Mobile Information Exchange and Integration: From Query to Application Layer

Van T.K. Tran **Raymond K. Wong** **William K. Cheung**[α] **Jiming Liu**[α]

School of Computer Science & Engineering
University of New South Wales
and National ICT Australia
Emails: trantkv@cse.unsw.edu.au
raymond.wong@nicta.com.au

[α]Department of Computer Science
Hong Kong Baptist University
Emails: william@comp.hkbu.edu.hk
jiming@comp.hkbu.edu.hk

Abstract

Due to the popularity of mobile devices, more and more commercial applications have been developed on these devices. While commercial applications are mostly backed by relational database systems, numerous database engines have been ported to or built on these devices, for example, SQLite. Since connectivity can be unstable or slow, applications such as iAnywhere have considered offline operations while data can be synchronized with the database server whenever the devices are online. On the other hand, while Web-based and XML content are very common these days, unfortunately, these mobile versions of database engines failed to fully support them. This paper considers a translation-based method with a decentralised versioning system in place to support offline operations. Web and XML contents are stored and versioned in a distributed manner and can be synchronized with each other without connecting to a server. The schema of these data can be automatically generated on device. With these schema, a translation engine which allows querying these data using SQL by translating the query to a corresponding XML query is facilitated. We believe this framework support mobile data applications on XML or Web data in a seamless manner. Finally, an initial prototype has been implemented and described in this paper.

Keywords: Decentralised Systems, Version Control, Mobile Data Management, Heterogeneous Information Exchange, Information Integration, XML, Query Translation, Schema Inference.

1 Introduction

The success of mobile devices and wireless communication technology has enabled various applications and activities to be ported from desktop platform into mobile fashion. Although these activities are widespread through a variety of disciplines, from professional to leisure, no matter what environment they are in, collaborative attribute plays a very important and essential part to improve performance and productivity. With the tremendous support of mobile technology, work environments have become very flexible and convenient. A group of people need neither stay at the same place to collaborate, nor attach to their PCs for their work. They can work and cooperate from anywhere as it is possible and convenient for them using their mobile devices.

Working in a collaborative manner, each person works on his/her own separated tasks, yet still closely related to his/her colleagues' tasks. It is also very likely that several people may have to work on a shared document. In such cases, it is essential for one to keep others up-to-date with the status of his/her work and vice versa. Before one makes a change to the shared document, he/she has to make sure that it is an updated document, so that changes are unlikely to be undone and redone. In other cases, the shared document may need to be reverted to its previous state. To manage all these scenarios of collaboration, version control systems are of great help.

In a collaborating environment, conflicts are unavoidable. When two collaborators attempt to update the shared document at the same time, conflicts could occur. Conflicts are generally resolved via communication between collaborators or by decisions of group leaders. The latter is less preferable for its inflexibility. The system should be flexible enough to allow two group members involved in the conflicts to discuss, reach a consensus and make a decision on how to control shared document versions. However, given a mobile teamwork condition, especially when communication is not as instant and convenient as in face-to-face manner, concurrency control is challenging.

Different architectures have been introduced and investigated to support work collaboration in mobiles, including centralised and decentralised systems. Centralised systems are server-client systems where all mobile clients are connected through a central server. The central server is responsible for controlling all communications and activities amongst mobile clients. Central servers and central authorities, however, are not always available for mobile collaborations. In such a situation, the concept of decentralised system appears to be very useful. Connection preference and quality sometimes also make decentralised systems more attractive than centralised systems. Version control in centralised systems is quite straightforward since all controls and management can be done in server-side, which is not much

different from applications for desktop platforms. On the other hand, version control in decentralised mobile systems is still a challenge.

There have been several decentralised version control studies around (Ellis & Gibbs 1989, Munson & Dewan 1996, Suleiman et al. 1998, Ionescu et al. 1999, Jiang et al. 2005, Su et al. 2007). These systems mainly focus on the decentralised aspect that a node can work without connection to the server. In other words, it is not necessary to connect to a server all the time for one to get work done. Instead he/she can just work on the local version and synchronise it with the server later when there is a connection. This paper, on the other hand, proposes a version control framework focusing on another feature of decentralisation, in which nodes can share documents in a peer-to-peer manner and without a central authority. We believe that this framework is much more suitable for mobile devices.

Inspired by the fact that Web-based and XML contents are very common in these days and, yet the mobile versions of database engines (mostly SQL-based, e.g., SQLite) failed to fully support them, this paper also investigates the notion of automatically generating the schema of these data on devices and introduces an XML Intelligent Agent to enable information exchange and information integration between heterogeneous data sources among the local repositories from individual mobile devices. Using the agent, a data source can query multiple data sources in its native query language and integrate the results obtained without any knowledge of the data representation, format or query language of the other heterogeneous data sources. The agent uses XML and iQuery (a derivative of XQuery) to represent queries and results. A mobile application queries the agent in a industry common query language (at the moment, most mobile data applications are still based on a subset of SQL). The agent translates the query to iQuery and routes the iQuery (or part-of) to other agents capable of processing the query. These agents translate the query to some native query language, execute it on the heterogeneous data source, obtain the result and return an XML representation of the result to the source agent. The source agent integrates all results and translates the final XML result representation to the data source's native result format (e.g., tabular, relational format) before returning it to the mobile applications.

This paper is outlined as follows: Section 2 gives an overview of related work in decentralised version control systems, query wrappers, and reference structural inference methods for XML data. Section 3 describes the motivating scenario that leads to the framework proposed and described in Section 4. Section 5 provides some experimental results to justify the proposed framework. Section 6 concludes the paper and discusses future work.

2 Related work

2.1 Version control and decentralised systems

Decentralised systems can be distributed or replicated systems. For either distributed or replicated systems, the copies of each object at all nodes must be kept consistent. Suleiman et al. (1998) presented an operational transformation algorithm that based on the notion of user's intention and using semantic properties of operations called forward and backward transpositions to serialise concurrent operations, in order to maintain the consistency amongst replicated copies. The forward and backward transpositions enable the equivalent histories to be characterised, in which forward transposition resolves the problem of concurrency operations and backward transposition changes the order of operations in the history without violating user's intention.

Jiang et al. (2005) proposed a semi-replicated architecture to maintain the consistency of the replicas in mobile environments. In this architecture, a central server acts as the agents of mobile sites and is used to backup a copy of the shared object, while mobile devices with limited resources hold only parts of the object. With this architecture, the agents that reside in the central server take full responsibility for managing operations across mobile sites, and preserving consistency of the replicated documents.

The SVK version control system introduced in (Kao 2003) is a decentralized version control system built with the robust subversion filesystem. It supports repository mirroring, disconnected operation, history-sensitive merging, and integrates with other version control systems, as well as popular visual merge tools. In other words, SVK is a way to work around the centralised design of SVN (CollabNet 2006) and should be seen as an extended client, not a replacement (Robert 2006). This SVK system is then used as a base system for our prototype presented later in this paper.

Many conventional solutions for version control on those decentralised systems have involved locking approaches. Munson & Dewan (1996) developed a framework based on a locking algorithm to prevent concurrent operations. In this framework, a user requests a lock on a particular object and the lock is held until the end of the transaction. In the meantime, other users can only read and receive updates but not modify that object. This algorithm focuses on conflict prevention rather than conflict management. Citro et al. (2007) overcame the above limitation by introducing a delayed post-locking algorithm. This algorithm is a variation of the conventional post-locking algorithm, in which the modified object is automatically locked when a conflict occurs.

The problem of locking approaches is the decreasing level of concurrency. Chianese et al. (2008) improved concurrency of the locking approach by considering frequent disconnection or inactivity periods of transaction. Besides locking approach, other strategies include operational transformation and multi-versioning (Suleiman et al. 1998, Citro et al. 2007, Ellis & Gibbs 1989). These strategies are aimed to manage and resolve conflicts in version control systems.

Generally, those above systems mainly focus on how a node can operate on its own without a central sever or how to resolve conflicts at a node, but they do not mention how nodes communicate and collaborate with one another. Ionescu et al. (1999) addressed this issue by introducing a replication architecture, in which if a change is made in a local object of a node, all other nodes will be notified to change accordingly. To use network resources efficiently, Su et al. (2007) designed an integrated consistency-control algorithm to decide a limited number of nodes get updated upon any changes in the network, instead of all nodes. This algorithm is defined based on the probability of contents selection and node update.

2.2 Query translation and information exchange

Many efforts have been invested in data conversion & query translation for information exchange and/or integration. For instance, NoDoSe (Adelberg 1998) can semi-automatically process input file using user-defined schema and GUI to specify a region in the input file for each object. It has an addition HTML parser specifically for HTML files. W4F wrapper (Sahuguet & Azavant 1999) also focus on parsing HTML to XML. It relies on the structure of HTML and its extraction language HEL is based on DOM.

The extraction rules introduced by Hammer et al. (1997), based partly on regular expression and nested structure of HTML documents, have variables for storing extracted data. Similar to (Hammer et al. 1997), as a wrapper generation tool, XWRAP (Liu et al. 2000) provides a component-based library to be used by the generated wrappers and there is an inductive learning based mechanism to determine the patterns of the document structures for wrapping. Ashish & Knoblock (1997) proposed an approach to automate the process of generating an extractor that can recognize the structure of HTML. This is done by identify tokens using HTML tags and regular expression. The nested structure of the input is detected using heuristics such as font size of heading and indentation.

In addition to data format transformation (which is needed to transform the source data and/or the output of the query to a required format), one major focus of this paper is on query language translation. Most recent efforts have been focused on translating XML queries to SQL, due to many proposals on implementing XML databases using relational database systems. For example, Krishnamurthy et al. (2004) presented an efficient method to translate XML query to SQL. Other work including (Yu 2004) focusing on rewriting the queries into a different form to facilitate efficient data integration. Different from most of these related work, since most mobile data applications have been built using an underlying SQL data access layer and most Web/mobile content are stored natively as XML or its related format, we present a query translation agent from SQL to XML. We argue that querying XML data using SQL is fundamentally more challenging and requires more 'intelligence'. This is due to the fact that the formation of an SQL query requires a good knowledge of a fixed, predefined schema. However, schemas are usually not availabled or stored with the content in the mobile devices. Even they are available, due to the heterogeneity of the data, it would be desirable to maintain a integrated version of the schemas of the stored content so that queries involving different content can be supported. Therefore, a lightweight, schema generation mechanism is implemented.

2.3 Structural schema inference

In order to provide a good and lightweight schema inference mechanism for the needs described above, we need to evaluate different alternatives. While most of the schema inference methods cost a similar amount of time to generate a schema, their degrees of accuracy vary. Numerous algorithms are implemented for evaluation in this paper. Due to the number of algorithms implemented, full descriptions of them are not presented here. Further description of these algorithms can be found from (Sankey & Wong 2001), and some related literature is included below.

The first known paper to address DTD generation using tradition grammatical inference methods was proposed by Ahonen (1996). The two methods proposed there are theoretically appealing, as they guarantee to infer languages falling within certain language classes. These classes are termed k-contextual and (k, h)-contextual, so named as they assume the structure to be inferred to have limited context. Another method applied to DTD generation (Young-Lai 1996), is derived from more recent work in grammatical inference. The base algorithm is known as Alergia, introduced in (Carrasco & Oncina 1994). The criterion for state equivalence in Alergia is based upon observed frequencies of transitions and finalities of a pair of states. The most recent work on schema inference by Bex et al. (2008) focused on a probabilistic algorithm that learns k-occurrence regular expressions for increasing values of k, and selects the one that best describes the sample based on a Minimum Description Length argument.

3 Motivating scenario

This section discusses a scenario that describes the motivation for our work. The scenario explains how our work is worthwhile. Let us consider a system consisting of two mobile nodes and a central server as in Figure 1. In this system, the version control function is done in the central server, placed within the Intranet of a company. The two client nodes collaborate with each other via the control of the server.

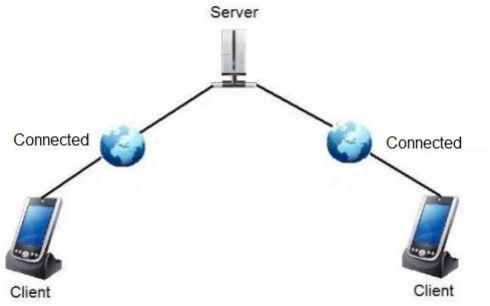

Figure 1: Centralised system

Let us now consider the situation when two employees of the company go overseas to demonstrate their product to a customer. The employees use these two mobile devices for the demonstration. Before the demonstration, one of them recognises that some changes need to be made in one mobile device, and the changes need to be updated in the other mobile. Because of the connection problem, they cannot connect to the server in the Intranet of the company back home, meanwhile they are located close to each other and good connection can be established with short-range transmission protocols like Bluetooth. It makes more sense to have them communicate directly with each other. In such a situation, the concept of decentralised system as in Figure 2 appears to be very useful.

The version control function implemented in this decentralised system will support the synchronization and update process between the two clients.

Furthermore, XML has grown in popularity as a Web publishing format, as a messaging format, as a data

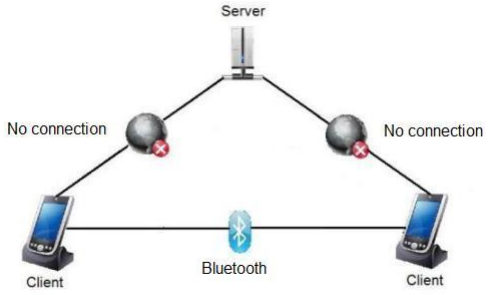

Figure 2: Decentralised system

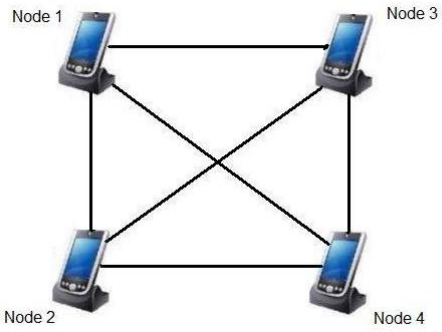

Figure 3: Proposed framework

exchange format and as a means of storing data. Data stored in XML format preserves the data representation, structure and its semantic meaning. Hence, more and more content stored in mobile devices are in XML. However, most mobile data management applications (e.g., iAnywhere) rely on an SQL access interface to the underlying content storage. Therefore, there is a need for a query wrapper that allows querying multiple XML repositories using SQL. It should also be possible for a mobile user to query / integrate multiple heterogeneous content files (possibly located in different devices) and use them as if they were a single data source, with a single global schema.

4 Proposed mobile framework

This section describes in detail our contribution, particularly:

- How mobile information is exchanged and integrated in the mobile application layer with our proposed decentralised version control framework (section 4.1)
- Down to the SQL data access layer, information exchange and integration amongst SQL databases have been studied quite intensively. However, most mobile contents are stored in XML or its related formats. This paper, therefore, focuses on the translation from SQL to XQuery to enable information exchange and integration amongst heterogeneous data sources in XML format using SQL (section 4.3)
- To achive this aim, schema generation for the underlying XML content is needed; therefore, a good and lightweight structural inference for XML data is required (section 4.2)

4.1 Decentralised version control framework

The proposed framework is a decentralised version control system (figure 3) that focuses on collaboration and consistency maintenance. The nodes communicate with one another in a peer-to-peer manner, given that good connection can be established with short-range transmission protocols like Bluetooth. Essentially, each node performs both server and client functionalities and behaviours of a version control system.

In a general version control system, the main and most popular functions are *checkout*, *update*, and *commit*. Within a traditional decentralised system, all nodes are treated equally. In other words, when a node performs *commit*, changes are applied to all other nodes. When a new node is added to the system, it can *checkout* from any random node in the system. *Update* is not really necessary here, since every node is kept current all the time.

However, not all nodes need to be kept updated all the time, since some of them may be just used for one-off tasks. To increase performance and effectiveness, our proposed framework consists of nodes not treated equally and not storing exactly replicated data. Particularly, when a node performs a change, it *commits* to its own server first, then signals other nodes that a new change has occurred in the system; however, only a few algorithm-decided peer nodes are notified to update their replicated documents accordingly. Other nodes will get updated upon requests. On the other hand, when a node requests to *checkout* or *update*, the requests will not be sent to any random nodes in the system, but to an algorithm-decided node.

In order to obtain those desired algorithm-decided nodes, our framework introduces the concept of *state table* and *active peer*, and then explain in details how several common functions (*checkout*, *update*, and *commit*) fit into this framework.

4.1.1 Active peers and state table

A *node j* is considered to be active when it gets updated, receives requests from other nodes, and makes requests to other nodes regularly. The active level of a node to another node is represented by a property called *active rate*. To decide the *active rate*, a *state table* is used. Each node maintains a *state table* for all of its connected peers. The *state table* of a *node i* consists of the identifier of any *node j* connected to it, the latest revision of *node j*, and the total number of made requests and received requests of *node j*. The *state table* also keeps a special field R_{all} representing the overall latest revision of the whole system, in order to identify its status (current or outdated) within the system. The *state table* must be updated regularly to ensure each node has its peers' latest information.

Let

- R_i denote the latest revision of *node i*
- R_j denote the latest revision of *node j*
- N_j denote the total number of made requests and received requests of *node j*
  ($N_j \geq 1 \ \forall j$ since every node must checkout at least one when first connecting)

- $A_i(j)$ denote the active rate of *node j* to *node i*

The parameters used in this algorithm can be easily located, collected and stored in the *state table*. The number of made and received requests of a node N_j illustrates its popularity and interaction frequency amongst its peers. The ratio $\frac{R_j}{R_i}$ defines the current state of *node j* compared with *node i*. These parameters altogether represent the active rate $A_i(j)$ of *node j* to *node i*.

$A_i(j)$ is decided as:

$$A_i(j) = \frac{R_j}{R_i} * N_j \quad (1)$$

The active rate $A_i(j)$ is used for the *commit* operation on node i. R_i is quaranteed to be the latest revision value by the operator at node i, hence $R_i = R_{all}$. The higher the ratio $\frac{R_j}{R_i}$ is, which means the higher R_j is, the more recently node j gets updated. The higher N_j is, the more traffic goes in and out of node j. All in all, the higher $A_i(j)$ is, the more active *node j* is, compared with *node i*.

Generally, this algorithm provides a simple yet practical and effective mechanism to decide the appropriate active nodes for versioning control operations.

4.1.2 Commit

Before an action is taken on *node i*, it is the responsibility of *node i*'s owner to make sure that *node i* is current or in other words, R_i is the overall latest revision. This can be obtained by simply comparing R_i with the special field R_{all} reserved in the *state table* of *node i*.

To commit a change, *node i* first commits the change to its own server, then informs all other nodes to update their special field R_{all}, and then star-synchronizes between its server and its connected nodes with highest active rates. This way, a node will always be able to maintain the updated value of R_{all} as long as it is kept connected in the system. On the other hand, if a disconnection occurs, after the reconnection, a node can retrieve the updated value of R_{all} from any other nodes that it connects to. This mechanism keeps all nodes be aware of the current state of the system.

This sequence of action is illustrated in figure 4.

In ideal cases, no other changes are made concurrently with *node i*, R_i is increased by 1 and becomes the highest revision value (R_{all}). Therefore, $R_j < R_i \ \forall j \rightarrow A_i(j) < 1 \ \forall j$.

If there exists j such that $A_i(j) \geq 1$, it is an indication that a conflict occurs. When a *commit* operation is performed, node i consists of the updated revision of the system, i.e. R_i equals to R_{all} and be the highest revision value. Hence, R_j must be either less than or equal to R_i, which makes $A_i(j) \leq 1$. Given a situation in which node j is up-to-date too, i.e. R_j equals to R_i; node j performs another *commit* operation concurrently with node i, and increases R_j by 1, meanwhile node i commits without checking R_i and R_{all} again. As a result, conflict occurs. In this case, R_j is greater than R_i, which makes $A_i(j) \geq 1$.

The conflict needs to be resolved first using appropriate strategies discussed in section 2, before *commit* can take place again.

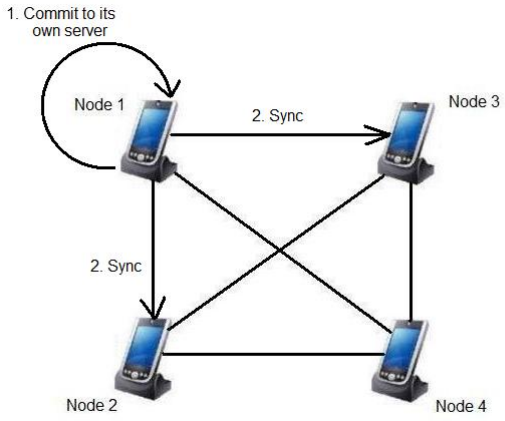

Figure 4: Operation: Commit

4.1.3 Checkout and update

Checkout and *update* are basically similar to each other and simplier than *Commit*.

When a node is added to the system, it performs *checkout* and *update* by repeatly comparing its connected peers' revision values with R_{all} of the system until a match is obtained, to make sure it gets updated with the lastest version of the shared documents. If none of its connected peers has the lastest revision, it can requests its highest revision peer to perform a different *update* request to be brought into the up-to-date state and then notify it to change accordingly.

If *node i* checkouts or updates from *node j*, N_i and N_j are increased by 1 and updated in the associated *state tables*.

4.2 Structural inference for XML data

Testing results presented in (Sankey & Wong 2001) revealed that the merge family methods (sk-strings method) performed well in some cases, and the optimisation methods (Ant Colony Optimisation - ACO) better in others. This led to the development of the sk-ANT Heuristic, based on the most successful method of each type.

The sk-ANT Heuristic: The motivation for this algorithm was to create a method that would be successful for a variety of input data sizes by combining the best features of both the sk-strings and ACO techniques. One consideration was to first run the sk-strings algorithms, and then use the results to seed the ACO optimisation. However, this approach suffers from several problems. Firstly, it is not practical to attempt all possible combinations of both algorithms. Thus we would be required to choose a limited number of models resulting from the sk-strings technique to seed the second stage of the process. The simplest way to achieve this would be to choose the best models, up to a reasonable number. These models will not necessarily lead to the best results, though, as they may have already been over-generalised. More importantly, by letting the sk-strings methods run to completion we would

lose many of the advantageous aspects of the ACO method. Most notably, its willingness to explore a greater breadth of the search space would be missed.

The sk-ANT method thus incorporates both the sk-strings and ACO heuristics at each step of the inference process. It is most easily described as a modified version of the ACO technique with the ants guided by an sk-strings criterion. The guiding is made progressively weaker as the model becomes smaller, to allow the advantages of the ACO method for smaller models to take effect. The main modification was made to the ant move selection of Algorithm 4, producing a new method as shown in Algorithm 5. The first major difference appears on line 4, where a merge must pass the sk-strings criterion to be considered. The outer while loop on line 2 and if statement on line 11 combine to progressively weaken the sk-strings criterion when it has become too strict. Eventually the criterion will be weak enough to let all merges pass, and the algorithm will behave identically to the original version.

Input: A set of all state pairs $merges$,
an ant heuristic $heuristic$,
an ant weighting function $weighting$,
a pheromone table $pheremones$
and an sk-strings criterion $skCriterion$.

Output: A state pair representing the chosen merge.

Method:

```
1.  choices ← [ ]
2.  while choices.size() = 0 do
3.    for merge in merges do
4.      if skCriterion(merge) then
5.        h ← heuristic(merge)
6.        p ← pheremones[merge]
7.        value ← weighting(h, p)
8.        choices.add((value, merge))
9.      end if
10.   end for
11.   if choices.size() = 0 then
12.     skCriterion.weaken()
13.   end if
14. end while
15. return stochasticChoice(choices)
```

Figure 5: sk-antMoveSelector Algorithm

4.3 XML intelligent agent - Query translation from SQL to XQuery

The agent is bound to one or more heterogeneous data sources and communicates with zero or more other agents. A data source queries the agent in its native query language and receives results in its native query result format. It is the agent that queries other heterogeneous data sources, communicates with other agents, integrates the results received and translates the results to the native result format of the data source. The aim is to hide the data representation used to exchange data inside the agent framework and allow data sources to query other heterogeneous data sources and integrate information using their native query language.

This section discusses SQL to XQuery translation, in the context of the XML Intelligent Agent framework. In fact, the agent translates the SQL query to iQuery (intermediate Query representation) that is in turn translated to XQuery or any other query languages. These are the steps for translating a SQL query to XQuery for further processing:

1. Convert to iQuery
   Agent A receives a SQL query, validates it and then converts it to iQuery. For a SQL query to be successfully validated, it must be:
   - specified in the SQL92 standard syntax.
   - qualify all expressions, if the query involves a join on more than one relation.
   - (optional) use Oracle's schema notation to specify the keywords used in locating a resource.

   If the query involves a join on more than one SQL relation, all expressions in the scope of that query need to be qualified. For example, (SELECT author FROM books) is valid but (SELECT author FROM books, articles) is invalid because the agent cannot figure out which relation contains the author column. The correct syntax is (SELECT books.author FROM books, articles). There needs to be a mechanism for specifying the keywords used in locating a resource. We borrow Oracle's schema identification notation to provide this feature. For example, a query of the form (SELECT author FROM BookSchema.Library) generates the keywords BookSchema and Library.

   To illustrate SQL to XQuery conversion, we use the following example:

```
SELECT book, author
FROM BookSchema.library
```

   The iQuery equivalent of the above SQL query is:

```
<result level=1>
{
FOR $library IN resource("bookschema, library")//??
RETURN
<result_tuple>
{$library//book}
{$library//author}
</result_tuple>
}
</result>
```

2. Resolve SQL Expressions
   Next, the iQuery is either processed by Agent A or sent to another agent that can process the query. For the purposes of clarity, we refer to the target agent that processes the iQuery as Agent B.

3. Resolve resource Expression
   Agent B receives the iQuery. First, it must find the XML documents required to process the query. This is a three step process:
   - For each resource keyword in each FOR clause of the iQuery:
   - Extract the resource keywords
   - Use these keywords to find the XML document
   - Replace the resource function with the XQuery document function. The argument of the document function is the name of the XML document.

The mechanism used to locate XML documents using resource keywords is not part of the agent framework. It depends on the heterogeneous resource. One method may be to concatenate all the keywords together to construct a filepath that identifies the document. Another method may be to lookup the keywords in a database that maps keywords to XML documents. Assuming, use of the first method, the result is:

```
<result level=1>
{
FOR $library IN
document("bookschema/library.xml")//??
RETURN
<result_tuple>
{$library//book}
{$library//author}
</result_tuple>
}
</result>
```

4. Process level Instruction
   SQL cannot handle composite types. XML documents however, are of a hierarchical nature and can be nested to any level of complexity. This is a problem. If the author element in bookschema/library.xml has child elements first_name, last_name, age; these child elements will be returned when the XQuery is processed. But SQL expects an atomic type in the author field, not a structure composed of first_name, last_name, age.
   iQuery solves this problem. The iQuery above contains an attribute in the result tag named level. A value of 1 requires the agent to confirm that the nodes selected in the RETURN clause of the XQuery have a depth of 1. The following cases are allowed:

   - node has no text node or children but only one attribute (in this case, the value of the attribute is used)
   - node has a non-empty text node and 0 or more child nodes and 0 or more attributes(in this case, the value of the text node is used)
   - node has no attributes, text node or child nodes (in this case, the "null" string is used)

   If all XPath expressions in the RETURN clause satisfy one of the above conditions, execution proceeds to the next step. If a selected node has no text node but has two or more child nodes or attributes, an error XML document is returned as the result.

5. Identify XML attributes
   The agent learns the elements and attributes in the target XML document. If the XQuery accesses an element which exists as an attribute in the XML document, the query is modified to access the attribute and not the element.

6. Process distinct Instruction
   In addition to the level attribute, the result tag may also contain an attribute called type. This attribute can only have one value - DISTINCT. It the type attribute is set, the XQuery is modified to ensure that the query result does not contain any duplicates.

7. Process XQuery, Return SQL Relation
   Lastly, the XQuery is forwarded to the XQuery engine, processed and the resulting XML document returned to Agent A. Agent A generates a SQL result representation from the XML document and returns it to the querying data source. This process is trivial. First the agent attempts to create a metadata SQL structure from the XML document. Nested child nodes inside an element node are interpreted as the result of a GROUP BY statement. Attributes of an element node are treated as child elements. Then the XML document is converted to a SQL result based on the derived metadata.

5 Experimental results

5.1 SQL to XQuery examples

To illustrate query translation, we make use of a small database with the Entity-Relationship and relational schema as in Figure 6

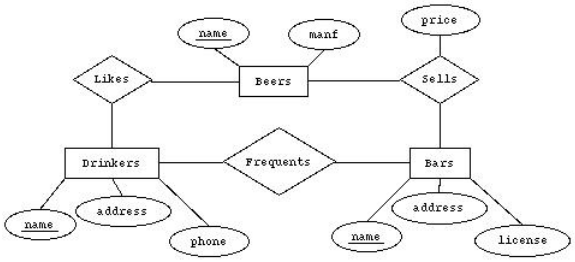

Figure 6: A sample relational database

The XML representation of the database in Figure 6 is a set of XML documents bound to an XQuery engine. Each relation in the database maps to an XML document.

SQL query to find the names and brewers of beers that John likes:

```
SELECT manf, name
FROM Beers
WHERE name IN
(
SELECT beer
FROM Likes
WHERE drinker = 'John'
)
GROUP BY manf, name
```

The iQuery equivalent to the above SQL query is:

```
<result>
{
FOR $beers IN
distinct(document("beers.xml"))//beer_tuple/manf
LET $XQ_FUN1_RESULT := XQ_FUN1()
LET $beers1 := document("beers.xml")
//beer_tuple[manf=$beers
AND name=$XQ_FUN1_RESULT]
WHERE not(empty($beers1))
RETURN
<result_tuple>
{$beers}
{$beers1/name}
</result_tuple>
}
</result>
```

in which:

```
DEFINE FUNCTION XQ_FUN1 () RETURNS sequence
{
FOR $likes IN document("likes.xml")
//like_tuple[drinker="John"]
RETURN $likes/beer
}
```

Figure 7 shows the XML Result representation.

```
<?xml version="1.0"?>
<result>
  <result_tuple>
    <manf>Caledonian</manf>
    <name>80/-</name>
  </result_tuple>
  <result_tuple>
    <manf>Sierra Nevada</manf>
    <name>Bigfoot Barley Wine</name>
    <name>Pale Ale</name>
  </result_tuple>
  <result_tuple>
    <manf>Lord Nelson</manf>
    <name>Three Sheets</name>
```

Figure 7: XML result representation

Figure 8 shows the SQL Result representation.

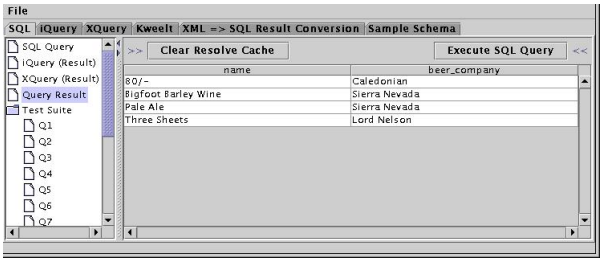

Figure 8: SQL result representation

Note that the length of the resulting XML document is minimized by grouping the results. But doing this does not affect the final SQL result. Also, note that the datatype sequence is identified in the function but it is not bound to a XML Schema or explicitly defined anywhere. This is impossible as the agent does not know the datatype of the sequence or the result of the function.

5.2 Inference algorithms

The small data set consisted of 100 sample files generated from random Probabilistic Finite State Automaton (PFSA) with a maximum of 5 states and an alphabet cardinality of 4. A total of 10 sample strings were generated for each PFSA, leading to Prefix Tree Automaton (PTA) with an average size of 42.15 states. The number of strings generated was deliberately kept small, as sparse data is an important problem in practical cases of inference. The average size of the models inferred by the algorithms ranged from 1.03 to 40.55 states, with the best models in terms of Minimum Message Length (MML) typically having between 2 and 5 states.

Figure 9 shows the success rates of several algorithms in inferring models with the lowest MML values. For each algorithm, two results have been shown. The first is the frequency of inferring the best model overall, by choosing the best of the algorithms attempts.

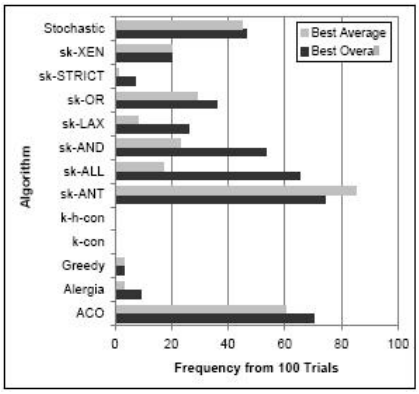

Figure 9: Success rates for small models

The second is the frequency of obtaining the best average performance, derived by averaging all of the algorithms attempts before ranking. The best overall performance is most important, whilst the best average indicates stability across diferring input parameters. The results clearly show that the sk-ANT algorithm performed best in terms of both rankings, particularly the best average ranking. The next two best algorithms were the original ACO method, and the sk-ALL heuristic (a combination of the results from all ak-strings variants.) This is one of the reasons those methods were chosen as the basis for sk-ANT. The other previously applied algorithms and reference methods performed quite poorly. Although the Stochastic method was able to infer some good models, it trailed significantly.

Algorithm	Average Deviation (%)	Worst Deviation (%)
ACO	3.41	41.50
Alergia	16.06	62.25
Greedy	91.45	459.04
k-contextual	19.43	53.43
(k, h)-contextual	17.11	45.39
sk-ANT	0.91	15.20
sk-ALL	1.69	17.50
Stochastic	5.53	45.01

Figure 10: Deviation from the best model inferred (small data set)

Statistics relating to the consistency of the algorithms over the 100 test cases were also gathered, in the form of comparisons against the best inferred models. Figure 10 shows both the average deviation and worst deviation in percent for each of the algorithms. The numbers were derived from the difference between the MML values of the best model inferred by a given algorithm as compared with the best model overall. We omit the individual sk-strings algorithms, preferring the combined sk-ALL results. The results show that the sk-ANT method is the best in terms of average and worst deviation, followed by the sk-ALL hueristic. Note that the worst case deviations may be too high for some applications. In such cases, using a combined approach with both the sk-ANT and sk-ALL algorithms would yield more consistent results.

5.3 Decentralised system

The SVK version control system mentioned previously in section 2 is used as a based system for our initial prototype. We created a C# application on top of the SVK system to handle communication between nodes. Some performance measurements have been carried out to evaluate the execution time of various file size, multiple files and multiple servers. Several nodes are set up to communicate with each other, sending and receiving requests.

Figures 11 and 12 show that the graphs of execution time of various file sizes and multiple files are almost linear. In other words, there is not much difference in terms of overhead between multiple small transactions and an equivalent big transaction. These experimental results provide a good justification for a version control framework, since collaborators can synchronise their work as frequently as possible, bit-by-bit without having to wait for the whole tasks done.

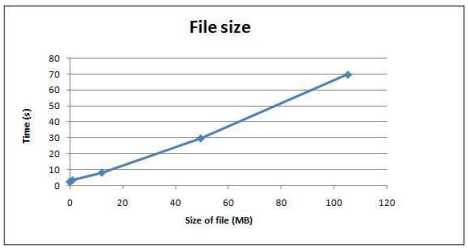

Figure 11: Execution time of various file size

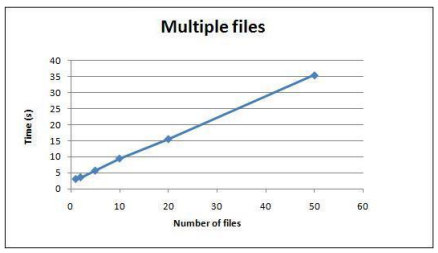

Figure 12: Execution time of multiple files

Figure 13 illustrates the same implication for execution time of multiple servers. The graph of execution time of multiple servers is also almost linear, i.e. the overhead of multiple single-server-synchronisations is not significant in comparision with one multiple-server-synchronisation. Therefore, it is not necessary for one transaction to involve a synchronisation with as many servers as possible to minimize the total transaction time; instead, single server transaction can be done as per requested any time without breaking the minimal total execution time. Ideally, the execution time, as a result, does not effect the active rate attribute of a node.

6 Conclusion

The emergent development of mobile devices and technology has significantly enhanced cooperation in many commercial applications.

Cooperation requires a sufficient version control system in mobile environment without a central server. In this paper, the proposed decentralised version control framework has successfully addressed this requirement and provides a simple yet efficient algorithm to support communication between mobile devices directly.

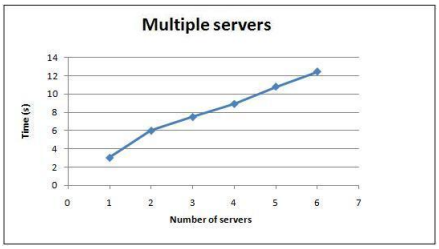

Figure 13: Execution time of one client and multiple servers

The paper has also presented a scalable, lightweight solution to enable information exchange and integration between heterogeneous data sources. The solution also allows querying heterogeneous data sources, especially in XML format, using a relational query language (for example, SQL). Thus most existing mobile data management applications can use the proposed framework for heterogeneous, XML content without expensive changes. In order to facilitate querying XML content with SQL, schema generation for the underlying XML content is needed. Therefore, we have also addressed the problem of structural schema inference for XML and evaluated various algorithms to infer schema for XML data on mobile devices.

Our algorithm for structural schema inference, as well as most of the existing ones, runs in batch mode, that is it has to be rerun whenever there are new documents added. For future work, we will investigate how to extend the existing inference technique to an incremental version, while still maintaining minimum memory and CPU usages.

Acknowledgement

William K. Cheung and Jiming Liu are partially supported by HKBU Faculty Research Grant FRG/03-04/II-70 for this work. Part of this work was done when Raymond K. Wong was visiting HKBU.

References

Adelberg, B. (1998), 'NODOSE—a tool for semi-automatically extracting structured and semistructured data from text documents', *In SIGMOD* **27**(2), pp. 283–294.

Ahonen, H. (1996), 'Generating grammars for structured documents using grammatical inference methods', Technical report, Department of Computer Science, University of Finland.

Ashish, N. & Knoblock, C. A. (1997), 'Wrapper generation for semi-structured internet sources', *In SIGMOD* **26**(4), pp. 8–15.

Bex, G. J., Gelade, W., Neven, F. & Vansummeren, S. (2008), 'Learning deterministic regular expressions for the inference of schemas from xml data', *In WWW '08: Proceeding of the 17th international conference on World Wide Web*, ACM, New York, USA, pp. 825–834.

Carrasco, R. & Oncina, J. (1994), 'Learning stochastic regular grammars by means of a state merging method', *In ICGI '94: Proceedings of the 2nd International Colloquium on Grammatical Inference*, **862**, Springer-Verlag, pp. 139–150.

Chianese, A., d'Acierno, A., Moscato, V. & Picariello, A. (2008), 'Pre-serialization of long running transactions to improve concurrency in mobile environments', *In ICDEW '08: Data Engineering Workshop* pp. 129–136.

Citro, S., McGovern, J. & Ryan, C. (2007), 'Conflict management for real-time collaborative editing in mobile replicated architectures', *In ACSC '07: Proceedings of the thirtieth Australasian conference on Computer science*, Australian Computer Society, Darlinghurst, Australia, pp. 115–124.

CollabNet (2006), 'Tigris.org: Open source software engineering tools'.
URL: *http://www.tigris.org/*

Ellis, C. A. & Gibbs, S. J. (1989), 'Concurrency control in groupware systems', *In SIGMOD* **18**(2), pp. 399–407.

Hammer, J., Garcia-molina, H., Cho, J., Aranha, R. & Crespo, A. (1997), 'Extracting semistructured information from the web', *In Proceedings of the Workshop on Management of Semistructured Data*, pp. 18–25.

Ionescu, B., Binder, J. & Ionescu, D. (1999), 'A distributed architecture for collaborative applications', *Pacific Rim Conference on Communications, Computers and Signal Processing, IEEE* pp. 525–529.

Jiang, B., Zhang, H., Chen, C. & Yang, J. (2005), 'Enable collaborative graphics editing in mobile environment', *In CSCWD* **1**, pp. 313–316 .

Kao, C.-l. (2003), 'The svk version control system'.
URL: *http://svk.elixus.org/*

Krishnamurthy, R., Kaushik, R. & Naughton, J. F. (2004), 'Efficient xml-to-sql query translation: Where to add the intelligence', *In VLDB*, pp. 144–155.

Liu, L., Pu, C. & Han, W. (2000), 'XWRAP: an xml-enabled wrapper construction system for web information sources', *Proceedings of the 16th International Conference on Data Engineering* pp. 611–621.

Munson, J. & Dewan, P. (1996), 'A concurrency control framework for collaborative systems', *In CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, ACM, New York, USA, pp. 278–287.

Robert, O. (2006), 'Dvcs or a new way to use version control systems for freeBSD'.
URL: *citeseer.ist.psu.edu/749273.html*

Sahuguet, A. & Azavant, F. (1999), 'Building light-weight wrappers for legacy web data-sources using w4f', *In VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 738–741.

Sankey, J. & Wong, R. K. (2001), 'Structural inference for semistructured data', *In CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, ACM, New York, USA, pp. 159–166.

Su, Z., Katto, J. & Yasuda, Y. (2007), 'Efficient consistency control for mobile dynamic contents delivery network', *International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications* pp. 171–173.

Suleiman, M., Cart, M. & Ferrie, J. (1998), 'Concurrent operations in a distributed and mobile collaborative environment', *Proceedings of 14th International Conference on Data Engineering*, pp. 36 – 45.

Young-Lai, M. D. (1996), 'Application of a stochastic grammatical inference method to text structure', Master's thesis, Computer Science Department, University of Waterloo.

Yu, C. (2004), 'Constraint-based xml query rewriting for data integration', *In SIGMOD*, pp. 371–382.