ViSMA: Extendible, Mobile-Agent Based Services for the Materialization and Maintenance of Personalized and Shareable Web Views[1]

G. Samaras, K. Karenos
Dept. of Computer Science
University of Cyprus
CY-1678 Nicosia, Cyprus
{cssamara, cs98kk2}@ucy.ac.cy

P. K. Chrysanthis
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
panos@cs.pitt.edu

E. Pitoura
Dept. of Computer Science
University of Ioannina
GR-45110 Ioannina, Greece
pitoura@cs.uoi.gr

Abstract

ViSMA (Views Supported by Mobile Agents) is a prototype set of extendible mobile-agent based services that allow the definition, materialization, maintenance and sharing of views created over remote web-accessible databases. The system's primary goal is to support mobile clients carrying lightweight devices of various connectivity and resources such as portable computers and PDAs by being completely deployed over mobile agent technology. In this paper we propose a multi-tier architecture on which the system is built and we present the system's features, which include view mobility, shareable and personalized view creation and customizable, automated view monitoring and updating. The system has been fully implemented and tested using three alternative client types.

1. Introduction

Data accessing in the recent years has been affected by three major trends: the vast amount of available sources, the increase of mobile and wireless clients and the need to support personalization and customization. New tools that can meet the presenting challenges must become available to users. ViSMA is one such system that provides the functionalities of defining, materializing and maintaining database views by taking advantage of mobile agent technology with additional features to support mobile and wireless clients. The role of mobile agents in ViSMA is twofold: Firstly, views are carried within mobile agents called **View Agents** [8] and may relocate themselves to reduce the distance from users that frequently request them. Secondly, they are used for migrating to a remote datasource and locally execute update propagation and query materialization operations, relieving remote clients and local database from performing this task while saving precious network resources.

[1] This work was partially funded by the Information Society Technologies program of the European Commission, Future and Emerging Technologies under the IST-2001- 39045 SeLeNe project and the IST-2001- 32645 DBGlobe project

The ViSMA system is fully implemented in Java, which is suitable for the development of mobile-agent based applications mainly due to its platform independence. We have also used Voyager ORB [6] to be the mobile agent platform. The Extensible and Flexible Library (XXL) [9] was used to instantiate the relational database functions executed by the agents. At the lower layer we utilized Tracker [2], an efficient location management system, to enable agent coordination and communication.

We have implement three alternative client components that aim at supporting clients of variant resource availability: the ViSMA Light Client Applet, the ViSMA Servlet Engine and the ViSMA Client Agent.

2. Architecture

In order to allow the creation and maintenance of distributed views we have designed a multi-tier agent-based architecture. The architectural components are distinguished based on their placement and functionality (Fig. 1) to: **Server Side**, **Database Side** and **Client Side** Components.

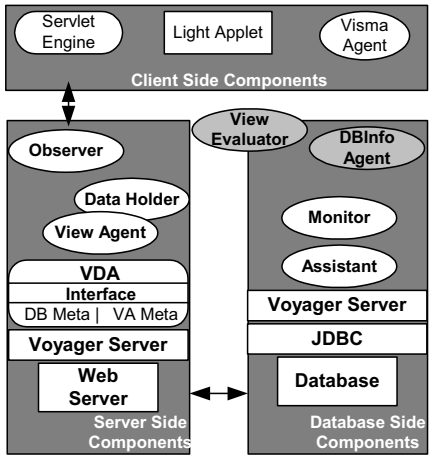

Figure 1: ViSMA Architecture

We also distinguish the agents described below into either *mobile* or *stationary*. Mobile agents may move from one network node to another in order to complete a predefined task while maintaining their state regarding both data as well as internal variables. Stationary agents migrate once to a specific node, remain ("park") at that location and periodically execute a task.

2.1. Server Side Components

The ViSMA server can be better characterized as a *View Dictionary*, which keeps meta-data on participating datasources and created views. With respect to views, metadata include the current location of the View Agent (VA) carrying the view as well as the view definition.

The lower levels of the View Dictionary (or ViSMA Server) include the **Voyager Server** (i.e. the mobile agent platform) and a **Web Server** that is used primarily for remote class loading by agents. It is also used for downloading the ViSMA applets, which are user side components, also discussed below.

The basic agent of the ViSMA Server is the **View Dictionary Agent (VDA)**. The VDA is the central communicational and coordinating agent, which is contacted by users, or agents receiving requests via the clearly defined ViSMA user-*access interface.*

The **View Agent (VA)** is another key agent of the ViSMA architecture. A VA is a mobile agent that is created once a user defines a view to be materialized and it is initialized by the VDA. A VA is responsible for creating, materializing and maintaining a view. During this process the VA initializes and dispatches a number of other supporting agents whose functionality is discussed in the subsequent paragraphs of this section. A fundamental VA feature is that its data can be shared among all ViSMA clients. Also, the VA it allows for its view to be queried by external entities. Therefore, sub-views can be extracted from a VA view. Finally, since a VA is a mobile agent this data can be carried with it as it moves. Therefore view migration is achieved.

A VA consists of a hierarchy of **Data Holders (DHs).** Each Data Holder Agent in the hierarchy is responsible for handling an individual *view fragment*. A DH will receive the definition of a view fragment and will become responsible to materialize and maintain it. The VA is then responsible to combine the fragments and produce the final view. Effectively, *DHs form the structure of the materialized view*.

Finally, **Observer Agent (OA)** is a mobile agent that is used for creating personalized, non-shareable sub-views by directly issuing user queries to a VA. The OA may communicate directly with its client and may move as its client moves.

2.2. Database Side Components

At each database site the **Mobile Agent Platform** (Voyager) must be installed in order to allow agent execution. This is actually the sole system requirement since all remaining functionality is dynamically deployed.

The **Assistant Agent** is a stationary agent that maintains a pool of connections to the database to serve visiting agents [7]. This agent provides transparent connectivity between the databases and the ViSMA agents. Changes to the database connectivity settings need only made known to the Assistant.

The **DB Info Agent** migrates to the remote database site, collects its metadata (schema, types), sends them to the VDA and finally self-destructs.

The **Monitor Agent** is another stationary agent sent by some DH to enable view maintenance. A Monitor agent "parks" at the remote site and periodically re-queries the database and sends changes to the DH. Thus DH may receive simultaneous updates from multiple Monitors.

Another basic functional agent is the **View Evaluator Agent (VEA)**. A VEA is a mobile agent that is sent by a DH to travel from one datasource to another and collect and combine data retrieved thus materializing a fragment of the final view (namely a *simple view*, see §3.2). Access to the databases is provided by each local Assistant. The VEA needs not be destroyed upon completion. On the contrary, it can be reused to re-execute the materialization plan, whenever necessary.

2.3. Client Side Components

A client component can be any entity that can contact the VDA directly or indirectly. We have implemented three alternative types or base clients, each of which can support different types of users. The **ViSMA Client Agent** provides each user with a private mobile agent that can interact in a P2P manner with other agents, without any VDA intercession. The second client type is **Applet-based** and enables any user with any Java supporting browser to download a *light* applet GUI to interact with the View Dictionary. Finally, for users with no Java support, the **ViSMA Servlet Engine** is provided. The Servlet engine is a middleware component that accepts standard HTTP requests through and replies in standard HTML making it highly appropriate for relatively low resource mobile devices such as PDAs and Smart Phones.

3. System Services

Throughout the rest of this section we will illustrate the functionality of the system using a concrete example. Consider the following distinct databases in a medical information system. For simplicity, and to avoid going into extreme implementation detail, we assume that each of the following databases is represented as a single table (Fig. 2).

Pa_Hospital

PatientID	Pressure	Temperature	Pulse

Ni_Hospital

PatientID	Condition	Respiration	Pressure

DoctorDB

PatientID	LastVisit	Diagnosis	Medication

PharmacyDB

Medication	Price	Available

Figure 2: Example Databases Schemas

A doctor visits two separate hospitals for patient treatments, one in Paphos ("Pa_Hospital") and one in Nicosia, Cyprus ("Ni_Hospital"). The hospital databases are updated several times a day regarding the condition of patients. The doctor also maintains her own database ("DoctorDB") to trace treatments related to her personal patients. "DoctorDB" database is also located in Nicosia but managed by different database management system. The PharmacyDB in Pittsburgh, USA keeps data on types of medication.

3.1. Datasource Registration

In order for a datasource such as "Pa_Hospital", "Ni_Hospital", "DoctorDB" and "PharmacyDB" to be available for access, the administrator must register its location to the ViSMA system. The administrator is simply required to download the database definition applet (or access the corresponding Servlet page if Java is not supported) and provide the database's location. Also he/she must provide valid access codes. Registration process is dynamic: Provided this information, the VDA dispatches an Assistant Agent, which migrates and connects to the database. The VDA also sends a DB Info Agent to collect the Database meta-data. This strategy allows for asynchronous operation and lets the VDA handle other incoming requests.

3.2. View Definition

During this process the user select the databases, tables and attributes to retrieve as well as sets the restrictions on each attribute. The user can also define which relation attributes are most important to her and select to specifically monitor those attributes for changes. View definition is assisted by retrieving the selected databases' metadata from the VDA locally to each client side component.

We deal with view definitions consisting of project, select and join operators. *Project-Select-Join (PSJ)* views are most common and, to a large extend, cover the requirements of the problem investigated [11]. We, thus, categorize views into either *simple* or *complex*. A simple view can be broken down to a succession of project-select-join operations and can be materialized by a single View Evaluator Agents (VEA). Complex views consist of at least two simple views combined with some operator such as "Union" or "Minus". Views can also be defined to extract data from a single database *(single views)* or be created over multiple databases *(multi-views)*.

Suppose that the doctor in our example has to leave for a conference in Athens but needs to keep track of her patients while away. She selects to define a view (e.g. using the Applet-Based Client from her laptop) before leaving. This *complex, multi-view* expressed in an SQL-like language can be:

```
DEFINE SHAREABLE ' Doctor-View ' AS
SELECT   Ni_Hospital.PatientID, Ni_Hospital.Pressure,
         DoctorDB.Diagnosis, PharamacyDB.Available
FROM     Ni_Hospital, DoctorDB, PharamacyDB
WHERE    DoctorDB.LastVisit > 1/1/2003
UNION
SELECT   Pa_Hospital.PatientID, Pa_Hospital.Pressure,
         DoctorDB.Diagnosis, PharamacyDB.Available
FROM     Pa_Hospital, DoctorDB, PharamacyDB
WHERE    DoctorDB.LastVisit > 1/1/2003
MONITOR Ni_Hospital.Temperature,
         Pa_Hospital.Pressure Every 5 minutes
```

Note the **MONITOR** line that defines the set of attributes to be monitored as well as the time interval between re-querying the sources for changes on these attributes.

3.3. Materialization Scheme

As soon as a view is defined, it is sent to the VDA. Before it reaches the VDA, the view definition needs to be pre-processed into a number of simple view fragments, which also form the view definition structure. This functionality is rather simple and it is provided by the client side component. This is feasible since, as mentioned in §3.2, all required metadata is downloaded locally. This relieves the VDA from having to structure each view definition. In our previous example the two 'SELECT-FROM-WHERE' clauses before and after 'UNION' represent two simple view definitions connected with the 'UNION' operator.

The VDA creates a View Agent (VA) and passes it the structured view definition. For each simple view in the view definition, the VA will create a Data Holder (DH) to administer this view. Each DH will create a VEA and provide a materialization plan. This is achieved using a **Query Builder** that can be programmed to optimize a query plan given database-related information which can be obtained from the View Dictionary as described in §2.1. ViSMA Query Builder currently creates a plan that attempts to minimize the size of data moved from one node to another by only carrying data that is absolutely necessary to the materialization process.

The final DH structure is tree-like, combining at each level a number of DH data. As a more generic example, consider the following complex query in which more than just two simple views are combined (Q1, Q2 and Q3 are simple view definitions.)

Q = (Q1 UNION Q2) MINUS Q3

Each of the query definitions is passed to a DH. Figure 3 shows how the DHs are structured within a View Agent (VA). Discontinues lines imply flow of data whereas continues lines imply function executed between this DH data and the data of the related DH.

Each of the simple queries is materialized by a VEA that travels to each datasource retrieving the requested data that is sent to the DH at completion. Figure 4 shows how a simple query (such as Q1, Q2 or Q3), referencing three distinct databases, is materialized. In our example view definition, this plan reflects the materialization of any one of the two 'SELECT-FROM-WHERE' clauses. Eventually each DH will receive a materialized view fragment.

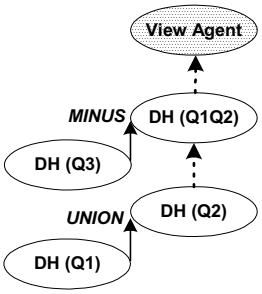

Figure 3: View Structure Based on Data Holders

Each DH will also send the necessary Monitors to any database that includes table attributes that the user selected for monitoring. For the example above, Monitor Agents need to be sent to 'Ni_Hospital' and 'Pa_Hospital', querying the 'Temperature' and 'Pressure' attributes respectively every 5 minutes as defined.

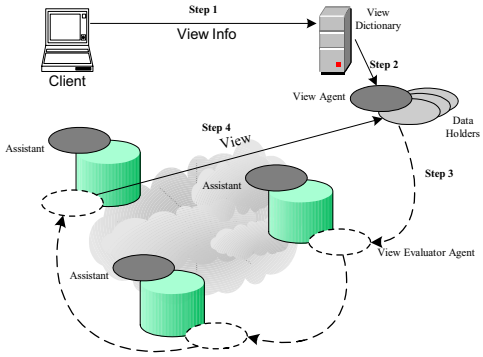

Figure 4: Simple View Materialization

An assumption is made here concerning the size of the materialized views: VAs are different from data warehouses in that they maintain limited size data from multiple sources. In a way, VAs form a distributed data warehouse that can relocate its pieces to better serve its clients. The size limitation is a logical assumption based on the fact that views created are mostly personalized and can be retrieved by many users who share the same interests. Currently we are working on view storing alternatives (such as using XML files and communicate them via standard HTTP) to enable larger size views to be stored and relocated dynamically.

3.4. Maintaining Views

As mentioned previously, Monitor Agents are used to provide updates for changes occurring at the datasources. Monitors reside at the database, connect and periodically re-query the database to retrieves the values of the attributes they monitor. A comparison is made between the last and the current results and the view difference (Δ View) is created. Only changes are sent to the DH, which can select to delete removed rows or resend a View Evaluator to rematerialize the view. Note that a DH may receive updates from Monitors located at different data sources (Fig. 5).

The algorithm used for this implementation is recomputational. In this scheme, we extend and adapt the Strobe Algorithm [11] to suit the needs of a mobile environment and mobile agent technology per se.

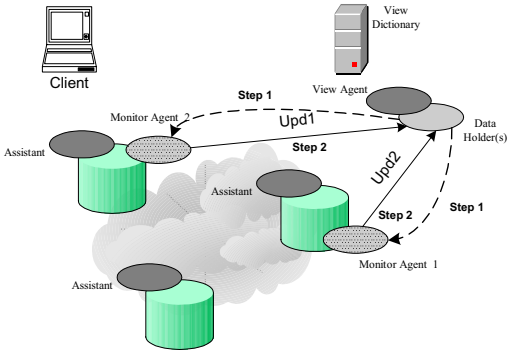

Figure 5: Datasource Monitoring

3. 5. View Data Retrieval

3.5.1. Shareable Views. Views created can be shared among all users of the system. A user may select to get any specific view's latest data by requesting it from the corresponding VA. In our example, the head-doctor may want to check on the status of the patients of the departing doctor. Retrieving a VAs data is done by requesting it either directly (ViSMA Agent client) or indirectly through the VDA (e.g. when using the Applet Client or

the Servlet Engine.) The result returned is presented to the user again depending on the client type that was used for the request. For example, the reply to a Servlet Engine request is in pure HTML.

3.5.2. Personalized View Creation. Users may be interested in a specific subset of some view. In this case, the system allows the creation of a personalized subview, which cannot be shared. In our previous example suppose that the doctor's assistant wants to define a view on the patients, whose pressure rises over 120/80, viewing the results from his PDA. He will use the Servlet Engine client to define the sub-view:

DEFINE PERSONALISED 'Assistant-View' **AS**
SELECT PatientID, Pressure
FROM View 'Doctor-View'
WHERE Pressure > 120/80

The query is sent to the VDA (Fig. 6), which creates an Observer that may move close to the VA to initially query it but will typically follow its creator to reduce delays and remotely re-query the VA for updates.

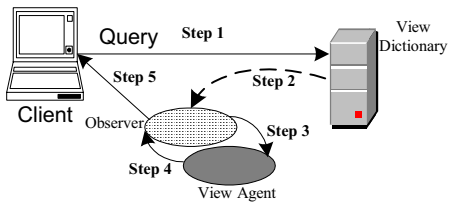

Figure 6: Personalized View Creation

3.6. View Mobility

Since a VA is mobile it can select to move closer to the group of users who use it most frequently. In our example, suppose that the doctor has connected again upon arrival to the conference. The VA either by detecting the new location or by noticing increased delays moves from Nicosia to Athens, at the local network (or the doctor's local machine), making request noticeably faster and reducing network traffic.

Within the ViSMA system we have incorporated a middleware subsystem called *Tracker* [2] that is used for efficient agent location tracking and for assessing the movement decisions.

3.7. View Deletion

When a user decides to delete a view she created, she sends a message to the VDA.

The VDA will notify the corresponding VA, which will hierarchically notify its DHs (Fig 7). Users may select to keep the last version within the Observer, however, generally views are considered invalid (outdated) if not deleted or updated for a specified period of time which is customizable and usually dependent on the type of the application for which the views are used.

4. Additional Features and Previous Work

Previous systems DVS [1] and VG [3] provide only basic ideas and have limited functionality. ViSMA is a full-fledged system that provides complete functionality by covering extensively all stages of the view manipulation processes. In addition, it follows a new materialization and maintenance philosophy based on the Data Holders as described in the system services section. Data Holders, being mobile themselves allow for the View Agent to relocate any one of the view fragments independently.

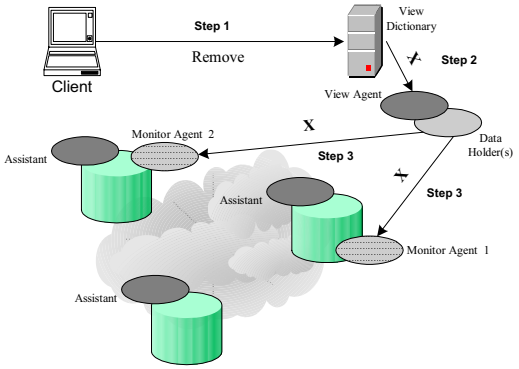

Figure 7: View Deletion

Another new feature is system extendibility, which can be visualized in two directions: Client extendibility and Functionality extendibility. Client extendibility can be achieved since ViSMA architecture allows additional types of users to have access to the View Dictionary via the request interface. The user side components need only decide how the data received will be handled (e.g., filtering the result to WML for a WAP-enabled device). Functional extendibility is provided for developers who may extend current agents to execute specific, system-defined functions viewing ViSMA as an Application Programmer Interface (API). For example developers may supply the Query Builder with additional query plan optimizations or recode the Monitor's update propagation method. Additionally we envision a *Multi-ViSMA* system, which may consist of collection of distributed View Dictionaries, each servicing a set of clients and databases. ViSMA Dictionaries may create ad-hoc networks and communicate in a P2P manner thus interconnecting different organizations.

View mobility is enhanced: Observers can monitor the movement of VAs and clients using Tracker and regulate their own location to minimize communicational costs.

We also incorporate ideas from [8] to further support view customization: During view definition, the user may explicitly select which columns to monitor for changes and at what re-querying frequency

Finally, VAs support querying over their data thus allowing for personalized sub-view creation and update transparently with respect to the original data sources. We also note that, with this system, non Java-enabled clients are supported while providing a user-friendly wizard-like GUI to assist view and sub-view definition which also allows for individual attribute monitoring settings.

5. Conclusion and Future Work

Recently, data access and retrieval have presented a number of challenges mainly due to the vastness of available data, the particularities of mobile and wireless computing and the need for personalization. In this work we present ViSMA, a functional tool that aims at assisting the dynamic definition, materialization and maintenance of web views. The primary design goal is to support mobile and wireless clients and be adaptable to the various types of user needs depending on the device capabilities and connectivity conditions. We propose a mobile-agent based architecture on which the system is build and describe how mobile agents can be dynamically deployed in providing a number of view manipulation services which are enhanced by view mobility as well as structured storage of views based on the Data Holder concept. We also highlight the fact that the system can be extended to accept new types of clients and may adjust its agent's functions to implement designer specific strategies by using PDAs and laptops.

Our first milestone in the development of ViSMA was to produce a functional prototype. As future work we intend to conduct an evaluation of the performance trade-offs concerning update efficiency versus query and retrieval of view fragments held by multiple DHs. Then we will work on enhancing the materialization plan at the Query Builders as well as improve our update propagation procedure with an incremental updating strategy. Finally we shall deal with system scalability issues with respect to the type of client and the type of query, which are a function of the view storage scheme and the Mobile Agent platform capabilities.

7. References

[1] C. Spyrou, G. Samaras, E. Pitoura, S. Papastavrou, and P.K. Chrysanthis: The Dynamic View System (DVS): Mobile Agents to Support Web Views. The 17th Int'l Conf. on Data Engineering, Mar. 2001.

[2] G. Samaras, C. Spyrou, E. Pitoura, M. Dikaiakos: Tracker: A Universal Location Management System for Mobile Agents, European Wireless, Feb 2002.

[3] G. Samaras, C. Spyrou, E. Pitoura: View Generator (VG): A Mobile Agent Based System for the Creation and Maintenance of Web Views. 7th IEEE Symp. on Computers and Communications, 2002.

[4] N. Roussopoulos Materialized Views and Data Warehouses: The 4th KRDN Workshop Athens, Greece, Aug.1997.

[5] O. Wolfson, P. Sistla, S. Dao, K. Narayanan, R. Raj: View Maintenance in Mobile Computing, 1995.

[6] Recursion Software Inc, Voyager ORB. At www.recursionsw.com/products/voyager/

[7] S. Papastavrou, G. Samaras and E. Pitoura: Mobile Agents for World Wide Web Distributed Database Access, IEEE TKDE, 2000.

[8] S. Weissman Lauzac, P. K. Chrysanthis. Personalized Information Gathering for Mobile Database Clients: The ACM Symp. on Applied Computing. Feb 2002.

[9] XXL (eXtensible and fleXible Library), Database Research Group, Univ. of Marburg, Germany. At www.dbs.mathematik.uni marburg.de/research/projects/xxl

[10] Y. Zhuge, H. Garcia-Molina, J. Hammer and J. Widom. View Maintenance in a Warehousing Environment. In *the* ACM SIGMOD Conf., 1995.

[11] Y. Zhuge, H. Garcia-Molina, J.L. Wiener: The Strobe Algorithms for Multi-Source Warehouse Consistency. The 4th International Conference on Parallel and Distributed Information Systems, 1996.