data acquisition and storage devices. For instance, sensors are now ubiquitous, and are used to capture different kinds of information ranging from water and energy consumption to variations in human mood and emotion.

Datasets availability introduce new opportunities for data consumers. For example, scientists can investigate new hypothesis or inspect the reproducibility of published results using datasets shared by their peers. Datasets production and availability pose challenges:

1. Cost affordable datasets collection, analysis, and curation to enable data exploitation.
2. Energy- and cost-aware data processing and manipulation (in terms of computational power, time, cost).
3. Transmission of large datasets along distributed process units avoiding bottlenecks.

To address these challenges the database community proposed scalable solutions for storing and querying large amounts of datasets. Through the linked data initiative, the semantic Web community created a large data corpus. The High-Performance community proposed new cloud-based techniques focusing on the elastic provisioning of computing and storage resources. The eScience community focused on practical aspects, e.g., data curation and reuse.

While useful, research contributions of the above communities are often fragmented, uncoordinated. Moreover, they are often point-based and disconnected solutions that do not cover the whole lifecycle of Big Data Management, which comprises the collection, curation, and exploitation of datasets. There is therefore a need for tight interactions between the different ICT fields to propose complementary and coordinated solutions covering the Big Data management lifecycle. We consider that the following research tasks will result in innovative and original results:

- Develop cost models that assess and predict resources needed for data storage and processing to allocate just enough resources. Investigate new elastic and energy-aware models for estimating (and allocating) resources necessary for storing and processing large datasets.
- Explore new strategies and algorithms to enable the transfer of large data sets between distributed processing units. We will explore techniques such as data streaming and process transfer instead of bulk data transfer, as potential solutions.
- Develop algorithms for transforming Big Data into Smart Data through annotation, indexing, abstraction, and summarization to facilitate data interpretation.
- Explore provenance as a means for enabling Big Data Publication: investigate how provenance information specifying data lineage can be used to leverage (and enforce) ownership, credit, citation, privacy, access policies of Big Data.
- Adapt existing data integration, mining, and querying techniques to Big Data sets to assist querying and analysis of large amounts of data sets.

CHAPTER 7

Publications

This is a selected list of the major publications of the topics of my research, ordered by relevance. They are referenced across the document through numbered references. A complete list of publications can be found in the extended version of my curriculum vitae.

Evaluation of hybrid queries by coordinating services

- 1 V. Cuevas-Vicenttin, G. Vargas-Solar, C. Collet, Evaluating Hybrid Queries through Service Coordination in HYPATIA, In Proceedings of the 15th International Conference on Extending Database Technology (EDBT Demo Session), Berlin, Germany, 2012 [acceptance rate: 22.5%]
- 2 V. Cuevas-Vicenttin, G. Vargas-Solar, C., Collet, Web Services Orchestration in the WebContent Semantic Web Framework, Proceedings of the 9th International Mexican Conference on Computer Science, IEEE, Mexicali, Mexico, 2008 (best paper award) [acceptance rate: 31.6%]
- 3 T. Delot, S. Ilarri, M. Thilliez, G. Vargas-Solar, S. Lecomte, Multi-scale query processing in vehicular networks, In Journal of Ambient Intelligence and Humanized Computing, Springer Verlag, ISSN 1868-5137, 2(3), 2011, pp. 213-226 [H Index: 7]
- 4 V. Cuevas-Vicenttin, C. Collet, G. Vargas-Solar, N. Ibrahim and C. Bobineau, Coordinating services for accessing and processing data in dynamic environments, In Proceedings of the OTM 2010 Conferences, COOPIS 2010, LNCS, 2010 [acceptance rate: 19.6%]
- 5 T. Delot, S. Ilarri, M. Thilliez, G. Vargas-Solar, M. Bellengier, Highly mobile query processing, In proceedings of The International Conference on Ambient Systems, Networks and Technologies, ACM, 2010 [acceptance rate: ca. 35%]
- 6 G. Montiel Moreno, J.L. Zechinelli-Martini and G. Vargas-Solar, Modelling autonomic dataspace using answer sets, Revista Iberoamericana de Inteligencia Artificial, 14(48), pp 3-14, 2010 (extended version)
- 7 G. Vargas-Solar, Noha Ibrahim, C. Collet, M. Adiba, J.M. Petit, T. Delot, Querying Issues in Pervasive Environments, Book chapter in Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications, Apostolos Malatras Editor, IGI Global, 2010

Reliable data services coordination

- 8 Valeria de Castro, Martin A. Musicante, Umberto Souza da Costa, Placido A. de Souza Neto, and Genoveva Vargas-Solar, Supporting Non-Functional Requirements in Services Software Development Process: An MDD Approach, In Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science, LNCS Springer Verlag, High Tatras, Slovakia, January, 2014 (to appear) [classification B1 in the CAPES of Brazil]

- 9 A. Portilla, Ch. Collet, G. Vargas-Solar, J.L. Zechinelli-Martini, L. Garcia-Banuelos, Contract based behavior model for services coordination, Lecture Notes in Business Information Processing, Springer Verlag, 2008 [acceptance rate : 13,5%]
- 10 T.H.G. Vu, Ch. Collet, G. Vargas-Solar, Defining and Modelling Secure Service-based Systems, In Proceedings of On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, LNCS 4275, Springer Verlag, Montpellier, France, November, 2006 [acceptance rate : 28%]
- 11 J. A. Espinosa-Oviedo, G. Vargas-Solar, J.L. Zechinelli-Martini and C. Collet, Securely coordinating services using contracts, In Proceedings of the 10th Mexican International Conference in Computer Science, Workshop WSEC, IEEE, 2009 [acceptance rate: 28,72%]
- 12 K. Belhajjame, G. Vargas-Solar, Ch. Collet, Intégration de services : une analyse structurée, "Ingénierie des Systèmes d'Information", Numéro spécial sur les Services Web, 2005
- 13 K. Belhajjame, G. Vargas-Solar, C. Collet, "Defining and coordinating open-services using workflows", In Proceedings of the Eleventh International Conference on Cooperative Information Systems (COOPIS03), LNCS, Catania Sicily, Italy, November, 2003 [acceptance rate: 1/4]
- 14 K. Belhajjame, G. Vargas-Solar, C. Collet, Building information systems by orchestrating open services, In Proceedings of the International Database Engineering and Applications Symposium IDEAS, IEEE, Montreal, Canada, July, 2005 [classification B1 in the list of the CAPES of Brazil]
- 15 K. Belhajjame, G. Vargas-Solar, C. Collet, "Pyros – an environment for building and orchestrating open services", In Proceedings of the International Conference on Services Computing SCC'05, IEEE, Florida, USA, July, 2005 [acceptance rate: 18.6%]
- 16 G. Vargas-Solar, C. Collet, "ADEES, Adaptable and extensible event service", In Proceedings of the 13th International Conference on Database Expert Systems and Applications (DEXA'02), France, September, 2002. [classification B1 in the list of the CAPES of Brazil]
- 17 C. Collet, G., Vargas-Solar, H., Grazziotin-Ribeiro "Active Services for data-intensive distributed applications". In Proceedings of the International Symposium (IDEAS'2000), IEEE, Yokohama, Japan, September, 2000 [classification B1 in the list of the CAPES of Brazil]

Data integration

- 18 G. Bruno, Ch. Collet, G. Vargas-Solar, Towards intelligent mediators configuration using ontologies, In Proceedings of the Second ICSNW'06, LNCS, Springer Verlag, Munich, Germany, March, 2006

Polyglot persistence

- 19 G. Vargas-Solar, C. Ferreira da Silva, P. Ghodous, J.L. Zechinelli-Martini, Moving energy consumption control into the cloud by coordinating services, International Journal of Computing Applications, Special Issue, 205(4), December, 2013 [1988-2013, H Index: 15]
- 20 J. Castrejon, G. Vargas-Solar, C. Collet, and R. Lozano, ExSchema: Discovering and Maintaining Schemas from Polyglot Persistence Applications, In Proceedings of the International Conference on Software Maintenance, Demo Paper, IEEE, 2013
- 21 J. Castrejon, G. Vargas-Solar, C. Collet, and R. Lozano, Model2Roo: Web Application Development based on the Eclipse Modeling Framework and Spring Roo, In Proceedings of the First Workshop on Academics Modeling with Eclipse ACME, Denmark, 2012

e-Science

- 22 G. Vargas-Solar, J.L. Zechinelli-Martini, V. Cuevas-Vicenttin, The e-GrOV data Grid: a step towards the Mexican Virtual Observatory, Special Issue on Data Management in Grid and P2P Systems, Journal of Computer Systems Science and Engineering, 23(2), 2008, pp. 107-119. [2006-2013, H Index: 8]
- 23 V. Cuevas Vicenttin, J.L. Zechinelli-Martini, ANDROMEDA: Building e-Science Data Integration Tools, In Proceedings of the International Conference on Database Expert Systems and Applications DEXA'06, LNCS 4080, Springer Verlag, Krakow, Poland, September, 2006 [acceptance rate: 90/296 = 30,4%]

Bibliography

- [ABC⁺76] Morton M Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, JN Gray, Patricia P. Griffiths, W Frank King, Raymond A. Lorie, Paul R. McJones, James W. Mehl, et al. System R: relational approach to database management. *ACM Transactions on Database Systems (TODS)*, 1(2):97–137, 1976.
- [ABD⁺89] Malcolm P Atkinson, Francois Bancilhon, David J DeWitt, Klaus R Dittrich, David Maier, and Stanley B Zdonik. The object-oriented database system manifesto. In *DOOD*, volume 89, pages 40–57, 1989.
- [ABR12] Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. Uniform access to non-relational database systems: the SOS platform. In *Advanced Information Systems Engineering*, pages 160–174. Springer, 2012.
- [ABW06] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.
- [Adi07] Michel Adiba. Ambient, Continuous & Mobile Data. Presentation, 2007.
- [AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering, 1995.*, pages 3–14. IEEE, 1995.
- [ASM⁺12] Foto N. Afrati, Anish Das Sarma, David Menestrina, Aditya G. Parameswaran, and Jeffrey D. Ullman. Fuzzy joins using mapreduce. In *ICDE*, pages 498–509, 2012.
- [Bat88] Don S Batory. Concepts for a database system compiler. In *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 184–192. ACM, 1988.
- [BBG⁺88] DS Batoory, JR Barnett, Jorge F Garza, Kenneth Paul Smith, K Tsukuda, BC Twichell, and TE Wise. GENESIS: An extensible database management system. *IEEE Transactions on Software Engineering*, 14(11):1711–1730, 1988.
- [BCL12] Vinayak Borkar, Michael J Carey, and Chen Li. Inside big data management: ogres, onions, or parfaits? In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 3–14. ACM, 2012.
- [BCVS06] Gennaro Bruno, Christine Collet, and Genoveva Vargas-Solar. Configuring intelligent mediators using ontologies. In *EDBT Workshops*, pages 554–572, 2006.
- [BDK92] François Bancilhon, Claude Delobel, and Paris C. Kanellakis, editors. *Building an Object-Oriented Database System, The Story of O2*. Morgan Kaufmann, 1992.
- [BEHK10] Dominic Battr’è, Stephan Ewen, Fabian Hueske, and O Kao. Nephele/Pacts: a programming model and execution framework for web-scale analytical processing. *Proceedings of the 1st Idots*, 2010.
- [BGP05] Sami Bhiri, Claude Godart, and Olivier Perrin. Reliable Web Services composition using a transactional approach. In IEEE International, editor, *O. e-Technology, e-Commerce and e-Service*, volume 1 of *eee*, pages 15–21, March 2005.

- [BJKS06] Rimantas Benetis, S. Jensen, Gytis Kariauskas, and Simonas Saltenis. Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal*, 15(3):229–249, September 2006.
- [BKKL07] Magdalena Balazinska, YongChul Kwon, Nathan Kuchta, and Dennis Lee. Moirae: History-enhanced monitoring. In *Proc. of the Third CIDR Conf*, 2007.
- [BKKM00] Sandeepan Banerjee, Vishu Krishnamurthy, Muralidhar Krishnaprasad, and Ravi Murthy. Oracle8i-the XML enabled data management system. In *Proceedings of the 16th International Conference on Data Engineering, 2000*, pages 561–568. IEEE, 2000.
- [Bla94] José A. Blakeley. Open object database management systems. In *SIGMOD Conference*, page 520, 1994.
- [Bla96a] José A Blakeley. Data access for the masses through ole db. In *ACM SIGMOD Record*, volume 25, pages 161–172. ACM, 1996.
- [Bla96b] José A Blakeley. OLE DB: a component DBMS architecture. In *Proceedings of the Twelfth International Conference on Data Engineering, 1996*, pages 203–204. IEEE, 1996.
- [BP98] José A Blakeley and Michael J Pizzo. Microsoft universal data access platform. In *ACM SIGMOD Record*, volume 27, pages 502–503. ACM, 1998.
- [BSSJ99] Rasa Bliujute, Simonas Saltenis, Giedrius Slivinskas, and Christian S Jensen. Developing a datablade for a new index. In *Proceedings of the 15th International Conference on Data Engineering, 1999.*, pages 314–323. IEEE, 1999.
- [CAB⁺13] Christine Collet, Bernd Amann, Nicole Bidoit, Mohand Boughanem, Mokrane Bouzeghoub, A. Doucet, David Gross-Amblard, Jean-Marc Petit, Mohand-Said Hacid, and Genoveva Vargas-Solar. De la gestion de bases de données à la gestion de grands espaces de données. *Ingénierie des Systèmes d’Information*, 18(4):11–31, 2013.
- [CAH05] Daniela Claro Barreiro, Patrick Albers, and Jin-kaio Hao. Selecting web services for optimal composition. In *International Conference on Web Services (ICWS05)*, 2005.
- [Cat10] Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2010.
- [Cat11] Rick Cattell. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
- [CBB⁺04] Christine Collet, Khalid Belhajjame, Gilles Bernot, Christophe Bobineau, Gennaro Bruno, Béatrice Finance, Fabrice Jouanot, Zoubida Kedad, David Laurent, Fariza Tah, Genoveva Vargas-Solar, Tuyet-Trinh Vu, and Xiaohui Xue. Towards a mediation system framework for transparent access to largely distributed sources. the mediagrid project. In *ICSNW*, pages 65–78, 2004.
- [CC96] Christine Collet and Thierry Coupaye. Primitive and composite events in NAOS. *Actes des 12e Journées Bases de Données Avancées*, pages 331–349, 1996.
- [CDF⁺91] Michael J. Carey, David J. DeWitt, Daniel Frank, Goetz Graefe, Joel E. Richardson, Eugene J. Shekita, and M. Muralikrishna. The architecture of the EXODUS extensible dbms. In *On Object-Oriented Database System*, pages 231–256. 1991.
- [CH90] Michael Carey and Laura Haas. Extensible database management systems. *ACM SIGMOD Record*, 19(4):54–60, 1990.
- [CJL08a] Ronnie Chaiken, Bob Jenkins, and Larson. Scope: easy and efficient parallel processing of massive data sets. *VLDB*, 2008.

- [CJL⁺08b] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, 2008.
- [Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.
- [Col00] Christine Collet. The NODS project: Networked open database Services. In *14th European Conference on Object-Oriented programming (ECOOP-2000)*, June 2000.
- [CR94] Panos K Chrysanthis and Krithi Ramamritham. Synthesis of extended transaction models using acta. *ACM Transactions on Database Systems (TODS)*, 19(3):450–491, 1994.
- [CV04] Christine Collet and Tuyet-Trinh Vu. QBF: A query broker framework for adaptable query evaluation. In *FQAS*, pages 362–375, 2004.
- [CVSGR00] Christine Collet, Genoveva Vargas-Solar, and Helena Grazziotin-Ribeiro. Open active services for data-intensive distributed applications. In *IDEAS*, pages 349–359, 2000.
- [CVVSCB09] Victor Cuevas-Vicenttin, Genoveva Vargas-Solar, Christine Collet, and Paolo Bucciol. Efficiently coordinating services for querying data in dynamic environments. *Mexican International Conference on Computer Science*, 0:95–106, 2009.
- [DCC⁺01] Stefan Dessloch, Weidong Chen, Jyh-Herng Chow, You-Chin Fuh, Jean Grandbois, Michelle Jou, Nelson Mendonça Mattos, Raiko Nitzsche, Brian T. Tran, and Yun Wang. Extensible indexing support in db2 universal database. In *Component Database Systems*, pages 105–138. 2001.
- [DDGJ01] Wijnand Derks, Juliane Dehnert, Paul Grefen, and Willem Jonker. Customized atomicity specification for transactional workflows. In *Proceedings of the International Symposium on Cooperative Database Systems and Applications*, pages 155–164. IEEE, April 2001.
- [DFKR99] Hasan Davulcu, Juliana Freire, Michael Kifer, and IV Ramakrishnan. A layered architecture for querying dynamic web content. In *ACM SIGMOD Record*, volume 28, pages 491–502. ACM, 1999.
- [DG92] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.
- [DG00] Klaus R Dittrich and Andreas Geppert. *Component database systems*. Morgan Kaufmann, 2000.
- [DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):1–13, 2008.
- [DGL86] Klaus R. Dittrich, Willi Gotthard, and Peter C. Lockemann. DAMOKLES - a database system for software engineering environments. In *Advanced Programming Environments*, pages 353–371, 1986.
- [DGP⁺07] Alan Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, Walker M White, et al. Cayuga: A general purpose event monitoring system. CIDR, 2007.
- [DKA⁺86] Peter Dadam, Klaus Kuespert, F. Andersen, Henk M. Blanken, R. Erbe, Juergen Guenauer, Vincent Y. Lum, Peter Pistor, and Georg Walch. A DBMS prototype to support extended NF2 relations: An integrated view on flat tables and hierarchies. In *SIGMOD Conference*, pages 356–367, 1986.

- [DKH92] Pamela Drew, Roger King, and Dennis Heimbigner. A toolkit for the incremental implementation of heterogeneous database management systems. *The VLDB Journal, The International Journal on Very Large Data Bases*, 1(2):241–284, 1992.
- [DM97] Stefan Dessloch and Nelson Mattos. Integrating SQL databases with content-specific search engines. In *VLDB*, volume 97, pages 528–537, 1997.
- [Dou96] Christophe Dousson. Alarm driven supervision for telecommunication network: li-on-line chronicle recognition. In *Annales des Télécommunications*, volume 51, pages 501–508. Springer, 1996.
- [Dum12] Edd Dumbill. Data markets compared, a look at data market offering from four providers. *O’Reilly Strata Making Data Work*, page <http://strata.oreilly.com/2012/03/data:markets:survey.html>, March 2012.
- [DW98] Desmond F D’souza and Alan Cameron Wills. *Objects, components, and frameworks with UML: the catalysis approach*, volume 1. Addison-Wesley Reading, 1998.
- [FDDB05] Marie-Christine Fauvet, Helga Duarte, Marlon Dumas, and Boualem Benatallah. Handling transactional properties. In Springer-Verlag LNCS, editor, *WISE 2005: 6th International Conference on Web Information Systems Engineering*, volume 3806, pages 273–289, Octobre 2005.
- [FGD98] Hans Fritschi, Stella Gatzui, and Klaus R Dittrich. FRAMBOISE: an approach to framework-based active database management system construction. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 364–370. ACM, 1998.
- [FLMS99] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 311–322, New York, NY, USA, 1999. ACM Press.
- [FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, June 2003.
- [Fro98] Stuart Frost. *Component-based development for enterprise systems: applying the SELECT perspective*. Cambridge University Press, 1998.
- [FS97] Mohamed Fayad and Douglas C Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.
- [GBDC03] Luciano García-Bañuelos, Phuong-Quynh Duong, and Christine Collet. A component-based infrastructure for customized persistent object management. In *DEXA Workshops*, pages 536–541, 2003.
- [GD87] Goetz Graefe and David J DeWitt. *The EXODUS optimizer generator*, volume 16. ACM, 1987.
- [GD94a] Stella Gatzui and Klaus R Dittrich. Detecting composite events in active database systems using petri nets. In *Proceedings Fourth International Workshop on Research Issues in Data Engineering, 1994. Active Database Systems.*, pages 2–9. IEEE, 1994.
- [GD94b] Andreas Geppert and Klaus R Dittrich. Constructing the next 100 database management systems: like the handyman or like the engineer? *ACM SIGMOD Record*, 23(1):27–33, 1994.
- [GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

- [GHKM94] Dimitrios Georgakopoulos, Mark Hornick, Piotr Krychniak, and Frank Manola. Specification and management of extended transactions in a programmable transaction environment. In *Proceedings. 10th International Conference Data Engineering, 1994*, pages 462–473. IEEE, 1994.
- [GJS92] Narain H Gehani, Hosagrahar V Jagadish, and Oded Shmueli. Composite event specification in active databases: Model & implementation. In *Proceedings of the International Conference on Very Large Data Bases*, pages 327–327. Citeseer, 1992.
- [GM93] Goetz Graefe and William J McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth International Conference on Data Engineering, 1993*, pages 209–218. IEEE, 1993.
- [GMPQ⁺97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of intelligent information systems*, 8(2):117–132, 1997.
- [Gra95] Goetz Graefe. The Cascades framework for query optimization. *Data Engineering Bulletin*, 18(3):19–29, 1995.
- [GSD97] Andreas Geppert, Stefan Scherrer, and Klaus R Dittrich. KIDS: Construction of database management systems based on reuse. *University of Zurich*, 1997.
- [GTB02] Attilio Giordana, Paolo Terenziani, and Marco Botta. Recognizing and discovering complex events in sequences. In *Foundations of Intelligent Systems*, pages 374–382. Springer, 2002.
- [Guz00] L Guzenda. Objectivity/DB—a high performance object database architecture. In *Workshop on High performance object databases*, 2000.
- [Hae05] Theo Haerder. DBMS architecture—still an open problem. In *BTW*, volume 65, pages 2–28, 2005.
- [HCL⁺90] Laura M. Haas, Walter Chang, Guy M. Lohman, John McPherson, Paul F. Wilms, George Lapis, Bruce Lindsay, Hamid Pirahesh, Michael J. Carey, and Eugene Shekita. Starburst mid-flight: as the dust clears [database project]. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160, 1990.
- [HFLP89] Laura M. Haas, Johann Christoph Freytag, Guy M. Lohman, and Hamid Pirahesh. Extensible query processing in Starburst. In *SIGMOD Conference*, pages 377–388, 1989.
- [HHE⁺09] Jean-Luc Hainaut, Jean Henrard, Vincent Englebert, Didier Roland, and Jean-Marc Hick. Database reverse engineering. In *Encyclopedia of Database Systems*, pages 723–728. 2009.
- [HKGT03] Marios Hadjieleftheriou, George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On-Line Discovery of Dense Areas in Spatio-temporal Databases. In *SSTD*, pages 306–324, 2003.
- [HPSR12] Fabian Hueske, Mathias Peters, Matthias J Sax, and Astrid Rheinl. Opening the Black Boxes in Data Flow Optimization. *VLDB*, 5(11):1256–1267, 2012.
- [HR83] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983.
- [HR85] Theo Haerder and Andreas Reuter. Architektur von datenbanksystemen fuer non-standard-anwendungen. In *BTW*, pages 253–286, 1985.
- [HR01] Theo Haerder and Erhard Rahm. *Datenbanksysteme: Konzepte und Techniken der Implementierung; mit 14 Tabellen*. Springer DE, 2001.

- [HTT09] Tony Hey, Stewart Tansley, and Kristin M. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.
- [IBY⁺07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59–72, 2007.
- [Jae96] U. Jaeger. SMILE - A framework for lossless Situation Detection. 1996.
- [KA98] David Krieger and Richard M Adler. The emergence of distributed component platforms. *Computer*, 31(3):43–53, 1998.
- [Kos08] Donald Kossmann. Building web applications without a database system. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 3–3, New York, NY, USA, 2008. ACM.
- [KSWU92] Harm Knolle, Gunter Schlageter, Erhard Welker, and Rainer Unland. TOPAZ: A tool kit for the construction of application-specific transaction managers. In *Objektbanken fuer Experten*, pages 254–280. Springer, 1992.
- [Lim12] Harold Lim. Stubby : A Transformation-based Optimizer for MapReduce Workflows. *VLDB*, 5(11):1196–1207, 2012.
- [LKD⁺88] Volker Linnemann, Klaus Kuespert, Peter Dadam, Peter Pistor, R. Erbe, Alfons Kemper, Norbert Suedkamp, Georg Walch, and Mechtild Wallrath. Design and implementation of an extensible database management system supporting user defined data types and functions. In *VLDB*, pages 294–305, 1988.
- [LM11] Josh Long and Steve Mayzak. *Getting Started with Roo*. O'Reilly, 2011.
- [LMP87] Bruce Lindsay, John McPherson, and Hamid Pirahesh. *A data management extension architecture*, volume 16. ACM, 1987.
- [LPM02] Iosif Lazaridis, Kriengkrai Porkaew, and Sharad Mehrotra. Dynamic queries over mobile objects. In *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '02, pages 269–286, London, UK, UK, 2002. Springer-Verlag.
- [LS88] Clifford A. Lynch and Michael Stonebraker. Extended user-defined indexing with application to textual databases. In *VLDB*, pages 306–317, 1988.
- [Luc02] David C Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.
- [Mat01] Friedmann Mattern. Ubiquitous computing. Presentation, 2001.
- [MBHT96] William J McKenna, Louis Burger, Chi Hoang, and Melissa Truong. EROC: a toolkit for building neat query optimizers. In *VLDB*, pages 111–121. Citeseer, 1996.
- [MCBD12] Lourdes Martínez, Christine Collet, Christophe Bobineau, and Etienne Dublé. The qol approach for optimizing distributed queries without complete knowledge. In *IDEAS*, pages 91–99, 2012.
- [Moh13] C Mohan. History repeats itself: sensible and nonsensql aspects of the NoSQL hoopla. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 11–16. ACM, 2013.
- [MP12] Fowler M. and Sadalage P. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 2012.

- [Mul12] C. Mullins. *Database Administration: The Complete Guide to DBA Practices and Procedures*. 2012.
- [MXA04] Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In Gerhard Weikum, Arnd Christian Koenig, and Stefan Dessloch, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 623–634. ACM, 2004.
- [ND95] Oscar Nierstrasz and Laurent Dami. Component-oriented software technology. *Object-Oriented Software Composition*, 1:3–28, 1995.
- [NM95] Oscar Nierstrasz and Theo Dirk Meijler. Research directions in software composition. *ACM Computing Surveys (CSUR)*, 27(2):262–264, 1995.
- [OHE96] Robert Orfali, Dan Harkey, and Jeri Edwards. *The essential distributed objects survival guide*. 1996.
- [OMG11] OMG. Business Process Model and Notation (BPMN). Rapport technique, OMG, <http://www.omg.org/spec/BPMN/2.0>, 2011.
- [ÖMS95] M. Tamer Özsu, Adriana Muñoz, and Duane Szafron. An extensible query optimizer for an objectbase management system. In *CIKM*, pages 188–196, 1995.
- [OPSY98] Steve Olson, Richard Pledereder, Phil Shaw, and David Yach. The sybase architecture for extensible data management. *IEEE Data Eng. Bull.*, 21(3):12–24, 1998.
- [ORS⁺08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.
- [PSB04] Peter R Pietzuch, Brian Shand, and Jean Bacon. Composite event detection as a generic middleware extension. *Network, IEEE*, 18(1):44–55, 2004.
- [PXK⁺02] Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. Comput.*, 51(10):1124–1140, October 2002.
- [RB99] Uwe Rohm and Klemens Bohm. Working together in harmony—an implementation of the corba object query service and its evaluation. In *Proceedings of the 15th International Conference on Data Engineering, 1999.*, pages 238–247. IEEE, 1999.
- [RS97] Mary Tork Roth and Peter M Schwarz. Don’t scrap it, wrap it! a wrapper architecture for legacy data sources. In *VLDB*, volume 97, pages 25–29. DTIC Document, 1997.
- [SABS02] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and Isolation for Transactional Processes. *ACM Transactions on Database Systems (TODS)*, 27(1):63–116, March 2002.
- [SC05] Michael Stonebraker and Ugur Cetintemel. ”one size fits all”: An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering, ICDE ’05*, pages 2–11, Washington, DC, USA, 2005. IEEE Computer Society.
- [Ses98] Praveen Seshadri. Predator: A resource for database research. *ACM SIGMOD Record*, 27(1):16–20, 1998.
- [SHWK76] Michael Stonebraker, Gerald Held, Eugene Wong, and Peter Kreps. The design and implementation of INGRES. *ACM Trans. Database Syst.*, 1:189–222, September 1976.

- [SKPO88] Michael Stonebraker, Randy H Katz, David A Patterson, and John K Ousterhout. The design of XPRS. In *VLDB*, pages 318–330, 1988.
- [SMWM06] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query optimization over web services. *VLDB '06, Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006.
- [SPSW90] Hans-Joerg Schek, H.-Bernhard Paul, Marc H. Scholl, and Gerhard Weikum. The DAS-DBS project: Objectives, experiences, and future prospects. *IEEE Trans. Knowl. Data Eng.*, 2(1):25–43, 1990.
- [SR86] Michael Stonebraker and Lawrence A. Rowe. The design of postgres. In *SIGMOD Conference*, pages 340–355, 1986.
- [SR01] Zhexuan Song and Nick Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, SSTD '01, pages 79–96, London, UK, UK, 2001. Springer-Verlag.
- [SRG83] Michael Stonebraker, W. Bradley Rubenstein, and Antonin Guttman. Application of abstract data types and abstract indices to CAD data bases. In *Engineering Design Applications*, pages 107–113, 1983.
- [SZD07] Ionut Subasu, Patrick Ziegler, and Klaus R Dittrich. Towards service-based data management systems. In *Workshop Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW 2007)*, pages 3–86130. Citeseer, 2007.
- [Szy97] CA Szyperki. Component software: Beyond OO programming, 1997.
- [Tiw11] Shashank Tiwari. *Professional NoSQL*. John Wiley & Sons, 2011.
- [TRV98] Anthony Tomasic, Louiqa Raschid, and Patrick Valduriez. Scaling access to heterogeneous data sources with disco. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):808–823, 1998.
- [TXC07] Yufei Tao, Xiaokui Xiao, and Reynold Cheng. Range search on multidimensional uncertain data. *ACM Trans. Database Syst.*, 32(3), August 2007.
- [VC04] Tuyet-Trinh Vu and Christine Collet. Adaptable query evaluation using qbf. In *IDEAS*, pages 265–270, 2004.
- [Vog09] Werner Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.
- [VSCGR00] Geneveva Vargas-Solar, Christine Collet, and Helena Grazziotin-Ribeiro. Active services for federated databases. In *SAC (1)*, pages 356–360, 2000.
- [VSIC⁺11] Geneveva Vargas-Solar, Noha Ibrahim, Christine Collet, Michel Adiba, Jean Marc Petit, and Thierry Delot. Querying issues in pervasive environments. *Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications*, pages 1–23, 2011.
- [VV04] K. Vidyasankar and Gottfried Vossen. A multi-level model for Web Service composition. In *ICWS*, pages 462–, 2004.
- [Wal11] Audrey Walters. An itunes model for data. *O'Reilly Strata Making Data Work*, page <http://strata.oreilly.com/2011/04/itunes:for:data.html>, April 10, 2011.
- [WBT92] David L. Wells, José A. Blakeley, and Craig W. Thompson. Architecture of an open object-oriented database management system. *IEEE Computer*, 25(10):74–82, 1992.

- [WCSO08] Hiroshi Wada, Paskorn Champrasert, Junichi Suzuki, and Katsuya Oba. Multiobjective Optimization of SLA-Aware Service Composition. *2008 IEEE Congress on Services - Part I*, pages 368–375, July 2008.
- [WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM, 2006.
- [Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.
- [WSCY99] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. Updating and querying databases that track mobile units. *Distrib. Parallel Databases*, 7(3):257–387, July 1999.
- [Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB’81: Proceedings of the seventh international conference on Very Large Data Bases*, pages 82–94. VLDB Endowment, 1981.
- [YB05] Eiko Yoneki and Jean Bacon. Unified semantics for event correlation over time and space in hybrid network environments. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 366–384. Springer, 2005.
- [YIF+08] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, volume 8, pages 1–14, 2008.
- [ZL01] Baihua Zheng and Dik Lun Lee. Semantic caching in location-dependent query processing. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, SSTD ’01*, pages 97–116, London, UK, UK, 2001. Springer-Verlag.
- [ZZP+03] Jun Zhang, Manli Zhu, Dimitris Papadias, Yufei Tao, and Dik Lun Lee. Location-based spatial queries. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data, SIGMOD ’03*, pages 443–454, New York, NY, USA, 2003. ACM.

Efficient, continuous and reliable Data Management by Coordinating Services

Genoveva VARGAS-SOLAR

Résumé

Les nouvelles architectures matérielles mettent à disposition des services à des différentes empreintes: services réduits pour les systèmes embarqués et illimité pour le cloud, par exemple ; certaines ajoutent aussi des contraintes pour accéder aux données, qui portent sur le contrôle d'accès et la réservation/affectation de ressources par priorités (e.g., dans la grille) et sur le coût économique (e.g., dans le cloud). Les applications qui utilisent ces architectures établissent aussi des préférences de qualité de service (service level agreement SLA) qui portent sur le temps de traitement, la pertinence et la provenance de données ; le coût économique des données et le coût énergétique qui implique leur traitement. Ainsi, la gestion de données doit être revisitée pour concevoir des stratégies qui respectent à la fois les caractéristiques des architectures et les préférences des utilisateurs (service level agreements SLA).

Notre recherche contribue à la construction de systèmes de gestion de données à base de services. L'objectif est de concevoir des services de gestion de données guidée par des contrats SLA. Il s'agit d'offrir des méthodologies et des outils pour l'intégration, le déploiement, et l'exécution d'un assemblage de services pour programmer des fonctions de gestion de données. La composition de services de données particulièrement lorsqu'il s'agit des services bases de données, doit respecter des critères de qualité de service (e.g., sécurité, fiabilité, tolérance aux fautes, évolution et adaptabilité dynamique) et des propriétés de comportement (par ex., exécution transactionnelle) adaptées aux besoins des applications.

Abstract

The emergence of new architectures like the cloud open new challenges for data management. It is no longer pertinent to reason with respect a to set of computing, storage and memory resources, instead it is necessary to conceive algorithms and processes considering an unlimited set of resources usable via a "pay as U go model", energy consumption or services reputation and provenance models. Instead of designing processes and algorithms considering as threshold the resources availability, the cloud imposes to take into consideration the economic cost of the processes vs. resources use, results presentation through access subscription, and the parallel exploitation of available resources.

Our research contributes to the construction of service based data management systems. The objective is to design data management services guided by SLA contracts. We proposed methodologies, algorithms and tools for querying, deploying and executing service coordinations for programming data management functions. These functions, must respect QoS properties (security, reliability, fault tolerance, dynamic evolution and adaptability) and behavior properties (e.g., transactional execution) adapted to application requirements. Our work proposes models and mechanisms for adding these properties to new service based data management functions.