Enforcing Interdependencies and Executing Transactions Atomically Over Autonomous Mobile Data Stores Using SyD Link Technology[1]

Sushil K. Prasad[*], Anu G. Bourgeois[*], Erdogan Dogdu[*], Raj Sunderraman[*], Yi Pan[*], Sham Navathe[**], Vijay Madisetti[***]
[*]Computer Science Department
Georgia State University, Atlanta, GA 30303
[**]College of Computing
[***]Department of Electrical Engineering
Georgia Institute of Technology, Atlanta GA 30332

Abstract

System of Mobile Devices (SyD) is a middleware we developed that can be used for implementing collaborative, mobile, and distributed applications over heterogeneous devices, data stores, and computing environments. Current prototype implementation of SyD consists of five modules. These modules provide ease of programming in the areas of distributed communication, remote method invocation, service publication and discovery, directory services, distributed service invocation and aggregation, event handling, collaborative link creation and enforcement. A central module is SyDLink, which allows SyD-based applications to create "coordination" links. Coordination links represent dependencies among heterogeneous devices and application components. Based on the underlying event-and-trigger mechanism, they allow automatic updates as well as real-time enforcement of global constraints and interdependencies. SyDLink objects provide the underlying mechanism in SyD to enforce atomic execution of distributed transactions. We explain and demonstrate the use of SyDLink objects via a running example, a collaborative SyD calendar application, throughout the paper.

1. Introduction

System on Devices (SyD) middleware technology was introduced [1] to address key problems of device heterogeneity, data format and network, and that of mobility. SyD combines ease of application development, mobility of code, application, data and users, network and geographical location independence, and the scalability required of large enterprise applications concurrently with the small footprint required by handheld devices. SyD uses the simple yet powerful idea of separating device management from the management of groups of users and/or databases.

The current technology for the development of such collaborative applications over a 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 specific to the type of device, data format, and the network. The data-stores are typically a centralized logical entity providing only a fixed set of 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. There are only a few existing middlewares which address the stated requirements. Even these are either not completely functional at this time, or enable only client-side programming on mobile devices, or are geared to a limited domain of applications, or are limited in group or transaction functionalities or mobility support.

The calendar application is an example of a typical SyD application in which several individuals maintain their independent schedule information in their hand-held and other devices [2-4]. The typical functionalities provided in such an application are: (i) set up meetings among individuals with certain conditions to be met such as a required quorum, (ii) set up tentative meetings which could not be set up otherwise due to unavailability of certain individuals, and (iii) remove oneself from a meeting or cancel an entire meeting resulting in automatic triggers being executed that may possibly convert tentative meetings into confirmed ones. Section 2 presents the current prototype implementation of SyD middleware and also introduces the SyD-based calendar application.

Creating and maintaining a dynamic group of entities, as in a meeting, is integral to SyD. We use SyD coordination links that can be employed by the SyD middleware for this purpose. The coordination links are abstract relationships among entities with underlying constraints and event-based triggers. These allow automatic updates and synchronization across independent data stores as well as on-the-fly establishment and enforcement of global constraints and interdependencies. *Subscription links* allow automatic flow of information from a source entity to other entities that subscribe to it. This can be employed for synchronization as well as for more complex changes. *Negotiation links* enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction.

Section 3 introduces the SyD coordination links, how they express interdependencies, and how transaction atomicity is achieved using links with examples from the calendar application. Section 4 presents the implementation and execution of SyDLink objects. Section 5 provides a comparison with existing calendar applications. Section 6 concludes with an overview and a comparison of SyD to existing distributed database technologies.

[1]This research was partially supported by State of Georgia's Yamacraw Embedded Software Contract #CLH49 and #DLN01. Embedded Software Contract #CLH49 and #DLN01.

Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)
0-7695-1921-0/03 $17.00 © 2003 IEEE

2. Overview of SyD

SyD is envisioned as a middleware that will enable rapid prototyping and implementation of distributed applications SyD is envisioned as a middleware that will enable rapid prototyping and implementation of distributed 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 (PDAs) 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.

Figure 1 depicts the layered architecture of SyD runtime environment in the current implementation. SyD in this environment is a middleware providing distribution transparency and management to SyD-based application development, therefore, greatly reducing the development, implementation, deployment, and maintenance time (software life cycle) for designers and programmers of distributed applications on heterogonous mobile devices and environments.

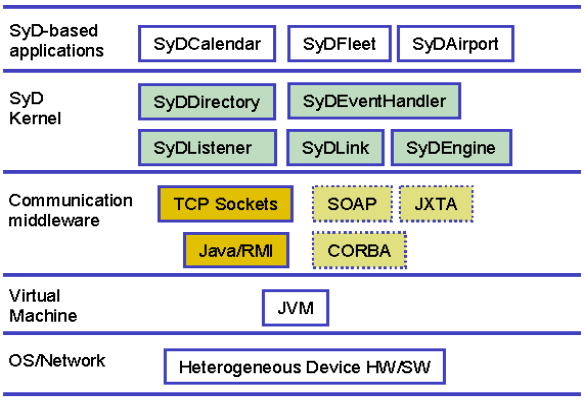

Figure 1. SyD Runtime Environment

SyD is a middleware located between applications and the communication services provided by primitive distribution middleware (Sockets, RMI, JXTA, CORBA, etc. [5-6]). Each layer depends on the services provided by a lower layer. Therefore, each layer hides complexities of the tasks provided in that layer from upper layers. This provides following advantages:

a. Distribution transparency: SyD modules provide location, access, resource sharing, and migration transparencies [21]. Application developers concentrate on application functionality, and business logic; distribution services are provided by SyD Kernel seamlessly.
b. Rapid application development: Detailed application distribution issues are hidden from designers and programmers (distribution transparency) therefore reducing design and development time.

SyD utilizes "primitive" distribution middleware technologies for remote method invocations, distributed object access, and registration. These could be Sockets, Java/RMI, CORBA, .NET, SOAP, etc [5-7,10,11]. In the current implementation, we have used TCP Sockets for small footprint and maximum flexibility. Future versions of SyD will be expanded to include other distribution middleware technologies, such as JXTA, for wider heterogeneity and acceptance.

The next lower layer in the current architecture is a JVM. We have developed SyD Kernel using Java and utilizing TCP Sockets for distributed communication layer. Consequently, applications developed using SyD run on a JVM. This also allows SyD applications to run on heterogonous devices and operating systems since JVM is available on many different platforms including small mobile devices (e.g., we employed Jeode JVM on iPAQs).

In this framework, applications are developed rapidly using SyD Kernel modules without any knowledge about lower layer services (primitive distribution middleware, OS/environment).

Figure 1 lists three sample mobile applications that we have developed using SyD middleware: a calendar application, a fleet application, and a price-is-right bidding game suitable to be played at an airport or a mall. Section 4 will provide more detail about the calendar application.

We have designed and implemented a modular SyD Kernel utility in Java. SyD Kernel includes the following five modules (Figure 1 and Figure 2):

SyDDirectory: Provides user/group/service publishing, management, and lookup services to SyD users and device objects. Also supports intelligent proxy maintenance for users/devices. SyDListener: Enables SyD device objects to publish their services (server functionalities) as "listeners" locally on the device and globally via the directory services. It then allows users on SyD network to invoke single or group services via remote invocations seamlessly (location transparency). SyDEngine: Allows users to execute single or group services remotely via SyDListener and aggregate results. SyDEventHandler: This module handles local and global event registration, monitoring, and triggering. SyDLinks: Enables an application to create and enforce interdependencies, constraints and automatic updates among groups of SyD entities.

A SyD-based application (SyDAppO object), such as SyDCalendar, SyDFleet, etc., typically has a server component and a client component. Such an application developed using SyD Kernel interacts with SyD Kernel

module APIs to get higher-level distribution services in the following fashion:

a. Publishing on SyDDirectory: Applications register and publish their information including location and service availability on SyDDirectory for other users to lookup and execute via SyDEngine. User/object groups can also be formed on SyDDirectory.
b. Registering services as listeners using SyDListener: SyDListener registers application methods as remote listeners for remote invocations locally in RMI registry and globally in SyDDirectory.
c. Execution via SyDEngine: Users can execute individual object's services remotely using SyDEngine. It is also used to execute a service on a group of objects. SyDEngine executes remote services by invoking SyDListener.

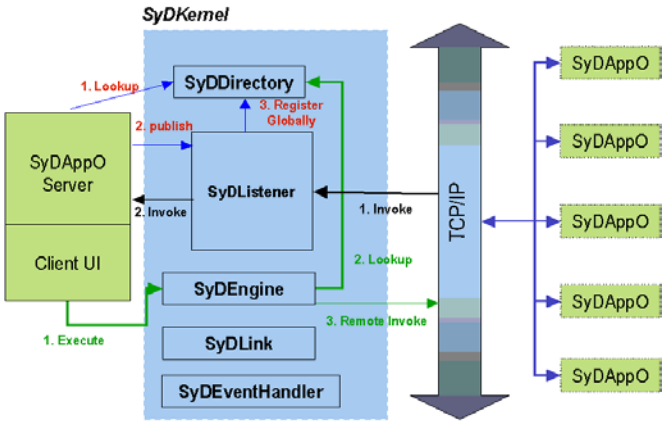

Figure 2. SyD Kernel architecture and the interactions between modules and application objects.

A SyD-based server application provides services to local user and also global users on the network. Global users access remote services by invoking methods remotely that are previously published as listeners. Therefore, user interface (client) and server application functionalities are separated in the implementation. Client interface allows users to invoke application services, locally or globally. A SyD-based application provides distribution transparency via SyDDirectory-based server applications. A SyDDirectory maintains user/service directories and upon request delivers the location information to requesters on the fly. In this framework, a requester of a remote service acts as a "client" of the remote service that is provided by a remote SyD-based application that acts as a "server". For this interaction, the client consults with the SyDDirectory to get remote user/service information about the remote "server".

2.1 SyD-Based Design of Calendar Application

SyD Calendar application is a typical collaborative application that we developed using SyD. Calendar application allows multiple users to share their calendars in real-time, setup group meetings, cancel meetings, and similar functions. Individual calendars can be maintained on small devices like PDAs (currently we are using Compaq iPAQ, and PalmPilots with WiFi wireless network connections), on PCs, or on servers. System allows users to register proxy devices for individual applications that will take care of disconnections. The following are some usage examples:

Scenario 1: *Meeting Setup*: A user wants to setup a group meeting. User enters the dates between which he or she wants to setup a meeting on his PDA and also the people whom he or she wants to call for the meeting. A list of open slots common to all the participants appears on the screen. Next, select an available slot, and the meeting is scheduled; calendar application updates the calendars of all participants.

Scenario 2: *Tentative Meetings*: In scenario 1, if some participants' calendars cannot be accessed (a usual case especially for users of mobile devices), calendar application interacts with the proxy devices of the individuals and sets up a meeting *tentatively*. Later, when the unavailable users come online, application polls their calendars and tries to update their calendars to setup the meeting permanently using the minimum participation criteria or other criteria.

In the above scenarios, coordination among individual users' calendars is achieved by SyDLink objects created on the fly between the user setting up a group meeting and participants of the meeting. Links are *tentative links* in the case actual calendars of participants cannot be accessed due to disconnectivity or unavailability (link is then between a SyD object and a SyD proxy object), or links are *permanent links* in the case actual participant's device can be accessed.

2.2 How is this application in current practice?

In current practice, the calendars of each user would have to be located explicitly and entries would have to be written explicitly by the committee calendar program to each unique database, after taking into consideration that it is a PDA, PC, or web-server, each with its own native communications mechanism. The resultant code would also not be portable, and very difficult to maintain. Methods that can trigger the committee calendar when individual calendars are changed cannot be easily written and their execution is not efficient.

3. The SyD Link Concept

As illustrated above, forming and managing dynamic groups of objects is one of the key aspects of SyD technology. In this section, we present SyD coordination links as a solution. A coordination link is an abstract relationship among a group of objects/databases (referred to as entities hereafter) with an underlying constraint and a set of event-triggered actions. In this section, we define links, how they express

interdependencies, and describe the different types of transactions that are possible.

3.1. Link Definition

A component of SyD, SyDLink, enables an application to create and enforce interdependencies, constraints and automatic updates among groups of SyD entities. A SyD coordination link is an entry in a data-store associated with an entity that has the following components:

A link is specified by its type (subscription / negotiation), its subtype (permanent / tentative), references to one or more entities, triggers associated with each reference (event-condition-action (ECA) rules), a priority, a constraint (and, or, xor), a creation time and an expiry time. Two types of links are considered at this point: subscription links and negotiation links. Subscription links are useful for automatic updates and synchronization by enforcing pre-order information dissemination in a directed acyclic graph.

Negotiation links, with specified logical constraints, enforce interdependencies between objects and its predecessors. The leaf level objects can safely start execution with assurance from predecessors that they are all reserved and will finish in time. Permanent links act as acquired locks or reservations that are released at the expiry time. Tentative links are essentially queued requests for a semaphore-like prioritized reservation or lock.

All link information is maintained in a link database that is stored locally by the user. This link database is created for a user when he/she installs a SyD application with link-enabled features. The application can maintain a logical connection by creating a link between various entities. All an application has to do is specify a list of users to be linked. For example, consider the calendar application linking users X, Y, Z based on their availability at a particular time. The SyDLink module will negotiate with each of the users and if and only if all the users are available at the specified time, will links be created between the users. If any user is not available at that time, then no links will be created. In this way, links can be created automatically based on availability.

3.2. Transactions Using Links

The existence of links enables us to perform many types of transactions over a collection of data stores. Permanent links allow for atomic transactions satisfying *real time transactions*. Tentative links, however, allow transactions to be carried out over a period of time without starvation. If a link is tentative, the action for the link is pending as the link is stored in a FIFO queue associated upon the link it is waiting upon. Once the permanent link that it is waiting upon, either expires or is deleted, then the first tentative link waiting in the queue is automatically converted to a permanent link, and the appropriate transaction(s) then take place. This situation is suitable for non-critical or *long running transactions*. For example, if link L1 has been caused to be tentative by permanent link L0, then when link L0 is deleted it triggers the automatic conversion of link L1 from tentative to permanent status and via the SyDEngine, it invokes the delete method on linked device. Consequently, all links logically associated with L0 are deleted in a cascading manner. All the links waiting on link L0 are maintained in a SyD_WaitingLink table. Once L1 is deleted, then the waiting link with the highest priority is converted to a permanent link. It is possible to have groups of links waiting on a particular link and deletion of the permanent link triggers automatic conversion of all links in the group with highest priority, from tentative to permanent.

As the devices we are considering are mobile devices, there are issues to consider such as loss of connection, power consumption, and weak connectivity. These temporary disconnections are implicitly tolerated by transferring control to a proxy and by partially completing a transaction, while leaving the subsystem consistent. This corresponds to a *flexible transaction* that could possibly be long running while the proxy is in control. Details regarding proxy synchronization are given in Section 4.

The constraint parameter enables *QoS/constrained transactions* to take place. For example each link with a deadline for abandoning a link execution trial can enforce a real time constraint. On invocation of a link, the overall deadline is passed as a constraint parameter where each intervening data store makes a local determination of its current execution delay (QoS), creates a tentative back link to its predecessor to reserve itself, and forwards the remaining time down the chain of negotiation links. If the constraint is violated on a forward traversal, then the initiator is informed. Else, each object in the path executes the sub-transaction, deletes its tentative back link, and passes the results back to its predecessor.

4. SyD Link Implementation and Execution

In this section we will describe how links are currently implemented in a SyD object and invoked when executing a method in a device object. We will also provide information regarding proxy synchronization. Refer to [22] for more details regarding the implementation of SyD middleware.

4.1. Link Implementation

As mentioned in the previous section, all link information is maintained in a link database that is stored locally by each user. Considering the Calendar application, it is dependant on SyDLinks in order to manage the interdependencies between various calendars. Cancel meeting especially involves following all the interdependencies and automatically converting a tentative meeting to permanent

based on priority. Using SyDLinks, the application can call deleteLink() which follows the following steps to achieve automatic triggering.

When the deleteLink() method is called on User1, it first checks to see if there are any associated waiting links in its database. If there are, then it will automatically convert the status of chosen waiting links from tentative to permanent through the SyDEngine. Next, it deletes the local link specified in the method call. The SyDListener module notifies the SyDEngine to invoke the deleteLink() method on all associated links located with other users. The calendar database of the user is then updated to reflect any deleted meetings and changed meetings. In order to invoke the necessary methods on the linked devices, the SyDEngine looks up the remote URL of the associated users from the SyDDirectory and invokes the methods remotely. The same steps are triggered and executed on each associated user.

4.2. Proxy Synchronization

Our SyD objects are assumed to always have web presence. At the first glance, this may seem unreasonable due to all of the disconnectivity issues associated with mobile devices. For this, if a SyD object *A* is down or disconnected, a proxy takes over the place of *A*. Once *A* comes back up, *A* takes over the proxy. The proxy and the SyD object act as a single entity for an outsider. This makes the SyD architecture fault tolerant and applicable to mobile environment and is transparent to the outside world.

To enable this, each device can either choose to register its own proxy, or have the SyDDirectory assign one to it. After a designated interval, the client object copies its data to the proxy object to synchronize with each other. If for any reason the client object is no longer available, then the SyDEngine automatically informs the SyDDirectory to update the URL information for the client to its proxy. Due to a possible time lapse, the information available at the proxy may be stale data. Now, any requests coming to the client will be directed to the proxy. Each application determines the method in which the proxy handles the data.

Considering the calendar application, each time data is copied from the client object to the proxy, all open slots are blocked by having the client reserve a meeting with itself. Therefore, while the proxy is active, there are no free slots available for any other user to set up a permanent meeting. However, other users may set up tentative meetings if desired. Then, when the client is available, all links that blocked times during the synchronization are deleted, triggering any tentative meetings to become permanent.

5. Comparison to Other Calendar Applications

There are a number of different calendar applications; Microsoft Outlook, Groupwise, and Lotus Notes for example. The calendar application presented here is not targeted to compete with the services of the above products. Instead, the application has been chosen to showcase the technical features of SyD. We will show that this calendar application implemented using SyD has a number of technically superior features over existing applications.

SyD implies a System on Devices that are tightly integrated. There is a global logic that defines the entire system. There are interdependencies between the databases – a property unique to the calendar application implemented in SyD. Most other calendar applications do not have any global logic or any interdependencies between the databases. SyD supports global querying, triggers, and constraints. The devices in SyD always have a web presence and can perform real time updates without human intervention. In SyD, device, network, and language independence can be achieved. These features make the calendar application implemented using SyD a technically superior product.

The calendar application implemented in SyD also includes some new design and implementation features, which are possible due to the many technical features of SyD. In other calendar applications each user stores a copy of every member's folder on his local machine. Each time a meeting needs to be set up, the initiator sends an e-mail to the required participants. The recipients then manually have to accept this meeting before it can be scheduled. There is no concept of priority (for either users or meetings), only the initiator of a meeting can cancel that meeting. There is no option of automatic rescheduling of meetings cancelled due to attendee unavailability and no authentication of users.

The calendar application using SyD overcomes all these shortcomings. Each user's local machine stores only that particular user's information. There are no copies of other user's information. This requires much less storage space. Each user is assigned a priority and each meeting is also assigned a priority depending on the must attendees. A cancelled meeting is automatically rescheduled and does not require any manual consent on the part of the user. Cancellations are based on priority and any cancellation automatically triggers a rescheduling.

A low priority meeting can also be bumped from its time slot to accommodate a meeting of higher priority. The low priority meeting is then automatically rescheduled. The entire process is automated and involves minimum human intervention. This calendar implementation also introduces a new concept of multiple 'OR' groups, wherein the initiator can specify that at least one member from each group must attend the meeting. All transactions are secure. There is an authentication process that verifies the validity of the users.

These new features are not implemented in the existing calendar applications. They could be incorporated into the existing applications, but the implementation would be ad-hoc and lack a systematic logic. That is the most

distinguishing feature of SyD as compared to existing middleware and database technologies [14-20].

6. Conclusions and Future Directions

A calendar application based on SyD technology is presented in this paper. We assume that personal data reside on mobile devices such as PDA or workstations. Hence, the system works in a wireless environment. Previously, SyD was proposed to address the key problems of heterogeneity of device, data format and network, and that of mobility, and to enable independent collections of information or databases to collaborate. In this paper, we illustrate the use of SyD through a calendar application. Forming and managing dynamic groups of objects is one of the key aspects of SyD technology. We presented SyD coordination links as a solution and demonstrated how the links are employed to establish meetings. A prototype implementation of the calendar application using some of the SyD features and its software architecture are also presented. Some issues involved in the calendar application such as scheduling options, database triggers, e-mail services, security, and directory service are discussed as well.

Compared with many existing calendar applications, many new features are introduced in our calendar application. Of course, they could be incorporated into the existing applications, but the implementation would be ad-hoc and lack a systematic logic. On the other hand, the SyD middleware implements a systematic global logic, because of which all these new features can be implemented systematically with ease in programming. Our implementation also considers factors such as low communication bandwidth and weak connectivity in a mobile environment, and small memory and low power in PDAs in our implementation. Hence, proxy and naming services are provided in our implementation.

7. References

1. "System of Databases (SyD): An Enabling Technology for Programming Applications on Multiple Mobile Databases," Patent application filed.
2. "A Calendar Application Based on SyD Technology," Yamacraw IAB presentation, October 2001. Patent application filed.
3. "System of Databases (SyD): A Model with Coordination Link Primitives and a Calendar Application," Yamacraw IAB presentation, April 2001. Patent application filed.
4. "System of Database (SyD): Architecture, Global Queries, Triggers, and Constraints," Yamacraw IAB presentation, April 2001. Patent application filed.
5. Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, IEEE Comm. Magazine, Feb., 1997.
6. T.F. Lunney and A. McCaughey, Component based distributed systems "CORBA and EJB in context", Computer Physics Communications, 2000, in press.
7. R. Sessions. COM and DCOM: Microsoft's Vision for Distributed Objects. Wiley, New York, 1997.
8. K. Ramamritham, Real-Time Databases, Distributed and Parallel Databases 1(1993), pp. 199-226, 1993.
9. 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.
10. SOAP: Simple Object Access Protocol, W3C recommendation, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
11. 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/
12. 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
13. Oracle Documentation Library http://tinman.cs.gsu.edu/~raj/oradoc/index.html
14. A. P. Sheth and J. A. Larson, Federated Database System for Mapping Distributed Heterogeneous, Autonomous Databases, ACM Computing Surveys, 22:3, September 1990, pp.183-236.
15. A. Sheth & J. Larson. Federated databases: architectures and integration. 22(3):182-236, 1990.
16. C. Yu, W. Sun, S. Dao, and D. Keirsey. Determining relationships among attributes for interoperability of multi-database systems. In 1 st International Workshop on Interoperability in Multidatabase Systems, pages 251--257, Kyoto, April 1991. IEEE Computer Society Press.
17. Witold Litwin, Leo Mark, Nick Roussopoulos: Interoperability of Multiple Autonomous Databases. ACM Computing Surveys 22(3): 267-293 (1990).
18. A. Vaduva, Rule Development for Active Database Systems. PhD thesis, University of Zurich, 1998.
19. E. Pitoura and B. Bhargava. A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases. ACM SIGMOD Record, 24(3): 44-49, September 1995.
20. M. H. Dunham and V. Kumar. Location dependent data and its management in mobile databases. Proc. of DEXA Wksp, pg 414-419, Vienna, Austria, Aug 1998.
21. Emmerich, Engineering Distributed Objects, Wiley, 2000.
22. "Implementation of a Calendar Application using SyD Coodination Links," S. K. Prasad, A. G. Bourgeois, E. Dogdu, R. Sunderraman, Y. Pan, S. Navathe, and V.

Madisetti, to appear in *Proc. Wksp on Internet Computing and E-Commerce*, *IPDPS,* 2003.

Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)
0-7695-1921-0/03 $17.00 © 2003 **IEEE**