Proceedings of the IASTED International Conference
COMMUNICATIONS, INTERNET, & INFORMATION TECHNOLOGY
November 18-20, 2002, St. Thomas, US Virgin Islands

MOBILE FLEET APPLICATION USING SOAP AND SYSTEM ON DEVICES (SYD) MIDDLEWARE TECHNOLOGIES[1]

S. K. Prasad[2], M. Weeks[2], Y. Zhang[2], A. Zelikovsky[2], S. Belkasim[2], R. Sunderraman[2], and V. Madisetti[3]
e-mail: mweeks@cs.gsu.edu
Computer Science Department
Georgia State University
Atlanta, Georgia 30303
United States

ABSTRACT

This paper presents a fleet management application with heterogeneity of devices and data, database synchronization, group transactions, peer-to-peer computing, and mobility support. We simulate a delivery service's fleet of trucks with PCs and hand-held devices, and describe how the system works together. To send messages between devices, we use the Simple Object Access Protocol (SOAP), allowing heterogeneous devices to communicate as peers. We examine current scenarios of our system, and discuss future enhancements. The ad-hoc nature of the solutions based on the existing technologies has led us to design a comprehensive middleware, namely, System on Devices (SyD), which we also describe briefly. SyD enables rapid development of collaborative applications, such as a fleet system, over heterogeneous, independent, data stores, devices, and wired and wireless networks, including those involving the web services.

KEY WORDS

Handheld Devices, Middleware, Embedded Software, Persistent Mobile Objects, System on Devices, SyD.

1. INTRODUCTION

Limitations of Current Technology: The current technology for development of a database application over a heterogeneous set of wired or wireless devices and networks has several limitations. Developing an application requires explicit and tedious programming on each kind of device, both for data access and for data communication. The application code is governed by the type of device, data format, and the network. The database server is typically a centralized logical entity providing only a fixed set of database services, with little flexibility for user-defined ad hoc services or the ability of user applications to dynamically configure a collection of independent data stores. Applications running across mobile devices are complex because of the lack of persistence of their data due to their weak connectivity.

Truck Fleet Application: In this system, a user may connect through the Internet to the Web and Data Center (WDC), request delivery of a package, and give information pertaining to the package, such as where the package can be picked up, its priority, and where it should be delivered. The WDC passes this on to a depot, which schedules a pick-up. A truck will arrive according to the schedule and take the package to its next stop. The trucks, depots and WDC form a heterogeneous network with different types of communications links between them. Each entity has its own database of varying capacity [1-4]. A truck's handheld device (HHD) holds the most current information about the packages, such as the package ID, its current location, and where it needs to go. A depot keeps similar, but more involved data, such as which truck has the package, the status of the truck (location, speed, heading), and where the package ultimately needs to be delivered.

We present the architecture of the fleet system as a hierarchy (Figure 1). The user connects to the Web and Data Center, which provides the user interface and serves as a repository for retired records. That is, once a package has been delivered, the trucking company will want to keep information about that package for a time, so the WDC acts as a backend database. The WDC connects to depots, and passes along information, such as the record for a package. Depots have wireless connections to their respective trucks, and the trucks can communicate with each other directly through their wireless devices.

Technical Merits: Following are the key technical aspects of our fleet management implementation: (i) Heterogeneous device, data format and language support (ii) Peer-to-Peer computing over dynamic groups (iii) Group transactions: Autonomous Database

[1] This research was funded by State of Georgia's Yamacraw Embedded Software research contract #BLA42 and #CLH49.
[2] Computer Science Department, Georgia State University
[3] Electrical and Computing Engineering, Georgia Institute of Technology

Synchronization based on triggering events; group querying, and (iv) Mobility support through proxies and directory service.

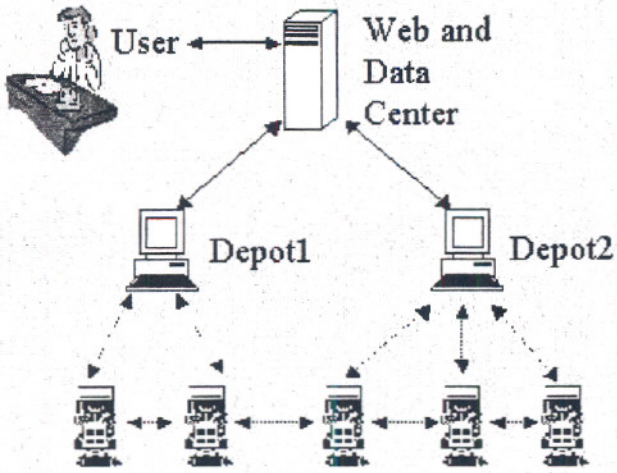

Figure 1. Communications in the fleet system

In this paper, we implement the fleet system by extending SOAP (Simple Object Access Protocol), a lightweight protocol that can be used to invoke a particular database procedure or query [5-6].

Currently, the database interactivity and collaboration can be achieved by rigorous application programming using existing middleware and database technologies [7]. One of the tasks of such an application is to interact with multi and mobile databases [8]. Due to the special conditions in the environment, many issues and problems need to be solved. For example, the networks may be temporarily disconnected and the communication bandwidth may be limited. Synchronization and update are also more difficult with mobile databases. There is a considerable effort that needs to be exerted to program the various databases and introduce the notion of collaboration and interactivity as outlined above.

Here, we identify several major technological issues that have not been addressed in the current existing systems.

1. Every station can act as a server and/or a client.

2. There is no single global schema. Each client has his own schema that is available to any other client. For security purposes, a client may selectively make portions of their schema available.

3. There is no concept of a single centralized or federated server. The transactions are with respect to an ad-hoc network that is created by establishing links among the clients as needed. These links may last only for the duration of a session or even a transaction.

4. Operations supporting cooperation among stations are provided, such as link operations.

5. The working environment is mobile and composed of weakly connected networks.

6. Applications can be shared by stations (downloaded, used, then deleted).

7. Identities can be established by users, giving as much (or as little) information as they desire.

We explain how these issues have been addressed in our fleet implementation by employing existing technologies. The ad-hoc nature of these solutions has led us to design a comprehensive middleware, namely, System on Devices (SyD). SyD enables rapid development of collaborative applications over heterogeneous data stores, devices, and wired and wireless networks, including those involving the web services.

This paper is organized as follows. Section 2 introduces the truck fleet application. Section 3 discusses its design and implementation. Section 4 briefly describes the System on Devices middleware architecture. Finally, Section 5 concludes the paper.

2. THE TRUCK FLEET APPLICATION

For our truck-fleet application, there are several scenarios that we examine in this paper. First, we present normal operations, which involves synchronization across autonomous databases. Next, we look at what happens if a traffic accident occurs. Following this is a description of the map utility, which demonstrates data collection from distributed sources.

2.1 Normal Operations

To start a delivery, a user would contact the Web and Data Center. The primary function for the WDC is to interface with the user, providing the shipping company's home-page, and allowing the user to order deliveries and track packages. Records for packages are stored here and "retired" after delivery. When a delivery is ordered, the WDC passes it on to a depot.

Every depot has a number of trucks associated with it. The depot's duties are 1) to store packages physically, as well as package information, until loaded onto trucks, 2) schedule the routes for the trucks to take, 3) serve as a broadcaster to trucks for traffic information, 4) provide detailed package information upon request, and 5) update WDC with all packages' information at regular intervals, such as at completion of a schedule.
The truck (with an HHD) picks up packages from a source and delivers them to a destination. However, a truck does not necessarily deliver the package directly; a package may have several stops before reaching its final destination. Since every truck is associated with a depot, its depot will make a schedule for it before the truck sets out.

A user may decide to check on his package. He would contact the WDC, and access information based upon the

package ID number. If the priority for the package is high enough, the WDC may contact the depot in charge of the package, which in turn contacts the truck. The truck responds back with the latest package information, as well as schedule information, such as the truck's current location, it's current (average) speed, the number of stops before the package is delivered, and any other information that could be used to predict the actual time of arrival.

2.2 The Accident Scenario – Peer-to-peer computing

Suppose one of the trucks unexpectedly becomes disabled, and needs help off-loading its cargo to another truck. This scenario demonstrates peer-to-peer computing in a mobile environment among a dynamically formed group of trucks. First, the disabled truck calls out to other trucks in the area to come help it. Then it decides which truck will help. In determining this, two parameters are used: *slack time*, the amount of free time in the called truck's current shift, and *emergency time*, the amount of time needed to finish the work left on the disabled truck.

The rules for action are as follows. The disabled truck collects information from all the other trucks, including slack time and position. Then it calculates the travel time for each of the other trucks to reach its current location. The disabled truck determines which trucks are available to provide help according to the condition: *(emergency time + travel time < slack time)*. If more than one truck is available, the one with the maximum *(slack time – travel time - emergency time)* will be chosen.

The disabled truck then informs the chosen truck. It will update its slack time and add the location of the accident to its schedule. Finally, it will add the disabled truck's delivery schedule to its own.

The technical implementation of these two scenarios is described in section 3 below. We are currently working on several enhancements, including optimization of the routing, updating the trucks' schedules due to traffic conditions, updating the trucks' HHDs remotely, and a truck rendezvous scenario. Currently, estimates of the delivery time are based on the location of a package and the distance between the source and the destination. More accurate delivery time estimations will depend on traffic updates, route selection and delivery sequence [9-13].

Map – Group Querying: The map is a tool that tracks (and displays) the location of all trucks in the fleet. Each depot maintains the position of each truck within its area and updates the corresponding coordinates (Figure 2).

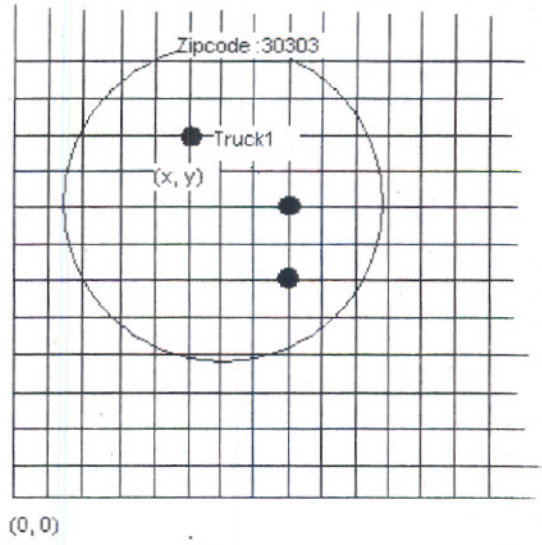

Figure 2. Map of the Truck Locations

Currently, the truck's location displayed on the map is the location at which it last delivered a package. Upon delivery of a package, the truck sends a message to the depot relaying the status of the package. The depot then uses this information to update the map.

The map is accessible through the WDC allowing customers to view the position of the delivery trucks within there area. The map also displays the direction in which each truck is heading, along with the estimated time to deliver the next package.

3 TRUCK-FLEET IMPLEMENTATION IN SOAP – HETEROGENEITY SUPPORTED

This section details the truck fleet's implementation. As with any problem, multiple potential solutions exist. We describe here how the fleet application is implemented using the SOAP messaging system.

SOAP, "Simple Object Access Protocol", is a lightweight communication protocol specification for invoking methods on servers [5, 6]. It allows communication between applications via HTTP/XML. SOAP is neither tied to any component technology nor to any programming language. Like its name implies, it is simple and extensible.

On the client side, the software does not connect to database directly. We use the server side to connect to the database and return the results to client side via SOAP packets. Therefore, we need to write and deploy the SOAP service, and then write a SOAP client to communicate with the SOAP server.

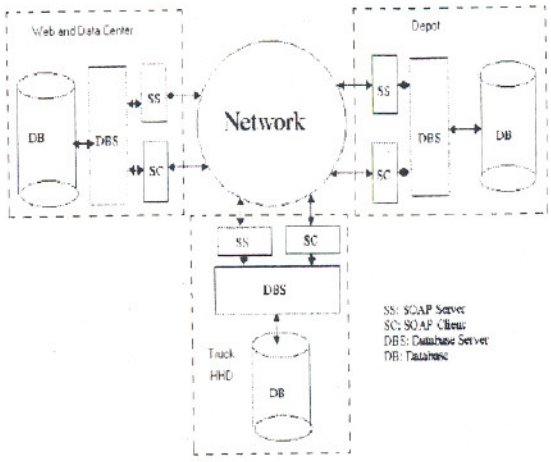

Figure 3. Fleet Architecture based on SOAP

In our fleet application each truck, depot and WDC act as both SOAP clients and servers. SOAP services are deployed in each truck, depot and WDC. When one entity needs to communicate with another entity or entities, it sends a SOAP request to the destinations. These requests should contain the host name and the SOAP method that needs to be invoked. Once the destination server receives the above information, the destination SOAP service will invoke appropriate methods and send the results back to the requester. For example, when a customer checks the status of his order, SOAP communication occurs as follows. First, the customer program sends a SOAP request to the WDC asking for the status of customer orders. Upon receiving the request, the WDC's SOAP server invokes the method requested by the customer program, performs database queries locally, and sends the results back to the customer program. Figure 3 shows how depots, WDC and trucks communicate with each other. In the figure, each entity has both a SOAP client (SC) and a server (SS), which interface with the database server (DBS), and the database itself (DB).

The customers' login web page is shown in Figure 4, simulated with the Palm OS. The client- side program will check the customers' identification by obtaining service from server side, and return the appropriate screen.

4. SYSTEM ON DEVICES MIDDLEWARE

This paper shows some of the problems inherent in a distributed system spanning different devices. To make the application development process easier, our research group is actively developing a middleware technology called System on Devices (SyD). SyD is a new technology that addresses the key problems of heterogeneity of device, data format and network, and that of mobility [1-4].

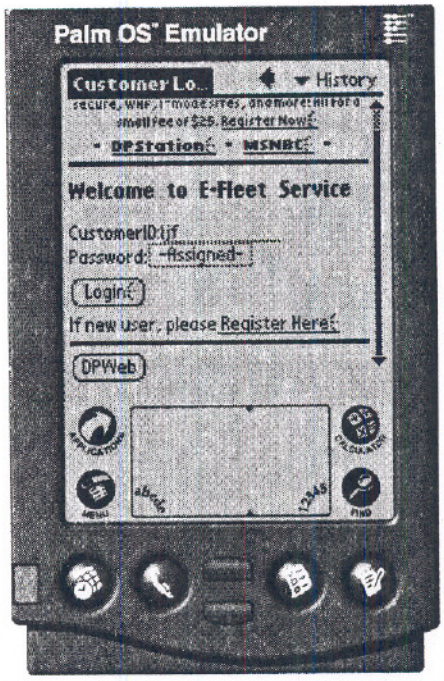

Figure 4 - Customers' Login Web Page on Palm Emulator

The System on Devices is a middleware to enable rapid implementation of applications that need a collection of heterogeneous, independent data stores to collaborate with each other. Each individual data store in SyD may be a web service, a traditional database such as relational or object-oriented, or may be an ad-hoc data store such as a flat file, an EXCEL worksheet or a list repository. These may be located in traditional computers, in personal digital assistants (PDAs) or even in devices such as a utility meter or a set-top box. These databases are assumed to be independent of each other, i.e. they do not share a global schema. The databases in SyD co-operate with each other to perform interesting tasks, and we envision a new generation of applications to be built using the SyD framework. The SyD architecture is captured in Figure 5.

The SyD architecture has three layers. At the lowest level, individual data stores are encapsulated by SyD Client Objects, referred to as SyDCOs. These client objects export the data that the client devices hold along with methods/operations that allow access as well as manipulation of this data in a controlled manner.

At the middle layer, there is SyD middleware, a logically coherent collection of services, APIs, and objects that facilitates the execution of application programs. At the highest level are the applications themselves that are independent of device, database and network. They rely on the directory services maintained by the SyD middleware. These applications include instantiations of

SyDCOs, SyDApp objects that are aggregations of SyDCOs, and SyDMW objects. The three-tier architecture of SyD enables applications to be developed in a flexible manner without knowledge of device, database and network details.

SyD software technology is characterized by the following:

1. Ordered stores of data, be they formal databases or ASCII lists.

2. SyD middleware (SyDMW) is responsible for making software applications (anywhere) aware of the named objects and their methods/services, executing these methods on behalf of applications.

3. SyD Applications (SyDApps) are written independent of the data formats or location of the information store.

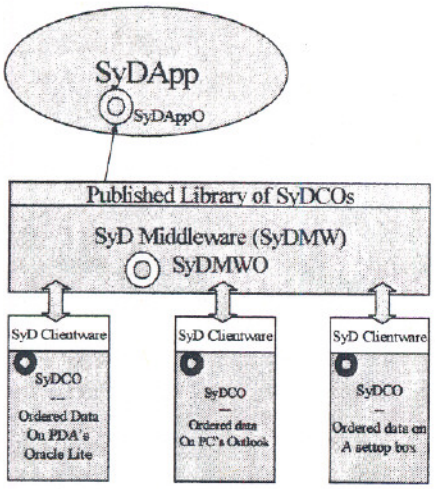

Figure 5. Architecture of SyD

SyDApps are thus truly portable, network and database independent, and are able to operate across multiple networks and multiple databases, relying on the SyDMW to provide the supporting services that translate the SyDApps code to the correct drivers, for both communications and computing.

SyD Infrastructure: The SyD infrastructure includes two major components: the SyD middleware and the SyD clientware. The responsibilities of SyD middleware include:

1. Directory and Ad-Hoc Group Management services: The SyD middleware maintains a directory of active devices, their proxies (if any), active applications, active groups of devices/users, etc. The middleware is also responsible for maintaining a directory of SyDCOs that have been published by the individual devices, as well as the SyD middleware objects (SyDMWOs) that may provide domain-specific services, such as financial services.

2. Communication Services: SyD Middleware is responsible for providing communication services across devices. Services such as broadcast and multicast to a group of devices are included in these services.

3. SyD Quality of Service Management: SyD middleware provides communication specific QoS support, such as reliability, quorum, delay, cost, fault-tolerance, and real-time constraints.

4. SyD Data Services: The middleware provides data services that include global querying and update of data stores managed by the SyDCOs.

The SyD clientware, which manages the devices, is responsible for creation and management of SyDCOs and for providing the execution engine for the SyDCO methods. SyDCOs are defined and implemented by the vendors of the devices or by developers specializing in device programming.

We are in the process of implementing SyD kernel and redeveloping the fleet system using the middleware primitives. We expect that web-based collaborative applications, such as this truck fleet management system, will be easier to develop and port across heterogeneous devices.

5. CONCLUSIONS

A mobile fleet application system running on different hosts (workstations) and hand-held devices has been implemented as a distributed mobile system on a wireless network. The Web and Data Center, depots and trucks are the three major components of the mobile fleet application system. Various triggered events such as truck accident and successful delivery are synchronized for correctness and consistency of the system.

In the future, more complex functions will be implemented, in conjunction with the System on Devices middleware. For instance, we plan to implement REC (Remote Execution Code) in the near future [14].

ACKNOWLEDGEMENTS

We would like to recognize and thank all of the graduate students who have made this work possible: Junfang Lei, Bing Liu, Janaka Balasooriya, Hui Liu, Pooja Bhatia, Arthi Hariharan, Bo Jin, and Yuanchen He.

REFERENCES

[1] M. Weeks, Y. Zhang, et al., "Mobile Fleet Application Based on SyD Technology," Proceedings of Yamacraw Industry Advisory Board Conference, April 2001.

(SyD): A Model with Coordination Link Primitives and a Calendar Application." Proceedings of Yamacraw Industry Advisory Board Conference, April 2001.

[4] Madisetti, R. Sunderraman, et al., "System of Database (SyD): Architecture, Global Queries, Triggers, and Constraints," Proceedings of Yamacraw Industry Advisory Board Conference, April 2001.

[5] SOAP: Simple Object Access Protocol, W3C recommendation, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[6] SOAP version 1.2 working draft, Editors: Martin Gudgin (DevelopMentor), Marc Hadley (Sun Microsystems), Jean-Jacques Moreau (Canon), and Henrik Frystyk Nielsen (Microsoft Corp.) July 9th, 2001 http://www.w3.org/TR/2001/WD-soap12-20010709/

[7] U. Dayal and H. Hwang. View definition and generalization for database integration in MULTIBASE: A system for heterogeneous distributed databases. IEEE Trans. Software Engineering, SE-10, 6, 628-644, 1984.

[8] E. Pitoura and B. Bhargava. A Framework for Providing Consistent and Recoverable Agent-Based SIGMOD Record, 24(3):44--49, September 1995.

[9] G. Reinelt, The Traveling Salesman: Computational Solutions for TSP Applications, Springer Verlag, Berlin, Germany, 1994.

[10] C. Malandraki and M. S. Daskin, Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms, Journal of Transportation Science, 26 (1992), pp. 185-200.

[11] M. Gendreau, F. Guertin, J.Y. Potvin and R. S'eguin, "Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-ups and Deliveries", Technical Report CRT-98-10, Centre de recherche sur les transports, Université de Montréal, 1998.

[12] S. Prasad, et al., "Mobile Fleet Communication System for Multiple Mobile Data-stores," Utility Patent filed April 23, 2002, Application Number 10/131,682.

[13] S. Prasad, et al., "Mobile Data-stores Enabled with Coordination-Link Primitives and a Calendar Application," Utility Patent filed April 23, 2002, Application Number 10/131,681.

[14] James W. Candler, Prashant C. Palvia, Jane D. Thompson and Steven M. Zeltmann, "The ORION Project: Staged Business Process Reengineering at FedEx", CACM, Vol.39, No. 2, February 1996, pp. 99-107.