View Generator (VG): A Mobile Agent Based System for the Creation and Maintenance of Web Views*

George Samaras
Dept. of Computer Science
Univ. of Cyprus
CY-1678 Nicosia, Cyprus
cssamara@ucy.ac.cy

Constantinos Spyrou
Dept. of Computer Science
Univ. of Cyprus
CY-1678 Nicosia, Cyprus
cspgcs1@ucy.ac.cy

Evaggelia Pitoura
Dept. of Computer Science
Univ. of Ioannina
GR 45110, Ioannina, Greece
pitoura@cs.uoi.gr

Abstract

The View Generator (VG) is a system that provides the necessary components for the definition, materialization, storage, maintenance and re-use of views over remote web-accessible databases. Through the system, clients identify web databases of interest, access their metadata and create personalized views that can be shared by other clients. Creating personalized views provides a more efficient way of data processing than directly issuing complex queries to the data sources especially in the case of light-weight and wireless clients that suffer from scarce local resources, limited bandwidth and high communication costs. VG's multi-tier architecture implemented using mobile Java agents allows automatic code deployment as well as dynamic relocation of views.

1. Introduction

The growing use of the Internet and the web provides users with large amounts of data, creating an increasing need for tools that will assist users in searching for and efficiently accessing information. Electronic commerce applications and the vast amount of business databases make this need even more important. Large and complex queries applied directly to the source databases are often quite expensive. Such queries are even more inefficient for wireless clients due to the severe limitations of the wireless links that are in general characterized by high communication costs, limited bandwidth and high latency [6]. The efficient creation and materialization of personalized views provides a scalable alternative.

In this paper, we present the View Generator (VG), a system based on mobile agents that allows the definition, creation and maintenance of materialized views over distributed databases. VG attempts to bring data as close as possible to the clients thus reducing the overhead of transmitting data over slow networks. In addition, it allows the sharing of views among the clients of the system as well as the creation of non-sharable personalized sub-views. In VG, access to remote materialized views, multiple level views and databases can be either synchronous or asynchronous [1]. With asynchronous communication, clients can pose queries, disconnect (voluntarily or not) and later re-connect to receive the results. Analogously, during peak periods, busy servers can postpone the processing of some asynchronous queries.

VG's multi-tier architecture consists of mobile agents, stationary agents and applets [3, 9]. It supports dynamic code deployment, thus minimizing the need for a-priori setting-up and installing client software components. Furthermore, the fact that all system components are extensions of mobile agents allows the system to efficiently adapt to the mobility patterns of its clients. We have implemented a location management system for mobile agents (called TRACKER) [11, 13], which, operating in synergy with VG, allows the efficient identification and location of clients and their views.

The remainder of this paper is organized as follows. Section 2 introduces the View Generator (VG) system, its architecture and components. Section 3 described the VG functionality and the supported operations. Section 4 presents view maintenance within the VG system, while Section 5 focuses on location management. Section 6 presents future work and Section 7 concludes the paper.

2. The View Generator Architecture

The View Generator (VG) system is based on mobile agents [3] and provides an infrastructure for the dynamic deployment of views over remote databases. It can be viewed as a multilevel dictionary that maintains information about the remote databases to allow clients to define and materialize such views. It also maintains information regarding previously materialized views, so

* This work has been partially funded by Cyprus Research Promotion Foundation and the IST project DBGlobe, IST-2001-32645.

that such views can be subsequently re-used or shared. Materialized views can be *mobile*, i.e., dynamically relocated following their clients and staying as close as possible to them. The VG system is implemented using the Java-based Voyager mobile agent platform [4].

The VG system supports the following operations: the identification and registration (to VG) of a web-accessible database, the de-registration of a database, the creation of a view over a single database as well as over multiple distributed databases, the re-use of an existing view, querying a view, the creation of personalized sub-views and the deletion of a view or sub-view. Beside these operations, the View Generator provides a number of support tools (wizards) to aid the definition, materialization and location of the various views or sub-views.

View Controller Agent while the *DB server* sub-system consists of the View Agent, Multi View Agent, Monitor Agent, DB Info Agent and Assistant Agent.

2.2 Client Side Agents of the VG System

The client side agents/applets are the components of the View Generator system that provide a simple and friendly Graphical User Interface (GUI) to the various tools of the system. The GUI consists of several wizards that help the clients in providing the system with the information necessary for the definition and execution of its various operations. The client side agents are also responsible for any kind of communication with the server side agents (i.e., the view controller or view agents). In particular, the View Client Applet, the View

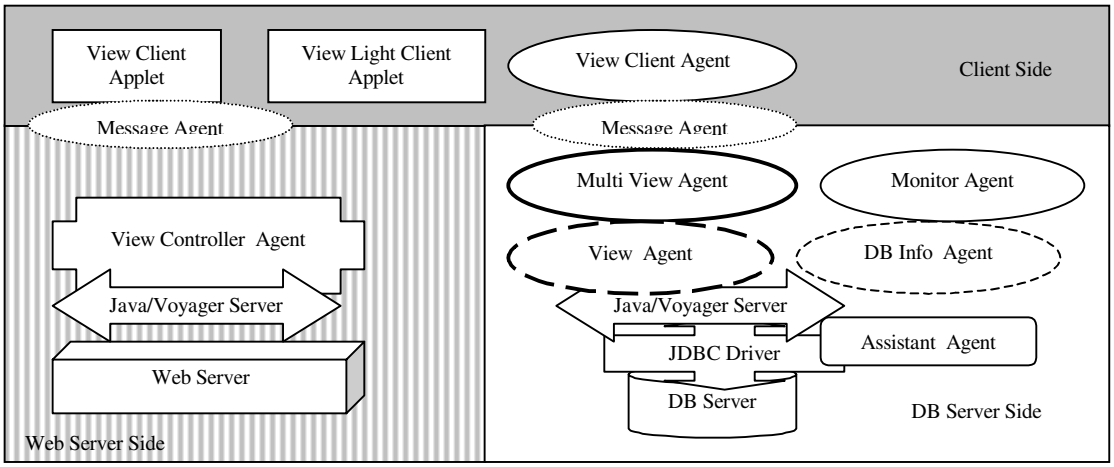

Figure 1: The View Generator architecture.

2.1 The View Generator Components

The View Generator system is composed of three types of objects: *applets, mobile agents* and *stationary mobile agents*. Although all agents are mobile, we call stationary the agents that, once sent to a specific location, they remain there until asked to be disposed. The stationary mobile agents constitute the fixed part of the architecture, while the mobile agents comprise the dynamic part of the architecture. Mobile agents are dynamically created and cooperate with each other and the stationary agents in creating, maintaining, and querying the views.

The applets and agents are distributed into the client and the server side parts of the VG system (Figure 1). The *client side* includes the following components**:** Message Agent, View Client Applet, View Light Client Applet and View Client Agent. The *server side* is composed of two sub-systems, the web server and the database (DB) server sub-systems. The *web server* sub-system includes the

Light Client Applet and the View Client Agent provide the interface to the VG operations. The Message Agent is responsible for the communication with the server side components.

The *View Client Applet* is downloadable from the VG web server. It provides the users with a simple and friendly GUI for performing VG operations. The *View Client Agent* resides at the client side and has the same functionality as the View Client Applet. This agent provides the GUI for the stand-alone application approach of the VG system. The stand-alone approach allows the client to access the various agents directly and not, as with applets, via a web server. With the use of this agent, the clients have also the ability, while on the move, to have their views relocated close to them (view relocation is described in more detail in Section 5).

For communicating with the server side agents, the view client components create a *Message Agent*. This agent is dispatched from the client side to the server side carrying the client request and data and returns back to the

client with the VG server's response [5]. The use of the Message Agent provides the client with the ability of asynchronous communication and disconnected operation. To this end, the client may assign to the Message Agent a specific operation and send it to the VG system. The client may then disconnect, voluntarily or not. Later, when the client reconnects, the Message Agent returns to it with the results of the operation.

Lastly, the *View Light Client Applet* is a light-weight version of the View Client Applet. It is a fast-downloadable applet for wireless clients that must fit into a small memory footprint. For communicating with the server agent, instead of using Message Agents, the View Light Client Applet uses direct messages.

2.3 Server Side Agents of the VG System

The server side agents are the components of the VG system that are responsible for performing its various operations.

2.3.1 Web Server Side Agents. The *View Controller Agent* (VCA) is the central agent of the system. It is a stationary agent. First, it is responsible for keeping track of the location of the various database servers, and for maintaining a snapshot of their metadata. Second, it keeps track of (a) all the materialized views (that is the location of the mobile agents that maintain each materialized view), and (b) the number of clients that are currently using a specific materialized view (for load balancing purposes). Finally, the VCA acts as a service point (an access point) for clients who wish to query an existing materialized view or create a new one.

Due to the applet security limitations, an applet can communicate directly only with objects running on the same web server. Thus, the VCA needs to be located at the web server.

2.3.2 DB Server Side Agents. This part of the VG system consists of two types of agents: agents that are holding views and agents that are providing database related information to other agents.

Agents Acting as View Holders. These agents include the View and Multi View Agents and the Monitor Agent. The *View Controller Agent creates the View Agent on demand*. View Agent is responsible for creating views on remote databases, executing queries on existing views, and maintaining materialized views. If the view is created over multiple databases the View Agent is called a *Multi View Agent*. A *Monitor Agent* is created when a client wants to be immediately informed about changes on the values of specific data items of a view. This agent cooperates with the corresponding View or Multi View Agent, and when an update occurs on the specified items, it informs the client. A Monitor Agent is a personalized sub-view. This view is a view of a view and is not sharable.

Database Related Agents. An *Assistant_Agent* is installed at each database system that participates in the VG system [5, 8]. This agent is initially created by the VCA and is sent to the site of the database server. Its task is to continuously provide incoming mobile agents with information related to the database so that they can connect to it and perform database queries. Such information may include an appropriate JDBC driver [10], a reference data source name for the database, and authentication keys (login and password). A *DBInfo Agent* is created on demand by the VCA and its task is to retrieve metadata from remote databases. It achieves this by dispatching to the remote database and then querying, with the aid of the Assistant Agent, its catalog.

2.4 Load Balancing.

VG synchronizes and coordinates the creation, update and use of each view. Since multiple clients may share a view, in the event that the corresponding View Agent becomes heavily loaded or need to move close to more than one client, the VG may clone (i.e., create a copy of) the View Agent. Thus, there may be more that one View Agents per view. Furthermore, there may be replicas of the view, in which case, the View Agent is responsible to choose the most appropriate replica for a given request, thus balancing the load on the various replicas of the view

3. VG Client Functionality

A VG client connects to the system through three different operational types. Once connected to the system, a VG client can add a new database to the system or delete an existing one. A VG client can also create new views, query existing views or delete views. This section describes client operation in detail.

3.1 Connection and Operation Types

We distinguish the clients of the VG system into two categories: *powerful clients* with no resource or connectivity limitations and *wireless clients* with limited resources and connectivity restrictions.

Type 1: *Connection and Operation through the View Client Applet*

This operational type is more appropriate for powerful clients that do not move often. To connect to the VG system, the client contacts the VG web server and downloads the View Client Applet. Then, the client can pose requests through the GUI of the View Client Applet. This type of operation utilizes the Message Agent to support asynchronous and/or disconnected operation. In particular, when the client initiates the first request, the

View Client Applet created the Message Agent to be used with the VCA (Figure 2: Step A). Each time the client poses a request; the Message Agent asynchronously carries it to the VCA (Figure 2: Steps B and C). Similarly the VCA delivers the result to the Message Agent (Figure 2: Step D), that in return carries it back to the client (Figure 2: Step E).

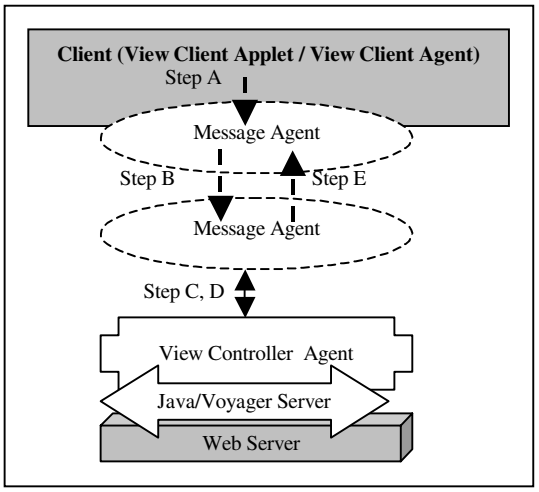

Figure 2: A client of type 1 or 2 communicates with the View Controller Agent.

Type 2: *Connection and Operation through the View Client Agent*

This operational type is more appropriate for powerful or wireless clients that move frequently. In this case, the client deploys the stand-alone application approach to avoid the applet's security restrictions (i.e., having all requests communicated only to the web server). The View Client Agent is downloaded to the client (either dynamically or it is pre-installed). When the client is connected to the VG, it can pose requests through the GUI of the View Client Agent. This type of operation also utilizes the Message Agent to support asynchronous and/or disconnected operation (Figure 2). Moreover when the client moves to another location, the client's View Agents are transparently moved as close as possible to the new location.

Type 3: *Connection and Operation through the View Light Client Agent*

This operational type is more appropriate for wireless clients that do not move frequently. In this case, the client downloads the View Light Client Applet from the web server. Once connected, the client can pose request through the View Light Client Applet. The View Light Client Applet communicates with the VCA with direct messages. This operational type provides only synchronous operation.

3.2 Database Related Operations

A VG client can add or remove a database.

Adding a Database. To add a new database to the VG system, the client must specify, by using the appropriate wizard of the VG GUI, the following information:

- The location of the database,
- The database server's JDBC driver,
- The data source name assigned to the database,
- The login and password, if requested,
- A unique name that is going to be used as an identifier of the database in the VG system.

The VG GUI communicates this information to the VCA through messages or Messenger Agents depending on its operational type (Figure 3: Step A). The VCA creates and sends an Assistant Agent to the database server, which resides there (Figure 3: Step B). Next, it creates and sends to the database server a DB Info Agent (Figure 3: Step C). The DB Info Agent, in cooperation with the Assistant Agent, collects the database metadata (Figure 3: Step D) and sends them to the VCA (Figure 3: Step E). It then disposes itself.

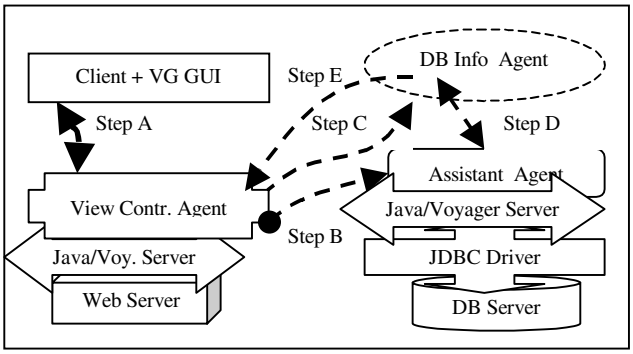

Figure 3: Adding a new database to the VG system.

Removing a Database. To remove a database from the VG system, the client communicates with the VCA through the appropriate wizard. The VCA informs the appropriate Assistant Agent, located at the database server, to dispose itself.

3.3 View Related Operations

Operations regarding views include creating new views, querying views and deleting views.

3.3.1 View Creation

A view is defined by a query over one or more relations, called base relations, of one or more databases. Views in the VG systems may or may not be materialized [7]. In the case of materialized views, the result of the

query that defines the view is stored in a View Agent. In the case of a non-materialized view, the View agent stores just the query that defines the view. To create a new view, a VG client, via VCA, searches for the appropriate base relations. (Figure 4: Step A and Figure 5: Step A).

Creating a Single View. To create a single view (a view for which all base relation belong to the same database), the client must specify, by using the appropriate wizard of the VG's GUI, the following information:

- The location of the database,
- A name and a textual description for the new view,
- The view update mode that specifies how the View Agent is going to maintain and keep the view's data up to date (see 3.3), and
- The query that defines the view.

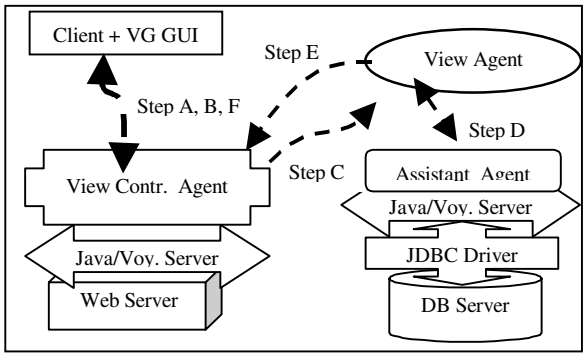

Figure 4: Creating a single view.

Then, the VG GUI communicates the request to the VCA (Figure 4: Step B). The VCA creates and sends a View Agent to the Database Server, which remains there (Figure 4: Step C). The View Agent, in cooperation with the local Assistant Agent, queries the database, retrieves the view's data (Figure 4: Step D) and sends them to the VCA (Figure 4: Step E). Next, the VCA sends the view's data to the client (Figure 4: Step F).

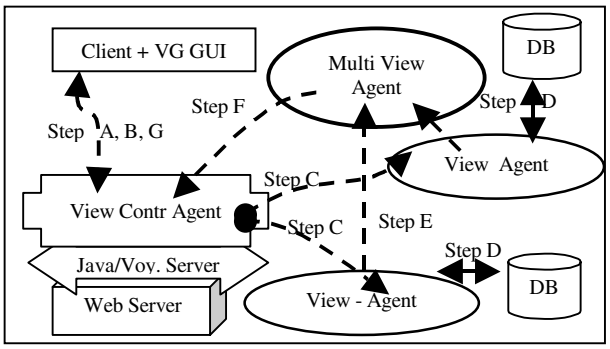

Figure 5: Creating a distributed view.

Creating a Distributed View. To create a distributed view, i.e., a view over relations of more than one database, first, as in the single view case, the client specifies all the necessary information for defining the view. Then, the VG GUI communicates with the VCA which in turn creates and sends View Agents to each database server and a Multi View Agent to one of them; all agents remain there (Figure 5: Step C). The View Agents, in cooperation with the local Assistant Agents, query the database, retrieve the appropriate data (Figure 5: Step D) and send them to the Multi View Agent (Figure 5: Step E) which than joins the results (Figure 5: Step F). Finally, the VCA sends the results to the client (Figure 5: Step G).

3.3.2 Querying Views. A VG client can query or monitor an existing view.

Querying a Single or a Distributed View. A client first specifies the search criteria, through a wizard, and the VCA determines the appropriate view. Then, it communicates with the corresponding View Agent (or Multi View Agent) that executes the query and sends the result to the VCA, which in turn passes it to the client.

Monitoring a Single or a Distributed View. VG clients may specify that they want to be notified about specific view updates. To this end, the VG system creates a Monitor Agent and places it at the same site with the corresponding View or Multi View Agent. (Figure 6). This agent monitors the view and when updates occur inform the client and supply him with the updated data. In some sense one can view the monitor agent as private sub-view over an existing view.

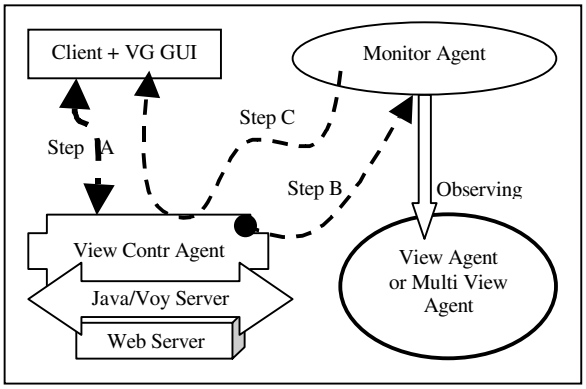

Figure 6: Monitoring a single or a distributed view.

3.3.3 Deleting Views. The procedure of deleting an existing single or distributed view is done by simply disposing the corresponding View or Multi View Agent

4. View Maintenance

In the case of updates at the base relations, the data at a view may become obsolete. Regarding updating the view,

the VG system supports two updates modes: periodic re-evaluation, and triggering. With the *periodic re-evaluation mode*, the View Agent (or Multi View Agent) re-evaluates the query that defines the view periodically. With the *triggering mode*, triggers are defined at the base relations that participate in the definition of the view. The trigger describes for which data updates, the View Agent must be notified.

The view update mechanism of VG is extensible. View Agents (or Multi View Agents) are capable of adding additional view update mechanisms dynamically. This is achieved through the notion of Task Handlers [2]. A *TaskHandler* is a Java object that implements a particular task, i.e., a specific view update operation. A TaskHandler library is a collection of such objects that are serializable and can travel along with the mobile agents (View Agents or Multi View Agents). Each View Agent (or Multi View Agent) can load one or more TaskHandlers and at run time choose which one to use.

5. Location Management

VG allows both View and Monitor Agents to move following their clients, in the case of a type 2 operational mode. A dynamic location management subsystem (TRACKER) is developed to aid the automatic acquisition of agent locations [11, 13]. The TRACKER system is a hierarchical and distributed location management middleware [12] which has the ability to manage the location of mobile agents that travel autonomously in the Internet. Being MASIF compliant and based on pure JAVA, TRACKER is independent of the semantics of any particular Java based mobile agent platform. A major advantage of this middleware is the ability: (a) to manage the location of any mobile agent and (b) to allow agents to locate other agents, independently of their native execution environment or implementation platform. TRACKER is quite generic, that is, able to dynamically incorporate various location mechanisms.

TRACKER consists of two agents, the TRACKERRegistry and the TRACKERAgent. There is one TRACKERRegistry at each node participating in the TRACKER system. TRACKERRegistries form a hierarchy. Each *TRACKERRegistry* is responsible for managing the location of all mobile agents residing locally at its node or in one of its children nodes. The *TRACKERAgent* is an abstract class that provides the operations related to location management (registration, deregistration to and from the TRACKER system). To support location management, all agents of the VG system are defined as subclasses of the TRACKERAgent abstract class. The VG system enhanced with the TRACKER location manager support load balancing: the (transparent from the clients) relocation of views close to the clients that are using them.

5.1 TRACKER Enabled VG Functionality

To use the TRACKER system, a TRACKERRegistry must be installed at each node participating in the VG system (Figure 7). The TRACKER hierarchy is created following the physical network topology. For instance in the hierarchy of Figure 13, nodes 2 and 3 are closer to node 1, nodes 4 and 5 are closer to node 2 and node 6 is closer to node 5. The View Controller Agent (VCA) is registered within the TRAcKER hierarchy. VCA registers to the local TRACKER Registry (TRACKER enabled node 2 in Figure 13). When a type 2 client (Client X) connects to the VG system, the View Client Agent registers to local TRACKERRegistry (TRACKER enabled node 6 in Figure 7).

The functionality of the VG system is now extended allowing the automatic relocation of views. In particular, the view related operations of the VG (creating, monitoring and querying a view) are now more flexible. For example in the case of the *creation* of a new view, after dispatching the View or Multi View Agents to the DB Servers and connecting to the local databases, the agents move as close as possible to the client (connected using the View Client Agent) that requests their data. In the case in which more than one client is using a specific view, the View Agent may clone itself. Similarly in the case of the *monitor* operation, the Monitor Agent moves close to the requesting client.

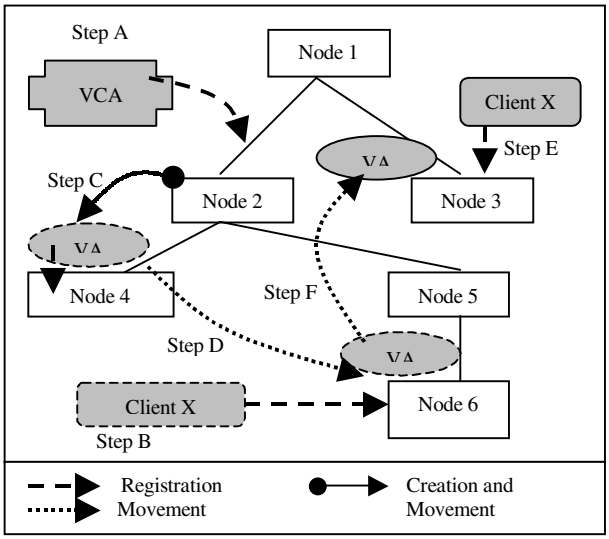

Figure 7: View and client location management example.

5.2 Example of TRACKER Enabled VG Operation

The TRACKER hierarchy is created following the physical network topology (Figure 7). The View

Controller Agent (VCA) is registered to the local TRACKER Registry, node 2 in our example (Figure 7: Step A). We assume that the VG Web Server resides at this node as well.

Client X connects to the VG System, using a View Client Agent. The View Client Agent registers to node 6 of the TRACKER System (Figure 7: Step B). Then, Client X performs a "create a single view" operation for which the base relations are on a database that is located at TRACKER enabled node 4. A View Agent (VA) is created by the VCA and is dispatched to this node (Figure 7: Step C). When the VA arrives and connects to the database, it registers to the local TRACKER Registry. The VA moves as close as possible to the client (to node 6), with the assistance of the TRACKER System (Figure 7: Step D). Assume that Client X disconnects and later re-connects to node 3 through a View Client Agent (Figure 7: Step E). Client X queries the view that the client has preciously created. VCA locates the VA for the view. Now, the VA may move closer to the client (to node 3), with the assistance of the TRACKER System (Figure 7: Step F).

6. Future Work

We are working on extending the VG system, with the ability to support clients without any Java support (e.g. handheld and palmtop PC with Windows CE operating system). To this end, we plan to add to the VG Web Server, a Servlet engine. This engine will work as a router between a client's non-Java enabled Web Browser and the View Controller Agent. The client will use an HTML forms as the VG GUI. Each time the client submits a request; this request will be transmitted to the VCA, through the Servlet engine, as an HTTP Request (text/HTML). The reply of the VCA will be transmitted back to the client, through the Servlet engine again as an HTTP Reply (text/HTML). This will provide an additional operational type (type 4) to the VG system.

7. Conclusions

The View Generator System provides the infrastructure for the dynamic creation, materialization, sharing and maintenance of views over web databases. The system is based on Java mobile agents, thus it takes advantage of the benefits provided by mobile agents including platform independence, mobility, autonomy, reactivity, communication and persistence. The system is appropriate for light-weight and wireless clients. It supports both synchronous and asynchronous communication, thus a client may pose a request, disconnect and connect later to receive the response. In addition, the system shares the workload between the clients to the servers, which are rich in recourses. Furthermore, the mobile agent oriented architecture, offers easy extension and improvement. Finally, the system can be easily deployed through the web.

8. References

[1] P. K. Chrysanthis, T. Znati, S. Banerjee and S.K. Chang. Establishing Virtual Enterprises by means of Mobile Agents. *Proc*. of the *10th IEEE Workshop on Research Issues in Data Engineering,* 1999.

[2] P. Evripidou, G. Samaras, E. Pitoura and P. Christoforos. The PacMan Metacomputer: Parallel Computing with Java Mobile Agents. *Fifth Generation Computer Systems special issue on JAVA in High Performance Computing,* 2000.

[3] C. G. Harrison, D. M. Chessm, and A. Kershenbaum. Mobile Agents: are they a good idea? Research Report, IBM Research Division, 1994.

[4] Objectspace Inc. The *Voyager* Mobile Agents Platform. Available at <http://www.objectspace.com/>

[5] S. Papastavrou, G. Samaras, and E. Pitoura. Mobile Agents for WWW Distributed Database Access. *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[6] E. Pitoura E., and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

[7] N. Roussopoulos, Materialized Views and Data Warehouses. *Proc*. of the *7th International Workshop on Knowledge Representation meets Databases,* 1997.

[8] G. Samaras, M. Dikaiakos, C. Spyrou and A. Liberdos. Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment .*The Joint Symposium.on Agent Systems and Applications and on Mobile Agents,* 1999.

[9] C. Spyrou, G. Samaras, E. Pitoura and E. Paraskevas. Mobile Agents for Wireless Computing: The Convergence of Wireless Computational Models with Mobile-Agent Technologies. (To appear in) *Journal of ACM/Baltzer Mobile Networking and Applications*, 2001.

[10] Sun Microsystems Inc. *JDBC drivers*. Available at <http://java.sun.com/products/jdbc/drivers.html>.

[11] C. Spyrou. "Creation of a Mobile Agents Location Management Mechanism", Master Thesis, Computer Science Department, University of Cyprus, June 2001.

[12] E. Pitoura, and Samaras, G., Locating Objects in Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering,* (TKDE) 2001.

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