Constructing and querying peer-to-peer warehouses of XML resources

Serge Abiteboul Ioana Manolescu Nicoleta Preda

INRIA Futurs & LRI, PCRI, France
firstname.lastname@inria.fr

Abstract

We present KADoP, a distributed infrastructure for warehousing XML resources in a peer-to-peer framework. KADoP allows users to build a shared, distributed repository of resources such as XML documents, semantic information about such documents, Web services, and collections of such items. KADoP2P builds on distributed hash tables as a peer communication layer, and ActiveXML as a model for constructing and querying the resources in the peer network. We describe KADoP’s data model, query language, and query processing paradigm.

1 Introduction

The increasing popularity of P2P architectures and Web services as a data exchange mechanism open up new possibilities for building very large-scale data management applications. We describe KADoP2P, a system for constructing and maintaining, in a decentralized, P2P style, a warehouse of resources. By resource, we mean: data items, such as XML or text documents, document fragments, Web services, or collections; semantic items, such as simple hierarchies of concepts; and relationships between the data and semantic items. KADoP allows a user to perform the following tasks:

• publish XML resources, making them available to all peers in the P2P network and in particular maintain indexing up to date;

• search for resources meeting certain criteria (based on content, structure as well as semantics of the data);

• declaratively build thematic portals from resources of the system.

KADoP leverages several existing technologies and models. First, it relies on a state-of-the-art Distributed Hash Table (DHT) implementation [6] to keep the peer network connected. Second, it uses the power of ActiveXML (AXML) [1], which allows specifying parts of a document as intensional (obtainable by activating or finding service calls). In this context, Active XML is used (i) for intensional indexing and (ii) for supporting a tool to declaratively specify a thematic portal, which can be thought of as a partially materialized view over the XML resources of the P2P network. Finally, KADoP employs sophisticated XML indexing and optimization techniques [3] (not covered here).

This document is structured as follows. Section 2 describes the KADoP’s data model, and Section 3 its query language. We present the system architecture in Section 4, and discuss related work and perspectives in Section 5.

2 KADoP data model for distributed data and knowledge

KADoP’s data model can be declined in two levels. The internal data model is generic, application-independent, and focused on simple resource types. The application-level data model can be built as a customized view on top of the internal data model, including e.g. more complex semantic relationships.

The internal data model  The KADoP internal data model, depicted in Figure 1, supports the types
of resources that can be published and searched for in our system. We distinguish two kinds of resources: data items, and semantic items. Data items (at left in Figure 1) correspond to various resource types:

- A page is an XML document. Pages may have associated DTDs or XML schemas describing their type; we treat DTDs as sources of semantic items (see further). Other formats such as PDF can be used; we ignore them here.

- We consider data with various granularities. A page fragment is a subtree of a page and a collection is a user-defined set of data items. Inside pages, we also consider element labels, attribute names, and (composed) words. We will ignore here issues such as stemming and detecting composed words.

- Finally, a web service is a function taking as input types XML fragments, and returning a typed XML fragment.

Any data item is uniquely identified by an PID (peer ID) and a name. The PID provides the unique name (logical identifier) of the peer that has published the data item, and where the item resides; names allow distinguishing between data items within the peer. Data items are connected by partOf relationships, in the natural sense: thus, a word is part of a fragment, a fragment part of a page etc. Furthermore, any type of data items can be part of collections. A data item residing on one peer may be part of a collection defined on another peer.

Semantic items, depicted at right in Figure 1, consist of concepts, connected by two types of relationships: partOf, and narrowerThan. A graph of concepts, structured with narrowerThan links, is called a (concept) hierarchy. A graph of concepts connected through partOf edges is called a (concept) schema.

Particular knowledge items are derived by an implicit conversion of DTDs associated to pages. From each element type \( \tau \) in the DTD, we create a concept \( c_\tau \); similarly, for every parent-child relationship between DTD types \( \tau_1 \) and \( \tau_2 \), we create a partOf relationship between \( c_{\tau_1} \) and \( c_{\tau_2} \). This implicit conversion reflects the fact that DTD types may correspond to concepts, and parent-child relationships in DTDs typically reflect complex object nesting.

A data item may be connected to a concept via an relatedTo relationship, reflecting the fact that the data item is pertinent to the concept. Such relationships may be produced in three ways. First, they may be specified by a user. Second, they may be inferred automatically between fragments matching a DTD type \( \tau \) and the corresponding concept \( c_\tau \). Third, they can be derived automatically e.g. by a document classifier, or using relevance functions: above a given threshold of relevance of a data item to a concept, a relatedTo relationship is created. We focus on the exploitation of relatedTo links.

**Application-level data model** For a specific application, it may be interesting to define semantically rich relationships between concepts. Let \( P_r;rel \) defined by peer \( P_r \), be a specialized binary relationship between concepts. To record this relationship, KADO creates a new concept \( P_r;rel \). Each time an application defines an instance of \( P_r;rel \) relationship, three new concepts are created: \( P_r;rel \), which stands for that instance of \( P_r;rel \) (p represents instance's identifier), and \( P_r;rel \), \( k \in 1, 2 \), which stand for the roles (two in this case) of concepts involved in rel. To connect \( P_1;rel \) to \( P_2;rel \) under \( P_1;rel \), KADO creates the following relationships: narrowerThan(\( P_r;rel \), \( P_r;rel \)), narrowerThan(\( P_r;rel \), \( P_r;rel \)), partOf(\( P_1;rel \), \( P_2;rel \)) where \( k \in 1, 2 \).

**Example** Figure 2 shows a sample instance of the KADO internal data model, over four peers corresponding to French database labs. A page is depicted as a tree; next to the root, we show the PID and the page name. Rounded boxes contain concepts and relationships between them, published on each peer. Collections, and collection memberships, are listed in italic font, and relatedTo statements appear in diamond-shaped boxes. Now assume on the Atlas peer we declare the binary relationship collaborates (collab). The instance collaborates(Geno:Person, LRI:People) of this relationship yields the concepts shown in (Figure 2, Atlas peer).

### 3 KADO query language

The KADO query language allows retrieving data items, based on constraints on the data item and on their relationship with various concepts. A KADO query \( Q \) is a tree pattern, whose nodes represent data items, and whose edges represent containment relationships among the nodes. Each node may be annotated with: (i) a data item name (tag, document, or collection name) \( n_i \) or with a * name; (ii) semantic constraints of the form relatedTo \( c \), where \( c \) denotes a concept, using a name, and either a PID or a *; (iii) textual constraints of the form contains \( w \),

**Figure 2**: Sample KADO instance of the internal data model, over four peers.
where \( w \) is a word. We distinguish a single \textit{return node} \( N_R \) of \( Q \).

Let \( G \) be the instance of the internal data model, consisting of data items, semantic items, and relationships from the whole KADoP network. A \textit{embedding} of \( Q \) in \( G \) is a function \( \phi \) from \( Q \) to \( G \), under the following conditions. First, \( \phi \) must preserve \textit{partOf} relationships among data items: if \( q_1 \) is a child of \( q_2 \) in \( Q \), then there is a chain of \textit{partOf} relationships going from \( \phi(q_1) \) to \( \phi(q_2) \) in \( G \). Second, \( \phi \) must preserve the node constraints specified by the query: for a query node \( q \), \( \phi(q) \) respects all the tag, text, or semantic constraints associated to \( q \).

We now define the \textit{exact} and \textit{extended} semantics of \( Q \). With exact semantics, for each embedding \( \phi \), \( Q \) returns the node obtained associated to \( N_R \) by \( \phi \). The \textit{extended} semantics of \( Q \) is obtained by relaxing the definition of embeddings, to take into consideration also \textit{narrowerThan} relationships. Let \( q \in Q \) be a query node, annotated with the constraint \textit{relatedTo} \( c \). With extended semantics, \( \phi \) can associate a node \( n \in G \) to \( q \) if \( n \) is an instance of a concept \( c' \), such that a chain of \textit{narrowerThan} relationships connects \( c \) and \( c' \).

\textbf{Example} Figure 3 (left) shows sample KADoP queries; query nodes are labelled by their names, and the returned node is shown in a dashed-line box. \( Q_1 \) returns the title of all papers containing “Toto”. Note that we refer to papers as instances of the concept “publications”. On the configuration in Figure 2, this concept is defined on the LRI peer; thus, \( Q_1 \) returns the titles of DBLP papers containing “Toto”, since they are instances of LRI:publications. \( Q_2 \) returns all project members, from labs that are members of MDP2P. In Figure 2, this denotes members of the AXML, MDP2P, and XIME-P projects of Geno.

\section{System architecture and functionalities}

A KADoP peer consists of several modules, depicted at right in Figure 3.

The DHT is in charge of providing the indexing of resources at the physical level. This is achieved by supporting a hash table, in which key-value pairs can be registered using \textit{put(key, value)}, while key lookup can be performed using \textit{get(key)}, with guaranteed bounds on the number of messages exchanged.

Users’ requests of publishing a resource are processed by the Publisher. First, the resource is serialised in a standard XML form and written in the local Resource Store. Resource \textit{storage} remains thus under the control of the publishing peer. However, the \textit{resource index} must be distributed over the DHT, to enable all peers to look up resources. To that purpose, the Publisher extracts from the resource a set of \textit{key-value} pairs describing the resource and its location on the peer, and calls the \textit{put} service of the DHT layer. The publisher is implemented as a set of AXML documents [1]; it maintains the P2P index up to date using periodic calls to the \textit{put} service.

To answer a user query, search keys are extracted from the query, and a set of corresponding calls to the \textit{get} DHT service are issued. The calls return locations of resources in the KADoP network, which may contribute to the query result. The Distributed Query Processor then calls the Communication Services, asking for the transfer of the relevant resources to the query peer, and combines them in the final query result. Every peer provides a Query Web Service, which takes as input a KADoP query, evaluates it as above, and returns its (XML-ized) result. A portal on a given topic can be easily built as an intensional AXML document: an XML document including calls to the Query Service [1].

\textbf{Key-value indexing of graph-structured resources} The crux of this architecture lies in the choice of the \textit{key-value} pairs to be inserted in the DHT P2P index. The idea is that \textit{search criteria} (such as category or tag names, precise words etc.) make up keys, while \textit{resource location} (PID and precise location of a resource in the peer) make up the associated values. The complete instance \( G \) of the internal data model over all peers is a graph, with 7 types of nodes and 3 types of edges (corresponding to the entities, resp. relationships in Figure 1). Key-value pairs are extracted from \( G \) as follows:

- For each node \( n \in G \) identified by PID:name, we compute the key \texttt{concat(nType,PID,name)}, where \( nType \) is a 3-bit prefix specifying the node type. This key can be used for a precise lookup, when both PID and name are known. We also compute the key \texttt{concat(nType.*,name)}, to be used for lookups by name only. With both keys, we associate the value \texttt{concat(PID,location)}.
• For each edge $e \in G$, connecting the nodes $n_1$ (identified by $\text{PID}_1$: name$_1$) and $n_2$ (identified by $\text{PID}_2$: name$_2$), we compute the keys $\text{concat(eType, PID}_1\text{, name}_1)$, and $\text{concat(eType, e, name}_1)$, where eType is a 2-bit prefix specifying the edge type. To each of those keys we associate the value $\text{concat(PID}_2\text{, location)}$; these key-value pairs allow to search for $n_2$, if $n_1$ is known (to traverse the relationship from $n_1$ to $n_2$). Finally, we create two symmetric key-value pairs, swapping the roles of $n_1$ and $n_2$.

**Intensional indexing** KADOP takes advantage of the intensional aspect of ActiveXML (parts of a document are specified by service calls) to intensionally index resources. The key of an intensional index entry is obtained just as above. However, the value associated to the key is not the location of a resource, but of a Web service that, when called, will return pertinent resources. Intensional indexing is much more concise: a single intensional index entry replaces several extensional (regular) entries, at the expense of one extra service call.

5  Related works and perspectives

We are currently implementing the system described here using Pastry [6] and AXML [1]. Our system is related to XML-based P2P frameworks [1, 7, 13] and semantic P2P networks [8, 10]. KADOP improves over [1, 13] by using a DHT layer allowing each peer to search for resources anywhere in the P2P network. Differently from [1, 13, 7], we set data and knowledge items on equal levels as first-class citizens. The SPIN project [12] addressed building a semantic warehouse of Web resources; in contrast, KADOP focuses on a decentralized P2P context. More motivations for the work presented here may be found in [2].

Semantic data management in P2P has been addressed in [5, 8, 10]; KADOP is different since it relies on a DHT symmetric network, providing each peer direct resource lookup. In contrast, the work described in [10] considers a two-layer hierarchical peer organization. Another difference is that KADOP’s query language simultaneously exploits structure, contents, and semantic information.

The conceptual part of our internal data model can be seen as a subset of models such as RDF [11] or description logics [8]; many languages and platforms for querying such resources exist [4, 9]. More restricted in this aspect, our query language does not directly support e.g. navigation along customized semantic relationships (although our P2P index does). We plan to explore the tradeoff between P2P efficient evaluation, and semantic expressivity, in the near future.

References


