

Implementation of a Calendar Application Based on SyD Coordination Links¹

Sushil K. Prasad*, Anu G. Bourgeois*, Erdogan Dogdu*, Raj Sunderraman*, Yi Pan*, Sham Navathe**, Vijay Madiseti***
*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 on Devices (SyD) is a specification for a middleware to enable heterogeneous collections of information, databases, or devices (such as hand-held devices) to collaborate with each other. This paper illustrates the advantages of SyD by describing a prototype calendar of meetings application. This application highlights some of the technical merits of SyD by exploiting the use of coordination links. Based on the underlying event-and-trigger mechanism, these links allow automatic updates as well as real-time enforcements of global constraints and interdependencies, not available with existing calendar applications. Additionally, the calendar application illustrates coordination among heterogeneous devices and databases, formation and maintenance of dynamic groups, mobility support through proxies, and performance group transactions across independent data stores.

1. INTRODUCTION

System on Devices (SyD) middleware technology was introduced in [1-4] to address the key problems of heterogeneity of device, data format and network, and that of mobility. SyD combines ease of application development, mobility of code, application, data and users, independence from network and geographical location, 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.

Limitations of Current Technology: 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, as further elaborated in Section 5.

The calendar of meetings 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 [1,3]. 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 contains an overview of SyD and illustrates a high-level design of the calendar application. Section 3 provides detail of the current prototype implementation.

Creating and maintaining a dynamic group of entities, as in a meeting, is integral to SyD. We propose SyD coordination links, which may 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 more complex changes. *Negotiation links* enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction. Section 4 introduces the SyD coordination links and gives a detailed logical design of the calendar application.

We also present a prototype implementation of the SyD middleware and the SyD-based calendar application. This illustrates some key SyD features such as coordination among heterogeneous devices and databases, formation and maintenance of dynamic groups, providing mobility support through proxies, and performing group transactions across independent data stores. The implementation is done in Java using Oracle database for data stores. The application is currently implemented on a collection of handheld devices on a wireless LAN. Section 5 presents details of the calendar application implementation. In Section 6, a comparison of the calendar application with the existing similar applications is presented. Section 7 concludes the paper.

2. OVERVIEW OF SYD

In this section, we 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 (PDAs), or even in devices such as mobile

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

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.

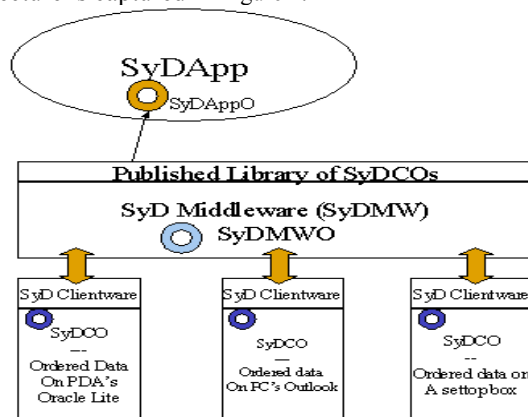


Figure 1. SyD Architecture

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 link with each other in an interdependent fashion to enable object composition and atomic transactions over multiple objects, provided by a linking module.

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 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. They rely only on these 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, be they formal databases or ASCII lists, stored on PDAs or on mainframes, 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. CURRENT PROTOTYPE IMPLEMENTATION

We have developed a prototype implementation of SyD middleware and several SyD-based applications. In this section, we describe the corresponding detailed architecture.

3.1 Detailed Architecture

Figure 2 depicts the layered architecture of SyD runtime environment in the current implementation. SyD in this environment, provides distribution transparency and management to SyD-based application development, therefore, greatly reducing the development, implementation, deployment, and maintenance time (software life cycle) for designers/programmers of distributed applications on heterogenous mobile devices and environments.

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.

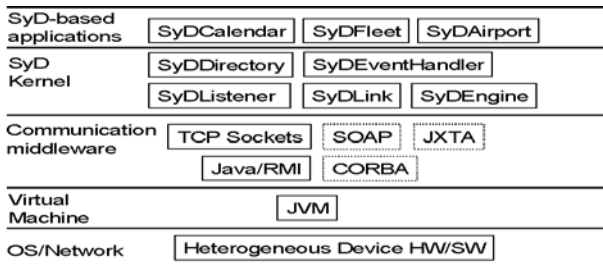


Figure 2. SyD Runtime Environment

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 foot-print 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 heterogenous 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 2 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 5 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 2 and Figure 3):

- 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 services (server functionalities) as “listeners” locally on the device and globally via directory services. It 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:

- 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.
- Registering services as listeners using SyDListener: SyDListener registers application methods as remote listeners for remote invocations locally in RMI registry and globally in SyDDirectory.
- 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 (group functionality). SyDEngine executes remote services by invoking the SyDListener module.

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 portion) 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”.

3.2 SyD-Based Design of Calendar Application

As an example of the SyD framework, we use an individual’s (say Phil) calendar as a basic SyD object. This calendar could be stored on a PC, a web server, or a PDA that supports the SyD clientware software. The SyD clientware names and publishes the SyD object, Phil_calendar_SyD, and registers this object with the SyD middleware, SyDMW. A SyDApp constructs an object called Calendars_of_phil+andy+suzy_SyDAppO that “links” together and defines a set of methods that can operate on the calendar objects of all three individuals, assuming Andy and Suzy also publish their calendar SyD objects for use by SyDApps via the SyDMW.

The SyDAppO called Calendars_of_phil+andy+suzy_SyDAppO may support the following methods: Find_earliest_meeting_time(), Change_meeting_time_to_next_available(), etc. The SyDAppO, Calendars_of_phil+andy+suzy_SyDAppO, would be instantiated from a general class called Calendars_of_committee_SyDAppC that could be provided by a vendor or written by users themselves.

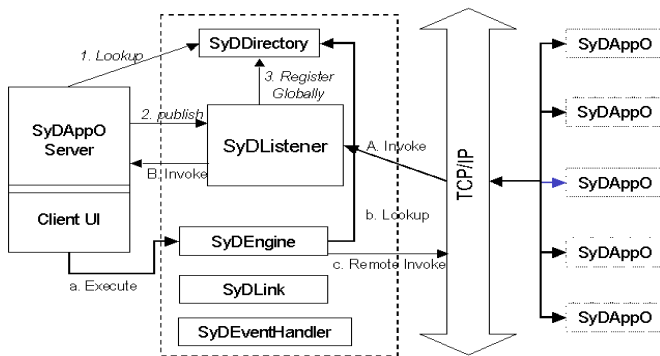


Figure 3. SyD Kernel architecture and the interactions between modules and application objects.

What is important to note is that the execution of the SyDApps is reliant on the SyDMW for locating, communicating with, and operating on the various SyD objects that could be distributed across the internet, intranet or an ad hoc heterogeneous network, and the SyDMW could be hosted on a local machine or hosted through an Application Service Provider (ASP). The SyDMW is also responsible for QoS issues as required by the SyDApps.

3.3 How does this application work 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 or a PC or a 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.

4. COORDINATION LINKS

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. We define the links and then show how the links are employed to establish meetings.

4.1 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 link creation time and a link expiry time. Subscription links are useful for automatic updates and synchronization and negotiation links, with specified logical constraints, enforce interdependencies.

4.2 Operations on Links

A link allows several kinds of group operations on a set of related entities. Let an entity X be linked to entities Y and Z, which may in turn be linked to other entities. A change in X may trigger changes in Y and Z, or X can change only if Y and Z can be successfully changed. We define two primary types of coordination links, namely, subscription link and negotiation link. Subscription link allows automatic flow of information from a source entity to other entities that subscribe to it. This can be employed for synchronization as well as more complex changes. Negotiation links enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction.

Following are the operations of SyDLinks:

1. Link database creation: 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.

2. Link creation: The application can maintain a logical connection by creating a link between the various entities. All an application has to do is specify a list of users who have to be linked. For example, if the application needs to link users X, Y, Z based on their availability at a particular time, then the SyDLink module will negotiate with all the users (1) and if and only if all the users are available (2) at that particular time, links will be created (3) between the users. If any user is not available at that time then no links will be created. So links can be created automatically based on the availability.

3. Automatic conversion of *tentative* links to *permanent* links: 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. 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 link in the group with highest priority, from tentative to permanent.

4. Link Deletion: Whenever SyD_deleteLink() method is invoked, two actions take place. First, it checks if there are any waiting links for the current link being deleted. If there are, then the waiting link (or group of waiting links) with the highest priority is converted from tentative to permanent status. Second, if the link being deleted is from user A to B, then SyD_deleteLink() is invoked on B via SyDEngine. Consequently, all links logically associated together are deleted in a cascading manner.

5. Method Invocation: When a subscription link is established between two entities X and Y then execution of a method M1 on X should automatically trigger execution of method M2 on Y. In order to facilitate this a SyD_LinkMethod table is maintained. All the details regarding the source and destination methods and the destination users is maintained in this table. The application programmer has to include a call to check whether the current method being executed is listed in the SyD_LinkMethod table. If it is, then the respective method names and user names have to be sent to the SyDEngine, to be executed remotely.

6. Link expiry: Periodically, the local event handler triggers a method which checks for links whose expiration times have been surpassed. All such links are automatically deleted.

4.3 Classification and Semantics of Coordination Links

In the following, the phrase "Change X" is employed to refer to an action on X; "Mark X" refers to an attempted change, which triggers any associated link without actual change on X.

Subscription Link: Mark A; if successful Change A then try: Change B, Change C . Note that a "try" may not succeed).

Negotiation-and Link: Change A only if B and C can be successfully changed (This implements an atomic transaction with "and" logic).

Semantics (may not be implemented this way):

Mark A for change and Lock A

If successful

Mark B and C for change and Lock B and C

If successful

Change A; Change B and C

Unlock B and C

Unlock A

Negotiation-xor Link: Change A only if exactly one of B and C can be successfully changed.

(implements atomic transaction with "xor" logic and can be extended to *exactly k out of n*).

Semantics:

Mark A for change and lock A

Mark B and C for change. Obtain locks on those entities that can be successfully changed.

If obtained exactly one lock

then Change A; Change the locked entities.

Unlock entities

Negotiation-or Link: Change A only if at least one of B and C can be successfully changed. (Implements atomic transaction with "or" logic and can be extended to *at least k out of n*)

Semantics:

Mark A for change and lock A

Mark B and C for change; Obtain locks on those entities that can be successfully changed.

If obtained at least one lock

then Change A; Change the locked entities.

Unlock entities

4.4 Cancel Meeting Scenario in SyD Calendar Application

The Calendar application 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.

1. Check to see if there are any associated waiting links.

2. If so,

2.1 Automatically convert status of waiting links from tentative to permanent through SyDEngine.

3. Delete the local link.

4. Invoke deleteLink on the rest of the associated links.

5. Update the calendar database of the user.

6. SyDEngine gets the remote URL of the associated users from the SyDDirectory Service and invokes the necessary method.

7. Repeat steps 1 through 6 for each associated user.

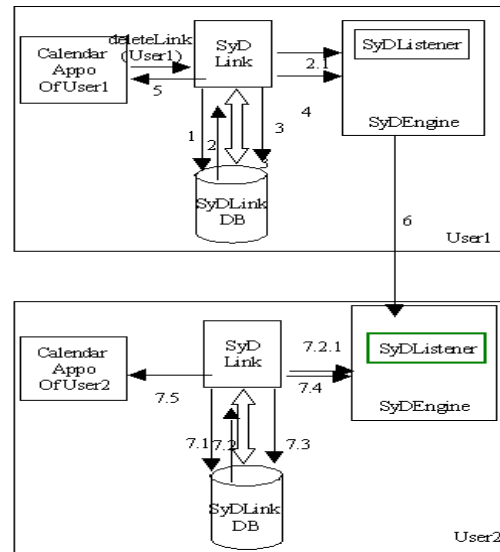


Figure 4. UML activity diagrams showing execution of SyD links for negotiation-or for three SyD objects A, B, and C where A is the activating object.

5. DETAILED DESIGN OF CALENDAR APPLICATION

A typical scenario for a meeting setup is that a user enters the dates between which he wants to setup a meeting and also the people whom he wants to call for the meeting. A list of open slots common to all the participants appears, and the meeting is scheduled with a click on the desired slot. There may be additional design criteria, such as *A* and *B* are must-attendees, but one of *C*, *D*, *E* would suffice. After finding an empty slot, the meeting can only be tentatively scheduled, because during the delay between the enquiry for the empty slots and the actual scheduling, the status of the participants may have changed. Also, someone may want to voluntarily change his schedule later, potentially causing the meeting to become tentative or get canceled. Likewise, a higher priority meeting may bump a previously scheduled meeting. A tentative meeting may be confirmed at a later time due to some cancellations. Also, an executive may want to delegate the task of scheduling a meeting to a staff who would be able to call the meeting with the transferred authority of his boss.

Here is a simple scenario for a meeting setup in the calendar application. User *A* wants to call a meeting between dates *d1* and *d2* involving folks *B*, *C*, *D* and himself. The first step is to find the empty slots in everybody's calendar. Finding empty slots can be carried out as follows: User *A* enters the date and the people to participate for the meeting in a GUI form. The system will then (i) query each table for free slots which fall between dates *d1* and *d2*, (ii) ensure that all participants confirm, before the subsequent actions would be valid, (iii) find common empty slots by intersecting the views returned from calendars, and (iv) present these slots to user *A*. User *A* then clicks on a desired empty slot. This causes a

series of steps. A negotiation-and link is created from user A 's slot to the specific slot in each calendar table. The link is triggered by any change in A 's slot. Choosing the desired slot attempts to write and reserve that slot in A 's calendar, and that triggers the negotiation-and link. The "action" of this link is to:

- (i) Query each table for this desired slot, and ensure that it is not reserved, and reserve this slot.
- (ii) If all succeed, then each corresponding slots at A , B , C and D create a negotiation link back to A 's slot.

Otherwise, for those folks who could not be reserved, a tentative back link to A is queued up at the corresponding slots to be triggered whenever the status of the slot changes. The forward negotiation-and link to A , B , C and D are left in place. In addition, back subscription links to A from others are created. These subscription links inform A on subsequent changes in the other participants and would help A decide (based on a threshold) to cancel this tentative meeting altogether or try another time slot.

Assume that C could not be reserved. Thus, C has a tentative back link to A , and others have subscription links to A . Whenever C becomes available (or its status changes for this slot), if the tentative link back to A is of highest priority, it will get triggered, informing A of C 's availability, and will attempt to change A 's slot to be reserved. This would trigger the negotiation-and link from A to A , B , C , and D , which in turn go through another round of negotiations to reserve each of the four folks. Assuming that all succeed (C already is available), all corresponding slots are reserved, and the target slots at A , B , C and D create negotiation links back to A 's slot. Thus, a tentative meeting has been converted to committed.

Now suppose, D wants to change the schedule for this meeting to another slot. This attempt by D would trigger its back link to A , which would trigger the forward negotiation-and link from A to A , B , C and D . If all succeed, then a new duration is reserved at each calendar with all forward and back links established. If not all can agree, then D would be unable to change the schedule of the meeting. Similar actions would take place if D simply wanted to cancel, or if D is involuntarily asked to bump this meeting by another entity. A higher priority request to D to commit to another meeting would bump this meeting, and then this meeting would become tentative, with D (instead of C) having a tentative link to A .

Suppose B is a supervisor (a higher priority entity). Then, as a result of the meeting schedule, A would not be able to establish a negotiation back link from B , but only a subscription back link. This retains B 's ability to change his schedule at will. If B does change his schedule, this change will trigger the subscription back link to A , and A will negotiate with A , C and D for the change. If one fails, then the meeting becomes tentative, which results in conversion of all back links to A to subscription links and conversion of the back link from B to A into a tentative link queued up at B 's slot awaiting change in B 's status.

As a second example, suppose A wants to schedule a meeting with a quorum of 50% among the faculty of Biology and at least two faculties from Physics and, in addition, B and C are must attendees. This meeting can be scheduled by

establishing a negotiation-and link to B and C , a negotiation-or link (at least k of n type) to all in Biology with n = size of biology faculty, and k = 50% of n , and a negotiation-or link to all in Physics with k = 2. On successful reservation of all entities, slots are reserved at the accepting entities and negotiation back links to A are established.

From the non-accepting entities, a tentative link back to A is established, thus allowing these faculty members to reserve their slots in future, if they become available. A cancellation by someone in Biology will trigger the back link to A and in turn will trigger the negotiation links from A to all. In particular, the negotiation-or link to Biology faculty will make a determination if 50% can make it, then grant the cancellation to the Biology faculty requesting it. If the quorum falls below 50% among Biologists, the negotiation-or links to the remaining Biologists are triggered, and only if an additional commitment is found, is the cancellation request granted.

5.1 Software Architecture and Implementation

Each user has a database embedded in his/her device (mobile or static) and we also assume that each device is always available. (At the first look, this assumption may seem unrealistic, but this can be implemented using a proxy if the device is inactive. In the calendar application, each user can initiate a meeting (either tentative or permanent) with the users he has access with, depending upon the availability of the time slots, can cancel a meeting (if he is the initiator of the meeting), or can drop out of the meeting if the constraints are still met. Setting up or cancellation of meetings triggers the update of local and remote database schemas. The security overhead for accessing remote databases can be overcome by using proper cryptographic techniques while sending the messages. The users involved in the meeting are notified about the details of the meeting using an e-mail message.

This application highlights the automatic scheduling of events, or in general, automatic triggering of the system of the databases. The coordination is accomplished by using the negotiation links as described in Section 3. All changes happen in real-time, which is one feature the current mobile devices are lacking.

5.2 Use of a Name Server and Proxy

Our SyD objects are assumed to always have web presence. For this, if a SyD calendar 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.

Further extending the idea of a proxy, if the SyD client object does not have the necessary capability to perform all the desired actions, this functionality could be made available from the proxy. Taking the example of a SyD calendar object A , if the embedded system on which A is being implemented does not have the capability of using a database server, the database server could potentially be placed on the proxy for A . The information regarding the proxies and peer SyD objects, proxies and their allotment to SyD objects should be available to all proxies and SyD objects. The use of SyDDirectory (Name server) solves this purpose. The main functionality of

the Name Server is to store information about all proxies and Syd objects and map each Syd object to at least one proxy.

In our implementation of the calendar system, the Syd calendar objects are implemented on the Compaq iPAQs with Windows CE operating system. The following steps take place in the calendar application:

1. The proxies register themselves with the Name Server when the application server starts.
2. The clients relay their information to the Name Server, and get back a proxy object, which acts as the proxy for it.
3. The client sends an HTTP request for the servlets on the application server at the proxy using the proxy object.
4. A client object is obtained at the proxy and is stored in the session of the application server.
5. For the whole session, the proxy contacts the client using the reference stored in the session.

The communication between the proxy, client, and the Name Server is achieved by using the Java RMI method calls. Vectors are used as the data structure to store the client and proxy information at the Name Server. A hash table is used for the mapping of clients and proxies.

5.3 Event-based Triggers

Oracle provides a special feature called Java Stored Procedures in 8i and later versions. In this, the Java classes are stored in the database as a procedure, and can be executed from inside the database as a trigger. Using this technique, one could connect to the remote peer databases and do the necessary updates following the triggering event [13]. The implementation of the current version undertakes Oracle triggers and Java store procedure route to implement the specific functionality, since all the Syd calendar objects are Oracle databases. But in the future versions, we will implement the triggers in the middleware.

One main disadvantage of the Oracle triggers is database portability. This kind of triggering could not be implemented on a client with a database other than Oracle. Our Syd model does not allow any dependencies on a specific database. In order to achieve this we would use middleware triggers. The triggers exist in the middleware and get executed upon a specific change in the database state. Assuming the database changes are occurring only from the Calendar User Interface, this could be easily achieved by checking for the necessary change in the database.

5.4 Security/Authentication

One of the primary issues in any distributed application is security. To make the calendar application secure, it has to be ensured that only authorized users have access to remote databases. In order to ensure secure transactions, the client application trying to access a remote database must be authenticated. For the purpose of authentication, each user is provided with a unique user id and password. Each user's database also has a table containing the user id and password of authorized users. A 32-bit key is used to encrypt the user id and password. Encryption is done using the Tiny Encryption Algorithm [22]. The encrypted user id and password are sent as parameters along with every request. On the server side, before processing the request, the user id and password are decrypted. These are then compared against a list of users who

have access permission. If it is a valid user, the request is processed and a response is generated.

6 COMPARISON TO EXISTING CALENDAR APPLICATIONS

There are a number of different calendar applications offered today; some of which are Microsoft Outlook, Groupwise, and Lotus Notes. The calendar application presented here is not targeted to compete with the services rendered by the above products. Instead, the calendar 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, which defines the entire system. There are various 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 email 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. There is also 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 a 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 the 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]. The SyD middleware implements a systematic global logic, because of which all these new features can be implemented with ease in programming.

7 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.

8 REFERENCES

1. S. Prasad et al., "Mobile Fleet Applications using SOAP and SyD Middleware Technologies," *Proc. Comm., Internet and Information Tech.*, 2002, pp. 426-431.
2. V. Madiseti, "SyD:A Middleware Infrastructure for Mobile iAppliance Devices," *iAppliancesWeb* 2002.
3. S. Prasad et al., "Enforcing Interdependencies and Executing Transactions Atomically over Autonomous Mobile Data Stores using SyD Technology," to appear *Proc. Mobile Wireless Networks, ICDCS* 2003.
4. W. Xie et al., "Supporting QoS-Aware Transaction in the Middleware for SyD," to appear *Proc. Mobile Distr. Computing, ICDCS* 2003.
5. Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 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. John Wiley & Sons, NY, 1997.

8. K. Ramamritham, Real-Time Databases, *Distributed and Parallel Databases* 1(1993), pp. 199226, 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 recomment., <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 Distr. Heterogeneous, Autonomous Databases, *ACM Comp. Surveys*, 22:3, 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, D. Keirse. Determining relationships among attributes for interoperability of multi-database systems. *The 1st Int. Workshop on Interoperability in Multidatabase Systems*, pg. 251-257, Kyoto, Apr 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, B. Bhargava. A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases. *ACM SIGMOD Record*, 24(3): 44--49, 1995.
20. M. H. Dunham and V. Kumar. Location dependent data and its management in mobile databases. *Proceedings of DEXA Workshop*, pages 414-419, Vienna, Austria, 1998.
21. W. Emmerich, *Engineering Distributed Objects*, Wiley, 2000.
22. D. Wheeler and R. Needham, "TEA, A Tiny Encryption Algorithm," *Fast Software Encryption: 2nd Int'l. Wkshp, vol. 1008 of Lec. Notes in Comp. Sc.*, pg. 363-366, 1994.