SyD: A Middleware Testbed for Collaborative Applications over Small Heterogeneous Devices and Data Stores*

Sushil K. Prasad,Vijay Madisetti[2], Shamkant B. Navathe[3], Raj Sunderraman, Erdogan Dogdu, Anu Bourgeois, Michael Weeks, Bing Liu, Janaka Balasooriya, Arthi Hariharan, Wanxia Xie[3], Praveen Madiraju, Srilaxmi Malladi, Raghupathy Sivakumar[2], Alex Zelikovsky, Yanqing Zhang, Yi Pan, and Saied Belkasim

Computer Science Department, Georgia State University
[2]School of Electrical and Computer Engineering, Georgia Institute of Technology
[3]College of Computing, Georgia Institute of Technology

Abstract. Developing a collaborative application running on a collection of heterogeneous, possibly mobile, devices, each potentially hosting data stores, using existing middleware technologies such as JXTA, BREW, compact .NET and J2ME requires too many ad-hoc techniques as well as cumbersome and time-consuming programming. Our System on Mobile Devices (SyD) middleware, on the other hand, has a modular architecture that makes such application development very systematic and streamlined. The architecture supports transactions over mobile data stores, with a range of remote group invocation options and embedded interdependencies among such data store objects. The architecture further provides a persistent uniform object view, group transaction with Quality of Service (QoS) specifications, and XML vocabulary for inter-device communication. This paper presents the basic SyD concepts and introduces the architecture and the design of the SyD middleware and its components. We also provide guidelines for SyD application development and deployment process. We include the basic performance figures of SyD components and a few SyD applications on Personal Digital Assistant (PDA) platforms. We believe that SyD is the first comprehensive working prototype of its kind, with a small code footprint of 112 KB with 76 KB being device-resident, and has a good potential for incorporating many ideas for performance extensions, scalability, QoS, workflows and security.

Keywords: Mobile Servers, SyD Coordination Bonds, Object and Web Service Coordination, Atomic Transactions, Application-Level QoS.

1 Introduction

Requirements for a Middleware Platform: There is an emerging need for a comprehensive middleware technology to enable development and deployment

* This Research was partially supported by Georgia Research Alliance's Yamacraw Embedded Software Contract #CLH49 and #DLN01. The SyD middleware and demo application codes can be downloaded from www.cs.gsu.edu/~yes.

of collaborative distributed applications over a collection of mobile (as well as wired) devices. This has been earlier identified as one of the key research challenges [3, 14]. Our work is an ongoing effort to address this challenge, and this paper reports our first prototype design and its implementation. We seek to enable group applications over a collection of heterogeneous, autonomous, and mobile data stores, interconnected through wired or wireless networks of various characteristics, and running on devices of varying capabilities (pagers, cell phones, PDAs, PCs, etc.). The key requirements for such a middleware platform are to allow:

1. Uniform Connected View: Present a uniform view of device, data and network to ease programmer's burden. Provide a device-independent and a persistent (always connected) object-view of data and services, so as to mask mobility and heterogeneity. Allow multiple data models and representations on device data stores.
2. Distributed Server Applications on Small Devices: Enable developing and deploying distributed server applications possibly hosted on mobile devices. Support atomic transactions across multiple, independent, and heterogeneous device-applications.
3. High-Level Development and Deployment Environment: Enable rapid development of reliable and portable collaborative applications over heterogeneous devices, networks and data stores. Provide a general-purpose high-level programming environment that uses existing sever applications and composes them to create, possibly ad-hoc, integrated applications rapidly.

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. It requires explicit and tedious programming on each kind of device, both for data access and for inter-device and inter-application communication. The application code is specific to the type of device, data format, and the network. The data store provides only a fixed set of services disallowing dynamic reconfiguration. Applications running across mobile devices become complex due to lack of persistence and weak connectivity. A few existing middlewares have addressed the stated requirements in a piecemeal fashion. Limitations include only client-side programming on mobile devices, a restricted domain of applications, or limited in group or transaction functionalities or mobility support, as further elaborated in Section 8.

SyD Solution: System on Mobile Devices (SyD) is a new platform technology that addresses the key problems of heterogeneity of device, data format and network, and 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 hand held devices. SyD separates device management from management of groups of users and/or data stores. Each device is managed by a SyD deviceware that encapsulates it to present a uniform and persistent object view of the device data and methods. Groups of SyD devices are managed by the SyD groupware that brokers all inter-device activities, and presents a uniform world-view to the SyD application. The SyD groupware directory service enables SyD applications to dynamically form groups of objects hosted by devices. The SyD groupware enables group communication and other functionalities across multiple devices. Section 2 presents the detailed SyD architecture.

Contributions and Significance: The primary contributions of our work presented here are broadly two-fold [10, 16, 17, 20–22, 26].

1. A proof-of-concept, working middleware, unique of its kind, that enables distributed server and collaborative applications over multiple small mobile devices, possibly hosting data stores. This is as much an engineering feat as is a middleware design. The foot-print of the entire SyD kernel code is 112 KB, out of which only 76 KB is currently device-resident; the rest is for directory and global event handling. For even smaller devices, this can be further reduced to 42 KB. The execution time work space used by SyD is 4-8 MB, exclusive of JVM and OS.
2. A methodology to rapidly develop and deploy robust distributed collaborative applications, and an execution platform to deploy such applications, while masking mobility and heterogeneity from application programmers, and allowing inter-device constraints and atomic transactions.

Hardware/Software Platform: Various technologies employed to prototype SyD kernel testbed are as follows:

1. HP's iPAQ models 3600 and 3700 with 32 and 64 MB storage running Windows CE/Pocket PC OS interconnected through IEEE 802.11 adapter cards and a 11 MB/s Wireless LAN.
2. Jeode EVM personal Java 1.2 compatible, implementing Java Virtual Machine; KVM/MIDP for cellphone emulator.
3. Instant DB version 3.26 on iPAQ for databases for various applications; Oracle 8i DBMS on PC as a directory server.
4. TCP Sockets for remote method invocation and JAVA RMI for local method execution through reflection.
5. μCode [15] version 1.03 as the mobile agent frame work on the iPAQ.
6. XML standard for all inter-device communication.

We will not be able to provide in-depth description of all aspects of SyD middleware, but will attempt to be complete while highlighting our key contributions. This paper is organized as follows. Section 2 provides a broad overview of SyD and presents the SyD prototype implementation and the current implementation architecture of SyD and SyD-based applications. The next three sections describe the three key aspects of SyD: Section 3 describes the SyD Listener module, which enables data hosting and serving capability in small, mobile, devices. Section 4 describes a range of options available in SyD for invoking individual and group remote methods. Section 5 then describes one key innovation, namely SyD coordination links/bonds which enable distributed object coordination and rapid ad-hoc application prototyping. Section 6 summarizes how SyD-based applications are developed in our framework, and mentions a few sample SyD applications. In Section 7, we attempt to place SyD middleware among the emerging middleware technologies and assess the level by which SyD currently meets its design goals. Section 8 presents the related work. Section 9 contains a summary and future work.

2 SyD Architecture Overview

In this section, we describe the design of SyD and related issues, and highlight the important features of its architecture. Each individual device in SyD may be a traditional database such as relational or object-oriented database, 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 in that they do not share a global schema. The devices in SyD cooperate 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 shown in Fig. 1.

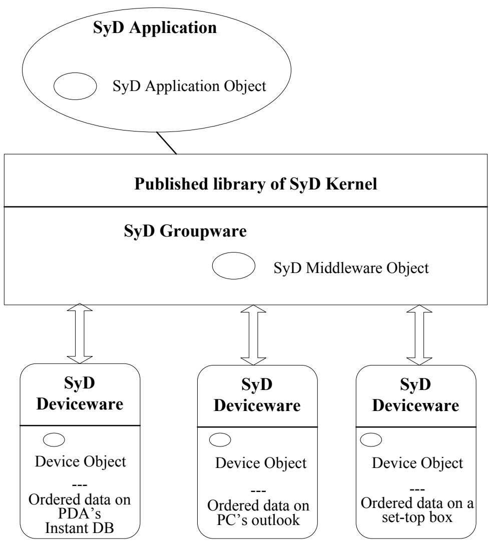

Fig. 1. SyD Framework

The SyD framework has three layers.

1. At the lowest layer, individual data stores are represented by device objects that encapsulate methods/operations for access, and manipulation of this

data. The SyD Deviceware consists 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. Object composition and execution of atomic transactions over multiple objects are provided by a bonding module.

2. At the middle layer, there is SyD groupware, a logically coherent collection of services, APIs, and objects to facilitates the execution of application programs. Specifically, SyD groupware consists of a directory service module, group transactions and global event support, with application-level Quality of Service (QoS).
3. At the highest level are the applications themselves. They rely only on these groupware and deviceware SyD services, and are independent of device, data and network. These applications include instantiations of server 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 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. SyD groupware provides the communications infrastructure between SyD Applications (SyDApps), in addition to providing QoS support services for SyDApps. SyDApps are applications written 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. 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 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.

Current Prototype Implementation: We have developed a prototype testbed of SyD middleware that captures the essential features of the SyD's overall framework and several SyD-based applications. We have designed and implemented a modular SyD kernel in Java, which includes the following five modules (Fig. 2):

1. The SyDDirectory provides publishing, management, and lookup services to SyD device objects and their proxies, and users.
2. The SyDListener enables SyD device objects to publish their services (server functionalities) as "listeners" locally on the device and globally via the directory services. It responds to invocations of server methods by the users and other application objects on SyD network.

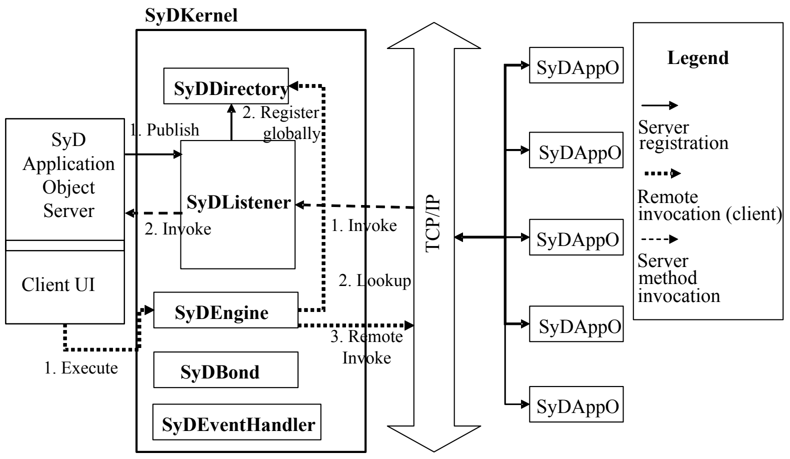

Fig. 2. SyD kernel architecture, and interactions among modules and SyD application objects (SyDAppO)

3. The SyDEngine allows users to execute single or group services remotely and aggregate results. Its sub-modules extend it with transactions with QoS.
4. The SyDBond module enables a SyD entity to link to other entities for automatic updates and to create and enforce interdependencies.
5. The SyDEventHandler module handles local and global event registration, monitoring, and triggering.

Disconnection Tolerance via Proxy: The SyDDirectory keeps track of application objects and their associated devices via location information (IP plus port number). Server applications can register their proxies, too. SyDDirectory actively maintains the availability of applications hosted on mobile devices versus their proxies via a “live bit” in the directory entries; this bit can be set/reset by an application on power-on/power-off or by SyDEngine on time-outs. The device’s location is also tracked and kept current in the SyD directory. If the device is inaccessible, due perhaps to an intermittent disconnection or battery discharge, then, after a timeout, the SyDEngine invokes the proxy application to complete the requested service and resets the “live-bit” of the application with the SyD Directory. The application on the mobile device, when online, synchronizes intelligently with its proxy updates and sets its “live-bit”. The assumption here is that an application running on a mobile device is most up-to-date. (Thus, a user calendar’s proxy allows a meeting to be only tentatively scheduled, which after synchronization with the mobile calendar may get converted into a permanent meeting; on the other hand, if mobile calendar is online, permanent meeting can be setup in real time). A detailed implementation of a proxy module incorporating and extending these functionalities is underway.

Applications with similar or inter-related services can be aggregated to form SyD groups for ad-hoc group functionality of services in SyD-based applications.

The SyDDirectory maintains entries for SyD groups by creating, changing, and deleting groups. The SyDEngine accesses group information to invoke group services.

Thus, SyD has some aspects of reflection-orientation through its directory service, which allows inspection, and adaptability through proxy management, group dynamics, and reference decoupling. We discuss the other key modules of SyD in the subsequent sections.

3 Uniform Object View - SyD Listener

Our SyDListener is a key component providing uniform object view of various data sources and server applications [17]. It enables application serving, and data store and web service hosting capabilities in small devices, which are traditionally thought of as mere client devices. It is effectively a stand-alone lightweight extended SOAP server enabling XML-based inter-device interactions. SyDListener is implemented as a multi-threaded application enabler with simple persistence management and asynchronous invocation functionality for (i) Personal Profile on Connected Device Configuration (CDC) devices, suitable for high-end PDAs [11], and (ii) J2ME Mobile Information Device Profile (MIDP) on Connected Limited Device Configuration(CLDC) devices, such as mobile phones [12].

The SyDListener module has dual responsibilities: (1) registering server applications for remote method invocations, and (2) performing local method execution and responding with results when remote requests are received [17]. Service registration is done locally to local object repository, and globally to SyDDirectory. There are three important classes in SyDListener module: SyDRegistrar, SyDListener, and SyDDelegate (Fig. 3). SyDRegistrar performs registration of server and its proxy, initially upon application deployment and after each reconnection or power-on of the device, ensuring that subsequent invocations are directed to the server object, not to its proxy. As part of the initialization of the server object, it typically should have a synchronization mechanism, possibly application-specific, with its proxy. The SyDListener class is instantiated at server side and performs listening for, parsing and invoking local methods in response to remote invocations. The SyDListenerDelegate is at the client side and performs communication with SyDListener, including transmission of request messages and receiving of results. The SyDListenerDelegate acts as an adaptor for SyDListener and hides all the communication details from clients. This means that the communication method, TCP sockets in this case, can also be changed without any influence on clients.

CDC/JVM version: The first version of SyDListener is built upon TCP sockets and Java RMI. SyDListenerDelegate and SyDListener communicate through TCP sockets. SyDListener locally accesses active service objects from RMI registry (Fig. 3). Inside the SyDListener, we integrate Java reflection mechanism with RMI. This implementation is single-threaded and only provides synchronous invocation. The footprint of this implementation of the whole listener module is 10.3 KB.

CLDC/KVM Version: The listener architecture provides flexibility in that

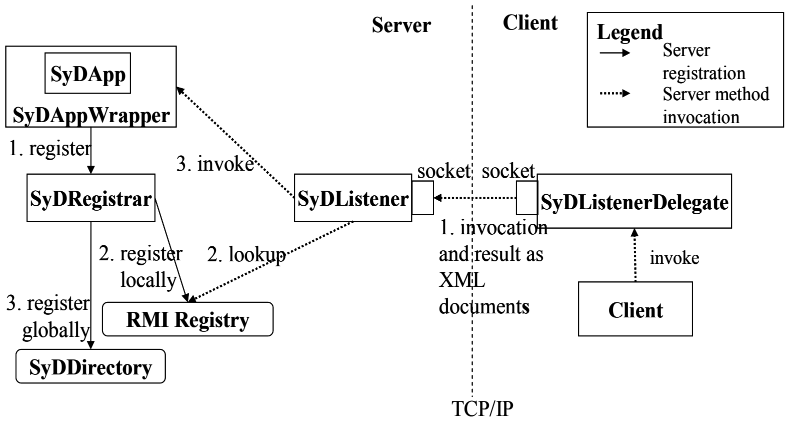

Fig. 3. SyDListener module architecture using TCP sockets

we could replace RMI with other technologies, and limit the change within SyDListener and the service application. This was employed to yield a second version of listener for CLDC devices, and is built upon J2ME Mobile Information Device Profile (MIDP). It utilizes the limited programming APIs available on MIDP and acts as a multi-threaded application server, which manages the lifecycle of SyD applications without using Java RMI. It provides simple persistence management to service objects and asynchronous method invocation to service clients. The footprint of the CLDC/KVM version is 19.5 KB, an increase when compared to that of CDC/JVM version because of the missing base functionalities in CLDC and the new functionalities such as multi-threading.

Figures 4 and 5, respectively, present the total connection time for round-trip response time from SyDListener using synchronous requests (on iPAQs) and asynchronous requests (on cell-phone emulator). We employ varying (i) exponentially distributed inter-arrival times of invocation from clients, and (ii) normally distributed service times (grain-size of method execution) at the SyDListener. There is a good amount of saving in air time, as well as opportunities for concurrency in client application with asynchronous invocation, but that does entail more programming burden.

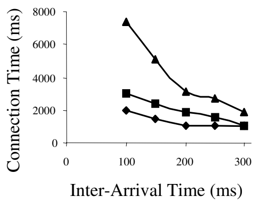

Fig. 4. Synchronous Invocation Model

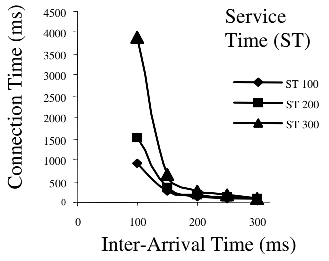

Fig. 5. Async. Invocation Model

4 Remote Methods and Transactions - SyD Engine and its Extensions

SyD recognizes the need for simplicity in method invocation on individual and groups of objects as well as the need for sophistication in light of mobility and heterogeneity of clients and servers. Therefore, it provides a range of mechanisms with sub-modules for tracking group invocations with real-time constraints, for adaptive transactions with application-level QoS and for on-server data processing with mobile capabilities. At the minimum, (i) language independence is enforced through a generic "invoke" method to SyDEngine, (ii) reference decoupling and group dynamicity are ensured through dynamic object/group IDs to address mapping via SyDDirectory, and (iii) XML vocabulary is employed to mask communication heterogeneity.

SyDEngine: The two major components of the SyDEngine are: the SyDDispatcher and the SyDAggregator. The SyDDispatcher module is responsible for dispatching method calls either on the local device or on to the remote device. At runtime, the SyDEngine looks up the SyD directory for current device location or proxy information and makes calls to the methods accordingly. The average round-trip time between SyDEngine on an iPAQ to SyD directory hosted on wireless LAN laptop is 218 ms out of which internal processing time of SyDEngine is only 7 ms. A local directory cache can cut this time further. The SyDAggregator module is responsible for aggregating multiple SyDDoC objects obtained from the SyDDispatcher. The possible operations currently implemented are: append, max, min and intersection, primarily over database tables. The SyDDoC utility provides a uniform data exchange capability throughout the SyD middleware and SyD-enabled modules. It is based on XML and is lightweight compared to DOM and SAX models of XML.

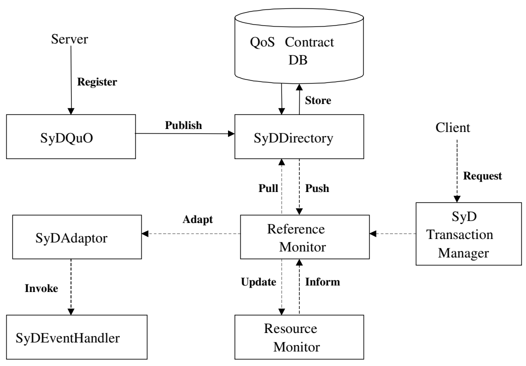

Fig. 6. QoS-aware Transaction

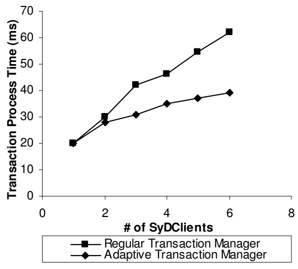

Fig. 7. Transaction Quality

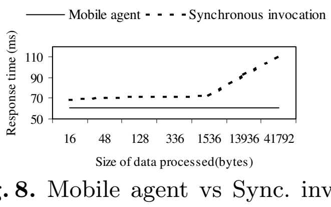

Fig. 8. Mobile agent vs Sync. invocation

The three modules, SyDDirectory, SyDListener and SyDEngine, along with the document utility module SyDDoc form the essential core of the SyD middleware, and account for a foot-print of only 60 KB with 42 KB being device-resident (the auxiliary technologies such as Jeode or Instant DB are not accounted for in this). The basic capabilities of SyDEngine are being enhanced by several extension sub-modules. For example, a client application can employ a TransactionTracker sub-module, residing off-device (possibly as a Web service object), which supports sophisticated asynchronous group invocations with real-time deadlines with timeouts [5]. Two additional sub-modules are as follows.

QoS-Aware Transactions: An extended architecture [25, 26] for the development and runtime support of QoS-aware transaction service is shown in Fig. 6. SyDQuO is the core component that describes and represents QoS contracts for transaction services. SyDDirectory provides QoS contract publish and lookup services for transactions. Reference Monitor watches all valid QoS contracts of currently running transactions and sends the updated resource thresholds to the resource monitor. Resource Monitor monitors the changes of resources such as CPU and memory. SyDAdapter manages the intra-transaction adaptation process while the Transaction Manager supports inter-transaction QoS. Fig. 7 shows preliminary performance results of the adaptive transaction manager, which sends load feedbacks to SyD clients whose transactions have the least priority to reduce the transaction transmission rates, thereby achieving better transaction rate overall. A detailed implementation of this sub-module is underway.

Agent-based Transactions: Mobile agent sub-module exploits the agent capability of on-server processing to save communication bandwidth. This submodule replaces the SyDEngine-SyDListener pair with μCode's Client and Server pair [15]. We have conducted preliminary experiments comparing the synchronous remote invocation (SI) through SyDEngine-SyDListener pair with the mobile agent (MA) [9]. Fig. 8 gives the comparison of SI with MA based on the size of the data processed. In the MA approach, data is processed at server sites and processed data is sent across the network. In the SI approach, data is collected from multiple devices and then processing takes place on the gathered data at the client and therefore results in higher response time. Effort is underway for SyDEngine to seamlessly switch to MA approach based on data size on agent-enabled SyD servers.

5 Distributed Coordination - SyDBond Module

A key goal of SyD is to enable SyD objects to coordinate in a distributed fashion. Each SyD object is capable of embedding SyD coordination bonds[1] to other entities enabling it to enforce dependencies and act as a conduit for data and control flows. Over data store objects, this provides active database like capabilities; in general, aspect-oriented properties among various objects are created and enforced dynamically. Its use in rapid configuration of ad-hoc collaborative

[1] Alternatively called "coordination links" [21, 22], or "Web bonds" in the context of Web services [18].

applications, such as inter-dependent set of calendars for a meeting setup [20], or a set of inter-dependent Web services representing airline, car rental, and hotel in a travel reservation application [4], has been demonstrated. The SyD bonds have the modeling capabilities of extended Petri nets and can be employed as general-purpose artifacts for expressing the benchmark workflow patterns [18, 19].

5.1 SyDBond Module

Coordination bonds enable applications to create contracts between entities and enforce interdependencies and constraints, and carry out atomic transactions spanning over a group of entities/processes. While it is convenient to think of an entity as a row, a column, a table, or a set of tables in a data-store, the concept transcends these to any SyD object or its component. There are two types of bonds: subscription bonds and negotiation bonds. Subscription bonds 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, needing data or event flows. Negotiation bonds enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction.

A SyD bond is specified by its type (subscription/negotiation), its status (certain/tentative), references to one or more entities, triggers associated with each reference (event-condition-action rules), a priority, a constraint (and, or, xor), bond creation and expiry times, and a waiting list of tentative bonds (a priority queue). A tentative bond may become certain if the awaited certain bond is destroyed. Let an entity A be bonded to entities B and C, which may in turn be bonded to other entities. A change in A may trigger changes in B and C, or A can change only if B and C can be successfully changed. In the following, the phrase "Change X" is employed to refer to an action on X (action usually is a particular method invocation on SyD object X with specified set of parameters); "Mark X" refers to an attempted change, which triggers any associated bond without an actual change on X.

- Subscription-and Bond: Mark A; If successful Change A then Try: Change B *and* Change C. A "try" may not succeed.
- Negotiation-and Bond: Change A only if B *and* C can be successfully changed.

Similar semantics can be defined with "or" and "xor" logic. A subscription bond from A to B is denoted as a dashed directed arrow from A to B. A negotiation bond from A to B is denoted as a solid directed arrow from A to B. A negotiation-and bond from A to B and C is denoted by two solid arrows, one each to B and C, with a "$*$" in between the arrows. Similarly, a "+" and a "$\wedge$" depict "or" and "xor" logic, respectively. A tentative bond, which is a negotiation bond in a waiting list, is shown as a solid arrow with cuts.

5.2 Modeling Dependencies Using Coordination Bonds

The modeling and execution capabilities of SyD bonds can be illustrated through typical scenarios of dependencies.

Producer-Consumer Dependencies: Fig. 9 shows how a classic relationship of a producer and consumer object can be bonded using two negotiation bonds. The *Place Order* method at a consumer object needs to ensure that the producer has enough inventories such that the corresponding *Accept Order* method will get executed successfully. Before guaranteeing this, the *Accept Order* probably will check the current and projected inventory. A negotiation bond is created from consumer to producer. This is the basic situation for deploying a negotiation bond. Once an order has been placed by the consumer and accepted by the producer, a subscription bond serves notice to *Dispatch Goods* method. Note that the bonds are useful within an object as well. Again before *Dispatch Goods* executes, it needs to ensure that consumers *Accept Delivery* method can be completed successfully (ensuring that enough space is available, for example) [18].

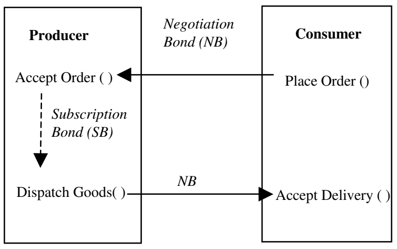

Fig. 9. Coordinating Producer-Consumer Objects

A Meeting Example: The potential of SyD bonds and their utility in modeling and enforcing contracts among coordinating objects can be further illustrated by a calendar of meeting example. For this application, we demonstrate here how an empty time slot is found, how a meeting is setup (tentative and confirmed), and how voluntary and involuntary changes are automatically handled. A simple scenario is as follows: A wants to call a meeting between dates d_1 and d_2 involving B, C, D and himself. The first step is to find the empty slots in everybody's calendar. A then clicks the desired empty slot. This causes a series of steps. A negotiation-and bond is created from A's slot to the specific slot in each calendar table (Fig. 10). Choosing the desired slot attempts to write and reserve that slot in A's calendar, triggering the negotiation-and bond. The "action" of this bond is as follows:

1. Query each table for this desired slot, ensure that it is not reserved, and reserve this slot.
2. If all succeed, then each corresponding slot at A, B, C and D creates a negotiation bond back to A's slot.

Else, for those folks who could not be reserved, a tentative bond back to A is queued up at the corresponding slots to be triggered whenever the status of the slot changes. The forward negotiation-and bond to A, B, C and D are left in place. Back subscription bonds to A from others are created to inform A of subsequent changes in the other participants and to help A decide to cancel this tentative meeting or try another time slot. Assume that C could not be reserved. Thus, C has a tentative back bond to A, and others have subscription bonds to A (Fig. 11). Whenever C becomes available, if the tentative bond 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 triggers the negotiation-and bond from A to A, B, C and D, resulting in another round of negotiation. If all succeed, then corresponding slots are reserved, and the target slots at A, B, C and D create negotiation bonds back to A's slot (Fig. 10). Thus, a tentative meeting has been converted to committed. Now suppose D wants to change the schedule for this meeting. This would trigger its bond to A, triggering the forward negotiation-and bond 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 bonds established. If not all can agree, then D would be unable to change the schedule of the meeting.

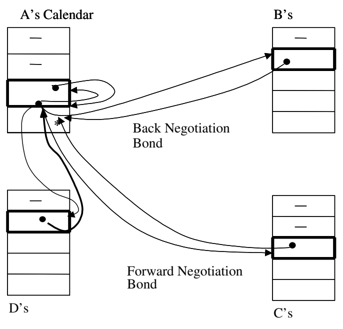

Fig. 10. A Scheduled Meeting

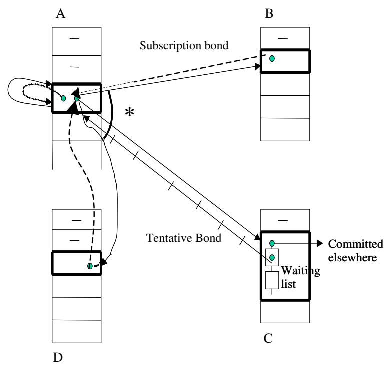

Fig. 11. A Tentative Meeting

5.3 Modeling Workflow Control Patterns

SyD bonds are being applied in distributed coordination among Web services, in particular, to express workflows (therefore, renamed Web coordination bonds). We illustrate the implementation of a few selected workflow patterns [1] using Web bonds. Further details are in [19].

Exclusive Choice (Xor-Split): Xor-Split is a point in a workflow where only one of possible paths is selected. Almost all the workflow modeling frameworks (except Petri net based) require considerable designer involvement to enforce XOR-Split. Web bonds eliminate this requirement by embedding *xor* logic among subscription bonds (Fig 12). If both paths get evaluated to true, only one will be selected. Negotiation bonds from B and C to A ensure that B and C could be executed only after A is completed.

Multiple instances with prior runtime knowledge: As number of instances is not known at the design time, most of the workflow models cannot

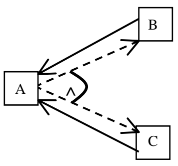

Fig. 12. Xor-Split

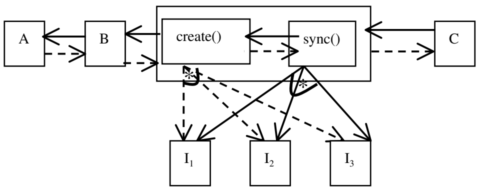

Fig. 13. Multiple instances with prior runtime knowledge

enforce this construct. Due to the dynamic creation and deletion facility of Web bonds, this can easily be enforced. Activity *B* passes the control to *create* sub-activity with instance creation parameters. Subscription bonds with *and* logic will be created with each instance (Fig. 13). At the same time, it makes sure that the sub-activity *Sync* creates negotiation bond with each instance. Having negotiation bonds to each instance, *Sync* activity ensures that it waits for all instances to be finished before passing the control to *C*.

6 Guidelines for Developing Collaborative Applications

SyD Methodology: SyD allows rapid development of a range of portable and reliable applications. The three-tier architecture of SyD enables development of applications in a flexible manner without explicit dependence on device, database, and network details. The steps involved in developing a SyD application are as follows.

Step 1: Model the application using SyD device objects, the device-based predefined or new objects, SyDMWOs, the available middleware objects, and SyDAppOs, new or previously designed application objects developed by an application platform developer. SyD directory aids in this search. Create, deploy and publish those server objects that are unavailable, by specifying their data and methods.

Step 2: Develop high level application server code by employing the objects/methods in Step 1. If new, composite server functionalities need to be published. Additionally, develop a separate client code, which typically is like a browser/GUI page.

Sample SyD Applications: Currently there are two key SyD applications, one each in the personal and enterprise domains. We implemented these using various technologies, including JDBC, SOAP, and SyD. The initial database programming using JDBC, and subsequently using SOAP, were carried out as student projects. These highlighted the cumbersome programming, lack of support for small devices, and the desired SyD-like features. The SyD-based development was by far the quickest, with more functionalities, due to high-level APIs of SyD (2-3 weeks each by 3-4 students), with comparable execution efficiencies.

Personal System of Calendar Application: The first is a calendar application in which each user has his own database that is either stored locally or on a proxy. The application can be logically divided into two parts, the server and the client. The server part includes all the methods that interact with the local data store and can be invoked remotely. The client part consists of the user interface which enables the user to interact with the application. One example of the SyD functionality is that the calendar application uses the SyDBond module to logically bond all members of a particular meeting together. A meeting can be rescheduled in real-time for all attendees by invoking the corresponding SyD Bonds by anyone participant [20, 21]. The time to setup a meeting between two iPAQs is about 1-2 secs, and increases slightly for larger groups. Automatic rescheduling and cancellation also have similar timing characteristics.

Enterprise Fleet Application: The second application is a truck fleet that operates an automated package delivery system. In this system, a user may connect through the Internet to the Web and Data Center (WDC), request delivery of a package, and give information pertaining to the package. The WDC passes this information to a depot, which schedules a pick-up [16, 23]. Average execution time for truck-to-truck communication querying about location using SyD is 885 ms, where as using JDBC is 1150 ms. Thus, the various abstraction layers (SyDEngine, Directory and Listener) are efficient and comparable to the older technology.

Developing Ad-Hoc Applications and Travel Application: SyD bonds can be employed to compose Web services (non-SyD-enabled) with an ad-hoc SyD application acting as a centralized coordinator. In [4], we demonstrate a travel application which allows for automatic rescheduling and cancellation of itineraries involving reservations for airline, hotel and car. Once an itinerary is decided and the trip is planned for the user, bonds are created and maintained in the user's SyDBond database. If the flight is canceled, then automatic cancellation of car and hotel reservations is triggered, thus easing the burden on the user to manually cancel all associated reservations. An ad-hoc application developer's nook provides a simple GUI-based interface for the application developer to initially set up and develop SyDBond-enabled coordinator applications.

7 Discussion

SyD has been driven by the current practical necessity to contribute fundamentally on certain aspects of middleware, database, Internet, mobile computing and related arenas. We briefly discuss SyD's place among the emerging middleware and distributed computing technologies, as well as where it is in terms of its goals. SyD as a middleware has a varying degree of flavors of object orientation (to mask heterogeneity), coordination orientation (for distributed coordination and ad-hoc group applications), and aspect orientation (distributed coordination among SyD objects with dynamic restructuring of embedded SyD bonds among coordinating objects). Additionally, it presents a reflection orientation with inspection, reference decoupling and dynamicity of groups through SyD-Directory, and adaptation through smart proxy management, real-time tracking and scheduling of sub-invocations and application level QoS.

The overriding philosophy of SyD has been to be simple and modular, from middleware design and implementation to application development and execution. The main goal has been to mask heterogeneity of device, data, language/OS, network and mobility for rapid collaborative application development. These have been achieved to the following extent: (i) device and data store through listener, (ii) language and OS through generic remote invocation semantics, (iii) network by XML vocabulary for all inter-device communication, (iv) mobility by (a) reference decoupling and replication/proxy through combined workings of SyD engine, directory service and registrar, (b) temporal decoupling through asynchronous invocation with various remote invocation options for mobile clients, and (c) an always connected object view through persistence/proxy mechanism for mobile hosts. A secondary goal has been to develop coordination mechanisms among objects to enable rapid ad-hoc application development, which has yielded SyD bond artifacts.

SyD's object model is persistent, is replicated with proxies, and encapsulates its distribution and replication policies, security, transaction support, etc. The object interface is generic XML/SOAP based for interoperability. Object reference is via XML strings dynamically bound through the directory service. Other aspects are as follows: (i) communication types are both synchronous and asynchronous; (ii) process model - listener is the object server, ensuring registration of objects and its proxies initially and after each disconnect; (iii) naming - object reference is generic XML using global id, id-to-address mapping is dynamic and location independent; (iv) synchronization - object implements its own synchronization mechanism for transaction support or for locking; SyD bonds are employed for inter-object coordination to enforce dependencies; (v) replication - proxy containing actively-managed replica or a functional substitute, object responsible for synchronization with proxy; used for persistence in connectivity for mobile objects; coordination bonds can be employed for coordination among object and its proxies; (vi) fault tolerance - faults/disconnections are supported through seamless switching between object and its proxies; varying level of transaction support and adaptive QoS properties are supported by SyDEngine and its sub-modules; (vii) security - objects encapsulate their own authentication mechanism; SyD relies on underlying network model for communication security.

8 Related Work

In this section we review several related middleware systems for mobile intelligent devices. Generally, these systems can be classified into P2P-protocol oriented systems and dynamic distributed applications (e.g. JXTA) or IP-based client-server applications (Jini, Microsoft .NET, IBM WebSphere Everyplace Suite). A large body of work in the heterogeneous database integration area has largely remained in the research domain without any specific products that can be named.

JXTA [2] is a set of open, generalized P2P protocols that allows any connected device on the network — from cell phone to PDA, from PC to server - to communicate and to collaborate as peers. Currently, JXTA provides a way of peer-to-peer communication at the level of socket programming. Proem [8, 7] is another mobile peer-to-peer platform supports developing and deploying mobile peer-to-peer applications. Compared to JXTA, Proem is geared more toward supporting mobile applications characterized by immediate physically proximal peers. SyD goes further in this arena by focusing on general collaborative applications involving intensive database operations and complex business logic. In contrast to Proem, SyD relies on proxies and provides mechanisms for reusing existing SyD applications and services. Jini [13], a more mature system compared to JXTA, uses Java's remote method invocation. Jini's limitations include the lack of scalability - Jini was originally designed for resource sharing within groups of about 10 to 100 - and that at least one machine on the network is required to run a full Java technology-enabled environment. Qualcomm's Binary Runtime Environment for Wireless (BREW) allows development of a wide variety of handset applications that users can download over carrier networks onto any enabled phone. Microsoft's .Net is a platform based on Web Services built using open, Internet-based standards such as XML, SOAP and HTTP. Communication among .NET Web Services is achieved through SOAP message passing. IBM WebSphere provides the core software needed to deploy, integrate and manage e-business applications. Web Sphere Everyplace Suite extends them to PDA's and Internet appliances. It supports client-side programming through WAP and allows the creation of discrete groups of users.

Table 1. SyD Comparison to existing middleware systems

Middlewares	Mobile Domain	Server on Mobile Host	Atomic Tranx	Workflow Modeling	Disconnection Tolerated	Platform Independence
SyD	Y	Y	Y	Y	Y	Y
.NET compact framework	Y	N	N	N	N	N
Jini	N	Y	N	N	N	Y
BREW	Y	N	N	N	N	Y
JXTA	X	Y	N	N	Y	Y
Proem	Y	Y	N	N	Y	Y
WebSphere Everyplace Suite	Y	N	N	Y	N	N

SyD supersedes the above technologies in terms of unique features such as orientation on mobile-specific applications, easy application to mobile devices, heterogeneity of data, simple middleware API, heterogeneous software/hardware, etc. Only SyD supports a normal database transaction model. Table 1 summarizes the important differences and similarities between SyD and above major related technologies.

Among research projects and experimental systems, the ICEBERG Project at U. C. Berkeley [24] is based on an open and composite service architecture based on Internet standards for flow routing and agent deployment. Cooltown is HP's vision of a technology future where people, places, and things are first class citizens of the connected world, wired and wireless [6]. The Wireless Messaging API (WMA) 1.0 extends the J2ME platform by providing application developers device-independent access to "short message service" and "cell broadcast service". MicroChai VM is a Java application environment that allows customers to download Java applications to mobile intelligent devices.

9 Conclusions

We have described the System on Mobile Devices (SyD) which is the first working middleware prototype supporting an efficient collaborative application development for deployment on a collection of mobile devices. Our prototype also supports peer-to-peer and server applications. One of the main advantages of SyD is a modular architecture which hides inherent heterogeneity among devices (their OS and languages), data stores (their format and access mechanism) and networks (protocols, wired or wireless) by presenting a uniform and persistent object view of mobile server applications and data-stores interacting through XML/SOAP requests and responses.

The paper has demonstrated the systematic and streamlined application development and deployment capability of SyD on three representative applications from disparate domains: a system of mobile fleet vehicles, a system of calendars, and a travel application.

The device-resident portion of our system has a small code footprint to be accommodated within mobile devices (76 KB). SyD employs seamless switching between a hosted application and its stable proxy to tolerate temporary disconnections and provide persistence.

The ongoing and future work involves porting SyD to devices other than iPAQs such as Palm Pilots and cell phones, obtaining a pure peer-to-peer version, possibly leveraging off JXTA's directory service, providing more robust QoS functionalities, and addressing dynamic group security issues.

Acknowledgments: To current and past students who contributed at various stages: S. Bhagavati, P. Bhatia, T. Chang, W. Chen, X. Chen, S. Desetty, B. Gamulkiewicz, J. Gong, J. He, Y. He, P. Jayanthi, W. Johnson, H. Liu, F. Tan, Y. Tang, and W. Zhong.

References

1. W. M. P. Aalst van der. Workflow patterns. 2003. `http://tmitwww.tm.tue.nl/research/patterns`,.
2. D. Brookshier. *JXTA: Java P2P Programming.* Sams, 2002.
3. Keith W. Edwards, Mark W. Newman, et al. Challenge:: recombinant computing and the speakeasy approach. In *Procs. of the 8th annual Intl. conference on Mobile computing and networking*, pages 279–286, Atlanta, Georgia, USA, 2002.

4. Arthi Hariharan, Sushil K. Prasad, et al. A framework for constraint-based collaborative web service applications and a travel application case study. In *Intl. Symposium on Web Services and Applications (ISWS)*, Las Vegas, June 21-24, 2004.
5. William G. Johnson. *Relaxed Transaction Model for Composite Web Services Using XML.* MS thesis, Computer Science Department, Georgia State University, Atlanta, 2004. `http://konya.cs.gsu.edu/~wjohnson6`.
6. T. Kindberg, J. Barton, et al. People, places, things: Web presence for the real world. In *Procs. 3rd Annual Wireless and Mobile Computer Systems and Applications*, page 19, Monterey, Dec, 2000.
7. G. Kortuem. Proem: A peer-to-peer computing platform for mobile ad-hoc networks. In *Advanced Topic Workshop Middleware for Mobile Computing*, Heidelberg, Nov 2001.
8. G. Kortuem, J. Schneider, et al. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks. In *First Intl. Conf. on Peer-to-Peer Computing (P2P2)*, pages 75–91, Sweden, Aug 2001.
9. Praveen Madiraju, Sushil K. Prasad, et al. An agent module for a system of mobile devices. In *Procs. of the 3rd Intl. Workshop on Agents and Peer-to-Peer Computing (AP2PC) in conjunction with Third Intl. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS).* LNCS, New York, July, 2004.
10. Vijay Madisetti. SyD: A middleware infrastructure for mobile iAppliance devices. EE Times Network, Nov 5, 2002.
11. Sun Microsystems. *Connected Device Configuration (CDC) and the Foundation Profile.* Technical White Paper, 2001.
12. Sun Microsystems. *Connected Limited Device Configuration (CLDC).* JSR-000139, May, 2000.
13. Jan Newmarch. *A Programmer's Guide to Jini Technology.* A Press, 2000.
14. Thomas Phan, Lloyd Huang, and Chris Dulan. Integrating mobile wireless devices into the computational grid. In *MobiCom*, pages 271 – 278, Atlanta, Sep, 2002.
15. Gian Pietro Picco. μcode: A lightweight and flexible mobile code toolkit. In *Mobile Agents, Procs. of the 2nd Intl. Workshop on Mobile Agents (MA)*, volume 1477, pages 160–171. Springer, LNCS, Stuggart, 1998.
16. S. K. Prasad, M. Weeks, et al. Mobile fleet application using SOAP and system on devices (SyD) middleware technologies. In *Communications, Internet and Information Technology (CIIT)*, pages 426–431, St. Thomas, Nov 18-20, 2002.
17. Sushil Prasad, Erdogan Dogdu, et al. Design and implementation of a listener module for handheld mobile devices. In *ACM Southeast Conf.*, Savannah, Mar 7-8, 2003.
18. Sushil K. Prasad and Janaka Balasooriya. Web coordination bonds: A simple enhancement to web services infrastructure for effective collaboration. In *37th Hawaii Intl. Conf. on System Sciences*, Big Island, Jan 5-8, 2004.
19. Sushil K. Prasad and Janaka Balasooriya. Web coordination bonds: A simple and theoretically sound framework for effective collaboration among web services. Technical report, CS-TR-04-01, Department of Computer Science, Georgia State University, June, 2004. `http://www.cs.gsu.edu/~cscskp/Pub/PB04TR.pdf`.
20. Sushil K. Prasad et al. Implementation of a calendar application based on SyD coordination links. In *3rd Intl. Workshop Internet Computing and E-Commerce in conjunction with the 17th Annual Intl. Parallel & Distributed Processing Symposium (IPDPS)*, page 242. IEEE Computer Society Press, Nice, April 22-26, 2003.

21. Sushil K. Prasad et al. Enforcing interdependencies and executing transactions atomically over autonomous mobile data stores using SyD link technology. In *Mobile Wireless Network Workshop held in conjunction with The 23rd Intl. Conf. on Distributed Computing Systems (ICDCS)*, pages 803–811, Providence, May.
22. Sushil K. Prasad, V. Madisetti, et al. System on mobile devices (SyD): Kernel design and implementation. In *First Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys), Poster and Demo Presentation*, San Francisco, May 5-8, 2003.
23. Sushil K. Prasad, M. Weeks, et al. Toward an easy programming environment for implementing mobile applications: A fleet application case study using SyD middleware. In *IEEE Intl Workshop on Web Based Systems and Applications, at 27th Annual Intl. Computational Software and Applications Conf. (COMPSAC)*, pages 696 – 703, Dallas, Nov 3-6, 2003.
24. Helen Wang et al. Iceberg: An internet-core network architecture for integrated communications. In *IEEE Personal Communications : Special Issue on IP-based Mobile Telecommunication Networks*, pages 10–19, 2000.
25. Wanxia Xie and S. B. Navathe. Transaction adaptation in system on mobile devices (SyD): Techniques and languages. In *Symposium of Database Management in Wireless Network Environments in the 58th IEEE Vehicular Technology Conf. (VTC)*, Orlando, Oct 7-10, 2003.
26. Wanxia Xie, Shamkant B. Navathe, and Sushil K. Prasad. Supporting QoS-aware transaction in the middleware for a system of mobile devices (SyD). In *1st Intl. Workshop on Mobile Distributed Computing held in conjunction with The 23rd Intl. Conf. on Distributed Computing Systems (ICDCS)*, pages 498–502, Providence, May 19-22, 2003.