Toward an Easy Programming Environment for Implementing Mobile Applications: A Fleet Application Case Study using SyD Middleware

S. K. Prasad, M. Weeks, Y. Zhang, A. Zelikovsky, S. Belkasim, R. Sunderraman
Department of Computer Science
Georgia State University
Atlanta, GA 30302-4110 USA

V. Madisetti
Electrical and Computing Engineering Department
Georgia Institute of Technology
Atlanta, GA 30332 USA

Abstract

This paper describes the advantages of SyD (System on Mobile Devices), a middleware technology for mobile devices and e-services, in terms of technology and programming. Features of SyD are illustrated here through our prototype application, a complex communication system for a trucking fleet that operates an automated package delivery system. The fleet system has been implemented in three ways, with SOAP, with JDBC, and with SyD. Our implementation experience shows that SyD greatly simplifies coding by allowing heterogeneous devices, peer-to-peer communications, group transactions based on triggering events, and mobility support through proxies and directory service.

1 Introduction

In this paper we discuss collaborative application development issues in the heterogeneous environment involving a variety of mobile devices. We have selected Java as the programming language since it allows a cross-platform environment. With it, however, there are many ways to solve a programming problem, especially one that requires networking and database accesses. Solutions include programming with JDBC (Java Database Connectivity) [2], SOAP (Simple Object Access Protocol) [1], and SyD (System on Mobile Devices) [7]- [12], our recently developed middleware for developing collaborative application over a collection of mobile heterogeneous devices.

To develop an application that requires integrating a set of devices with different capabilities (e.g. mobile phone, PDA, PC, and server), and data distributed on these devices, requires solving many problems. For each device, the programming may be different for both data storage and communications, for example, the selection of database packages that work with small devices like a PDA is severely limited. Groups may need to be formed spontaneously, and deal with unexpected situations. Also, a reliable system will allow automatic updates, by a triggering mechanism. Finally, weak connectivity means that data may not be available when needed. In short, the application requires the following technical needs to be met:

I. Heterogeneous device, data format and language support,

II. Peer-to-Peer computing over dynamic groups,

III. Group transactions: Autonomous Database Synchronization based on triggering events; group querying, and,

IV. Mobility support through proxies and directory service.

A practical application that demonstrates the technical problems mentioned above is a truck fleet that operates an automated package delivery system. In our application, a user would connect through the Web and Data Center (WDC) web-page, 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 in the general geographic area of the package, which schedules a pick-up. A truck will arrive according to the schedule set by the depot, and take the package to its next stop, which could be the package's destination, or some intermediate point between the package's source and its destination.

Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)
0730-3157/03 $ 17.00 © 2003 IEEE

The trucks, depots and WDC form a heterogeneous network with different types of communications, and databases of varying capacity. The WDC and depots are assumed to have a wired network connection, but depots have wireless connections to their respective trucks. Also, the trucks can communicate with each other directly through their wireless devices.

In this paper, we discuss the implementation of the fleet system in the following three ways. First, we extend SOAP (Simple Object Access Protocol), a lightweight protocol that can be used to invoke a particular database procedure or query [1]. Second, we use JDBC (Java Database Connectivity) [2] to carry out the transactions in remote databases. It is more restrictive than SOAP, but gives a basis for comparison. Finally, we introduce SyD (System on Mobile Devices) as an alternative to the two above solutions.

The primary contributions of our work presented here is a methodology to rapidly develop robust distributed collaborative applications, and an execution platform to deploy such applications, while masking mobility and heterogeneity from application programmers. We illustrate each of the steps involved in developing and deploying a SyD application through the fleet management application.

This paper is organized as follows. Section 2 briefly introduces SyD, followed by section 3, which details how an application is developed using SyD. Section 4 discusses the fleet application and its implementation. Finally, Section 5 concludes the paper.

2 An Overview of SyD

In this section, we briefly describe the design of SyD and related issues, and highlight the important features of its architecture. SyD is envisioned as a system that will enable rapid prototyping and implementation of applications that need a collection of heterogeneous, independent databases to collaborate with each other in a mobile environment. Each individual device in SyD may be 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 or even in devices such as a utility meter or a set-top box. These devices are assumed to be independent of each other, i.e. they do not share a global schema. The devices 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 1.

The SyD architecture has three layers.

1. At the lowest layer, individual data stores are encapsulated by device objects. These device objects export the data that the devices hold along with methods/operations that allow access, as well as manipulation of this data in a controlled manner. This is enabled by SyD Deviceware consisting of a listener module to register objects and to execute local methods in response to remote invocations, and an engine module to invoke methods on remote objects. SyD device objects also have inherent capabilities to link with each other in an interdependent fashion to enable object composition and atomic transactions over multiple objects, provided by a linking module.

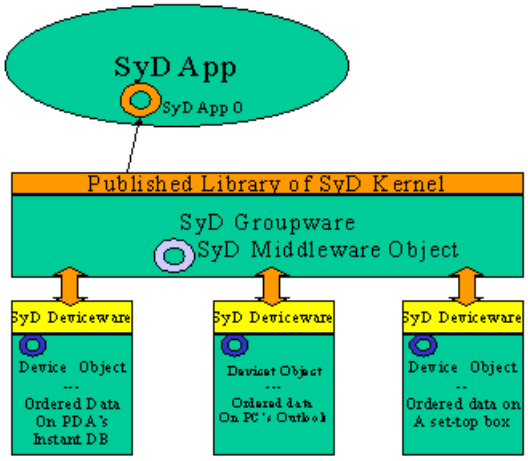

Figure 1. Architecture of SyD

2. At the middle layer, there is SyD groupware, a logically coherent collection of services, APIs, and objects that facilitates the execution of application programs. Specifically, SyD groupware consists of a directory services module, portion of engine module for group service invocation and result aggregation, and an event handler module for global events.

3. At the highest level are the applications themselves (SyDApp). They rely only on groupware services, and are independent of device, database and network. These applications include instantiations of SyDApp objects that are aggregations of the device objects, and SyD middleware 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, thus, characterized by the following: data stores, middleware for communications, data and method access, and the applications that take advantage of SyD. Ordered stores of data are supported by SyD deviceware that allows the construction, naming, and publication of device objects, that operate on these data stores through methods. SyD groupware is responsible for making software applications (anywhere) aware of the named objects and their methods/services, executing these methods on behalf of applications, allowing the construction of SyD Application Objects (SyDAppOs) that are built on the device objects, and providing the communications

infrastructure between SyD Applications (SyDApps), in addition to providing QoS support services for SyDApps. SyDApps are applications written by and for the end users (human or machine) that operate on the SyDAppOs alone and are able to define their own services that utilize the SyDAppOs, without directly depending on either the location or the type of database or type of device (PDA or mainframe) where a certain information field is stored. The SyD groupware provides only a named device object for use by the SyDApps, without revealing the physical address, type or location of the information store.

SyDApps are, thus, truly portable, network and database independent, and are able to operate across multiple networks and multiple devices, relying on the middleware to provide the supporting services that translate the SyDApps code to the correct drivers, for both communications and computing. SyDApps can also decide on their own features and services they offer, without depending on individual databases residing on remote computing devices to offer those services. The SyD architecture, thus, is compatible with and extends the currently emerging web-services paradigm for Internet applications.

3 How to Develop a SyD Application

SyD middleware technology can use the client/server model as well as peer-to-peer. The section below demonstrates the client/server model.

3.1 Client Code and Server Code

It is a time consuming and laborious task to develop applications using existing technologies such as JXTA, BREW, and compact .NET. Current technology provides little flexibility for user defined ad-hoc services and makes it difficult to dynamically configure a collection of independent data stores. SyD technology offers a flexible solution that addresses key problems of heterogeneity of device, data format, network, and mobility. Empowered by ease of application development, mobility of code, data and users, SyD encapsulates different devices by persistent object addressing both the device and data heterogeneity. Each application that is hosted on a mobile device maintains its proxy object on a wired device adding fault tolerance to disconnection in mobile environment. The dynamic directory seamlessly switches between proxy and application in case of disconnection. An ad hoc application can be developed according to the following two steps:

1. Model the application using SyD objects using the provided rich set of SyD APIs. Incorporate SyD directory functionality to publish objects along with their data and methods in the SyD directory service.
2. Develop high-level application server code by employing the objects/methods in Step1. If new, composite, server functionalities need to be published. Else, develop only a separate client code, which is typically similar to a browser/GUI page.

3.2 Design Aspects

The design of a collaborative application within the SyD environment proceeds in two fronts: the design of services offered by individual devices (server side) and the design of consumers of these services (client side). For some applications these two design phases may be done by two independent entities: the service providers publish their services and the clients subscribe to these services in their application code. For other applications, the services code is driven by the client requirements and may be designed by the same entity.

The SyD framework enables rapid design and development of both server and client code. Server functionality is exposed as Web services and published with the SyD Directory and client code subscribe to these services and incorporate them within their logic. The SyD IDE provides templates for server as well as client code and enables rapid application development. The application designer may choose one of several server templates as a starting point for the design and implementation of the services. The IDE also allows for these services to be published with the directory server. Client templates are also available as starting points for client code. These templates include code for invoking the services as well as GUI elements for user interaction. High-level primitives allow programmers to seamlessly call remote methods, possibly over a multiple devices. The underlying complexities of messaging, fault-tolerance, etc., are handled by the SyD engine on the client side and SyD listener on the server side.

4 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.

4.1 System Operations

The Web and Data Center (WDC) is contacted by the user to request package transport. It also allows the user to track packages [6]. Upon request for delivery, the WDC passes it on to a depot. Every depot has a number of trucks associated with it. The depot's duties are to store packages physically, as well as package information, until loaded

onto trucks, to schedule the routes for the trucks to take, to serve as a broadcaster to trucks for traffic information, to provide detailed package information upon request, and to update WDC with all packages' information at regular intervals, such as at completion of a schedule. The truck driver, with a Hand-Held Device (HHD), picks up packages from a source and delivers them to a destination. However, a package may have several stops before reaching its final destination. If a user wants to check on a package, he would contact the WDC. 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.

Each element of the system has different capabilities. The WDC, depots, and trucks are likely to have servers, standard PC's, and HHD's, each with varying processor speeds, communications bandwidth, and storage facilities.

4.2 The Accident Scenario

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 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: $emergencytime + traveltime < slacktime$. If more than one truck is available, the one with the maximum $slacktime - traveltime - emergencytime$ 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 next.

4.3 Implementation with SyD

The goal of SyD is to provide an integrated programming and deployment platform. This goal allows a programmer to have a uniform view of device, data and network, even though these things are likely to be completely different. Also, the objects should appear to be persistent, even though the data (and perhaps even the services themselves) could be hosted on one or more mobile devices. Forming, maintaining, and manipulating groups should be easy to do. We have implemented this middleware, and this "fleet of trucks" application shows the power and flexibility of SyD. It has a footprint of 112 KB, only 76 KB of which is device resident. Representative code is given below.

Method Invocation:

```
SOAP
(Server side)
method name: getAns(String Query, int
 columns))
conn =DriverManager.getConnection
("jdbc:oracle:thin:@192.168.1.100:1521:sid1",
 user,pass);
Statement stmt = conn.createStatement ();
ResultSet rset=stmt.executeQuery(Query);

(Client Side)
String query="select * from Customers
 where customerID = '"+customer + "'
 and password = '" +_password + "'";
CreateSoapEnvelop(String soapUrl,
 String soapMethod, String soapService,
 Call soapCall, String Query);
Response res=soapCall.invoke(soapUrl,"");

JDBC
conn=DriverManager.getConnection
 ("jdbc:oracle:thin:@192.168.1.100:
 1521:sid1",user,pass);
Statement stmt = conn.createStatem;
 String query="select * from Customers
 where customerID = '"+customer +
 "' and password = '" +_password + "'";
ResultSet rset =
 stmt.executeQuery(query);

SYD
Vector paramvalue = new Vector();
usernamelist.add("mike");
method
paramtype.add("java.lang.String");
method
paramvalue.add("packet#");
doclist = dispatcher.invoke(
 usernamelist,"findlocation",
 paramtype,paramvalue);
```

The SyD kernel was developed in Java. It contains a Directory service, which provides user, group and service publishing, lookup service, and intelligent proxy management. The Listener service sits on device and enables them to act as servers by listening to remote invocation requests. Next, the Engine component allows users to execute services remotely and aggregate the results. An Event Handler handles local and global events. The Link component enables an application to create and enforce interdependencies, constraints, and automatic updates among groups of SyD entities.

For the fleet application, the Fleet Application Object (FleetAppO) registers its services in the SyD Directory service using SyD Listener that sits on the device. The FleetAppO invokes methods on other SyD objects using SyD Engine, which in turn calls SyD listener to invoke methods on remote objects.

Step 1. SyD application registers its service with the SyD Registrar and SyD Directory Service

Step 2. SyD Appo calls SyD Engine to utilize the service of other users

Step 3. SyD Engine calls SyD Listener, which in turn calls the SyD Listener in other devices

Step 4. SyD Listener in other devices call the related service and finally return the results back to the calling SyD Engine

Step 5. SyD Engine aggregates and analyzes the results and returns to the user

4.3.1 Development of WDC and Depot Using SyD

The WDC and Depots sit at fixed locations. They are modeled with application objects (SyDObjects), and have the Listener deployed on each WDC and Depot device. Next, the WDC and Depots are automatically registered in the local SyD Listener and directory service. 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. Figure 2 shows how the truck fleet system's components interact. 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.

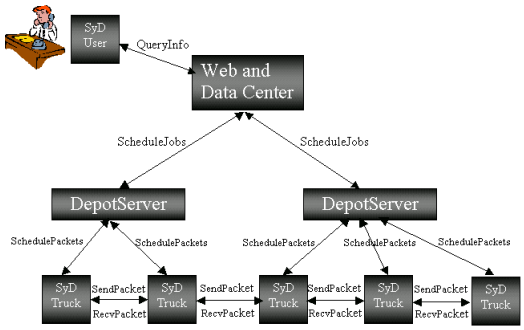

Figure 2. Software Architecture of the Fleet System

4.3.2 Development of Truck Using SyD

The trucks will be deployed on mobile devices (such as an iPAQ). Each truck is modeled as a SyD Object. The trucks are grouped together depending on the depot that they report to. When a truck registers itself in the directory service, it will specify the group name so that group functionality can be invoked.

```
Package syd.sydapp;
Class SyDTruckAppO extends ..{
   // Create the SyD Appl. Object
   SyDTruckAppO truck = new
      SyDTruckAppO();

   // Publish In the Directory
   publisher.
   createPublishUserMethodsRequest
      (userID, userPasswd, userURL,
      proxyID, appName, methods,
      returnTypes, params);
   String pub = publisher.getString();
   SyDRegistrar registrar = new
      SyDRegistrar (portNum,
      directoryServerName,
      directoryServerPort);
   registrar.register(truck, pub);

   // Dispatch using SyD engine
   dispatcher.invoke(usernamelist,
      <methodname>, paramtype,
      paramvalue);
}
```

5 Conclusion

Based on the functionality and architecture of SyD, the mobile fleet application system running on different

hosts (workstations) and hand-held devices has been implemented. The new SyD application system is 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 fleet SyD. The mobile fleet application system runs correctly and smoothly in terms of SyD consistency, dynamic map, wireless operations, and distributed synchronization.

The development of SyD will enable companies to greatly improve performance through the use of technology [7], by making wireless applications for heterogeneous and distributed devices easier to develop.

Acknowledgments: This research was supported by the State of Georgia's Yamacraw Embedded Software research contract #BLA42 and #CLH49.

We would like to recognize and thank all 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] SOAP: Simple Object Access Protocol, W3C recommendation, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[2] John Ellis, Linda Ho, and Maydene Fisher, JDBC 3.0 Specification Proposed Final Draft 4, Sun Microsystems, October 25, 2001. See http://java.sun.com/products/jdbc

[3] Steve Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, February, 1997.

[4] K. Ramamritham, "Real-Time Databases, Distributed and Parallel Databases," 1(1993), pp. 199-226.

[5] M. H. Dunham and V. Kumar, "Location dependent data and its management in mobile databases," *Proceedings of DEXA Workshop*, Vienna, Austria, August 1998, pages 414-419.

[6] 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.

[7] Sushil K. Prasad, Vijay Madisetti, et al. 2003. "A Middleware for Collaborative Applications over a System of Mobile Devices (SyD): An Implementation Case Study," Technical Report CS-TR-03-01, Department of Computer Science, Georgia State University, July, 16 pages.

[8] Vijay Madisetti, "SyD: A middleware infrastructure for mobile iAppliance devices," *EE Times Network*, verb-http://www.iapplianceweb.com/story/OEG20021105S0031, November 5, 2002.

[9] S. K. Prasad, M. Weeks, Y. Zhang, et al., "Mobile Fleet Application Using SOAP and System on Devices (SyD) Middleware Technologies," *Communications, Internet and Information Technology (CIIT 2002)*, St. Thomas, Virgin Islands, USA, November 18-20, 2002, pages 426-431.

[10] Sushil K. Prasad, Erdogan Dogdu, et al., "Design and Implementation of a listener module for handheld mobile devices," *Proceedings of 41st Annual ACM Southeast Conference*, Savannah, Georgia, March 7-8, 2003.

[11] Sushil K. Prasad, Anu G. Bourgeois, et al., "Implementation of a Calendar Application Based on SyD Coordination Links," *Proceedings of The Third International Workshop on Internet Computing and E-Commerce* at the 17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, 22-26 April 2003.

[12] Wanxia Xie, Shamkant B. Navathe, and Sushil K. Prasad, "Supporting QoS-Aware Transaction in the Middleware for a System of Mobile Devices (SyD)," *Proceedings of 1st International Workshop on Mobile Distributed Computing*, at 23rd International Conference on Distributed Computing Systems (ICDCS'03), Providence, Rhode Island, May 19-22, 2003.