Université Pierre et Marie Curie - Paris 6

Document de Synthèse

présenté pour obtenir

L'HABILITATION À DIRIGER DES RECHERCHES

Mention Informatique

par

Dan VODISLAV

TITRE

Intégration, partage et diffusion de données sur le Web
(Data integration, sharing and distribution on the Web)

Soutenue le 29 octobre 2007

JURY

Rapporteurs :	Peter BUNEMAN	University of Edinburgh, Royaume Uni
	Georges GARDARIN	Université de Versailles Saint-Quentin-en-Yvelines
	Jacques LE MAITRE	Université du Sud Toulon-Var
Examinateurs :	Anne DOUCET	Université Pierre et Marie Curie, Paris 6
	Marie-Christine ROUSSET	Université de Grenoble
	Michel SCHOLL	Conservatoire National des Arts et Métiers, Paris

Remerciements

Je voudrais remercier tous ceux qui m'ont aidé, de près ou de loin, à franchir cette étape si importante.

Michel, qui m'a accueilli dans Vertigo et auprès duquel j'ai appris tant de choses. Merci Michel pour ton amitié, pour tes encouragements, pour la générosité et la chaleur avec laquelle tu sais partager ton expérience.

Peter Buneman, *Georges Gardarin* et *Jacques Le Maitre*, qui ont accepté le travail ardu de rapporteur.

Anne Doucet et *Marie-Christine Rousset*, qui ont accepté de faire partie du jury, ainsi que *Pierre Sens*, qui a suivi ma candidature à l'Université Paris 6.

Tous mes collègues de l'équipe *Vertigo*, avec lesquels j'ai partagé tant de bons moments au long des années: Bernd, Philippe, Vassilis, David, Valérie, Michel C, Nicolas, ainsi que Irini, Cédric, Julien, Imen, Nouha, Radu, Sébastien, François et les autres. J'ai eu la chance de faire partie de cette équipe formidable où il fait bon travailler. Merci aussi à mes nouveaux collègues de la grande et sage famille *Wisdom*, avec qui j'ai pu lancer de nouveaux projets très prometteurs.

Serge, qui m'a accueilli dans Gemo et m'a donné la possibilité de travailler sur des sujets passionnants avec lui et les autres membres de son équipe: *Ioana*, *Gabriel*, *Nicoleta* et les autres.

Sophie, avec qui j'ai travaillé durant les belles années de l'aventure Xyleme et dont j'ai appris énormément de choses. Merci à toute l'équipe *Xyleme* pour l'opportunité qui m'a été offerte de partager cette expérience de recherche unique avec des gens formidables: Guy, Jean-Pierre, Grégory, Mihai, Fred, Bruno, Markos, Laurent, Patrick, David, Hugo, Ramon, Gérald, Amir, François, Joëlle et plein d'autres, trop nombreux pour pouvoir les citer tous.

Pierangelo, *Imen*, *Radu* et *François*, mes doctorants, qui ont subi avec bonne humeur mes débuts dans l'encadrement de thèses.

Mes collègues du laboratoire CEDRIC et du département d'informatique du CNAM, qui m'ont aidé à découvrir les subtilités, les délices et les affres du métier d'enseignant-chercheur.

Mes étudiants, qui ont subi stoïquement mes premiers pas dans l'enseignement et auxquels je dois en très grande partie le plaisir d'enseigner.

Mes parents, auxquels je dois tant, ma belle-mère préférée pour ses encouragements et son aide.

Et surtout *Bogdana*, *Elisa* et *Alexandra*, qui sont mon point d'équilibre, sur lequel on peut tout construire.

Contents

Chapter 1

Introduction

This report presents my research activity since the end of my PhD thesis, in January 1997. During this period, I have been an Assistant Professor at the Conservatoire National des Arts et Métiers (CNAM), in Paris, member of the CEDRIC Laboratory in Computer Science. I also held a joint position of scientific adviser in the Xyleme S.A. company (2000-2006), then a position of associate member of the Gemo group at INRIA Futurs (since 2005).

My research activity is divided into *two distinct periods*, addressing ***two different research fields***: *multimedia user interfaces* and *data management on the web*. A salient point in my research activity is *the thematic reconversion* I operated between these two domains in 1999.

Multimedia user interfaces. My work in this research field covers my PhD (December 1993 - January 1997) and the succeeding period until 1999. After my PhD thesis, entitled "Visual programming for animation in user interfaces", I continued my work on the modeling of structured multimedia interfaces, as an Assistant Professor at CNAM and member of the Multimedia team of the CEDRIC laboratory.

Data management on the web. In September 1999, I started my *thematic reconversion* by joining the Vertigo database team of the CEDRIC laboratory. The occasion was the startup of the Xyleme project at INRIA, whose goal was the creation of a distributed, very large scale repository, able to store all the XML documents on the web and to provide services for acquisition, indexing, querying, change management, semantic integration, etc. Initiated by the Verso group at INRIA, the project included database teams from CNAM - CEDRIC (Vertigo), from Paris Sud - LRI (IASI) and from the Mannheim University.

Since 1999, my research activity in the field of *data management on the web*, has been organized around three main axes:

1. Web-scale integration of XML data
2. Views for heterogeneous XML data
3. Peer-to-peer architectures for data sharing

This introduction presents a global overview of all the themes and sub-themes of my research activity, while the next chapters focus on the period after my thematic reconversion, encompassing the most significant research contributions. The last chapter presents the conclusions and future work for my research activity.

1.1 Multimedia user interfaces

My PhD thesis [Vod97a] was addressing the problem of using animation for illustrating interaction in graphical user interfaces. The main challenge of this work was to define a model for animated user interfaces and to simplify the specification, the programming of interactive animations.

The result was *HandMove*, a general model for animated user interface components, that allows visual programming. The main idea of the model is to represent animation through *abstract trajectories* describing continuous evolution and through *spots* on these trajectories, describing sudden changes. Animation can be based on various types of stimuli: timer ticks, input device signals, application events. Sudden changes are triggered by events, generated by predicates on the animated objects, by the external application or by explicit event generation.

The spatial metaphor used by HandMove is appropriate for visual programming and provides an intuitive method for specifying animation by drawing. The HandMove tool provides a visual editor for both graphical objects and animated behavior and a run-time visualizer of the resulting behavior. HandMove allows describing animated scenes, that can be ”encapsulated” as interactive and animated user interface components, then used at the same level as buttons, text fields and menus in user interfaces. This work has been realized in collaboration with NSL, a French software editor specialized in X/Motif user interface tools and the HandMove tool has been integrated with XFaceMaker, the NSL's graphical user interface builder.

After my PhD thesis, I continued for some time to work on formalizing the HandMove visual model and on improving the HandMove tools - the results of this work have been presented in [Vod97b].

My research activity during this period was directed toward the modeling of multimedia user interfaces as structured objects. In the same line as the HandMove model, this work aims at describing in a *declarative* manner both graphical objects and behavior as *structured entities*.

The main result is the definition of a general model for interactive animations (IAN) in multimedia documents, presented in [VV00]. The model naturally integrates usual multimedia elements (text, image, video, sound) with motion and graphical changes of geometric objects, but also with user and inter-object interaction. Animation is defined in a declarative and structured manner, through a scenario composed of event-condition-action rules. The purpose is to enable both *easy authoring* through declarative techniques and *database management* of animations as structured objects. Database management of IANs allows building libraries of parametrized animations and retrieving them on the basis of specific criteria: scenario elements, animation events, object properties, etc. We proposed an extension of the OQL language [Cat97] for object-oriented databases, adapted to querying collections of IANs.

Another research direction in this field was the study of *3D interactive scenes*, described by using the VRML language [vrm97], that already provides a declarative model, as a tree of transformation nodes. However, VRML does not provide the appropriate abstraction level for interaction and animation, that mostly rely on external scripts to be programmed. In [CTV00], we proposed an extension of VRML that introduces new, higher level transformation nodes for interaction, for animation and for event routing between objects.

A last point addressed in this topic, was the *integration of database applications through the user interface*. We considered the problem of building applications over data describing the same objects, but coming from different sources and having major differences in their nature and their processing tools. The application was the integration of XML data describing monuments with geographical information about their location, in the context of a collaboration with the French Department of

Culture. This work represents the subject of the CNAM engineer thesis of Julio Fernandes (1998) and has been presented in [AVFC98].

1.2 Data management on the web

My research in the field of data management on the web deals, at a general level, with the problem of exploiting very large amounts of data distributed over a large number of sources on the web - sources with possibly heterogeneous structures, with various degrees of autonomy and various processing capabilities.

A central theme in my research is the problem of ***data integration***, that aims at providing a unified, transparent access to data stored in heterogeneous and autonomous sources. The access is based on an homogeneous, global schema and on mappings between this global schema and the data sources, defining an *integration view*. This problem has been extensively studied in the literature, addressing issues such as data integration architectures and models, query rewriting, mapping generation, source capabilities, quality of data, etc. My research on this topic has focused on the integration of *heterogeneous XML data*, at *very large scale or medium scale* and on *providing simple and intuitive access to the integrated data*, in order to make it accessible to novice users and to offer support for rapid application development.

The other main theme I addressed is that of ***distributed architectures for data sharing on the web***, focusing notably on peer-to-peer (P2P) systems for content distribution, sharing, updating and querying in web communities.

In this context, my research work has followed three main directions:

1. First, I addressed the problem of *XML data integration at the scale of the web*. This work has been realized in relation with the *Xyleme project*, that aimed at creating a distributed XML repository able to store all the XML documents on the web and to provide semantic integration of data, among other services. The challenge was to define scalable techniques for data integration at the web scale, concerning query rewriting, methods for automatic view generation, view distribution over the sources, etc.

2. Next, I considered the problem of building *simple views over heterogeneous XML data*, in order to make the access to such data as simple as querying a table through a query form. The challenge is to find a good compromise between expressive power and simplicity of querying and of maintaining the view when new sources are added.

3. Finally, I addressed the problem of using peer-to-peer (P2P) architectures for building *scalable systems for content management on the web*. The challenge is to design efficient and robust P2P systems, where peers cooperate and share resources (processor, disk space, bandwidth, etc.) in order to provide scalable global functionalities, such as indexing and querying content metadata, content distribution, notification on changes, etc.

An important characteristic of my research activity is that it is ***strongly related to system implementation***. In the Xyleme project, then later as a scientific adviser for the Xyleme company, I coordinated and participated to the implementation of several modules of the Xyleme software: the *Xyleme Views* semantic integration module, the *MapGen* mapping generation tool, the *XyView* module for application views, the *XMLizer* tool for extracting XML structure from plain text documents, etc. In my collaboration with the Gemo team at INRIA, I coordinated and participated to the implementation of the *EDOS* P2P platform for Linux packages distribution and to several improvements in the *ActiveXML* platform.

The next sections give a brief description of my activity in each of the three research axis.

1.2.1 Web-scale integration of XML data

My work in this axis corresponds to the period between 1999 and 2003, as a participant to the Xyleme project (1999-2000), then as a scientific adviser for the Xyleme S.A. start-up company, created in September 2000.

The Xyleme project, initiated by Serge Abiteboul and Sophie Cluet (Verso group at INRIA) and François Bancilhon (ex-CEO of O2 Technology) in September 1999, was organized as an open network of researchers, including the Verso group at INRIA, the Vertigo team from CNAM, the IASI group from LRI and the database group from the Mannheim University. As explained above, the goal of the project was the creation of a kind of "Google XML", i.e. a distributed repository, able to store all the XML documents on the web and to provide services for acquisition, indexing, querying, change management, semantic integration, etc.

During this period, I have been in charge (with Marie-Christine Rousset from LRI) of the semantic integration module in the Xyleme project (1999-2000), I supervised (with Sophie Cluet) the PhD thesis of Pierangelo Veltri (1999-2002), the Master thesis of Jean-Pierre Sirot (2000) and the engineer thesis (Polytechnical Institute of Bucharest) of Anne-Marie Dumitrache (2003).

The difficulty of realizing data integration in Xyleme comes from *the scale*: for any type of content, a huge amount of XML documents, with many different XML structures, is available on the web. We proposed a view mechanism adapted to web-scale data integration, that provides a homogeneous XML schema to access the repository and that guarantees scalability by distributing the view definition, query rewriting and execution over the repository machines and by using a scalable query rewriting algorithm. This model has been published as a VLDB Journal paper [ACM+02], a book chapter [CCT+05], two conference papers [CVV01, ACV+01], and has been the subject of the PhD thesis of Pierangelo Veltri [Vel02] and of a patent submission. I also coordinated (with Sophie Cluet) and participated to the implementation of this view mechanism as a software module (*Xyleme Views*) in Xyleme.

One of the major problems in web-scale data integration is the creation of the view and its maintenance over the time. We defined methods for *automatic generation of view mappings*, essential for building views at this scale. We proposed a set of algorithms for generating node-to-node mappings between tree structures, based on linguistic similarity of tag names and on the structural context of nodes. This method has been published as a DKE journal paper [DRR+03], a conference paper [RSV01] and has been the subject of the Master thesis of Jean-Pierre Sirot. I coordinated the implementation of the *MapGen* tool in Xyleme, for semi-automatic generation and maintenance of view mappings.

The *integration of the view mechanism with query processing* in the Xyleme system raised some interesting issues. We choose a transparent integration of views in the query processing mechanism, allowing to handle views as ordinary XML document structures. The difference appears in the generated execution plan, that includes view-specific operators. This query processing model has been presented in several conference papers ([ACV+00], and partially in [CVV01] and [ACV+01]). We also added a grouping operator to the Xyleme query language, very useful for query rewriting on views. This point has represented the subject of the engineer thesis of Ana-Maria Dumitrache, that I supervised.

I also considered during this period the problem of *producing XML documents from unstructured content*. XML-ization tools are essential in XML management systems by enabling the use of the huge quantities of unstructured data, unexploitable with traditional databases. I supervised the design

and implementation of the *XMLizer* tool (2003) in Xyleme, able to learn recurrent patterns in plain text files and to extract XML structures from these patterns. The *XMLizer* has received a grant for innovative projects from ANVAR.

A detailed description of my work in this research axis is presented in Chapter 2.

1.2.2 Views for heterogeneous XML data

My research in this axis corresponds to the period between 2003 and 2007 and corresponds to the subject of the PhD thesis of Imen Sebei. This work can be divided in two parts: the first one, in collaboration with Xyleme, focused on views over an XML repository (the *XyView* model), while the second one, within the framework of the *ACI SemWeb*, addressed views over *open systems*, composed of autonomous sources with unpredictable behavior (the *OpenXView* model). The XyView model was also included in the platform designed in the context of the *WebContent* project.

The *ACI SemWeb* project [sem04] (2004-2007), gathered database teams from PRISM Versailles, LIP6 Paris, LSIS Toulon, LIRIS Lyon, LINA Nantes and CEDRIC Paris, and aimed at designing methods and tools for querying the Semantic Web in XQuery. In SemWeb, I have been in charge of the P2P research axis in this project, and (with Elisabeth Metais) of the CEDRIC part. The *RNTL WebContent* project [web06] (2006-2009) aims at building an advanced platform for enriching and exploiting web documents and gathers teams from INRIA, CEA, EADS, Thales, LIP6, PRISM Versailles, Exalead, Xyleme, etc. I was in charge for Xyleme in this project and I managed the integration of the Xyleme software in the WebContent platform (2006).

In this axis, I supervised, in addition to the PhD thesis of Imen Sebei (2003-2007), the Master thesis of François Boisson (2006).

This topic reconsiders the problem of large scale data integration in a different context. The goal is to make the access to heterogeneous XML data as simple as possible, through universal relation-like views, in order to simplify queries for novice users and to provide support for rapid application development. The difference with web-scale views in Xyleme is that we address application-oriented views, at a lower scale, where a better expressive power can be considered, e.g. by including joins and transformation functions in the view definition.

The *XyView* model allows rapid application development on top of heterogeneous, schema-free XML data stored in an XML repository. A XyView view is structured at three levels: (i) the *physical level*, providing tree structures extracted from documents, (ii) the *logical level*, providing homogeneous tree structures over unions of physical views, and (iii) the *user level*, providing a flat structure (table) over joins of logical views. This view structure, that separates unions from joins, and the use of one-to-one mappings between levels instead of queries, make XyView views easier to create and to maintain than query-based views. Also, query rewriting is more flexible and allows e.g. discarding useless duplicates in answers in a natural way.

This work has represented the subject of the first part of Imen Sebei's PhD thesis and of two conference papers, [VCCS05] and [VCCS06]. I supervised and participated to the implementation of the XyView model as *a software module in Xyleme*, that provides: an API for view management and query translation, a tool for graphical editing of views and a tool for automatic generation of web form applications on top of XyView views.

The *OpenXView* model is adapted to *open systems*, where data sources are autonomous, variable in number and with unpredictable changes, e.g. P2P systems. XyView is not appropriate in this context, because the three-level view structure may be forced to change too often, e.g. when new sources are published. OpenXView provides a mixed ontology/tree integration schema (concepts having tree-structured attributes), more flexible than XyView, with *implicit joins* based on concept

key attributes. OpenXView dynamically determines for each query the set of sources concerned by the query, and the joins and unions between them, unlike XyView and other integration view systems, where most of these elements are predetermined. In order to limit the number of possible query rewritings, we proposed equivalence and containment criteria for rewritings based on minimal covers for query elements.

The work on OpenXView has represented the subject of the second part of Imen Sebei's PhD thesis and of the Master thesis of François Boisson. It was presented in two conference papers ([BSSV06b] and [BSSV06a]), and published in a journal paper [BSSV07]. An OpenXView prototype is currently under development.

A detailed description of my work in this research axis is presented in Chapter 3.

1.2.3 Peer-to-peer architectures for data sharing

My research in this axis corresponds to the period between 2004 and 2007 and corresponds to my collaboration with the Gemo team at INRIA Futurs - I am an associate member of Gemo since 2005. This work is realized in relation with the software platform developed by Gemo, composed of *ActiveXML* [AMT06], a P2P platform for dynamic XML documents (containing web service calls), and of *KadoP* [AMP05], a distributed XML management system, built on top of ActiveXML and of a distributed hash table (Pastry).

Part of this work has been realized within the framework of the *EDOS* European project [EDO04] (2004-2007) and of the *WebContent* French RNTL project [web06] (2006-2009). *EDOS* aims at creating a platform for producing, testing and disseminating free/open-source software. I have been in charge (with Serge Abiteboul) of the work package 4, that aims at building a P2P platform for content dissemination, and gathering teams from INRIA, Mandriva, University of Tel Aviv, University of Geneva and CSP Torino. In the *WebContent* project, that aims at building an advanced platform for enriching and exploiting web documents, I am currently participating for Gemo to the design of a P2P architecture for scalable data management.

In this axis, I am supervising the CIFRE PhD thesis of Radu Pop (2005-2008), in collaboration with Mandriva. I also supervised on the same topic the CNAM engineer thesis of Eric Darondeau (2004-2005) and the Polytechnique engineer internship of Ming Hoang To (2006).

The goal of my research in this axis is to design and implement scalable systems for content management on the web, based on P2P architectures where peers cooperate and share resources in order to transparently provide global functionalities [AP07]. The system should provide its services in a transparent way, being the only responsible for load balancing and resource sharing.

The EDOS distribution system belongs to this category. It replaces the classical "hierarchy of mirrors" distribution architecture and provides scalable functionalities for the dissemination of Mandriva Linux packages within the community network: publishing, indexing and querying of metadata, optimized content dissemination, notification on changes, etc. The EDOS distribution system represents the subject of the PhD thesis of Radu Pop and is presented in four conference papers ([ADP$^+$07c], [ADP$^+$07a], [ADP$^+$07b], [APVV07]). The system has been implemented on top of KadoP and BitTorrent [Coh03]; I supervised and participated to its implementation, that will be the object of a demonstration at VLDB 2007 [ADP$^+$07c]. Also, the EDOS distributed system is the basis of the distributed architecture adopted in the WebContent project.

In the same research axis, I participated to the improvement of the ActiveXML platform [weba], by providing support for coupling with an XML database. Initially, the ActiveXML platform was using a simple persistence mechanism, based on XML files and in-memory query processing and updates. Coupling it to an XML database allows using the database capabilities for storage, querying

and updating, and guarantees scalability in terms of data managed by a peer. This work has represented the subject of the CNAM engineer thesis of Eric Darondeau (2004-2005), for a coupling with Xyleme, then of the Polytechnique engineer internship of Ming Hoang To (2006), for a generic coupling with any XML database supporting XQuery and XUpdate.

A detailed description of my work in this research axis is presented in Chapter 4.

1.2.4 Summary of publications

Web-scale integration of XML data

[ACM+02] V. Aguilera, S. Cluet, T. Milo, P. Veltri, and D. Vodislav. Views in a large scale XML repository. *VLDB Journal*, 11(3):238–255, 2002.

[DRR+03] C. Delobel, C. Reynaud, M.-C. Rousset, J.-P. Sirot, and D. Vodislav. Semantic integration in Xyleme: a uniform tree-based approach. *Data & Knowledge Engineering Journal*, 44(3):267–298, 2003.

[CCT+05] M. Cannataro, S. Cluet, G. Tradigo, P. Veltri, and D. Vodislav. Using views to query XML documents. In *Encyclopedia of Database Technologies and Applications*, pages 729–735. IDEA Group Reference, 2005.

[CVV01] S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale XML repository. In *VLDB*, pages 271–280, 2001.

[RSV01] C. Reynaud, J.-P. Sirot, and D. Vodislav. Semantic integration of XML heterogeneous data sources. In *IDEAS*, pages 199–208, 2001.

[ACV+01] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying a web scale XML repository. In *SEBD*, pages 105–118, 2001.

[ACV+00] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying XML documents in Xyleme. *ACM SIGIR workshop*, 2000.

Views for heterogeneous XML data

[BSSV07] Francois Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav. Source identification and query rewriting in open XML data integration systems. *IADIS International Journal of WWW/Internet*, 5(1), pages 29–44, 2007.

[VCCS06] Dan Vodislav, Sophie Cluet, Grégory Corona, and Imen Sebei. Views for simplifying access to heterogeneous XML data. *OTM Confederated conferences/CoopIS*, pages 72–90, 2006.

[BSSV06b] Francois Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav. Scalability of source identification in data integration systems. *IEEE SITIS*, 2006.

[BSSV06a] Francois Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav. Query rewriting for open XML data integration systems. *IADIS WWW/Internet*, pages 133–141, 2006.

[VCCS05] Dan Vodislav, Sophie Cluet, Grégory Corona, and Imen Sebei. XyView: Universal relations revisited. *Bases de Données Avancées (BDA)*, pages 357–372, 2005.

Peer-to-peer architectures for data sharing

[ADP+07c] S. Abiteboul, I. Dar, R. Pop, G. Vasile, D. Vodislav, and N. Preda. Large scale P2P distribution of open source software. Demonstration, in *VLDB*, pages 1390–1393, 2007.

[ADP+07a] S. Abiteboul, I. Dar, R. Pop, G. Vasile, and D. Vodislav. EDOS distribution system: a P2P architecture for open-source content dissemination. In *IFIP Intl. Conference on Open Source Systems (OSS)*, pages 209–215, 2007.

[ADP+07b] S. Abiteboul, I. Dar, R. Pop, G. Vasile, and D. Vodislav. Snapshot on the EDOS distribution system. *FOSDEM Workshop, Free & Open source Software Developers' European Meeting*, 2007.

[APVV07] S. Abiteboul, R. Pop, G. Vasile, D. Vodislav. Scalability Evaluation of a P2P Content Distribution System. *Bases de Données Avancées (BDA)*, 2007.

Multimedia user interfaces

[VV00] D. Vodislav and M. Vazirgiannis. Structured interactive animation for multimedia documents. *IEEE Symposium on Visual Languages (VL)*, 2000.

[Vod97b] D. Vodislav. A visual programming model for user interface animation. *IEEE Symposium on Visual Languages (VL)*, pages 344–351, 1997.

[AVFC98] B. Amann, D. Vodislav, J. Fernandes, and G. Coste. Browsing SGML documents with maps : The French 'Inventaire' experience. *International Conference on Database and Expert Systems Applications (DEXA)*, 1998.

[CTV00] P. Cubaud, A. Topol, and D. Vodislav. Les limites de VRML pour les comportements interactifs: étude de cas. *ERGO-IHM*, 2000.

Chapter 2

Web-scale integration of XML data

This chapter presents a synthetic view of my work in the field of web-scale integration of XML data, realized in relation with the Xyleme system. After a brief presentation of the context of this research axis, we present the Xyleme web-scale integration model, the query rewriting problems, the automatic view generation algorithms and we end up with a presentation of the work done for extracting XML elements from unstructured text.

2.1 Context: data integration on the web

Data integration is the problem of providing unified and transparent access to data stored in multiple, autonomous and heterogeneous data sources [Len02]. This problem, extensively studied for many years, acquired a new dimension in the context of the web. Various data integration architectures have been proposed, among which *mediators* [Wie95], *warehouses* [Wid95], and more recently *peer-to-peer* [KP05] (that may be seen as a generalization of mediator/warehouse architectures), are the most common ones.

Compared to warehouses, mediators provide a more flexible architecture. Data is leaved under the control of each source and the relation between the global model and sources is defined through *mappings* that enable *query rewriting* (from queries on the global model to queries on sources) and *results typing* (from the source type to the global type). Warehouses provide better performance by *materializing the integration view* from data gathered from sources, transformed and merged following the global model and stored in a common repository, but have to deal with the problem of *data freshness*.

The definition of the relation between the global model and local source models (the data integration view) mainly follows two different approaches. The *global-as-view* (GAV) approach considers that all the sources are known in advance (close world assumption) and the global model is defined as a view over the set of local source models. A GAV model definition has the form $g \mapsto q_s$, i.e. each element g of the global model is defined as a query (view) over the local sources. This corresponds to the classical database integration approach and has the advantage of very simple query rewriting algorithms. Data integration systems such as Tsimmis [GMPQ$^+$97], YAT [CCS00], Enosys [PBO$^+$03], XMLMedia [GMT02], Agora [MFK02] use a GAV integration schema.

The *local-as-view* (LAV) approach considers that the global model is fixed and describes an application domain, independently of the data sources. Each data source describes its contribution as a local view over the global model, independently of other data sources. A LAV model definition has the form $s \mapsto q_g$, i.e. each element s of the the local model of each source is defined as a query

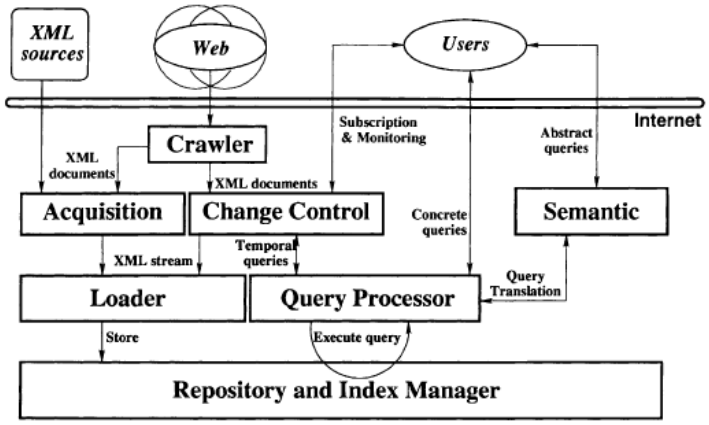

Figure 2.1: Xyleme functional architecture

(view) over the global model. LAV has the advantage of easily integrating new sources (open world assumption) and are consequently more appropriate for data integration on the web. The main problem of LAV system is the complexity of query rewriting, known as the problem of answering queries using views [Hal01] (sources are views over the global model), that is discussed in more details in Section 3.1. Systems such as Information Manifold [LRO96], Tukwila [IHW02], Picsel [GLR00], STYX [ABFS02] are based on LAV integration schemas.

More recent approaches for defining the integration view include *global-local-as-view* [FLM99] (GLAV) and *both-as-view* [MP03] (BAV). GLAV can express both GAV and LAV with the same type of mapping rules; a GLAV model definition has the form $q_s \mapsto q_g$. BAV uses reversible schema transformations sequences to express the relation between global and local schema elements, which can be used at query rewriting.

In this context, we addressed in Xyleme the particular problem of building a data integration system *at the scale of the web*, i.e. scalable in both number of sources and number of users. The challenges concerned more data heterogeneity, the size of the integration view and the methods for creating and maintaining it, than the expressive power of the view or query rewriting. Also, we addressed the problem of integrating *XML data*, stored in a *native XML repository* and did not consider the problem of XML views built on top of relational storage, such as in SilkRoute [FKS+02], XPERANTO [CKS+00] and many others. At that time, with the significant exception of web-search engines for text documents, data integration systems only addressed small- or medium-scale integration. Recently, new approaches for data integration at the web scale were proposed, such as MetaQuerier [CHZ05] and PayGo [MCD+07].

2.2 The Xyleme data integration model

2.2.1 The Xyleme system

Xyleme [ACFR02] is basically a warehouse of XML documents, acquired from the web or from local sources; the warehouse is distributed in a cluster of computers connected through a fast speed local network. Xyleme is able to store all the XML documents from the web and to provide a set of advanced services on top of this data, illustrated in the functional architecture presented in Figure 2.1.

The *Repository and Index Manager* module allows storing, retrieving and indexing XML documents. Storage is based on a native XML database and indexing on a fast, in-memory full-text XML

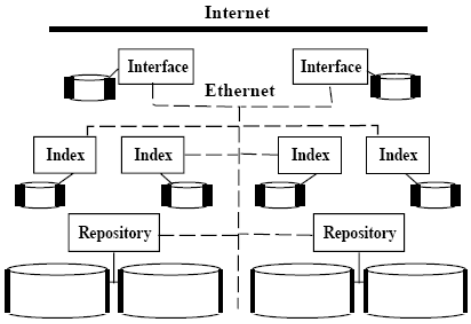

Figure 2.2: Distributed architecture in Xyleme

index structure. The *Acquisition and Crawler* module inspects the web and local content to collect XML documents, which are loaded in the repository by the *Loader* module. The *Change Control module* is responsible of monitoring of document changes, version management and subscription of temporal queries. The *Semantic module* provides a homogeneous integrated and mediated schema on the heterogeneous XML documents stored in the repository. The *Query Processor module* enables to query the XML repository as a database. In particular, it translates a query in terms of the semantic layer into another one computable on the stored documents.

My work in Xyleme concerns the Semantic and Query Processing modules, as described in the next sections. More details on the other Xyleme modules are presented in [KM00, MAA+00, MACM01].

Scalability is guaranteed by the distribution of the repository and of the index on several computers, such as presented in Figure 2.2, that illustrates the distributed architecture of Xyleme.

- *Repository machines* (RM) are in charge of storing the documents. Data is clustered according to a semantic classification, each RM storing potentially several clusters of semantically related data (e.g., art, and literature), in order to reduce the number of machines that have to be accessed to evaluate a particular query.
- *Index machines* (XM) have large memories that are mainly devoted to indexes. Clusters are partitioned on index machines so as to guarantee that: (i) all indexes reside in main memory; and (ii) each XM is associated to only one RM.
- *Interface machines* (IM) are connected to the Internet. They are in charge of running applications. Whereas the number of RMs and XMs depends on the warehouse size, the number of interface machines grows with the number of users.

2.2.2 The data integration model

The Xyleme data integration model uses *a mixed mediator-warehouse* solution, that combines benefits from both approaches. XML files from the web are stored and indexed in the local repository, without being transformed to a global model. This allows fast query processing and the use of the same query language for querying data from all the sources. Freshness problems are handled by the Change Control Manager, that monitors changes in data sources and can trigger the reloading of the data source into the repository.

We use a LAV integration schema, to be able to deal with the variable number of sources on the web. As explained above, Xyleme data is organized in *clusters*, built through semantic classification

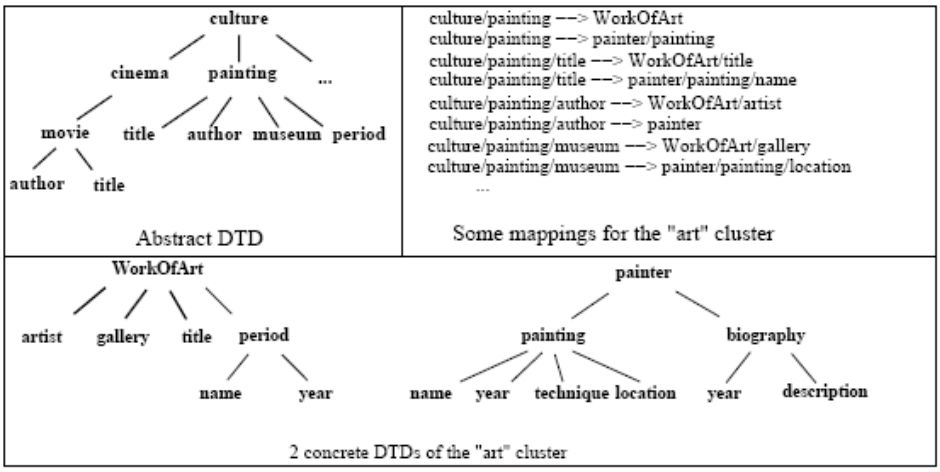

Figure 2.3: Example of Xyleme data integration view

of documents. A view is defined over an *abstract cluster*, composed of a set of real clusters. The global schema is a tree, called *abstract DTD*, describing a homogeneous structure of XML documents accessible through the integration view. Each XML source is described by a local tree schema, called *concrete DTD*. Concrete DTDs can be automatically extracted from a cluster by building tree data summaries, similar to Lore data guides [GW97]. Figure 2.3 presents an example of integration view for cultural information, built over the abstract cluster ”culture” composed of real clusters ”art”, ”literature”, ”cinema” and ”tourism”.

Mappings are expressed as a set of two-way correspondences between a node in the abstract DTD and a node in a concrete DTDs, represented as paths (path-to-path mappings). This simple representation has the advantage to be *simple to generate automatically*, which is essential for web-scale view creation and maintenance, and to be *reversible*, which simplifies query rewriting. Actually, a path-to-path mapping $p_{global} \leftrightarrow p_{local}$ has the following meaning: all the XML elements that match p_{local} in the source data are included in the answer to the global query p_{global}. This corresponds to a GLAV definition of the view and is also close to BAV because of its reversibility.

The problem of path-to-path mappings is that when the global query is composed of several paths (tree queries), the way of combining the interpretation of each individual path cannot be expressed by a general rule (constraint) and may vary from a source to another. A solution would be to use *tree-to-tree mappings*, that express correspondences between a subtree of the abstract DTD and a subtree of a concrete DTD, but the problem is that such mappings cannot be automatically generated with precision.

With the constraints of web-scale views, the choice of path-to-path mappings is a good compromise. We define algorithms for automatic mapping generation (see Section 2.4 below) and a set of general rules for combining mappings in tree queries at query rewriting (see Section 2.3 below). This pragmatic solution provides incomplete answers in the general case, but can be considered as acceptable in web-scale applications.

A novelty in the Xyleme integration system is *the distribution of views* over the local Xyleme cluster. The idea is to follow the data distribution schema for clusters over Repository machines: for each cluster stored on a RM, all the view mappings concerning that cluster will be stored on the Index machine (XM) associated to the RM. This method guarantees both storage and query processing scalability for views, by sharing the storage and processing effort among XMs, as explained below.

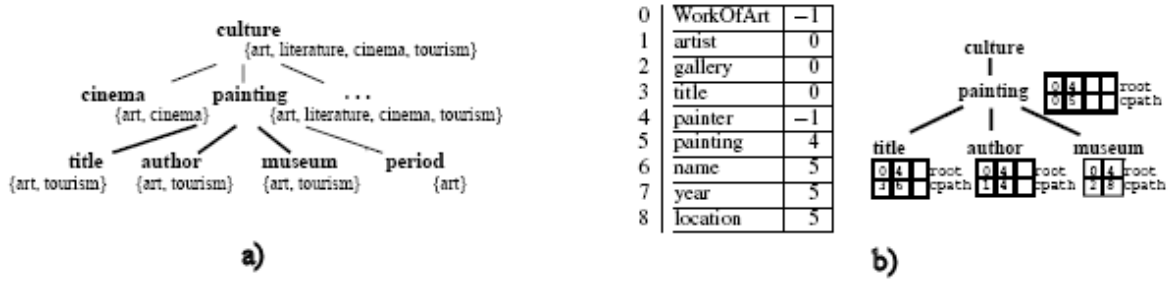

Figure 2.4: View structures on an interface machine (a) and on an index machine (b)

2.3 Query rewriting in Xyleme

View representation Figure 2.4 presents the distribution of the view over the Xyleme network. Interface machines store *a global description of the view*, composed of the abstract DTD tree, where each node is annotated with the set of clusters that contain data corresponding to that node. Index machines store *a local description of the view*, composed of the set of concrete DTDs for the clusters indexed by that XM, together with the set of mappings between these concrete DTDs and the abstract DTD.

The forest of concrete DTDs on an XM is represented by a table, with one entry per tree node, containing the tag name and the entry number for the node's parent, e.g. in Figure 2.4 entry 5 represents the path *painter/painting* in the concrete DTD of root entry 4. Mappings are represented by the abstract DTD tree, where each node is annotated with the list of local concrete DTD nodes mapped to it. A concrete DTD node is represented by a couple of integers $(root, cpath)$, where $cpath$ and $root$ are respectively the entry numbers for the node and for the root of its concrete DTD, e.g. the abstract path ***culture/painting*** is mapped to *WorkOfArt* in the concrete DTD of root entry 0 and to *painter/painting* in the concrete DTD of root entry 4. Lists of $(root, cpath)$ couples are sorted in ascending order. This representation is optimized for the query rewriting algorithm presented below.

Queries on views are similar to queries on real XML data, but address the tree structure of the abstract DTD. The main algebraic operator in the Xyleme query algebra is *PatternScan*, that filters a cluster through a query pattern tree [AYCLS01], e.g. the query pattern tree in Figure 2.5 asks for the titles of paintings of Van Gogh at the Orsay museum. The difference between view queries and normal queries is the introduction of two specific operators for query rewriting:

- **AQT** (Abstract Query Translator): executed on the Interface machine, transforms a *PatternScan* on an abstract cluster into a *union* of *PatternScan* on real clusters and sends them only to the corresponding Index machines (see Figure 2.5).

- **A2C** (Abstract to Concrete): executed on the Index machine, transforms a *PatternScan* on the abstract DTD received from the AQT into a *union* of *PatternScan* on concrete DTDs. Each resulting *PatternScan* is then executed on the local Index machine, as for a normal query. Figure 2.6 shows one possible translation of the abstract tree query.

Query rewriting Query rewriting in A2C considers several constraints, which reduce the number of rewritings to a reasonable amount for web-scale applications, by keeping only the most "pertinent" rewritings:

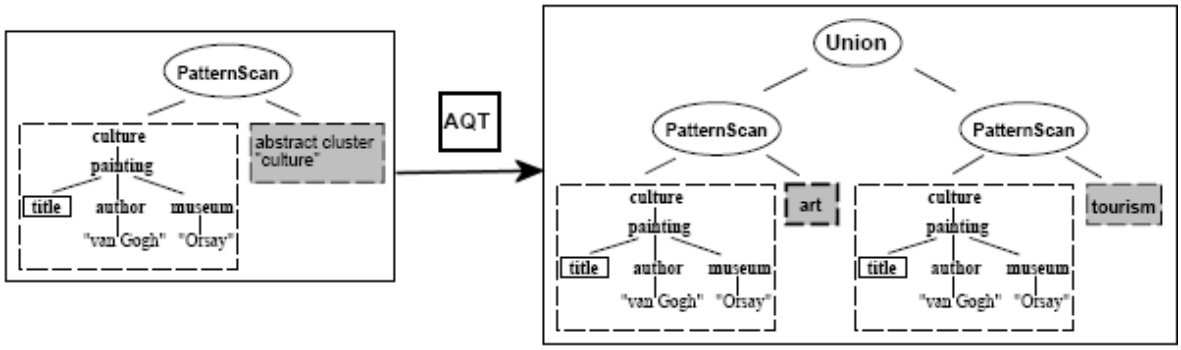

Figure 2.5: AQT operator

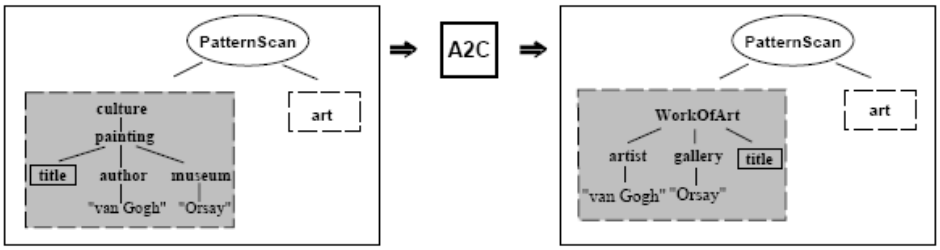

Figure 2.6: A2C operator

1. When rewriting an abstract query tree, all the paths must be translated into paths of a same concrete DTD, i.e. there are no joins between documents.

2. Rewriting must preserve the descendant relationship, e.g. if abstract paths ap_1 and ap_2 are translated into concrete paths cp_1, respectively cp_2 and ap_1 is a prefix of ap_2, then cp_1 must be a prefix of cp_2. This rule preserves only the most semantically close rewritings.

3. An abstract path should not have two mappings along the same concrete path, i.e. ap should not be mapped to both cp_1 and cp_2, such that cp_1 is a prefix of cp_2. However, if this happens (very rarely in practice), only the most specific (the longest) path is considered as valid. This rule allows reducing the complexity of query rewriting.

The idea of the algorithm, detailed in [ACM+02] is to consider all the branches of the abstract query tree, starting from the leaf up to the root, and to build rewritings of each branch, then "join" them at the junction nodes. Constraints 2 and 3 above guarantee that the cost of finding one rewriting for a branch is proportional to the branch length. Moreover, since mappings are sorted by concrete DTD, branch join can use traditional mergesort algorithms, thus keeping the cost of finding all the tree rewritings linear in the number of concrete DTDs. We shown in [ACM+02] that the average cost of producing all the rewritings is $C_{rewriting} = O(khm^l)$, where k is the average number of mappings per abstract path, h is the height of the query tree, l is the number of leaves in the query tree and m is the average number of mappings for a path in one concrete DTD. In conclusion, query rewriting is scalable with the view size (which is proportional with k).

Improvements A different approach for query rewriting was presented in [DRR+03], based on the idea of precomputing and storing all the branch translations, together with mappings, distributed on

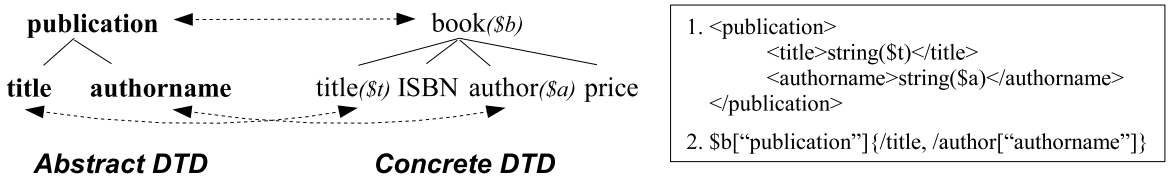

Figure 2.7: Results typing

the Index machines. Then, query rewriting only selects the branches that correspond to the current query and computes the branch join.

Other enhancements of the query rewriting algorithm have also be proposed in [ACM+02]. One of these is allowing *joins by links* between documents. The idea is to mark in the global view structure, stored on the Interface machines, what abstract paths have mappings to concrete paths containing links. At query rewriting, a join-by-link plan may be generated each time a concrete DTD can only answer a part of the query tree. Another enhancement is the definition of several levels of *query relaxation*, to be activated when a query does not get enough answers. Query relaxation is obtained by eliminating some of the constraints imposed by the A2C algorithm: descendant relation preservation, join of branches at all the junction nodes, translation for all the nodes of each branch, etc.

Results typing Another problem addressed in this context is *the typing of query results*. The answers to an abstract query come from heterogeneous sources and should be transformed in order to respect the homogeneous types of the abstract DTD. This is important, e.g. when query results are handled by programs that need precise and homogeneous types in order to be able to process data.

In Xyleme, we faced the problem of transforming XML results following the abstract XML types, with a query language that does not support nested queries, nor set variables. The Xyleme query language is equivalent to only a subset of full-text XQuery, but sufficient for most applications and highly optimized.

Figure 2.7 presents a fragment of an abstract DTD and one concrete DTD with the corresponding mappings. ”publication” elements in the view correspond to ”book” elements in the documents, however, ”book” elements contain some extra information (ISBN and price). A query on the view, which asks for ”publication” elements, should only extract title and author information from ”book” elements. We introduced a *grouping operator* in the Xyleme query algebra and language, that, for a given element, allows grouping together its subelements (at any level) specified by the parameters. Moreover, this operator allows *renaming* the selected subelements.

On the right side of Figure 2.7 two return expressions for result typing are presented. The first one corresponds to the classical approach of building the result tree from fragments. The problem is that variables *$t* and *$a* are monovalued, so if a book has a title and several authors, it will produce one result for each author. The second expression, presented with a simplified syntax, corresponds to our grouping operator. It extracts from books (renamed to ”publication”) only title (not renamed) and author (renamed to ”authorname”) subelements, that remain grouped together for each book. This operator allows expressing subelements through path expressions and can be nested. This grouping operator can also be seen as a *selective projection* operator, because it allows projecting only a part of the composition subtree.

2.4 Automatic generation of mappings

The use of automatic techniques for building and maintaining views is essential for views at the web scale, which cannot be handled manually. As shown above, the choice of the view model, based on path-to-path two-way mappings is strongly influenced by this issue.

We proposed a general method for automatic generation of path-to-path mappings between two tree structures, presented in [RSV01, DRR$^+$03] and based on two main ideas:

- Two paths can be mapped only if *their ending tags are similar*. We used the WordNet thesaurus [Fel98] to determine word similarity. WordNet provides several semantic relations, such as synonymy, hypernymy (generalization), meronymy (composition), etc. E.g., *culture/painting/museum* can be mapped to *WorkOfArt/gallery*, because *museum* and *gallery* are synonyms. Additional techniques for dealing with composed words (e.g. *ProductName*) are also defined, but not detailed here.

- Two paths can be mapped only if *their contexts are similar*. Ending tag similarity is not enough, e.g. *artist/name* should not be mapped to *museum/name*, because they represent the name of two different entities. We considered that the context of a path, if it exists, is one of its prefixes (the context of a node is one of its ancestors). A path p can be mapped to a path p' only if the context of p is mapped to the context of p'. In practice, we cannot define the context for all the concrete paths, but we can do it for abstract paths, whose number is limited. In this case, the rule is: an abstract path ap can be mapped to a concrete path cp only if the context path of ap is mapped to *a prefix* of cp.

Part of these techniques are similar to those developed in Cupid [MBR01], Clio [HMH01], COMA [DR02] and other schema matching methods [RB01], proposed at the same period with our work on Xyleme.

We realized several prototypes and a graphical tool for Xyleme, called *MapGen* (Figure 2.8), able to generate, edit and update mappings for Xyleme views. MapGen allows editing the abstract (virtual) DTD tree and annotating each abstract node (path) with:

- *A set of similar words*, used for tag similarity. Instead of using an on-line thesaurus, such as WordNet, we choose to manually build this set, using thesauri and dictionaries only at annotation time. The advantages are a faster mappings computation (no thesaurus queries at run-time) and more precision, by selecting only the appropriate similar words in the given context and by including domain-specific knowledge.

- *The context node*, if exists, e.g. *Article/Text* is the context of *Article/Text/Paragraph* in the example in Figure 2.8.

- *The identifier mark*, that indicates that the current node is an identifier property of its context node, e.g. *museum/name* is an identifier property of *museum*. This information helps finding mappings such as *museum/name* $\leftrightarrow$ *art/gallery*, even if *name* and *gallery* are not similar words.

Notice that the manual annotation of the abstract DTD is not scale dependent. After the view administrator annotates the abstract DTD, it can launch the automatic mapping computation, globally or concrete DTD by concrete DTD, can visualize and edit the results, can adjust annotations, save the view, etc.

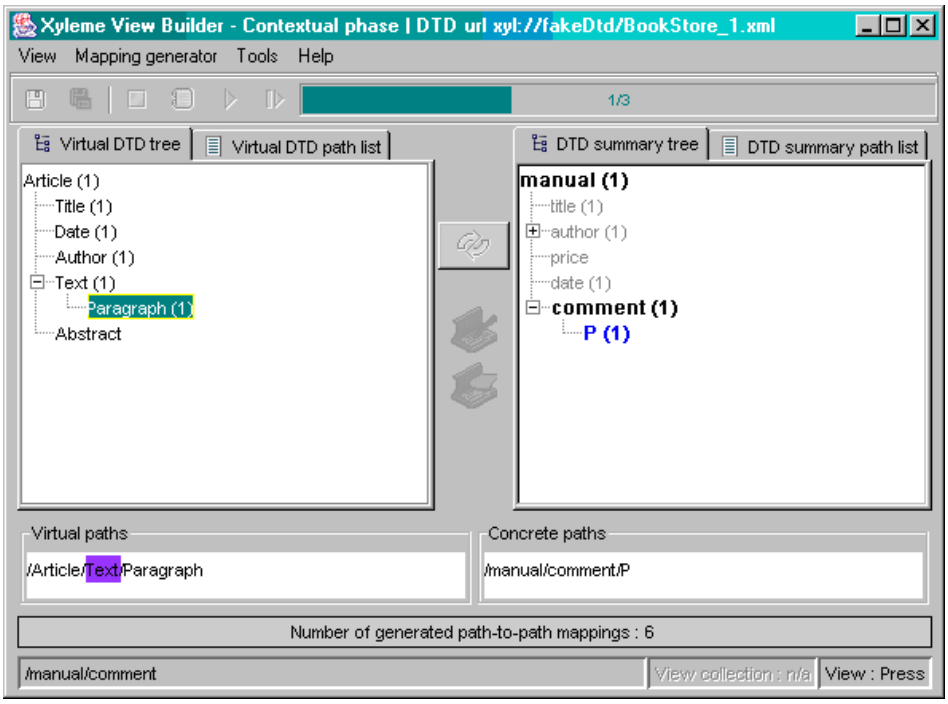

Figure 2.8: The *MapGen* view management tool

2.5 XML-ization of unstructured documents

If XML documents can be easily produced from structured data such as relational databases or spreadsheets, extracting meaningful XML from weakly structured content is a hard problem. Huge quantities of unstructured or weakly structured documents in various formats (PDF, Word, HTML, etc.) are only exploited as plain text. The transformation of such large amounts of data in XML has to be handled through appropriate tools and cannot be done manually.

We developed a Xyleme tool, called *XMLizer*, that allows identifying recurrent patterns in plain text documents, learning these patterns, annotating elements to be extracted, then using these patterns to extract XML structures from a document corpus, while accepting small variations in data with respect to the learned patterns.

The functional architecture of the *XMLizer* tool is presented in Figure 2.9. During the learning phase, documents from the learning corpus are processed to discover recurrent patterns. *The cleaning module*, parameterized with a pre-processing grammar, extracts only those parts of the documents concerned by the XML-ization process, and splits them in unit blocks (e.g. lines, paragraphs, etc.) The *model discovery module*, also parameterized with a token grammar, extracts a *pattern of tokens* (model) from each block and realizes a clustering of patterns by similarity. This module also detects repetitions inside a pattern (lists) and asks validation to the user in these cases. Clusters of patterns are stored in the *knowledge base*, where the user can delete, split, merge, modify them. Each cluster is characterized by a *regular expression pattern*, that the user annotates in order to specify what parts are to be extracted. The user specifies the target XML structure and the transformation of the extracted parts into this structure. Also, the user may indicate parts to be stored in *thesauri* (e.g. country names, person names), that are used for validation at run-time.

At run-time, documents to be XML-ized are split in blocks through the same cleaning module,

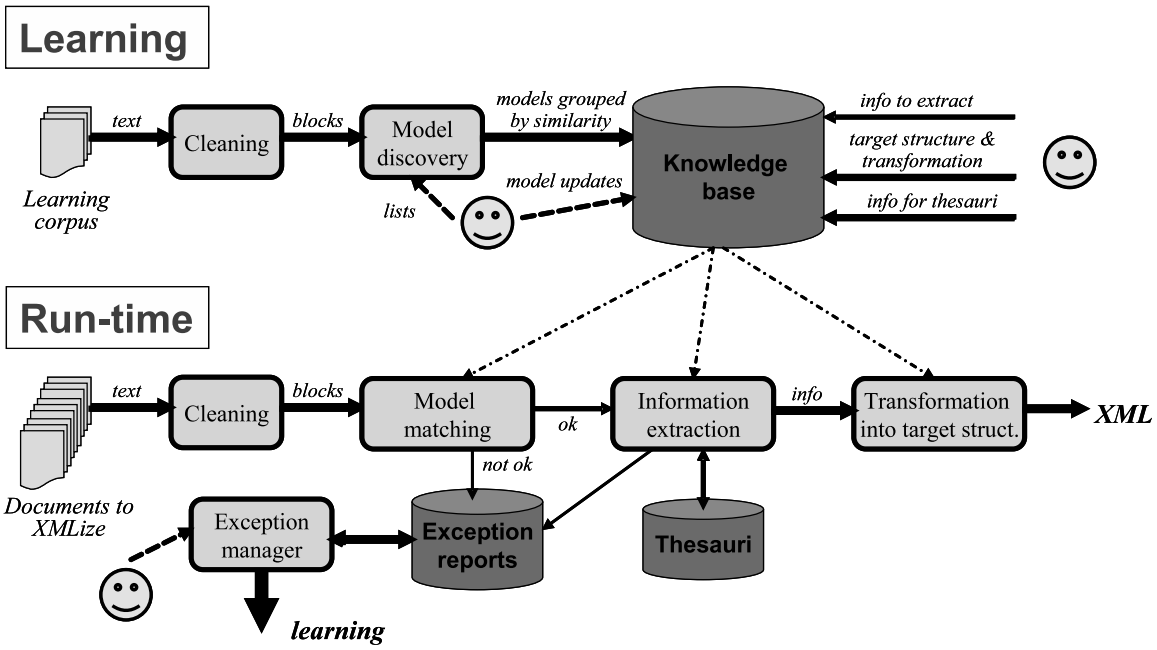

Figure 2.9: Functional architecture of the *XMLizer* tool

then patterns extracted from blocks are matched with those stored in the knowledge base. In case of successful match, information is extracted from the block, validated with the thesauri (if necessary) and transformed into XML. A salient characteristic of the *XMLizer* tool is that it allows *approximate matching*, i.e. if the pattern is close enough to one pattern in the knowledge base, the tool accepts it and realizes then an approximate extraction. Such situations are frequent in practice, e.g. an address that contain the floor number while addresses in the learning corpus did not, etc.

If no match is found or the approximate extraction cannot be done, an exception is generated and stored in the exception database. The *exception manager* allows to the user an off-line visualization of the exception context and possibly new document examples to feed the learning process.

The *XMLizer* belongs to the family of XML wrapping tools, such as W4F [SA99], Lixto [BFG01] and many other commercial tools. Unlike these tools, that generally use existing structures in documents (e.g. HTML tree structure, Word styles, etc.), the *XMLizer* is able to find structures in plain text, to learn patterns and lists, and to allow approximate matching.

Chapter 3

Views for heterogeneous XML data

This chapter presents the main elements of my work on views for heterogeneous XML data, in the context of the PhD thesis of Imen Sebei. After a brief presentation of the context of this research axis, we present the two parts of this work: the *XyView* model, for application views in Xyleme, and the *OpenXView* model, for open systems on the web.

3.1 Context: views, query rewriting and rapid application development

The work in this research axis is closely related to the data integration issues in Xyleme, presented in Chapter 2, but addresses a different context. We focus on application-oriented views for data integration, at a lower scale, thus in a more traditional data integration environment, without the limitations of web-scale views and where a better expressive power can be considered. If we still consider heterogeneous XML sources, the goal is here to make the access to this data as simple as possible, by exploring universal relation-like views for novice users and for rapid application development.

With such expressive application views, in the case of a (G)LAV integration schema, the problem of answering user queries is more complex than for Xyleme views. This issue, also known as the problem of answering queries using views [Hal01, Len02] (sources are views over the global model), has two aspects: (i) *query answering*, which consists of describing the set of data elements in sources that correspond to a user query, and (ii) *query rewriting*, which consists of translating the user query into the source query language. Many query rewriting algorithms, covering various data models and constraint types have been proposed, e.g. Bucket, Inverse rule and MiniCon [Hal01, PH01] for relational data, constraint-based algorithms for relational and XML data [DT05, YP04], etc. A particular case corresponds to higher-level models, such as Styx [ABFS02] or Yacob [SGS05], using ontology global schemas, where the rewriting algorithm is based on *implicit joins* between sources using key elements (Styx) or common attributes (Yacob).

The problem of simplifying the access to (heterogeneous) XML data was addressed in different ways. Users that want to query a collection of XML documents may have difficulties in manipulating XML structures and query languages, more complex than in the relational model. This is true not only for novice users, but also for application programmers that are not XML experts. To simplify query formulation, systems like XQBE [ABCC03] and Xing [Erw03] use visual specification of XML queries based on tree patterns, but users still need to handle XML structures, express joins, etc. Other systems allow writing queries with minimal knowledge about the structure of documents: keyword search in XML data [CMKS03, HPB03, GSSB03] or tag and keyword search [LYJ04]. Such systems are apropriate for interactive user queries, but not adapted for application development over

heterogeneous XML documents, because of their limited expressive power (e.g. no joins) and lack of precision.

Among the tools for rapid development of web applications over XML data, *Qursed* [PPV02] comes the closest to our application development context. Qursed enables rapid development of interactive applications over XML data, based on web query forms and reports. It provides a visual editor, which roughly takes an HTML query form (input for the user), a report template (output for the user) and an XML Schema describing the data. The application view is defined by mappings between input query fields and XML data, then between XML data and report output. The limitations of Qursed concern the management of heterogeneous or schema-free XML data and the development of non-interactive applications. Commercial products, such as BEA Liquid Data [BEA], provide advanced environments for data integration and web application development, focusing on specialized programmers. A major limitation in such tools is the lack of support for creating views over multi-source heterogeneous data; BEA Liquid Data uses query-based views, inappropriate for heterogeneous XML management, and is not designed to mix many sources into a single view.

In this context, our approach for simplifying the access to heterogeneous XML data is to define *universal relation-like views*, that provide a very simple query interface for users and application programmers; querying the view is as simple as querying a table through selection and projection operators. Unlike keyword- and tag-based search, view-based queries are precise, the price to pay being the creation and the management of this view. A possible approach for creating table views over XML data would be to shred XML in relations, physically (like many RDBMS today) or virtually ([HJL$^+$04]), then to create a relational view on top. This solution, that may work efficiently for homogeneous XML documents, with no structural variation, is not appropriate in our context: we want to build views over heterogeneous and possibly schema-free XML, stored in any system supporting an XML query language.

We also explored the best way of defining such application views and shown that mediator-based views are more appropriate than traditional query-based views for integrating heterogeneous XML data. Mediator-based views, defined through a set of mappings between structure elements, provide a more structured view model, easy to edit and to modify through visual, intuitive tools. Moreover, mediator-based views are adapted to the integration of many sources and are more flexible than query-based views at query rewriting for e.g. discarding useless elements and joins.

We proposed two such view models for heterogeneous XML integration:

- ***XyView***, adapted to data stored in XML repositories supporting an XML query language. XyView is based on a GAV integration schema, but accepts a limited set of source updates that require low view update effort.
- ***OpenXView***, adapted to open systems, where sources are autonomous and can publish new content in an unpredictable manner. OpenXView uses a (G)LAV integration schema adapted to a large number of heterogeneous sources. Query rewriting is based on *implicit joins* between sources, that are not handled appropriately by classical rewriting algorithms such as MiniCon [PH01].

3.2 The XyView model

3.2.1 Motivation and goals

Huge amounts of data produced by companies, such as notes, contracts, emails, progress reports, minutes, and other documents, are largely unexploited because they do not fit with traditional databases

```
<!-- Document 1: National league result -->
<GameResult>
   <WireHeading> ... </WireHeading>
   <Description> Real Madrid 1 - Valencia 0
   </Description>
   <Date> 2004-05-22 </Date>
   <Team>
      <Name> Real Madrid </Name>
      <Scored> 1 </Scored>
      <Scorer><PlayerName> Zidane </PlayerName>
              <Count> 1 </Count>
      </Scorer>
   </Team>
   <Team>
      <Name> Valencia </Name>
      <Scored> 0 </Scored>
   </Team>
</GameResult>
```

```
<!-- Document 2: Inter-countries game -->
<Result Date="2004-03-15">
    <Summary> France 1 - Spain 1 </Summary>
    <Scorers>
        <Player Goals="1">
            <Name> Zidane </Name>
            <Country> France </Country>
        </Player>
        <Player Goals="1">
            <Name> Raul </Name>
            <Country> Spain </Country>
        </Player>
    </Scorers>
</Result>
```

```
<!-- Document 3: Sports encyclopedia -->
<Encyclopedia>
  <Football>
    <Player><Name> Zidane </Name>
            <Biography>...</Biography>
    </Player>
    ...
  </Football>
  ...
</Encyclopedia>
```

Sample queries on football documents

Q_1: "Games in which Zidane scored more than once"
Q_2: "The biography of Zidane"
Q_3: "Biographies of scorers from games on 2004-09-08"

Figure 3.1: Examples of documents and queries

and tools. The advent of XML provides the opportunity to change that, by storing such data in XML repositories so as to be able to query them with tools more sophisticated than full text search engines. However, if data is heterogeneous and schema-free, querying the repository may be a difficult task. The XyView model addresses the problem of simplifying queries on such data and enables developing, easily and quickly, simple query API (web services) or user interfaces (web forms) over these repositories.

Consider the example of a sports news company that handles several types of news wires, which are well formed XML documents extracted from text files, with no global schema, and with different structures. Figure 3.1 shows two such wires about football, containing results from national leagues (Document 1) and results from international games (Document 2). The news company wants to build an application that queries through simple web forms the various football results wires and a sports encyclopedia with detailed information about football players (Document 3). Document similar to these three categories are stored in a single XML repository, in collections identified respectively by *NationalURI*, *InternationalURI* and *EncyclopediaURI*.

The application queries, as those in Figure 3.1, may concern football results (Q_1), player biographies (Q_2), or both (Q_3). These apparently simple queries are in fact rather hard to program in XQuery as illustrated by Figure 3.2 for Query Q_3 (where the result is supposed to be a string).

We want to simplify the task of users that query this repository, by allowing them to view the database as something as simple as a query form consisting of fields that can be used to filter or extract data. The solution we propose borrows from the universal relation paradigm [Ull83]: *XyView* provides the means to easily view a set of heterogeneous XML documents as a single array that can be queried through simple selections and projections. Besides the fact that the elements of this array are XML trees instead of atomic types, two fundamental differences with classical universal relations exist:

- *The array is not defined by a query*, but by a set of mappings and joins between structure elements, such as in mediator views. The problem with universal relations is that, due to join operations, projections generate many duplicates that are not always easy to remove, or alternatively, joins can also be the cause of missing information. This is usually solved by introducing outer-joins but at the cost of having to deal with null values.

```
union(
   For $doc1 in collection(NationalURI),
       $var1 in $doc1/GameResult,
       $doc2 in collection(EncyclopediaURI),
       $var2 in $doc2/Encyclopedia/Football/Player,
       $var3 in $var2/Biography
   Where $var1/Date = xs:date('2004-09-08') and
         $var1/Team/Scorer/PlayerName = $var2/Name
   Return string($var3),
   For $doc1 in collection(InternationalURI),
       $var1 in $doc1/Result,
       $doc2 in collection(EncyclopediaURI),
       $var2 in $doc2/Encyclopedia/Football/Player,
       $var3 in $var2/Biography
   Where $var1/@Date = xs:date('2004-09-08') and
         $var1//Player/Name = $var2/Name
   Return string($var3) )
```

Figure 3.2: Query Q_3 expressed in XQuery

By defining it through a set of mappings and joins, the view is not equivalent to a query, but to a *virtual set of queries* that are generated on the fly to fit the user query, avoiding useless joins and useless paths in the view structure. The virtual set of queries is deduced from the set of view mappings and joins; XyView uses a GAV integration schema, where the role of query rewriting is mainly to discard all useless view elements, in relation with the user query.

- *The view definition is highly structured*, organized at several levels. The task of the view administrator is much simpler when the view is defined through mappings and joins, because it may be created, edited and maintained through intuitive, visual tools.

  Moreover, we introduce an intermediary view level, the *logical data view* (LDV), that deals with *data heterogeneity* by providing homogeneous tree structures for groups of heterogeneous sources, and that separates join and union operations in the view definition, thus providing a simpler view structure. This level also separates the view definition in two mapping layers: a GAV one, from the user view to the LDV, and a GLAV one, from the LDV to source data, thus providing a better control on view updates, e.g. when new sources are added.

The practical goal of XyView is to optimize the productivity of web application programmers, who are not database experts and to simplify as much as possible the task of creating and maintaining such views. We created a set of tools on top of the Xyleme repository, which can easily be adapted to any system supporting XQuery. These tools cover the view editing process, but also automatic generation of web form applications based on these views.

3.2.2 The data model

A XyView view is organized at three levels, connected through mappings and join predicates. Figure 3.3 presents the view levels in the case of the sports news example (mappings and join predicates are not shown).

1. The first level deals with schema-free data, by defining ***physical data views*** (PDV) that summarize XML access paths to useful information in documents. The example view contains three PDVs that correspond to the three different document structures: *National*, *International* and *Encyclopedia*.
2. The second level deals with heterogeneity, by defining integrated ***logical data views*** (LDV) over *unions* of physical data views with similar contents. The example view contains two LDVs, *Game*, which provides a homogeneous structure for game results in PDVs *National* and *International*, and *Encyclopedia*, which covers the last PDV.

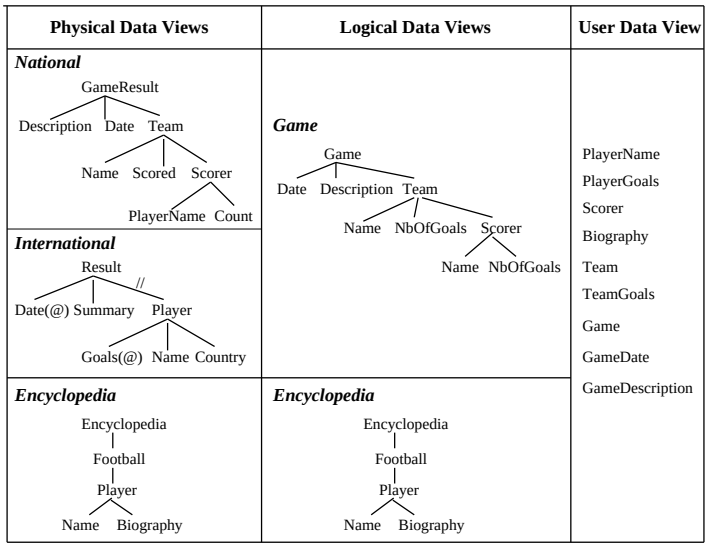

Figure 3.3: View levels for the sports news example

3. The third level defines the ***user data view*** as *joins* between logical data views. Its structure is composed of a set of *concepts*, that the user wants to query as a table. This table represents the join between the two LDVs (the scorer name in *Game* must be the same as the player name in *Encyclopedia*).

Physical Data Views (PDV) Physical data views represent the structure of XML data as it is stored in the repository. We consider that data is organized in collections of XML documents (clusters) and that each collection provides a *data summary*, i.e. a set of tree structures representing the access paths to documents in that collection. Data summaries, similar to Lore data guides [GW97], represent *implicit DTDs* for the XML documents in the collection and allow handling schema-free data.

All the elements in a data summary do not necessarily correspond to elements in the user view. A PDV may discard useless elements, by removing branches or by creating shortcuts in long branches by using descendant connections ($//$). E.g., in the view example in Figure 3.3, the *WireHeading* subtree has been removed from PDV *National*, while in PDV *International*, element *Scorers* has been discarded from the path to *Player*, because it is useless and removing it introduces no ambiguity; the edge leading to *Player* is marked with $//$. Note that there is an edge above each root element in the example figure, not shown for simplicity; this edge may also be marked with $//$ if the root element is discarded. Tree simplification eases the view design process, by keeping only useful access paths from possibly cumbersome document structures. Also, $//$ shortcuts significantly improve query processing of the final XQuery, by reducing the number of structural conditions to check.

Actually, the notion of PDV is more general than its illustration in XyView. Generally speaking, a PDV is an intermediary view level between data sources and the global level. If a data source $S = (S_S, C_S)$ is defined by a collection of XML documents C_S and a tree schema S_S, and the global level is defined by a global schema S_G, then a PDV p over the data source S is defined by:

- *A PDV tree structure* S_p, obtained from the source tree structure S_S by eliminating useless nodes, as explained above. The PDV tree structure is a tree with labeled nodes and edges, where edge labels may be "/" (default value) or "//".
- *A collection of documents* C_p over which the PDV is defined, its default value being the data source collection C_S.

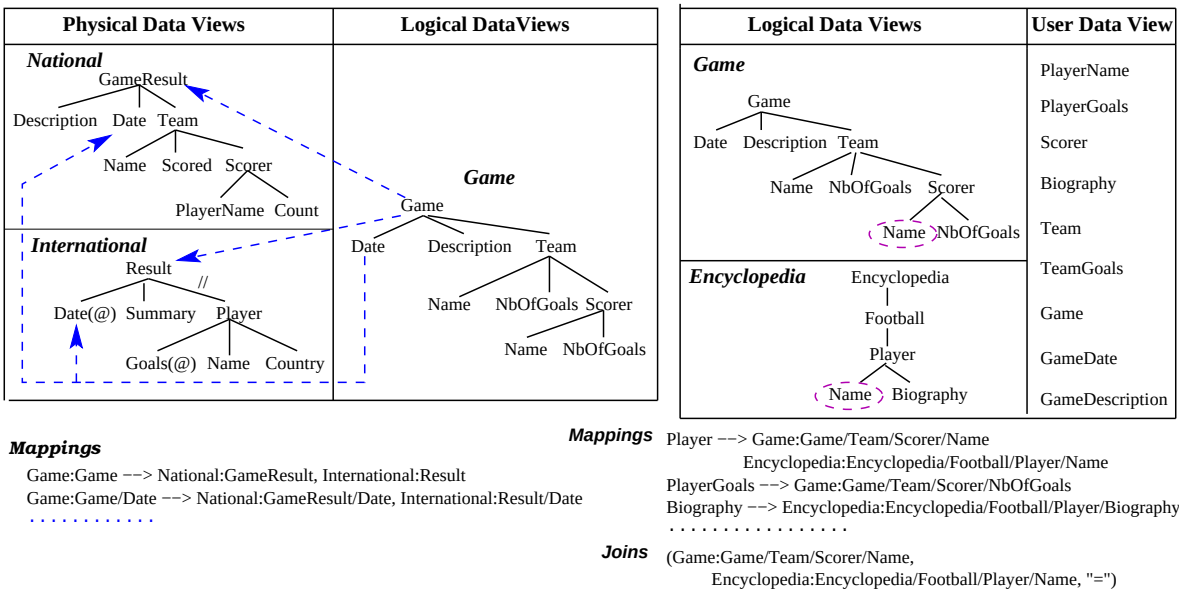

Figure 3.4: From physical to logical data view, from logical to user data view

- *An element-to-element mapping* between the global schema S_G and the PDV structure S_p, such that any element of S_G is mapped to *at most one* element in S_p. Therefore, the mapping between S_G and S_p can be defined by a *partial function* defined on the set of elements in S_G, with values in the set of elements in S_p.

The mapping constraint in the PDV definition is essential. Figure 3.4 illustrates XyView LDV-PDV mappings, represented as sets of GLAV path-to-path relations, similar to those in Xyleme views described in Section 2.2.2. The difference with Xyleme views comes from the mapping constraint: *a LDV path cannot be mapped to more than one path in a PDV*. This rule guarantees that any set of paths in a LDV (any subtree) has *at most one translation* into a set of paths (subtree) of a PDV. In Xyleme views this was not true, and an abstract tree pattern could have many translations, filtered by the set of constraints we defined; the problem is that some valid rewritings may be lost, while some of the rewritings obtained may be wrong. This is acceptable for web scale applications, but not for the XyView context.

In XyView, a single rewriting is possible in each PDV, thus precision is guaranteed by the view designer that created the PDV. But this also means that a document structure may lead to several PDVs and the price to pay is that the view designer must carefully extract the PDVs from the data summary structures and maintain them in case of changes. An example is presented in Figure 3.5: for the LDV on the left, two PDVs can be extracted from the document structure (we consider the obvious path-to-path mappings). Notice that Xyleme views would produce no rewriting of the LDV tree, because the *workOfArt - style* descendant relationship is not preserved in the document structure. Notice also that automatic PDV extraction, by combining path-to-path mappings, could be considered, but user validation cannot be avoided. In our example, the *gallery/style* element may correspond to the style of paintings and not to the one of the other art items; only the view administrator can decide if *gallery/style* should be included or not in the second PDV.

For query answering over PDVs, the semantics of a PDV p over a collection of documents c may be summarized as follows. The interpretation of any tuple $[pn_1, ..., pn_k]$ of nodes of the PDV tree is given by the evaluation of a *query pattern tree* over the collection c. The query pattern tree is the

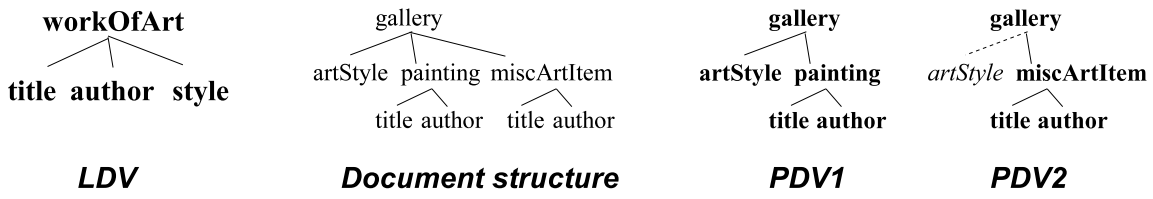

Figure 3.5: An example of multiple PDVs in a document structure

subtree of p that has $pn_1, ..., pn_k$ as leaves and that asks all of them for projection. Intuitively, the elements in the returned tuples are *the closest possible* wrt the PDV, because they have the closest possible common ancestor in the document (because of the pattern tree structural constraints).

Logical Data Views (LDV) Logical data views are defined by a group of PDVs, a homogeneous tree structure for those PDVs and a set of path-to-path mappings between the LDV structure and the PDV structures, illustrated in Figure 3.4 and discussed above.

LDVs provide a homogeneous XML structure for query results, i.e. the value of a concept mapped to an LDV element has the XML structure of that element in the LDV tree. For query answering, the semantics of an LDV l over a set of PDVs P may be expressed as follows. The interpretation of any tuple $[ln_1, ..., ln_k]$ of LDV elements in l is ***the union*** of the intepretations of tuples $[pn_1, ..., pn_k]$ for all PDVs $p \in P$, where pn_i is the (unique) element of p mapped to ln_i, $\forall i \in \{1, ..., k\}$. If some ln_i has no mapping in p, the interpretation of $[pn_1, ..., pn_k]$ is considered to be empty.

This union semantics of LDVs and the GLAV mappings between LDVs and PDVs allow flexible updates of the view inside each LDV group. E.g., to add a new PDV to a LDV group, one have to only compute the mappings between the new PDV and the LDV.

User Data View The user data view consists of a set of typed concepts, their mappings with LDV elements and a set of predicates that are used to join the LDVs. Figure 3.4 illustrates the user data view for the sports news example. Mappings, of the form *concept-to-LDV path*, are similar to LDV to PDV mappings, but join specification, as a set of predicates relating paths in the joined LDVs, makes the global mapping between concepts and LDVs a GAV one (an LDV depends of other LDVs because of the joins). The example shows several mappings and the unique join (by equality) between the player names in the two LDVs.

Concepts are typed by the view designer, for instance *PlayerGoals* is an integer, *GameDate* is of type date, *PlayerName* is a string, while *Game* is an XML element.

The semantics of a user data view can be intuitively summarized as follows. The interpretation of a tuple of concepts is a n-ary join of partial tuples of LDV nodes, found in LDVs through mappings.

View construction The components of a view depend each other, e.g. PDVs depend on the data summaries and on mappings to LDV elements (in order to respect the unique mapping constraint), LDVs depend on the set of corresponding PDVs and on the concepts mapped to them, etc. In this context, the view definition flow could respect the following schema:

1. Fix the view domain, i.e. the set of document collections in the repository, then extract the data summaries for these collections.
2. Establish the set of concepts (the user view), based on the knowledge that the view designer has about data in the repository and the application domain.
3. For each data summary, establish the mappings between tree nodes and concepts, then extract a set of PDVs from the data summary structure by respecting the unique mapping constraint.

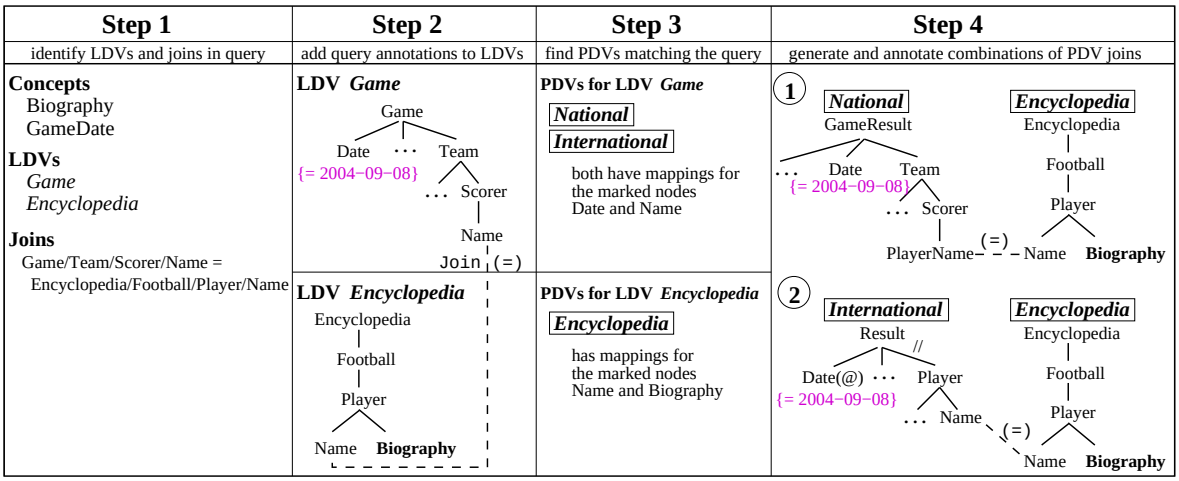

Figure 3.6: Steps for translating Query Q_3: `Select` *Biography* `Where` *GameDate*=2004-09-08

4. Group PDVs by the set of concepts they cover (i.e. the concepts mapped to the PDV); a group contains PDVs that cover the same (or almost the same) set of concepts.
5. For each group, build an LDV structure and create the mappings to the PDVs.
6. Create the concept to LDV mappings and define the join predicates.

3.2.3 Query rewriting

As is the case with universal relations, the query language supported at this level consists of selections and projections over concepts, where selection predicates depend on the concept type. For instance, Query Q_3, that returns biographies of scorers from games on 2004-09-08, has the form:

`Select` *Biography* `Where` *GameDate* = 2004-09-08

Query rewriting mainly consists of a GAV rewriting of the user query into a join of LDVs, then into a union of joins of PDVs. At each rewriting level, useless view elements (joins, tree paths) are discarded. The last step is to transform the PDV rewriting into an XQuery over the repository. Figure 3.6 illustrates this rewriting process for Query Q_3.

Step 1 identifies the sets of concepts (C_Q), LDVs (L_Q) and joins (J_Q) involved in query Q. Basically, C_Q is composed of concepts addressed in the selection or projection clauses of Q, L_Q of LDVs that have mappings to concepts in C_Q and J_Q of joins between LDVs in L_Q. This allows eliminating from the rewriting all the LDVs not in L_Q and joins not in J_Q. Other semantics for L_Q are possible in XyView, e.g. by removing redundant LDVs (those contributing with concepts already provided by other LDVs in L_Q), or by including LDVs that appear along some join path between LDVs in L_Q.

Step 2 consists of adding query annotations to the nodes of LDVs from L_Q. Annotations include projection marks, selection conditions and join marks, obtained from the corresponding query concepts, through mappings. This produces an equivalent rewriting of the query at the LDV level, as an n-ary join of LDV pattern trees.

Step 3 detects for each LDV in L_Q the set of PDVs that provide a non-empty interpretation of the annotated LDV nodes.

Step 4 generates all the combinations of PDVs found at Step 3; this produces the rewriting at the PDV level, as an union of n-ary joins between PDVs (one join per combination). LDV trees are replaced by PDV trees and node annotations are obtained from LDVs by using LDV-PDV mappings.

The final step produces the equivalent XQuery rewriting. First, each PDV tree is minimized, by keeping only paths that lead to annotated nodes. The resulting tree patterns are used to generate a union of flat FLWR XQueries, expressing the structural constraints in each PDV tree and joins between these trees. The final XQuery obtained for Query Q_3 is presented in Figure 3.2 above. More details on the rewriting algorithm are given in [VCCS06].

Comparison with query-based views XyView differs from standard view mechanisms relying on query composition: the view is not defined by an a priory query, but by mappings that allow an opportunistic adaptation of the view to the user query.

A query-based view must provide a full view over the document structures for all the concepts. A good candidate for the view query is *the user query that projects all the view concepts*, i.e. its translation through the previous algorithm. A query on this view faces problems such as ***data loss*** and ***duplicates***, due to the presence of unnecessary joins and PDV paths in the view definition wrt the user query.

For instance, a query that asks for player biographies would not return biographies of players that are not recorded as scorers, because of the join between LDVs *Game* and *Encyclopedia*, unnecessary for this query. Using outer-joins avoids data loss, but introduces null values and duplicates. Similarly, when asking for game descriptions, the view only provides games for which all the concepts in the *Game* LDV tree can be instantiated, i.e. games without scorers are not returned.

Also, the query asking for player biographies will produce duplicates because of the unnecessary join - the player's biography is returned each time it appears as a scorer. The same is true for the query on game descriptions - the same game description is returned for each scorer of the game. Duplicates can be eliminated through *distinct* operations, but this has a cost and it is sometimes difficult to distinguish between good (existing in data) and bad duplicates.

Typing Query results may be typed in several ways: (i) *flatten as strings*, such as for Query Q_3 in Figure 3.2, where the *string* function is applied to the element, (ii) *unchanged*, i.e. returning the elements without transformation, thus producing heterogeneous results, or (iii) *typed following the LDV tree structure*.

Typing is specified through *LDV node transformations*, described as node annotations. Consider for instance a query close to Q_3, that asks for *scorer information* (name and number of goals) from games on 2004-09-08. Results typed following the LDV should be *Scorer* elements, composed of *Name* (string) and *NbOfGoals* (integer) subelements. This is obtained by annotating each LDV node following its type. E.g., the *Scorer* node is annotated as follows:

Return: `<Scorer> $1 $2 </Scorer>`

Symbol "$i" indicates the i-th child of the current node in the LDV tree, which will be recursively typed following its own annotation. Note that XyView adapts the recursive tree typing of an element to each PDV: in the tree result, only branches that appear in the PDV are considered. For instance, if a PDV does not have a node corresponding to *NbOfGoals*, *Scorer* results in that PDV will only have a *Name* subelement.

We also use symbol "$$" to indicate the XML data element that corresponds to the current node. For instance, the annotation of a leaf node (such as *Name*) for LDV typing would be:

Return: `<Name> string($$) </Name>`

Note that this annotation mechanism allows expressing the three typing methods above and enables additional customization of results typing. More details on LDV node annotations in XyView are given in [VCCS06].

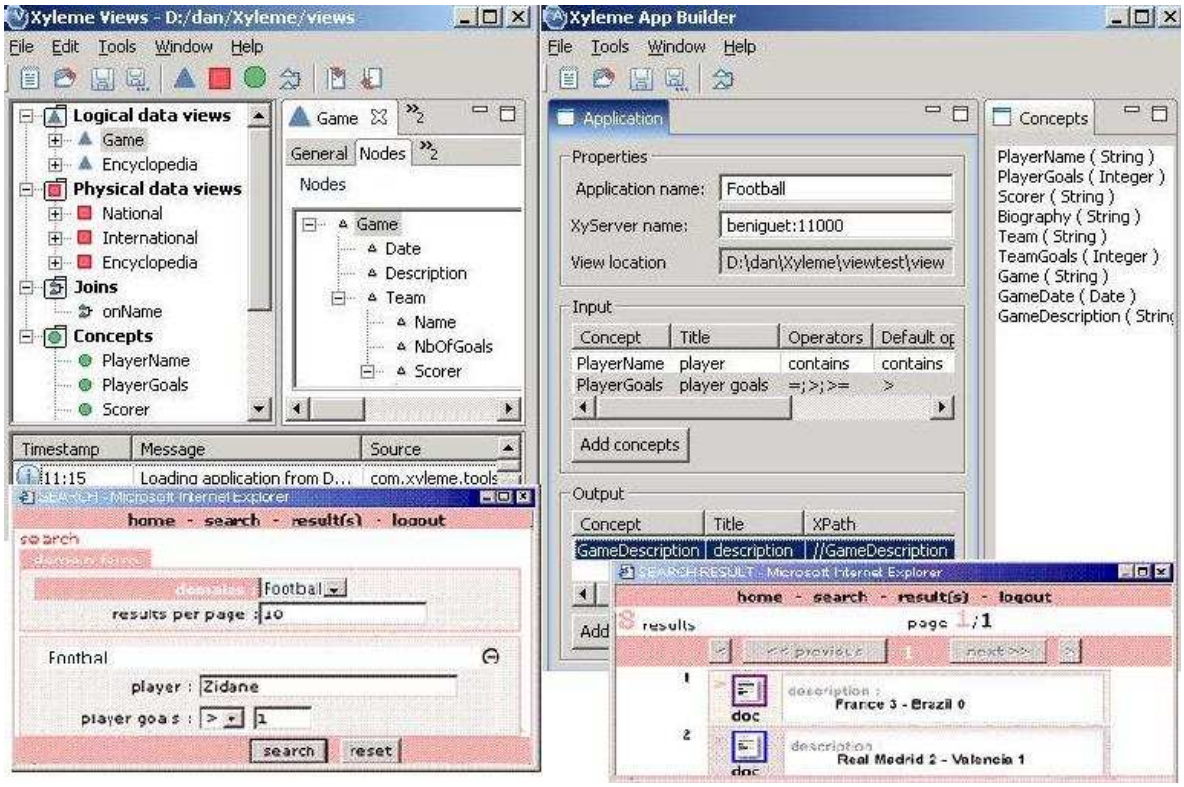

Figure 3.7: XyView editor and web-form generator

3.2.4 The XyView system

XyView has been implemented as a set of tools for rapid development of web applications over the Xyleme XML repository, but can be easily adapted to any system that supports XQuery. The XyView system is composed of the following modules:

- *A view editor* (upper-left window in Figure 3.7) that enables visual creation and modification of XyView views, through editors for each view component: PDVs (using data summary extractors), LDVs, concepts, mappings, joins, etc. Views are saved in a persistent form, as a set of XML files.
- *A run-time environment* that provides a simple Java API for using XyView views in web form or web service applications. The XyView API allows creating/modifying views, loading/saving views from/in the persistent form, building and translating user queries against the view. Note that XyView does not interfere in the communication between the application and the XML repository, but simply provides a query rewriting service; this architecture simplifies the adaptation of XyView to any system supporting XQuery.
- *A web-form application generator* (upper-right window in Figure 3.7) that provides a graphical environment for creating simple web-form applications over the Xyleme repository. The system automatically generates the HTML query form (bottom-left window) and the application servlets producing the query report (bottom-right window).

3.3 The OpenXView model

3.3.1 Motivation and goals

OpenXView [BSSV06a, BSSV06b, BSSV07], addresses the problem of XML data integration in *open* integration systems, over *a large number of sources*, where users may freely publish data in order to share information on common interest topics. A typical example is peer-to-peer [KP05] communities sharing structured content, such as XML data. The key characteristic of open integration systems is *source autonomy* in publishing data, leading to *frequent and unpredictable changes* in data and in the composition of the set of sources, and to *data heterogeneity* for documents whose structure was independently designed by different users.

A view model for heterogeneous XML data such as XyView is not appropriate in this context, because of the frequent and unpredictable changes in sources, which may lead to a reorganization of the view structure (the grouping of PDVs by LDV, the structure of LDVs, joins, etc.)

OpenXView is an evolution of XyView towards *a more flexible model*, where the intermediary LDV level disappears, where union groups and joins between them are automatically generated at query rewriting, depending on the user query. It uses a GLAV integration schema, appropriate for large scale integration in open systems.

The global integration model adopted by OpenXView is a mix of *ontologies* and *tree-like XML schemas*, that combines the advantages of these two most common global models for XML data integration [HIMT03]. The advantage of tree-like schemas is a lower model mismatch with source data, thus simplifying mapping generation, query rewriting and results typing. The advantage of ontologies is a better expressive power for the global model, mappings and queries, but mapping generation and mapping update are generally too complex for large scale integration. We propose a *a hybrid integration schema*: a simple ontology, where concepts have attributes organized in hierarchies (such as in XML structures), but may be connected through two-way ”relatedTo” relationships, more flexible for mapping constraints than ”partOf” XML relations.

On the source side, users publish *Physical Data Views* (PDVs), such as in XyView. Mappings, similar to LDV-to-PDV mappings in XyView, are directly expressed between the global schema and PDVs. They are generated together with PDVs at publication time.

Concerning user queries, OpenXView has the same goal as XyView: a very simple query language for heterogeneous XML data, based here on selections and projections over concept attributes in the ontology. Unlike XyView, *joins between documents are* ***implicit*** and are based on *concept key attributes*, defined for each concept in the ontology. This completely hides the way concepts are fragmented among sources, to both users that express queries and to those that publish data, thus providing a higher abstraction level in data management.

For now, we addressed in OpenXView the model definition and the problems of query answering and query rewriting; other issues, such as data publication, schema maintenance and query optimization are left for future work. We followed *two main goals* in OpenXView query rewriting: (i) to provide algorithms that are scalable with the number of sources, and (ii) to introduce criteria for limiting the number of query rewritings to a ”most pertinent” subset, their number being very large due to the use of implicit joins.

The closest related work to OpenXView are XML integration systems such as Styx [ABFS02], and Yacob [SGS05], that use an ontology as a global schema and implicit joins. Styx uses concept keys for implicit joins, but its query rewriting algorithm does not scale with the number of sources. Yacob uses both explicit and implicit joins, but defines a special (and very expensive) algebraic operator to compute concept extensions through implicit joins, instead of producing XQuery rewritings. Tradi-

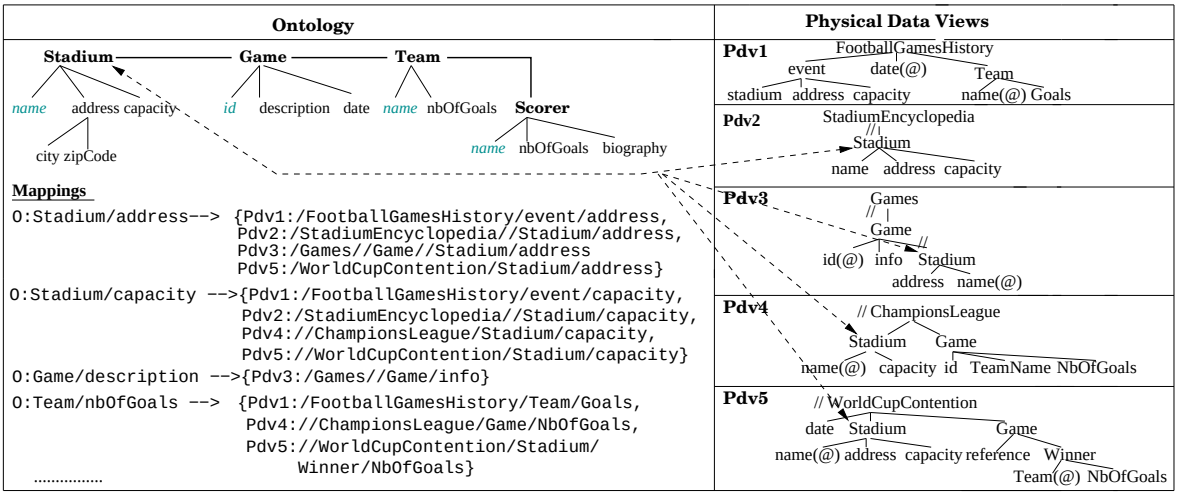

Figure 3.8: An example of OpenXView ontology, Physical Data Views (PDV) and mappings

tional query rewriting algorithms for LAV systems, such as MiniCon [PH01], or Chase & Backchase (C&B) [DT05] do not work in the case of implicit joins.

3.3.2 The OpenXView data model

We give here an informal presentation of the OpenXView data model elements, illustrated by an example.

Ontology, concepts, attributes An OpenXView ***ontology*** is a labeled graph, whose nodes, called *concepts*, have unique labels representing the concept name.

Each ***concept*** has *a set of attributes*, organized in a *composition hierarchy* represented as a tree with labeled nodes. By convention, the root of this tree has the same label as the concept. Attributes are not shared between concepts.

Attributes are *typed*; types may be *atomic* (integer, date, string, etc.) or *composed* (XML element). Only leaf attributes may have atomic types, all non-leaf attributes are composed.

Each concept has ***a key***, composed of a subset of its leaf attributes. Intuitively, each key value identifies an "instance" of the concept in the real data, as defined below.

Each edge in the ontology graph represents an untyped "relatedTo" ***relation between concepts***, that implies a relation between the concept instances, as explained below.

Figure 3.8 presents an example of OpenXView ontology, composed of four concepts (*Stadium*, *Game*, *Team* and *Scorer*), each one having a set of attributes organized in a composition hierarchy, e.g. the stadium address is composed of a city and a zip code. For simplicity, no distinction is made between a concept and the root of its attribute tree. Concepts keys are represented in italic font, e.g. *name* for concept *Stadium*, *id* for concept *Game*. There are relations defined between concepts (*Stadium - Game*, *Game - Team* and *Team - Scorer*), represented by an untyped edge (edge with no label) in the ontology graph.

Data sources, mappings OpenXView ***data sources*** are *collections of XML documents* that publish a *tree-like schema* over these documents. We use here the same source model as in *XyView*, where a data summary structure is extracted from the set of documents.

Also, we adopt the XyView's notion of ***Physical Data View*** (PDV), presented in Section 3.2.2. The only difference is that in OpenXView the global model above PDVs is the ontology, while in XyView it was the LDV. Each data source produces upon publication a set of PDVs over its collection of documents, and the *mappings* between the ontology and each PDV.

Figure 3.8 describes five PDVs, for documents about football games. Nodes annotated with (@) represent attributes. Mappings between PDVs and ontology attributes are presented here grouped by attribute. Each ontology node has at most one corresponding node in each PDV, e.g. *Stadium/address* is mapped to */FootballGamesHistory/event/address* in *Pdv1*, to */StadiumEncyclopedia//Stadium/address* in *Pdv2*, etc.

Concept instances, constraints Intuitively, ***a concept instance*** is identified by a given value for the concept key. ***A concept occurrence*** corresponds to an occurrence of a concept instance (key) in a document. There may be *several concept occurrences of a concept instance* in the source data. A concept occurrence is composed of all the elements corresponding to the concept's attributes "accompanying" the key element (as explained below). A concept occurrence may be *complete* wrt a set A of the concept's attributes, i.e. it contains all the attributes in A, but in many cases it is *partial* (some attributes are missing). *Joining* concept occurrences on the concept key attribute allows obtaining complete values. Note that obtaining a single value for a concept instance (by combining occurrences for the instance key) depends on several modeling choices, such as accepting or not multiple values for an attribute (e.g. several descriptions for the same game), or the way of handling contradictory attribute values (e.g. different capacity values for a stadium in two different stadium occurrences).

Figure 3.9 presents an example of XML document for *Pdv3*. Note that the multiplicity for PDV elements is always "*" (zero-or-many), so not all the elements are always present. Also, other elements, not related to the ontology, may exist, e.g. *referee*. Given the mappings with the ontology, the document contains three occurrences for concept *Game* (for instances of keys "N5", "N8" and "I22") and two for concept *Stadium* ("Gerland" and "Vélodrome"). E.g. the occurrence of *Game* for key "N5" is composed of nodes 3, 4 and 5, corresponding to the attributes of *Game* "accompanying" node 4 that holds the key. All these occurrences are partial wrt the set of all the concept attributes (date missing for *Game*, capacity for *Stadium*), complete values may be obtained by merging them with other occurrences for the same key value.

Ontology relations between concepts and between attributes express ***constraints*** to be satisfied in data sources, in order to enforce semantics. *A "partOf" relation* between attributes constrains the associated XML elements in a source document to have a similar "partOf" relation (descendant relation preservation). Our choice is to *check "partOf" relations between attributes at publishing time* and to only accept mappings that respect this constraint. E.g., in Figure 3.8, if *Game* is mapped to */Games//Game* in *Pdv3*, only mappings for *Game/description* to descendants of */Games//Game* are accepted (here */Games//Game/info*). This constraint allows representing concept occurrences as subtrees, e.g. nodes 3, 4 and 5 in Figure 3.9 form a tree.

By contrast, *"relatedTo" relations* between concepts are less restrictive and are not checked at publishing time. They express a relation between concept instances and may have a different meaning from an application domain to another. For simplicity, we define relations between instances as follows: *for two related concepts c_1 and c_2, an instance i_1 of c_1 is related to an instance i_2 of c_2 iff there exists an occurrence o_1 of i_1 "directly connected" to an occurrence o_2 of i_2 in some document* d *belonging to a PDV* p *having mappings to both c_1 and c_2*. Direct connection is defined as follows: occurrences o_1 and o_2 are directly connected if they are "the closest" possible wrt the PDV, i.e. *they have a common ancestor at the same level as the lowest common ancestor in the PDV*. Other definitions

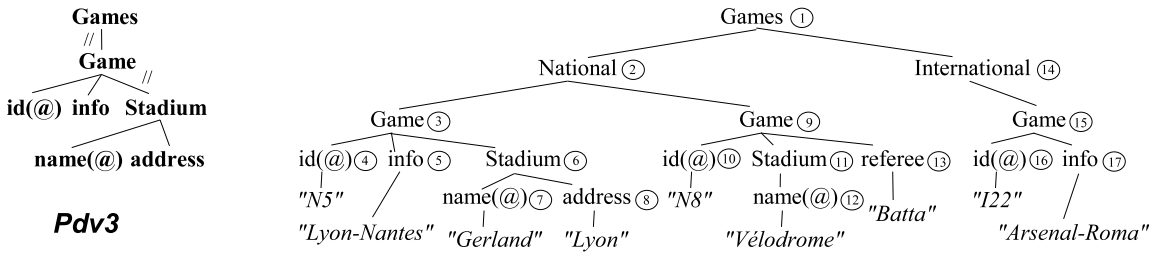

Figure 3.9: An example of XML document for Physical Data View *Pdv3*

are possible, e.g. adding descendant relation constraints between occurrences.

In the example of Figure 3.9, the PDV nodes corresponding to concepts *Game* and *Stadium* have the lowest common ancestor (LCA) in element *Game*. Hence, the *Game* occurrence of key "N5" is directly connected to the *Stadium* occurrence of key "Gerland" (LCA: *Game*), but not to the occurrence of key "Vélodrome" (LCA: *National*). Also, *Game* occurrence "I22" is not directly connected to any *Stadium* (the LCA with both *Stadium* occurrences is *Games*). This example document produces two couples of related *Game* - *Stadium* instances, "N5" with "Gerland" and "N8" with "Vélodrome". Other couples of related instances may be found in other documents.

3.3.3 Query answering and query rewriting

An OpenXView query Q is a conjunctive selection/projection query over the set of ontology attributes:

Q: `Select` a_1, ..., a_n
`Where` $cond_1(a'_1)$ `and` ... `and` $cond_m(a'_m)$

where a_i and a'_j $1 \leq i \leq n$, $1 \leq j \leq m$ are attributes and $cond_j$ are predicates over the attribute value, compatible with the attribute type.

We give here a sketchy semantics for OpenXView queries and for the query rewriting algorithm, based on the following simplifying hypothesis:

- Concept keys are composed of a single attribute.
- Concept attributes are single valued.
- All the attributes of a concept are at the same level, their tree structure may be considered afterwards for building a structured result.

We consider the following *notations*:

- For an ontology $\mathcal{O}$, $concepts(\mathcal{O})$ is the set of concepts (nodes), $rel(\mathcal{O})$ is the set of "relatedTo" relations (edges) in the ontology graph and $attr(\mathcal{O})$ is its set of attributes.
- For a concept c, $attr(c)$ is the set of its attributes and $key(c)$ is its key attribute.
- For a set of concepts C, $rel(C)$ is the set of "relatedTo" relations between the concepts in C.
- For a PDV p: $coll(p)$ is the collection of documents over which p is defined, $tree(p)$ is its tree structure and $nodes(p)$ is the set of tree nodes.
- The mapping between ontology $\mathcal{O}$ and PDV p is a function $mapping_p : attr(\mathcal{O}) \rightarrow (nodes(p) \cup \{nil\})$, where the special value nil means "no node".
- For a PDV p, $attr(p)$ is the set of ontology attributes mapped to p, i.e. $attr(p) = \{a \in attr(\mathcal{O}) | mapping_p(a) \neq nil\}$
- For a query q, $attr(q)$ is the set of attributes that appear in the query, $concepts(q)$ is the set of concepts of these attributes and $rel(q)$ is the set of "relatedTo" relations between these concepts.

Query answering The interpretation of a tuple of PDV nodes $[n_1, ..., n_k]$ belonging to a PDV p is the same as considered in XyView (Section 3.2.2), i.e. the evaluation of a query pattern tree over the data source collection of documents, satisfying the "closeness" constraint of the tuple's elements.

The interpretation of a tuple of attributes $[a_1, ..., a_n]$ in the ontology wrt a PDV p is the interpretation of the corresponding PDV nodes: $I([a_1, ..., a_n], p) = I([mapping_p(a_1), ..., mapping_p(a_n)], p)$

If some $mapping_p(a_i)$ is nil (attribute a_i has no corresponding node in p), the interpretation is the empty set.

The interpretation of a tuple of attributes $[k, a_1, ..., a_n]$ belonging to a same concept c, in a set of PDVs P, where $k = key(c)$, is given by *the union of all the possible joins (on the key) of partial tuples from different PDVs*. Let us note $A = \{a_1, ..., a_n\}$, where $k \notin A$ and let $Partition(A)$ be the set of all the partitions of A. An element of $Partition(A)$ has the form $\{A_1, ..., A_m\}$, where $A_i \subset A$. By convention, if $A_i = \{a_{i_1}, ..., a_{i_l}\}$, then $[k, A_i]$ stands for $[k, a_{i_1}, ..., a_{i_l}]$.

$$I([k, a_1, ..., a_n], P) = \bigcup_{\substack{\{A_1,...A_m\} \in Partition(A) \\ \{p_1,...,p_m\} \in P, A_i \subset attr(p_i)}} I([k, A_1], p_1) \bowtie_k ... \bowtie_k I([k, A_m], p_m) \quad (3.1)$$

The interpretation of a "relatedTo" relation between concepts c_1 and c_2 in a PDV p is given by all the couples of directly connected occurrences of c_1 and c_2 in the documents of PDV p:

$$I(r_{c_1c_2}, p) \stackrel{\Delta}{=} I([key(c_1), key(c_2)], p) \quad (3.2)$$

The same relation in a set of PDVs P is interpreted as follows:

$$I(r_{c_1c_2}, P) = \bigcup_{p \in P} I(r_{c_1c_2}, p) \quad (3.3)$$

The interpretation of a tuple of attributes belonging to several concepts $c_1, ..., c_m$ in a set of PDVs P is given by an independent interpretation for each concept, combined with the interpretation of the "relatedTo" relations. That means that only combinations of concept instances that respect the "relatedTo" relations are accepted.

$$I([k_1, a_{1,1}, ..., a_{1,n_1}, ..., k_m, a_{m,1}, ..., a_{m,n_m}], P) = \times_i (I([k_i, a_{i,1}, ..., a_{i,n_i}], P)) \quad \bowtie_{k_1,...,k_m} \quad (\bowtie_{r \in rel(\{c_1,...,c_m\})} I(r, P)) \quad (3.4)$$

where $k_i = key(c_i)$ and $a_{i,j}$ are attributes of c_i.

Finally, the interpretation of a query q is obtained by considering the tuple composed of $attr(q)$ and of the keys of concepts in $concepts(q)$, and by realizing the selection and projection operations on the interpretation of this tuple.

Query rewriting The query answering expressions above already provide a basic query rewriting algorithm. It is easy to show that the resulting rewriting is equivalent to *a union of unit rewritings*, by transforming formula 3.4 into a union of joins.

A unit rewriting does not contain unions, it is composed of a n-ary join of PDVs that can produce answers to the query. Joins on each concept key are necessary to build tuples with all the concept's query attributes, by merging concept occurrences. Additional joins for the "relatedTo" relations may be necessary to include PDVs that provide an interpretation for these relations.

We say that *a PDV p covers a set of attributes A* if $attr(p) \supseteq A$. A PDV covering A can provide tuples composed of all the attributes in A. *A PDV p covers a "relatedTo" relation* between two concepts c_1 and c_2 if p's structure can satisfy the corresponding constraint. With the "relatedTo" interpretation chosen above, p must simply cover the keys of c_1 and c_2.

A unit rewriting for a query q is described:

- For each query concept $c \in concepts(q)$, by a set of covering PDVs joined on $key(c)$ and a partition of the concept's query attributes ($attr(c) \cap attr(q)$) among these PDVs. The partition expresses what attributes each PDV "provides" in the result. This corresponds to a term of the union in Formula 3.1.
- For each "relatedTo" relation $r \in rel(q)$, by a PDV that covers r.

In other words, a unit rewriting must specify, for each attribute or relation in the query, the PDV that provides that element. In the example of Figure 3.8, a possible unit rewriting for Query Q_e: **Select** *Game/description, Stadium/name, Stadium/capacity* would be:

- For concept *Game*, *Pdv3*, that provides *Game/description*.
- For concept *Stadium*, *Pdv3*, that provides *Stadium/name* and *Pdv4*, that provides *Stadium/capacity*.
- The relation between *Game* and *Stadium* is provided by *Pdv3*.

A unit rewriting can be easily be translated into a simple FLWR XQuery expression, e.g. for the example above, the XQuery rewriting would be (results typing is not considered here):

```
For    $doc3 in collection(Pdv3URI), $doc4 in collection(Pdv4URI),
       $g3 in $doc3/Games//Game, $n3 in $g3//Stadium/@name, $s4 in $doc4//ChampionsLeague/Stadium
Where  $g3//Stadium/@name = $s4/@name
Return <Result> {$g3/info, $n3, $s4/capacity} </Result>
```

The number of unit rewritings produced by Formula 3.4 may be very large. We aim at reducing them to a smaller, "most pertinent" subset, and at exploring, in future work, the *ranking* of unit rewritings, in order to execute them following an order that produces the most pertinent results first.

We introduce several ***unit rewritings equivalence and containment criteria***, based on reasonable symplifying hypothesis:

Equivalence: Two unit rewritings having the same sets of covering PDVs for each concept and the same PDVs for the "relatedTo" relations *are equivalent*, whatever the partition of attributes among covering PDVs is. This is based on two simplifying hypothesis: (i) for a concept instance (key value), attribute values found in any occurrence are the same or equivalent, and (ii) any partition of attributes between the covering PDVs produces the same set of concept instances. In practice this is not always true, especially for the second point (tuples with more attributes match less occurrences), but we consider that the differences are insignificant.

Containment: For two unit rewritings ur_1 and ur_2, such that the set of covering PDVs for each concept in ur_1 is included in the corresponding set of ur_2, and that have the same PDVs for the "relatedTo" relations, we consider that ur_2 *is contained in* ur_1. The idea is that, for a concept covered by a set of PDVs, the set of concept instances produced by merging occurrences is the *intersection* of the sets of instances provided by each PDV. Adding a new PDV will reduce this set of instances to a subset.

The consequences of these criteria on query rewriting can be summarized in two points: (i) in unit rewritings, *only the set of covering PDVs for each concept matters*, not the partition of the attributes among these PDVs, and (ii) for each concept one must find *the minimal set of covering PDVs*.

Query rewriting algorithm The OpenXView query rewriting algorithm is based on the above criteria and on two additional "pertinence" *restrictions* that reduce the number of unit rewritings:

1. The global set of PDVs that covers all the concepts and relations in the query must also be minimal. This limits the dispersion of query attributes among sources that could produce less pertinent results.

2. A "relatedTo" relation between concepts c_1 and c_2 must be provided only by PDVs belonging to the sets of covering PDVs of c_1 and c_2. Two variants are possible: (i) belong to both sets (strict condition), and (ii) belong to one of them (relaxed condition).

The containment criterion and restriction 1 above raise an essential issue in the OpenXView query rewriting algorithm: the computation of ***minimal covers***. *A cover* of a set of attributes A with a set of PDVs P is a subset cp of P that covers A, i.e. $\bigcup_{p \in cp} attr(p) \supseteq A$. *A minimal cover* is a cover where no PDV can be removed without breaking the coverage.

The rewriting algorithm for a query q consists of the following steps:

1. Reduce the size of the problem, by creating *equivalence classes of PDVs*. Two PDVs p_1, p_2 are equivalent if they cover the same subset of query attributes, $attr(p_1) \cap attr(q) = attr(p_2) \cap attr(q)$. Computing minimal covers will use equivalence classes instead of PDVs. This reduces the size of the problem from N PDVs (that may be very large) down to $2^k - 1$ equivalence classes (empty coverage class is discarded), where k is the number of query attributes.
2. Compute the set MC of *global minimal covers* (of $attr(q)$).
3. For each concept c in $concepts(q)$, compute the set MC_c of *concept minimal covers* (of the query concept attributes $attr(q) \cap attr(c)$).
4. Compute the set CMC of *combinations of minimal covers of query concepts* that are *compatible with the global minimal covers*,
   $CMC = \{[mc_1, ..., mc_n] | mc_i \in MC_i, \cup_i mc_i \in MC\}$, where $concepts(q) = \{c_1, ..., c_n\}$.
5. For each combination of minimal covers $cmc \in CMC$, compute the set RC_{cmc} of *combinations of "relatedTo" covers*, by considering that each relation $r_{c_i c_j} \in rel(q)$ can be provided by elements in $mc_i \cap mc_j$ (strict condition) or in $mc_i \cup mc_j$ (relaxed condition).
6. Each couple $(cmc, rc_{cmc}) \in CMC \times RC_{cmc}$ provides a unit rewriting over equivalence classes, that can be easily transformed into a set of unit rewritings over PDVs, by considering all the combinations of PDVs in the equivalence classes.

Consider the case of the example query Q_e above, where $attr(Q_e)$ = {*Game/description, Stadium/name, Stadium/capacity*}. There are two equivalence classes: cl_1={*Pdv1, Pdv2, Pdv4, Pdv5*} that covers *Stadium/name* and *Stadium/capacity*, and cl_2={*Pdv3*} that covers *Game/description* and *Stadium/name*. There is a single cover, which is a minimal cover, composed of both classes, $MC = \{(cl_1, cl_2)\}$. For concept *Game*, cl_2 provides the only cover, which is minimal, $MC_{Game} = \{(cl_2)\}$. For *Stadium*, there are two possible covers (cl_1) and (cl_1, cl_2), but only (cl_1) is minimal, so $MC_{Stadium} = \{(cl_1)\}$. The only combination of minimal covers is $cmc = [mc_{Game} = (cl_2), mc_{Stadium} = (cl_1)]$, which is compatible with MC, because $mc_{Game} \cup mc_{Stadium} = (cl_1, cl_2)$ is a global minimal cover. The *Game-Stadium* relation is covered by *Pdv3* in cl_2, and *Pdv4, Pdv5* in cl_1. There is no way to cover this relation under strict conditions ($mc_{Game} \cup mc_{Stadium} = \emptyset$), but it can be covered under relaxed conditions, by cl_2 (*Pdv3*) or cl_1 (*Pdv4, Pdv5*).

The final result is composed of several unit rewritings:

- *Game* and $r_{Game-Stadium}$ in cl_2 (i.e. *Pdv3*), *Stadium* in cl_1 (i.e. *Pdv1* or *Pdv2* or *Pdv4* or *Pdv5*). This produces four PDV unit rewritings.
- *Game* in cl_2 (i.e. *Pdv3*), *Stadium* and $r_{Game-Stadium}$ in cl_1 (i.e. only *Pdv4, Pdv5* because of the $r_{Game-Stadium}$ relation). This produces two PDV unit rewritings.

Minimal cover algorithm We proposed in [BSSV06b] a very efficient algorithm for minimal cover computation, which is a critical operation at query rewriting. The basic idea is to *precompute* all the possible minimal covers for a set A of k elements, with subsets of A, for k = 1, 2, 3, etc. Figure 3.10 presents an example of minimal cover computation for a query of size k=3.

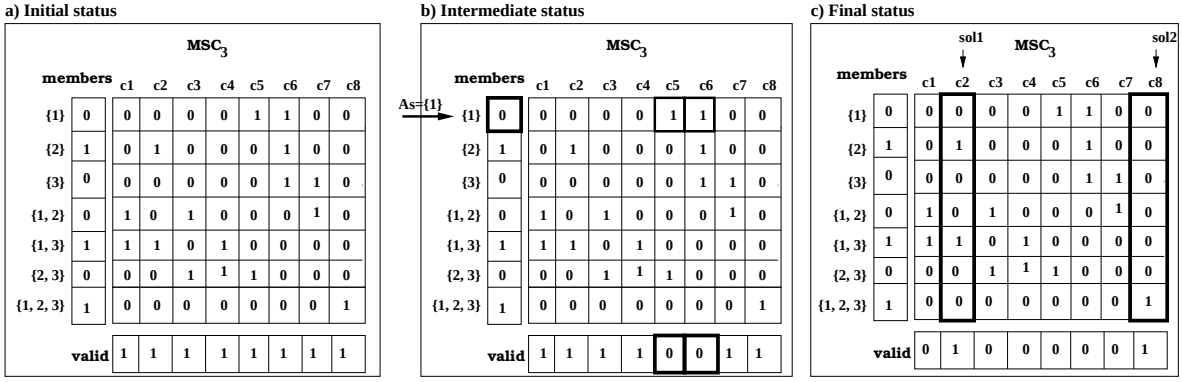

Figure 3.10: Minimal cover computation algorithm

We consider that the set of precomputed minimal covers holds in memory. In practice, we use a bitmap MSC_k to store the precomputed minimal covers for sets of size k. The bitmap has $2^k - 1$ lines (one for each possible subset, excepting for $\emptyset$) and as many columns as minimal covers. The example in Figure 3.10 uses MSC_3, which has $2^3 - 1 = 7$ lines and 8 columns (there are 8 possible minimal covers for sets of size 3, named here c_1, ..., c_8). A "1" value in a cell that corresponds to a subset A_s and a cover c means that A_s belongs to c. Hence, columns represent the composition of minimal covers and lines indicate to what minimal covers belongs each subset.

Given a query q of size k, Step 1 in the query rewriting algorithm above computes the set of PDV equivalence classes. Each equivalence class is identified by the set of query attributes it covers; if the set of k query attributes is represented by $K = \{1, 2, ..., k\}$, an equivalence class is identified by a subset $A_s \subseteq K$.

The minimal cover algorithm uses, besides MSC_k, two bit vectors (Figure 3.10):

- *members*, indicating the set of equivalence classes produced by Step 1. In the example, only 3 classes exist, covering respectively the sets of query attributes $\{2\}$, $\{1,3\}$ and $\{1,2,3\}$.
- *valid*, indicating the valid minimal covers (all are valid in the beginning).

The algorithm considers all the "bad" members (equivalence classes that do not exist) and invalidates the minimal covers containing them ("bad" minimal covers). In the example, for the "bad" member $A_s = \{1\}$, minimal covers c_5 and c_6 are invalidated. At the end, only "good" minimal covers remain valid; in our example there are two solutions $c_2 = [\{2\}, \{1,3\}]$ and $c_8 = [\{1,2,3\}]$.

This algorithm is very efficient compared to traditional branch-and-bound methods that would compute minimal covers starting with the set of equivalence classes. Its drawback is the amount of memory necessary for storing the bitmap, because its size grows very fast with k. For the amount of RAM memory available in today computers, the bound is $k \leq 8$, which is reasonable for the applications we have in mind.

Chapter 4

P2P architectures for data sharing

This chapter presents my activity in the field of distributed architectures for data sharing and content distribution on the web, in the context of my collaboration with the Gemo group at INRIA Futurs. After a brief presentation of the context of this research axis, we present the two parts of this work: the *EDOS* content distribution system, and the improvements to the *ActiveXML* platform.

4.1 Context: P2P data management and content distribution systems

Peer-to-peer (P2P) architectures [KP05, LCP+05] provide effective means for resource sharing (processor time, bandwidth, memory, disk space) in a distributed environment. They provide scalable and robust solutions for the development of large scale web applications. Unlike grid architectures, P2P is characterized by a large autonomy of peers, which may freely enter or leave the system.

More and more popular, P2P architectures are used in various application types: file sharing (Napster, Kazaa), P2P computing (Seti@home), communication (Skype, Jabber), file transfer (BitTorrent), distributed databases (PIER [HHL+03], Edutella [NWQ+02], KadoP [AMP05]), etc. Following the degree of centralization, they may be centralized (e.g. Napster), decentralized, or hybrid (e.g. JXTA). The most usual classification concerns *the overlay network*, which may be *unstructured* (each peer only knows a set of neighbors - Gnutella [Rip01], Piazza [HIM+04], SomeWhere [ACG+04]) or *structured* in various configurations, the most usual being *distributed hash tables* (DHT) (Chord [SMK+01], CAN [RFH+01], Pastry [RD01]) and *hierarchical* (e.g. MediaPeer [DGY05]).

In this context, we are interested by decentralized and hybrid P2P architectures for the management of large amounts of semistructured data. We focus on structured networks, notably DHTs, able to guarantee efficient routing and distributed query processing. The Gemo group at INRIA Futurs developped a P2P platform for data management, based on two main modules.

- *ActiveXML* [AMT06, weba], that represents both (i) *a model for dynamic XML documents*, including service calls in XML documents to represent dynamic changes in their content, and (ii) *a P2P platform* for storing, querying and sharing such documents in a distributed environment. The ActiveXML model provides a declarative framework for the specification of distributed data management applications. Each ActiveXML peer stores local documents, maintains their content by calling embedded service calls when specified, and provides web services for other peers by querying local documents. The ActiveXML peers form a decentralized, unstructured P2P network, connected by the service calls in documents.
- *KadoP* [AMP05, webb], that provides distributed indexing of (Active)XML documents, by using a DHT (Pastry [RD01]). Peers may publish their ActiveXML documents, which are

indexed at the XML element level, then the index entries are distributed in the network by using the underlying DHT. KadoP provides efficient distributed query processing over the published documents.

We are interested in using P2P architectures, in particular the ActiveXML and KadoP modules, for building *distributed systems for content sharing* within a web comunity. Such systems, called *data rings* in [AP07], provide global functionalities to users and are responsible *as a whole* for all the internal management of the content sharing functionalities: indexing, replication, reorganization, etc.

In the particular case of open-source software distribution (e.g. Linux), very large amounts of data (tens of Gigabytes) must be disseminated to a very large community of developers and users (thousands of members). Moreover, content is frequently updated to new versions of the software modules. Current software distribution architectures are mainly centralized or based on sets of mirrors. While centralized systems are not scalable, current mirror-based systems face difficulties to guarantee reasonably uniform performances, because the effort is not uniformly shared between mirrors for realizing the system functionalities. Another major problem of such systems is the difficulty to globally ensure the freshness of content in the system.

Among the current distribution architectures for Linux, the most popular open-source software, only few propose improvements to the classical architecture. The most noticeable are Red Hat Network [Wit], that adds notification channels to maintain freshness, and Conary [Con], that uses distributed versioning repositories to minimize downloads for updates. Currently, the use of P2P architectures for content distribution mainly addresses load balancing and bandwidth sharing (Coral [FFM04], Codeen [Cod]). We want to extend this primary use by adding new P2P functionalities, such as *distributed information system* based on XML metadata indexing and querying, and efficient *file sharing* and *multicast dissemination*, similar to BitTorrent [Coh03].

4.2 The EDOS content distribution system

We present here the content distribution solution proposed in the context of the EDOS European project [EDO04]. EDOS stands for "*Environment for the development and Distribution of Open Source software*" and addresses the production, management and distribution of open source software packages. The EDOS *distribution system* proposes a P2P dissemination architecture including all the participants to the distribution process: publishers, mirrors and end-users. The system has been implemented as an application to the distribution of Mandriva Linux packages.

Compared to existing software distribution systems, EDOS introduces several key improvements:

- a P2P architecture providing resource sharing, load balancing and robustness;
- advanced information system capabilities, based on distributed indexing and querying of XML content metadata;
- efficient dissemination based on clustering of packages and multicast techniques;
- support for maintaining freshness on updates, by using subscription / notification techniques (pub/sub).

4.2.1 System overview

The goal of the EDOS distribution system is to efficiently disseminate open-source software (referred as *data* or *content*) through the Internet. Published by a main server, data is disseminated in the network to other computers (mirrors, end-users), that get copies of the published content. EDOS is

articulated around a distributed, P2P information system that stores and indexes *content metadata*. This metadata-based information system allows querying and locating data in the EDOS network.

Data model There are *three kinds* of data units in the EDOS system:

- **Package**: main data unit type, represented by an RPM file;
- **Utility**: individual file used in the installation process;
- **Collection**: it groups packages, utilities or subcollections, to form a hierarchical organization of data.

A release is a set of data units that form a complete software solution - it corresponds to a full Linux distribution. Its content is described by a collection.

Content dissemination is initiated by *publishing* data units in the system. Publishing consists in generating metadata for each data unit and indexing it in the distributed system. Periodically, the main server publishes a new release. Updates to the current release are realized by publishing new versions of packages or utilities. When the time comes, a ”publisher” decides to transform the current status of the current release into a new release.

Metadata management is a key issue in the distribution process. We built a global, distributed information system about data to be disseminated in the network. This system is fed with content metadata, which may include not only content properties, but also information on production, testing, statistics, etc. Distributed management of metadata is justified by its size and by the high rate of queries and updates it supports. We classify metadata properties in three main categories:

- *identifiers* - in our case, the name and the version number uniquely identify a data unit.
- *static properties*, that do not change in time for a content unit, e.g. size, category, checksum, license, etc.
- *changing properties*, i.e. properties that may vary in time: *location* of replicas in the network, collection *composition*, distribution statistics, etc.

The XML structure chosen for EDOS metadata is a compromise between efficiency requirements for both *query processing* (that requires large XML files, containing all the elements addressed in a query) and *metadata updates* (that need small files). We choose to create separate XML files for each package (package properties) and for each release (release composition). Changing properties use service ActiveXML service calls to compute the current value. For efficiency reasons, replica location management uses a separate mechanism.

Actors and roles Peers of the EDOS P2P distribution system may be classified in *three categories*:

1. **Publishers**: They publish new content in the network, manage flash-crowd dissemination and the pub-sub system. In EDOS there is typically a single Publisher, but the architecture permits several ones.
2. **End-users** (Clients): They download content from other peers, query the system, subscribe to data changes. They also participate to the network by storing and providing their local data for downloads. To query the metadata, they need an entry-point into the indexing network of the Mirrors.
3. **Mirrors**: They provide all the functionalities of the End-users. Besides that, they participate in the indexing network. Typically, these are *trusted*[1] and reliable servers providing some guaranteed quality of service.

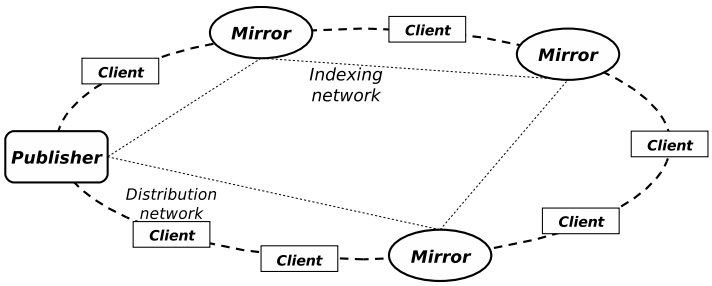

Figure 4.1: Actors in EDOS P2P content distribution

Figure 4.1 presents the actors in the P2P distribution network. Actors are connected in *two distinct networks*:

- *The distribution network*, composed of all the peers - they store, download and share EDOS data, i.e. software packages, utilities and collections.
- *The indexing network*, composed of trusted peers (Publisher and Mirrors) - they store the index on content metadata. For security reasons, Clients are not allowed to participate in metadata and index sharing, but can provide content, whose validity may be automatically verified by using the checksum metadata property.

Usage scenarios **Flash-crowd situations** generally happen when new, popular and large size content is published (here a new release), and many peers (Clients or Mirrors) want to get this content as soon as possible. Flash-crowd distribution uses efficient dissemination methods, based on clustering of data units and multicast. Each peer asking for some portion of the new release may already have some of the packages - therefore it computes a *wish list* containing only the missing data units. Based on the wish lists gathered from peers, the Publisher computes *clusters* of data units to be disseminated. Instead of downloading individual data units, peers download clusters (where a cluster is a set of packages that are requested by a common set of peers), in a global *multicast* process.

The flash-crowd dissemination of a new published release is described by the following steps:

1. Peers interested in the new release subscribe to a special channel for new release publication.
2. The Publisher publishes the new release and notifies all peers that subscribed to that release. Among the metadata published for the new release, its *composition* (identifiers of data units) is necessary for each peer to determine the set of data units to download.
3. Notified peers decide if they are finally interested in the new release or not. The peers compute the delta between the new release and the content they already have. This delta, called *wish list*, composed of the identifiers of data units to be downloaded, is sent to the Publisher.
4. The Publisher waits for wish lists during a predefined window of time. Then it computes clusters of data units, based on the set of collected wish lists.
5. The Publisher published the computed clusters of data units and starts ”torrents” for disseminating them.
6. Each peer gets in parallel (via multicast techniques) a set of clusters that covers its wish list.

Off-peak distribution and query corresponds to periods between flash-crowd situations. During these periods, Publishers may publish updates to the current release and the other peers may query

[1]The correctness of a file is guaranteed by its signature and checked at download. The correctness of metadata is guaranteed by the fact that the mirrors are trusted peers.

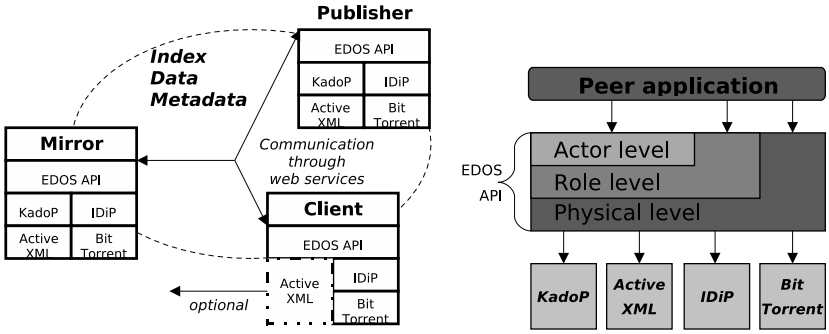

Figure 4.2: EDOS software modules & API structure

the system, download query results, subscribe to distribution channels, receive notifications on such channels and download software updates.

4.2.2 Software architecture

EDOS distribution functionalities are implemented as a *Java API*, based on a set of external software modules (Figure 4.2):

- **ActiveXML** [ABC+04] provides an extended XML format for EDOS metadata and storage for metadata documents published in KadoP. Web service calls embedded in ActiveXML documents may be used to represent intensionally changing information (e.g. statistics on the distribution process) or package dependencies.
- **KadoP** [AMP05] allows publishing, indexing and advanced querying of EDOS metadata. KadoP is the core of the EDOS information system.
- **IDiP** [MZ07] implements functionalities for the flash-crowd usage scenario: content clustering and multicast dissemination using BitTorrent.
- **BitTorrent** [Coh03] is an efficient file sharing and downloading system. We use a slightly modified version of *Azureus*, a Java implementation of BitTorrent, for multicasting and downloading from multiple replicas.

The structure of the EDOS distribution API is presented in Figure 4.2. The API is organized into three levels:

1. **Physical level**: it provides EDOS peer basic functionalities. It is composed of several modules: a *content manager*, an *index manager*, a *channel manager*, a *dissemination manager*, etc. Programming distribution applications at the physical level requires more effort, but offers the best flexibility.
2. **Role level**: it is built on top of the physical level, provides a default implementation for each role in the distribution network, i.e. publishing, downloading, replicating, querying, and subscribing.
3. **Actor level**: it provides a default implementation for each actor kind (Publisher, Mirror or Client), by combining several roles.

Implementation and tests The first version of the EDOS distribution system has been implemented as a set of Java/JSP web applications on top of the EDOS API. Each peer in the EDOS network runs a Java web application. Peer applications use a Tomcat web server for deployment, with Axis for web

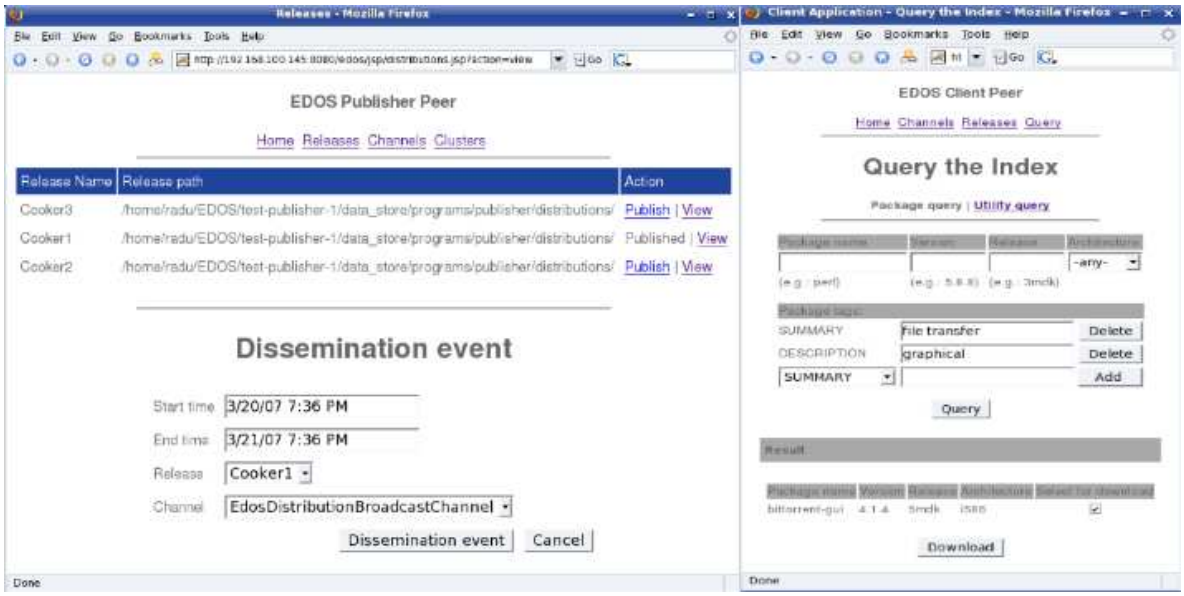

Figure 4.3: Publisher and Mirror/Client web applications

services. The functionalities of each EDOS peer are accessible through a JSP user interface, running in a web browser.

The Publisher web application (left side of Figure 4.3) allows publishing new content, managing subscription channels and driving flash-crowd dissemination. Mirrors and Clients have similar user interfaces (right side of Figure 4.3), allowing queries, downloading, subscriptions to channels and notification handling.

Tests with the first prototype demonstrated the relevance of P2P-based solutions for large-scale content distribution, the ability of managing very large amounts of metadata with KadoP and the improvements brought by IDiP for flash-crowd dissemination. Tests have been realized on several hundreds of peers on the Grid'5000 [gri03] network to evaluate the scalability of EDOS publishing, querying and downloading. Other tests are planned to be realized on the Internet, by using the Mandriva network infrastructure. More details are presented in [EDO].

The last developments in EDOS brought several improvements:

- In massive publication of metadata, which is critical in EDOS. Publishing a new release is time consuming because it implies the publication of a large amount of metadata. We found that the best method for accelerating publication is to split content between several Publishers that work in parallel.

- In security, by introducing peer authentication, and by providing support for firewall/NAT traversal.

- In the user interface, by providing new front-end, based on Java Eclipse RCP standalone applications.

The EDOS distribution framework is also used as a starting architecture in the *WebContent* RNTL project, where the goal is to provide a distributed solution for storage, querying and transformation of web documents.

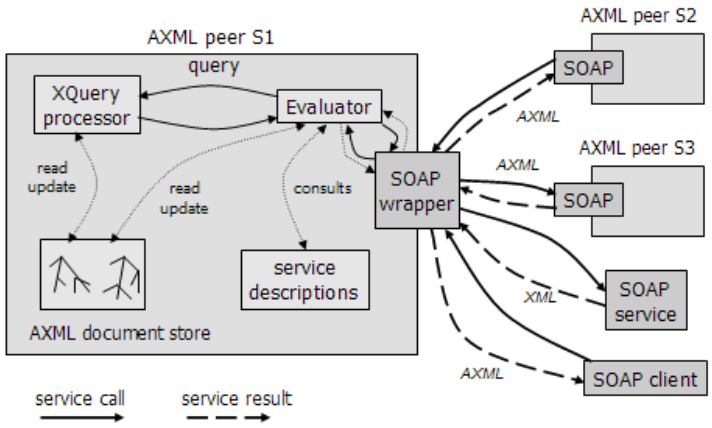

Figure 4.4: ActiveXML peer architecture

4.3 Improvements to the ActiveXML platform

The ActiveXML platform [weba] was realized as an open-source software and is used by several research teams across the world. It is based on the peer architecture presented in Figure 4.4. Each peer is composed of the following main modules:

- *The document store*, which provides persistent storage for local ActiveXML documents.
- *The evaluator*, which triggers service calls embedded in ActiveXML documents and updates accordingly the documents with the service call results.
- *The query processor*, mainly used for service requests, evaluates queries or executes update queries over the document store.

The main role of an ActiveXML peer is the *management of the dynamic content of its local ActiveXML documents*, by scheduling and triggering the execution of the embedded service calls. This function is realized by the Evaluator module.

The following example presents a very simple ActiveXML document containing service calls.

```
<temperature xmlns:axml="http://www-rocq.inria.fr/verso/AXML">
    <axml:sc service="weather.com/temperatures" mode="replace" frequence="every day">
      <axml:param name="place">
        <city> Paris </city>
      </axml:param>
    </axml:sc>
</temperature>
```

It contains an element *temperature*, whose value is not fixed, but provided by a call to a web service (*weather.com/temperatures*), receiving a parameter (*place*) and returning the current temperature at the given place. Parameters may be ActiveXML documents, containing other service calls or queries to local documents (evaluated by the query processor). The ActiveXML peer storing this document will execute the embedded service call *every day* (*frequence* attribute) and insert the result in the document, by replacing the previous result, if exists (*mode* attribute). The updated document has the following structure:

```
<temperature xmlns:axml="http://www-rocq.inria.fr/verso/AXML">
    <axml:sc service="weather.com/temperatures" mode="replace" frequence="every day">
      <axml:param name="place">
        <city> Paris </city>
```

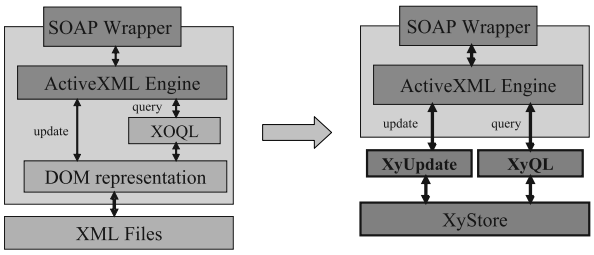

Figure 4.5: Coupling of ActiveXML with Xyleme

```
      </axml:param>
    </axml:sc>
    <value date="2007-06-29"> 15 </value>
</temperature>
```

A second role of an ActiveXML peer is to be a *web service provider*. The services provided by the peer are generally implemented as *queries over the local documents*, evaluated by the query processor. The peer maintains a WSDL-like description of the services it provides.

Peers communicate with each other only by the mean of web service invocations, through their SOAP wrapper modules. They can exchange XML data with any web service client/provider, and ActiveXML data with ActiveXML peers.

The improvements that we brought to the ActiveXML platform addresses **the third role** of an ActiveXML peer, concerning *the persistence of local documents*. The initial ActiveXML implementation used a simple persistence mechanism, based on XML files stored in the file system and loaded in memory, where updates to documents are immediately written on disk. Query processing and document updates are realized on the DOM in-memory representation.

This basic storage architecture is not scalable and limits the use of ActiveXML peers to small volumes of data. The solution is *the use of an XML database*, in order to use the database capabilities for storage, querying and updating for providing scalable management of data on the peer.

We designed and implemented ***two variants of coupling ActiveXML with an XML database***. The main difficulty is to re-design the ActiveXML peer without broking the other functionalities. *The first variant*, realized by Eric Darondeau during his CNAM engineer thesis, provides a coupling of ActiveXML with Xyleme. Figure 4.5 presents the ActiveXML architecture before and after the coupling with Xyleme. The in-memory storage, querying and updating of data is replaced with the database functions: the XyStore module for storage, XyQL for querying, XyUpdate for updates. The other peer functionalities are preserved, the connection with the database documents being realized through document and service call identifiers, assigned by the system and exploited through the query and update languages.

The second variant, realized by Ming Hoang To, during his Politechnique engineer internship, is a generalization of the coupling with Xyleme. This solution provides the *coupling with a generic XML repository*, providing querying and updating functionalities, based on the XQuery and XQuery Update languages. Its validity was verified by instantiating the repository on top of the *eXist* XML database.

Chapter 5

Conclusions and future work

5.1 Conclusions

My research activity since the end of my PhD thesis, in January 1997, has been milestoned by ***a thematic reconversion*** in 1999, from the domain of *multimedia user interfaces* to the one of *data management on the web*.

Two main topics have been addressed during this period: *data integration on the web* and *distributed architectures for data management*. The former encompasses two issues: *web-scale integration of XML documents* and *application views for simplifying the access to heterogeneous XML data*. The latter explores *peer-to-peer architectures for content sharing and distribution on the web*. A noteworthy feature of my research activity is the *design and implementation of several software modules*, included in commercial products (Xyleme), in open-source software for Linux distribution (EDOS for Mandriva Linux) or for the research community (ActiveXML platform), or in research prototypes.

Web scale data integration has been studied in the context of the Xyleme system. We proposed a solution for XML data integration at very large scale, based on a global XML schema (abstract DTD), GLAV path-to-path mappings between global and local XML schemas, a method for distributing the view among Xyleme machines, a method for including view querying into the general query processing mechanism and a scalable algorithm for query rewriting. This system has been implemented as part of the Xyleme software, together with tools for automatic generation of mappings (MapGen) and for extracting XML from unstructured text, in order to feed the XML repository.

Application views for simplifying the access to heterogeneous XML data have been explored in two different cases: XML repositories supporting a limited set of changes in data sources (the XyView model) and open systems, dealing with autonomy of sources that may enter/leave the system or change their content at any moment (the OpenXView model). The XyView model has been implemented as an application module in Xyleme, together with tools for view editing and for generating web-form applications on top of views. In OpenXView, we explored for the time being query rewriting and query answering issues.

Peer-to-peer architectures for content sharing and distribution on the web have been studied in the context of the ActiveXML framework and tools for distributed management of data, where several improvements of the ActiveXML platform have been realized. We designed EDOS, a P2P system for the distribution of open-source software modules to a large community of users. EDOS homogeneously shares the effort between all the network nodes, to provide scalable, global functionalities such as metadata management and querying, content dissemination and change notification.

5.2 Future work

The management of data distributed across the web continues to be a very important issue and a major challenge in today systems. My future work will address the following points, detailed below:

- **A continuation** of the ongoing work on *data integration in open systems* (the OpenXView model) and on *data ring systems* (in the WebContent project, inspired by the EDOS content management architecture).

- **New directions** in web data management and distributed web architectures: *RSS feeds management* (ROSES project), *management of data with evolving schema in diachronic (archive) databases* (DIACHRON project), and *distributed management of multimedia data for content-based querying* in heterogeneous networks (DISCO project).

Concerning the **collaboration with other research teams**, the creation in 2007 of the *Wisdom* PPF ("Plan Pluri-Formation"), which gathers the database research teams from the CEDRIC (CNAM), LIP6 (Paris 6) and LAMSADE (Paris 9) laboratories, enforces the collaboration with these teams. The ROSES and DISCO projects mentioned above have been initiated and submitted by *Wisdom* to the French ANR MDCO 2007 call. I will also continue my collaboration with the *Gemo* group at INRIA Futurs, in the context of my research on data ring system and of the RNTL WebContent project (2006-2009).

At the European level, I am starting a collaboration with Vassilis Christophides from FORTH Heraklion (Grece) in the context of the DIACHRON project on diachronic databases, that will be submitted to the FP7 call of European projects. A collaboration with Michalis Vazirgiannis from the Athens University, on the topic of distributed query processing in P2P systems, has been initiated through the submission in 2007 of a proposal of Marie Curie ITN (Initial Training Network).

Data integration in open systems: I intend to continue the work on data integration in open systems with the OpenXView model, by addressing other issues, such as *source publishing* (e.g. automatic generation of mappings and extraction of PDVs from a source schema, inference of ontology updates), *ranking of query rewritings*, *query optimization* for OpenXView execution plans on top of various system architectures (repository, P2P with various overlay networks).

Data ring systems: The work initiated in collaboration with the Gemo group at INRIA Futurs, around the ActiveXML framework and in the context of the EDOS project, will continue with the elaboration of new data ring systems for different types of applications. Data rings are P2P systems for community content sharing, that are responsible for the internal organization of data and processing among the peers (indexing, replication, etc), and that transparently offer global system functionalities to end-users. Two goals can be outlined in this research direction:

- The design of a general declarative architecture for data ring system specification based on ActiveXML, and of an algebra to implement these declarative specifications.
- The design and implementation of data ring systems based on this general architecture. This is the goal of the RNTL *WebContent* project (2006-2009), in which I participate to the design and implementation of a solution for distributed storage, querying and transformation of web documents.

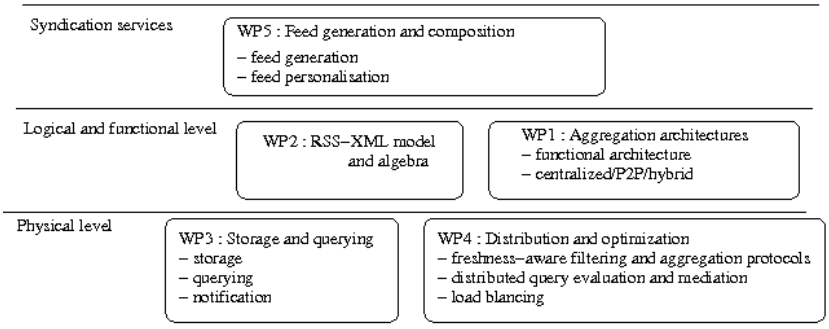

Figure 5.1: Tasks in the ROSES project

RSS feeds management: I am interested in considering new dimensions in XML data integration, such as time and change events. RSS feeds, generalized on the today web, provide a very large number of XML data sources with changing content on the web. The RSS model adds time stamped change events to XML data, in a data model that is a mix of changing documents and data streams. RSS comes also with a web syndication model for using RSS feeds, that reduces the time lag between publishing and consuming data on the web and that may constitute a general framework for data sharing on the web.

RSS feeds management is the object of the *ROSES* project (2008-2010), submitted to the French ANR MDCO 2007 call. I am co-leader (with Bernd Amann from LIP6) in this project that gathers database teams from Wisdom (CEDRIC and LIP6), PRISM Versailles and LSIS Toulon, together with the *2or3things* company that created the Blasfeed RSS management software. The goal of ROSES is to apply modern web database techniques to the management of RSS feeds, encompassing localization, integration, querying and composition of RSS data. More precisely, the project will study (see Figure 5.1) the design of basic services (store, refresh, filter, notify) and optimization techniques for feed management in centralized and distributed environments, the definition of a general RSS model and algebra, of different system architectures, and of high-level services, methods and tools for feed generation and composition.

My main points of interest in this project are the definition of a general RSS-XML model at several levels (physical and logical algebra, query language, view language), adding a temporal end event-oriented dimension to XML models, and the declarative specification of high-level RSS-XML services as views over RSS feeds and XML data. A PhD thesis co-directed by Bernd Amann and myself will be funded by this project.

Diachronic databases: The temporal dimension of data management is also present in diachronic databases, which aim at perpetual preservation of their content. A diachronic database must be able to manage the evolution of both data content and structure over the time. It must allow queries about any past state, or ”longitudinal” queries about the evolution of a given element in time. Diachronic databases must be distributed to ensure robustness; they must be human and machine readable and independent of software packages.

The design of diachronic databases is the goal of the *DIACHRON* STREP project proposal, where the Vertigo group participates together with the University of Edinburgh, FORTH Heraklion, the National Technical University of Athens, CNR Italy and others. In this context, I am interested in exploring the management and querying of data whose structure changes in time. Changes in the data structure could be represented with mappings, in a similar fashion to data integration models. The

user should be able to query the current data structure, and to get answers from past data, organized following a different schema. Also, queries about structure changes themselves should be allowed.

Distributed management of multimedia content: Database management of multimedia content (image, text, sound, video) is generally realized in centralized environments. Content-based queries in such databases use multidimensional index structures to accelerate query processing. When the multimedia content comes from several sources, P2P architectures are a natural choice for building scalable, robust and efficient multimedia management systems. Recently, methods for distributing the query index over several peers in a network have been proposed.

I am interested in defining methods for distributed content-based indexing and querying of multimedia files in *heterogeneous networks* in terms of indexing schema and access rights. This corresponds to real life situations, where content providers use their own local indexes and need a strict control over access to data, but want their content to be retrieved by user queries over a set of such sources. The idea would be to define a content summary structure, exported by each peer in a common format and indexed in a distributed way in the network. Query processing uses in a first step this distributed summary to detect relevant sources, then uses the result of the first step and local indexes to find the final results.

This research problem is the subject of the PhD thesis of François Boisson, started in January 2007, directed by Michel Crucianu and myself. It represents also one of the main issues in the *DISCO* project (2008-2010), submitted to the French ANR MDCO 2007 call, and that gathers research teams from Wisdom (CEDRIC and LAMSADE), INRIA Lille and IRCAM, in collaboration with Europearchive and the French Réunion des Musées Nationaux.

Bibliography

[ABC+04] Serge Abiteboul, Omar Benjelloun, Bogdan Cautis, Ioana Manolescu, Tova Milo, and Nicoleta Preda. Lazy Query Evaluation for Active XML. In *SIGMOD*, 2004.

[ABCC03] Enrico Augurusa, Daniele Braga, Alessandro Campi, and Stefano Ceri. Design and Implementation of a Graphical Interface to XQuery. *Proceedings ACM Symposium on Applied Computing*, pages 1163 – 1167, 2003.

[ABFS02] Bernd Amann, Catriel Beeri, Irini Fundulaki, and Michel Scholl. Querying xml sources using an ontology-based mediator. *CoopIS/DOA/ODBASE*, pages 429–448, 2002.

[ACFR02] Serge Abiteboul, Sophie Cluet, Guy Ferran, and Marie-Christine Rousset. The xyleme project. *Computer Networks*, 39(3):225–238, 2002.

[ACG+04] P. Adjiman, P. Chatalic, F. Goasdoué, M.C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting. In *ECAI*, pages 945–946, 2004.

[ACM+02] V. Aguilera, S. Cluet, T. Milo, P. Veltri, and D. Vodislav. Views in a large scale xml repository. *VLDB Journal*, 11(3):238–255, 2002.

[ACV+00] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying xml documents in xyleme. *ACM SIGIR workshop*, 2000.

[ACV+01] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying a web scale xml repository. In *SEBD*, pages 105–118, 2001.

[ADP+07a] S. Abiteboul, I. Dar, R. Pop, G. Vasile, and D. Vodislav. Edos distribution system: a p2p architecture for open-source content dissemination. In *IFIP Open Source Systems (OSS)*, 2007.

[ADP+07b] S. Abiteboul, I. Dar, R. Pop, G. Vasile, and D. Vodislav. Snapshot on the EDOS distribution system. *FOSDEM Workshop, Free & Open source Software Developers' European Meeting*, 2007.

[ADP+07c] S. Abiteboul, I. Dar, R. Pop, G. Vasile, D. Vodislav, and N. Preda. Large scale p2p distribution of open source software. In *VLDB*, 2007.

[AMP05] Serge Abiteboul, Ioana Manolescu, and Nicoleta Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. In *ICDE*, 2005.

[AMT06] Serge Abiteboul, Ioana Manolescu, and Emanuel Taropa. A framework for distributed xml data management. In *EDBT*, pages 1049–1058, 2006.

[AP07] Serge Abiteboul and Neoklis Polyzotis. The Data Ring: Community Content Sharing. In *Conference on Innovative Data Systems Research (CIDR)*, pages 154–163, 2007.

[APVV07] S. Abiteboul, R. Pop, G. Vasile, and D. Vodislav. Scalability evaluation of a p2p content distribution system. In *Bases de Données Avancées (BDA)*, 2007.

[AVFC98] B. Amann, D. Vodislav, J. Fernandes, and G. Coste. Browsing SGML documents with maps : The French 'Inventaire' experience. *International Conference on Database and Expert Systems Applications (DEXA)*, 1998.

[AYCLS01] Sihem Amer-Yahia, SungRan Cho, Laks V.S. Lakshmanan, and Divesh Srivastava. Minimization of tree pattern queries. *ACM SIGMOD*, 2001.

[BEA] BEA Liquid Data. *http://www.bea.com*.

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *VLDB*, pages 119–128, 2001.

[BSSV06a] Francois Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav. Query rewriting for open XML data integration systems. *IADIS WWW/Internet*, pages 133–141, 2006.

[BSSV06b] Francois Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav. Scalability of source identification in data integration systems. *IEEE SITIS*, 2006.

[BSSV07] Francois Boisson, Michel Scholl, Imen Sebei, and Dan Vodislav. Source identification and query rewriting in open XML data integration systems. *IADIS International Journal of WWW/Internet*, 2007.

[Cat97] R. G. Cattell. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.

[CCS00] Vassilis Christophides, Sophie Cluet, and Jérôme Siméon. On wrapping query languages and efficient xml integration. In *SIGMOD Conference*, pages 141–152, 2000.

[CCT+05] M. Cannataro, S. Cluet, G. Tradigo, P. Veltri, and D. Vodislav. Using views to query xml documents. In *Encyclopedia of Database Technologies and Applications*, pages 729–735. IDEA Group Reference, 2005.

[CHZ05] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR*, pages 44–55, 2005.

[CKS+00] Michael J. Carey, Jerry Kiernan, Jayavel Shanmugasundaram, Eugene J. Shekita, and Subbu N. Subramanian. Xperanto: Middleware for publishing object-relational data as xml documents. In *VLDB*, pages 646–648, 2000.

[CMKS03] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSEarch: A Semantic Search Engine for XML. *Proceedings VLDB*, 2003.

[Cod] Codeen. http://codeen.cs.princeton.edu.

[Coh03] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of P2P Systems*, 2003.

[Con] Conary Software Provisioning System. http://wiki.rpath.com/wiki/Conary.

[CTV00] P. Cubaud, A. Topol, and D. Vodislav. Les limites de VRML pour les comportements interactifs: étude de cas. *ERGO-IHM*, 2000.

[CVV01] S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale xml repository. In *VLDB*, pages 271–280, 2001.

[DGY05] Florin Dragan, Georges Gardarin, and Laurent Yeh. Mediapeer: A safe, scalable p2p architecture for xml query processing. In *DEXA Workshops*, pages 368–373, 2005.

[DR02] Hong Hai Do and Erhard Rahm. Coma - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[DRR+03] C. Delobel, C. Reynaud, M.-C. Rousset, J.-P. Sirot, and D. Vodislav. Semantic integration in xyleme: a uniform tree-based approach. *Data & Knowledge Engineering Journal*, 44(3):267–298, 2003.

[DT05] Alin Deutsch and Val Tannen. Xml queries and constraints, containment and reformulation. *Theor. Comput. Sci.*, 336(1):57–87, 2005.

[EDO] EDOS Work Package 4 deliverables. http://www.edos-project.org/xwiki/bin/view/Main/Deliverables.

[EDO04] EDOS project: Environment for the development and Distribution of Open Source software, 2004. http://www.edos-project.org.

[Erw03] M. Erwig. Xing: A Visual XML Query Language. *Journal of Visual Languages and Computing*, pages 5–45, Februray 2003.

[Fel98] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[FFM04] Michael Freedman, Eric Freudenthal, and David Mazieres. Democratizing Content Publication with Coral. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[FKS+02] Mary F. Fernandez, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, and Wang Chiew Tan. Silkroute: A framework for publishing relational data in xml. *ACM Trans. Database Syst.*, 27(4):438–493, 2002.

[FLM99] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, pages 67–73, 1999.

[GLR00] François Goasdoué, Véronique Lattès, and Marie-Christine Rousset. The use of carin language and algorithms for information integration: The picsel system. *Int. J. Cooperative Inf. Syst.*, 9(4):383–401, 2000.

[GMPQ+97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The tsimmis approach to mediation: Data models and languages. *J. Intell. Inf. Syst.*, 8(2):117–132, 1997.

[GMT02] Georges Gardarin, Antoine Mensch, and Anthony Tomasic. An introduction to the e-xml data integration suite. In *EDBT*, pages 297–306, 2002.

[gri03] GRID'5000 plate-forme de recherche exprimentale en informatique, 2003. http://www-sop.inria.fr/aci/grid/public/Library/rapport-grid5000-V3.pdf.

[GSSB03] Lin Guo, Feng Shao, Jayavel Shanmugasundaram, and Chavdar Botev. XRANK : Ranked keyword search over XML documents. *Proceedings SIGMOD*, 2003.

[GW97] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *Proceedings of the 23rd VLDB Conference*, pages 436–445, 1997.

[Hal01] Alon Halevy. Answering queries using views: A survey. *The VLDB Journal*, pages 270–294, 2001.

[HHL+03] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with pier. In *VLDB*, pages 321–332, 2003.

[HIM+04] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

[HIMT03] A. Halevy, Z. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. *Proceedings WWW*, 2003.

[HJL+04] Alan Halverson, Vanja Josifovski, Guy Lohman, Hamid Pirahesh, and Mathias Mörschel. ROX: Relational over XML. *Proceedings VLDB*, 2004.

[HMH01] Mauricio A. Hernández, Renée J. Miller, and Laura M. Haas. Clio: A semi-automatic tool for schema mapping. In *SIGMOD Conference*, page 607, 2001.

[HPB03] Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword proximity search on XML graphs. *Proceedings ICDE*, 2003.

[IHW02] Zachary G. Ives, A. Y. Halevy, and D. S. Weld. An XML query engine for network-bound data. *The VLDB Journal*, 2:380–402, December 2002.

[KM00] Carl-Christian Kanne and Guido Moerkotte. Efficient storage of xml data. In *ICDE*, page 198, 2000.

[KP05] Georgia Koloniari and Evaggelia Pitoura. Peer-to-peer management of xml data: issues and research challenges. *ACM SIGMOD Record*, 2005.

[LCP+05] Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[Len02] Maurizio Lenzerini. Data integration: a theoretical perspective. *PODS*, 2002.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.

[LYJ04] Yunyao Li, Cong Yu, and H.V. Jagadish. Schema free XQuery. *VLDB*, 2004.

[MAA+00] Laurent Mignet, Serge Abiteboul, Sébastien Ailleret, Bernd Amann, Amélie Marian, and Mihai Preda. Acquiring xml pages for a webhouse. In *BDA*, 2000.

[MACM01] Amélie Marian, Serge Abiteboul, Gregory Cobena, and Laurent Mignet. Change-centric management of versions in an xml warehouse. In *VLDB*, pages 581–590, 2001.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. *Proceedings VLDB*, pages 49–58, 2001.

[MCD+07] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[MFK02] Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering xml queries on heterogeneous data sources. *VLDB*, 2002.

[MP03] Peter McBrien and Alexandra Poulovassilis. Data integration by bi-directional schema transformation rules. In *ICDE*, pages 227–238, 2003.

[MZ07] Tova Milo and Tal Zur. Boosting Topic-Based Publish-Subscribe Systems with Dynamic Clustering. In *SIGMOD*, 2007.

[NWQ+02] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: a p2p networking infrastructure based on rdf. In *WWW*, pages 604–615, 2002.

[PBO+03] Yannis Papakonstantinou, Vinayak Borkar, Maxim Orgiyan, Kostas Stathatos, Lucian Suta, Vasilis Vassalos, and Pavel Velikhov. XML queries and algebra in the Enosys integration platform. *Data & Knowledge Engineering.*, 44(3):299–322, 2003.

[PH01] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *The VLDB Journal*, pages 182–198, 2001.

[PPV02] Yannis Papakonstantinou, Michalis Petropoulos, and Vasilis Vassalos. QURSED: Querying and Reporting Semistructured Data. *Proc. SIGMOD*, 2002.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[Rip01] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing*, pages 99–100, 2001.

[RSV01] C. Reynaud, J.-P. Sirot, and D. Vodislav. Semantic integration of xml heterogeneous data sources. In *IDEAS*, pages 199–208, 2001.

[SA99] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-sources using w4f. In *VLDB*, pages 738–741, 1999.

[sem04] ACI SemWeb: Interrogation du web sémantique avec XQuery, 2004. http://bat710.univ-lyon1.fr/ semweb.

[SGS05] K.-U. Sattler, I. Geist, and E. Schallehn. Concept-based querying in mediator systems. *VLDB Journal*, 14(1):97–111, 2005.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Conference*, pages 149–160, 2001.

[Ull83] J. D. Ullman. Universal Relation Interfaces for Database Systems. *Proceedings IFIP*, 1983.

[VCCS05] Dan Vodislav, Sophie Cluet, Grégory Corona, and Imen Sebei. XyView: Universal relations revisited. *Bases de Données Avancées (BDA)*, pages 357–372, 2005.

[VCCS06] Dan Vodislav, Sophie Cluet, Grégory Corona, and Imen Sebei. Views for simplifying access to heterogeneous XML data. *OTM Confederated conferences/CoopIS*, pages 72–90, 2006.

[Vel02] P. Veltri. *A view mechanism for a large scale XML repository*. PhD thesis, University Paris 11, Orsay, 2002.

[Vod97a] D. Vodislav. *Programmation visuelle pour l'animation dans les interfaces homme-machine*. PhD thesis, CNAM Paris, 1997.

[Vod97b] D. Vodislav. A visual programming model for user interface animation. *IEEE Symposium on Visual Languages (VL)*, pages 344–351, 1997.

[vrm97] VRML97 - the Virtual Reality Modeling Language, International Standard ISO/IEC 14772, 1997. *http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/*.

[VV00] D. Vodislav and M. Vazirgiannis. Structured interactive animation for multimedia documents. *IEEE Symposium on Visual Languages (VL)*, 2000.

[weba] ActiveXML web page. http://activexml.net.

[webb] KadoP web page. http://gemo.futurs.inria.fr/projects/KadoP.

[web06] WebContent: The Semantic Web Framework, 2006. http://www.webcontent-project.org/.

[Wid95] J. Widom. Research problems in data warehousing. In *CIKM '95: Proceedings of the fourth international conference on Information and knowledge management*, pages 25–30, 1995.

[Wie95] G. Wiederhold. Mediation in information systems. *ACM Comput. Surv.*, 27(2):265–267, 1995.

[Wit] Sean Witty. Best Practices for Deploying and Managing Linux with RedHat Network.

[YP04] Cong Yu and Lucian Popa. Constraint-based xml query rewriting for data integration. *SIGMOD*, pages 371–382, 2004.