A Methodology for Engineering Collaborative Applications over Mobile Web Objects using SyD Middleware

Sushil K.Prasad, Anu G. Bourgeois, Praveen Madiraju, Srilaxmi Malladi, and Janaka Balasooriya
Department of Computer Science
Georgia State University
Atlanta, GA 30302
{sprasad, anu, cscpnmx, cscsrmx, cscjlbx}@cs.gsu.edu

Abstract

Future web applications will be more collaborative and will use the standard and ubiquitous Internet protocols. We have previously developed System on Mobile Devices (SyD) middleware to rapidly develop and deploy collaborative applications over heterogeneous and possibly mobile devices hosting web objects. In this paper, we present the software engineering methodology for developing SyD-enabled web applications and illustrate it through a case study on a System of Calendar application, with implementation on iPAQs and its performance metrics study. SyD-enabled web objects allow us to create a collaborative application rapidly with limited coding. In this case study, the modular software architecture allowed us to hide the inherent heterogeneity among devices, data stores, and networks by presenting a uniform and persistent object view of mobile calendar objects interacting through XML/SOAP requests and responses. The performance results we obtained show that the application scales well as we increase the group size and adapts well within the constraints of mobile devices.

Keywords: Object and Web Service Coordination, SyD Coordination Bonds, Mobile Web Objects, Collaborative Applications

1. Introduction

Rapid development of collaborative distributed applications by leveraging off existing web entities will be key to bringing the Internet's collaborative potential to the users at large. Such collaborative applications span domains as diverse as personal applications (travel, calendaring and scheduling), enterprise e-commerce applications (supply chains, work flows, and virtual organizations), and scientific biomedical applications (biomedical data and process integration, and experiment workflows). The constituent autonomous entities, the sub-applications, and the coordinating applications themselves, are usually hosted on heterogeneous and autonomous, possibly mobile platforms [9]. There is an emerging need for a comprehensive middleware technology to enable development and deployment of these collaborative distributed applications over a collection of mobile (and wired) devices. This has been identified as one of the key research challenges recently [4, 12]. Our work is an ongoing effort to address this challenge, and in [15], we reported the design of System on Mobile Devices (SyD) middleware and its prototype implementation[1].

The current technology for the development of such collaborative web applications over a set of wired or wireless devices 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. A few existing middlewares have addressed some of the requirements of a comprehensive middleware [2, 6, 7, 8, 21]. For example, Proem [8] is one such platform for developing and deploying peer-to-peer (p2p) collaborative applications in a mobile ad-hoc networking environment. Commercial products such as .NET compact framework [11] and J2ME are also popular. In [1], authors describe issues related to service composition in mobile environments and evaluate criteria for judging protocols that enable such composition. ISAM [21] supports mobile collaborative applications using Java-based middleware. Yet another group of services such as Chef [3], Global-MMCS [20], and CAROUSEL [10] support collaboration primarily among people, not applications. The limitations of existing middlewares include: only

[1] http://www.cs.gsu.edu/~yes

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 [15]. SyD supersedes the existing technologies in terms of unique features such as orientation on mobile-specific applications, enabling servers on handhelds, heterogeneity of data/devices, simple middleware API, etc. Only SyD supports a normal database transaction model. We have a methodology for any generic application development and primitives to enforce constraints among web objects.

Rapid Web Application Engineering: In this paper, we describe SyD's high-level programming methodology to rapidly engineer group web applications over a collection of heterogeneous, autonomous, and possibly mobile data stores and sub-applications. A key goal of SyD is to enable SyD objects to coordinate in a distributed (and centralized) fashion. Each SyD object is capable of embedding SyD coordination bonds [5,15,16] (or ``Web bonds'' in the context of web services [13, 14]) to other entities enabling it to enforce dependencies and act as a conduit for data and control flows (Section 2). The methods provided by SyD enable developing collaborative applications rapidly and easily.

We demonstrate this software engineering methodology by showing how to develop and deploy a personal system of calendars application. In this distributed application, each user has his own database that is stored locally or on a proxy. The application logically bonds all members of a particular meeting together. A meeting can be rescheduled in real-time for all attendees by triggering the underling SyD bonds by any one participant [16]. The performance results we obtained for this application on iPAQs show that it scales well as we increase group size and fits well within the constraints of mobile devices.

SyD naturally extends to enabling collaborative applications across web-based objects. The SyD objects are stateful, web-based, and have interfaces like web services for method invocations. Furthermore, all method invocations and their responses in SyD employ SOAP-like XML envelopes. Therefore, SyD objects, their interactions, and the underlying techniques discussed in this paper have a direct bearing on web services and their compositions and coordination.

Section 2 briefly describes our background work on SyD middleware and the logical design of calendar application. Section 3 describes the generic SyD-based software engineering methodology and illustrates its steps through calendar application case study. It also describes specific deployment details of calendar application on iPAQs. Section 4 provides performance metrics and Section 5 concludes our paper.

2. SyD Architecture and Coordination Bonds - Background

In this section, we describe the design of System on Mobile Devices (SyD) and related issues, as well as highlight the important features of its architecture. (Refer to [15] for more details.)

2.1. SyD Architecture Overview

SyD uses the simple yet powerful idea of separating device management from management of groups of users and/or data stores. The SyD framework has three layers to accomplish this task. At the lowest layer, individual data stores are represented by device objects that encapsulate methods/operations for access, and manipulation of this data (SyD Deviceware). At the middle layer, there is SyD Groupware, a logically coherent collection of services, APIs, and objects to facilitate the execution of application programs. At the highest level are the SyD Applications themselves. They rely only on 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.

We have developed a prototype test bed of SyD middleware that captures the essential features of SyD's overall framework and several SyD-based web applications. We have designed and implemented a modular SyD kernel in Java as depicted in Figure 1. The SyD Kernel includes the following five modules:

1. **SyDDirectory**: Provides user/group/service publishing, management, and lookup services to SyD users and device objects. Also supports intelligent proxy maintenance for users/devices.
2. **SyDListener**: Provides a uniform object view of device services, and receives and responds to clients' synchronous or asynchronous XML-based remote invocations of those services [15]. Also allows SyD device objects to publish their services locally to the listener and globally through the directory service.
3. **SyDEngine**: Allows users/clients to invoke individual or group services remotely via XML-based messaging and aggregates responses. This yields a basic composer of mobile web services.

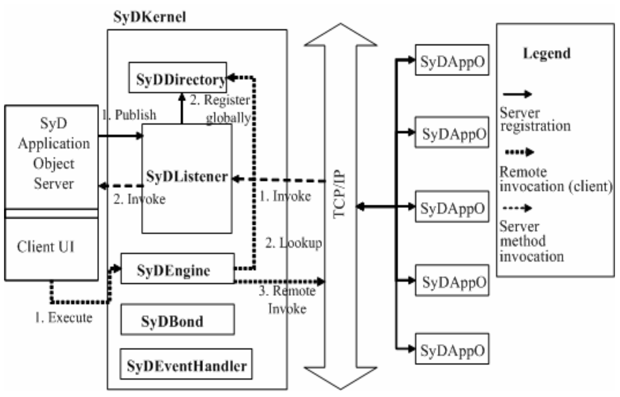

Figure 1. Interaction among modules of SyD Kernel [15]

4. **SyDBond**: Enables an application to create and enforce interdependencies, constraints and automatic updates among groups of SyD entities and web Services [13, 16].
5. **SyDEventHandler**: Handles local and global event registration, monitoring, and triggering.

2.2. SyD Coordination Bonds

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 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 applications, such as a set of calendars for a meeting setup [16], or a set of inter-dependent web services in a travel reservation application [5], 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 [13, 14].

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), status (certain/tentative), references to one or more web entities, triggers associated with each reference (event-condition-action rules), priority, constraint (and, or, xor), bond creation and expiry time, 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 Bond:* Mark *A*; If successful Change *A* then Try: Change *B*, Change *C*. A ``try" may not succeed.
- *Negotiation-and Bond:* Change *A* only if *B* and *C* can be successfully changed.

Using SyD bonds, 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 [16]. A simple scenario is as follows: *A* wants to call a meeting involving *B, C, D* and himself. After the empty slots in everybody's calendar found, a "negotiation-and bond" is created from *A*'s slot to the specific slot in each calendar table shown as solid lines (Figure 2). 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 to:

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

Else, for those individuals 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. Assume that *C* could not be reserved. Thus, *C* would have a tentative bond back to *A*, and others have subscription bond to *A*. 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 (Figure 2). Thus, a tentative meeting would be converted to permanent. Now suppose *D* wants to change the schedule for this meeting. This would trigger its back 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 bond established. If not all can agree, then *D* would be unable to change the meeting.

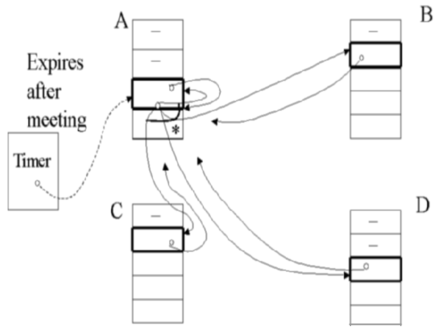

Figure 2. A scheduled meeting [16]

3. Designing Collaborative Applications

Collaborative SyD Applications: Collaborative group applications leverage off multiple constituent web entities, where each of those entities is a server application/component or an object or a data store. A centralized coordinator application resides on one host and composes or configures multiple SyD objects (which are themselves typically distributed). Composition is by invocation of method calls of constituent objects. Configuration employs the SyD coordination bonds to establish flow and dependency structure between the coordinator application and the constituent objects. A distributed coordinator application primarily employs SyD bonds among constituent SyD objects and thus is co-hosted distributively alongside them. The calendar of meetings application illustrates a distributed coordinator application.

3.1. Generic Design Methodology

SyD middleware provides components to aid easy development of collaborative applications which span from centralized to pure distributed. Collaborative applications interact with each other and in the process come across data dependencies or control dependencies or both depending on the nature of application. The SyD components provide an effective way of collaboration with heterogeneous peer devices and also provide a way to enforce dependencies. SyD bonds provide methodologies to enforce data and control dependencies in such application scenarios. The challenge is to associate SyD bonds in an early stage of application design for its effective use. In fact, one can follow standard UML design methods to design applications [19] and then insert bond artifacts at appropriate design phases as required. We will explain the design process of a collaborative application [17] with SyD middleware and SyD bonds at hand based on UML for distributed objects to model collaborative applications.

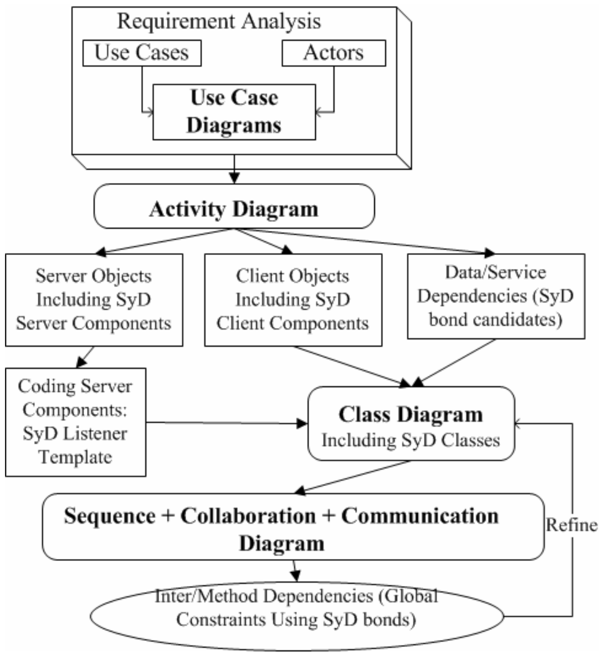
Requirement Analysis
Use Cases
Actors
Use Case Diagrams
Activity Diagram
Server Objects Including SyD Server Components
Client Objects Including SyD Client Components
Data/Service Dependencies (SyD bond candidates)
Coding Server Components: SyD Listener Template
Class Diagram Including SyD Classes
Sequence + Collaboration + Communication Diagram
Refine
Inter/Method Dependencies (Global Constraints Using SyD bonds)

Figure 3. Generic collaborative application design process

The steps for designing distributed applications using the concepts of SyD are as follows (Figure 3):

Step 1: A requirement specification is given by the user of the application system describing the way the system is expected to work.

Step 2: A requirement analysis is carried out to identify actors and use cases. An actor is an external entity (person, another system or object), which uses the system. Use cases are either text descriptions or flow descriptions of how actors interact with the system in all scenarios encountered in the applications. From use cases and actors, use case diagrams are drawn. Use case model diagrams show interaction between actors and all use cases.

Step 3: Activity diagrams are developed based on use case diagrams, use cases and actors. UML activity diagrams are equivalent to flow charts and data flow diagrams in object-oriented paradigm. In activity diagrams, the data flow spans across use cases and allows one to identify data and method inter-dependency of the use cases at an abstract level. These data and control dependencies can be analyzed and modeled using SyD bonds.

Step 4: Next step is the identification of classes and class diagrams. Class diagrams represent the static behavior of the system. Class diagrams describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and object. The methods that have a data resource object as an attribute might be "SyD-bondable" methods as it is likely to enforce interdependency. The persistence or non-persistent data objects with dependencies can be modeled using SyD methods to automate any method invocation needed for the application. Dynamic behavior of the system is modeled using sequence diagrams and collaboration diagrams. Both these diagrams help to identify inter-service dependencies at method level where we can apply SyD bonds to enforce them. Such design can further be clarified using communication diagrams that show the message flow between objects.

Once all the objects, data, data dependencies, and control dependencies have been identified and modeled using SyD and other components, implementation can begin. Server logic can be coded starting from SyD-listener skeleton which is middleware specific. Client coding can be started using SyDBond, SyDEngine, SyDDoc directory logic which is application specific. Figure 3 shows our collaborative application design process.

3.2. Designing Calendar Application – Case Study

Here, we illustrate the design process of a distributed calendar application. We will limit the discussion to particular scenarios in the system wherever appropriate.

Step 1: The requirements specification details the view of user and addresses the aspects of the benefits of the new system, interaction with other systems and system functionality. Based on the specification, several different use cases are identified for calendar application. The use cases of interest are: *get available times*, *setup meeting*, *cancel meeting*, *view calendar*, *reschedule meeting*, *create bond*, and *delete bond.*

Step 2: For the cancel meeting, the text description of use cases is given in Table 1. This can be represented in a pictorial view as use case diagram. The interaction between the actors and all use cases of the system can be given in a use case model diagram.

Table 1: CANCEL_MEETING Use Case

Use Case	**CANCEL_MEETING**	
Participating Actors	Application, Initiator, System	
Entry Condition	1.	Cancel meeting option is selected by the Initiator or is invoked by system.
Flow of Events	2.	System invokes CANCEL_MEETING
	3.	Confirmation of cancel meeting sent to all attendees.
	4.	System checks for any associations waiting on the initiator.
	6.	All the associations waiting up on are now converted to confirmed status.
	7.	All the associations are informed of the change.
Exit Condition	8.	Return to main menu.

Step 3: The activity diagram for cancel meeting follows these steps. For the calendar application, the method call for *cancel meeting* checks for any dependencies associated in its execution. The presence of confirmed dependencies will result in its successful execution. However, in case of tentative dependencies, a reschedule is triggered resulting in an automatic execution of the scenario "conversion of status", in case of no conflicts. These method dependencies indicate place holders for SyD methods [13, 14].

Step4: The methods *cancel meeting* (attendeelist, starttime,endtime), *reschedule*(attendeelist, starttime, endtime), *confirm meeting*(attendeelist, starttime, endtime), etc., executed in a calendar application result in the update of dependent data objects. These data dependencies indicate place-holders for SyD Bonds.

3.3 Case Study Implementation Details

The design of the calendar application has been implemented on HP iPAQ H3600 and H3700 series running windows CE operating system. Here, we describe implementation details providing insights into the development process. These details logistics and device-level details should help developers of similar applications for mobile devices.

*Step 1***:** We implemented SyD Middleware (as a java package) and Calendar code using java JDK 1.3. The system user interface was designed using JAVA Applets. We used Oracle8i as the back end database for storing SyD bond and application specific tables. All were implemented on a PC. Calendar application code interfaces with SyD Middleware application code for executing method calls (SyDEngine), listening for incoming method calls (SyDListener), and making directory service calls (SyDDirectory).

Step 2: We installed JVM for iPAQ, Jeode EVM Version 1.9. We ported the SyD Middleware code and calendar application code on the iPAQ using Microsoft ActiveSync version 3.5 [18] and set the classpath appropriately.

Step 3: After downloading the SyD Middleware, we installed and ran the middleware components on the iPAQ. This involves: (i) running a directory server (Oracle server) on a PC connected via a wireless network with the base iPAQ and (ii) running *listener.lnk* file (located in /syd/sydlistener path), which continuously listens for incoming method calls.

Step 4: We then installed the calendar application code itself. To do this, we executed the *CalRegistrar.lnk* file, which registers the application with SyDDirectory, followed by the application GUI to implement the various scenarios (set up meeting, cancel meeting and reschedule meeting).

4. Experiments and Performance Metrics

Here, we report experiments and performance metrics we obtained on the calendar application.

4.1. Experimental Hardware/Software Setup

We ran our experiments on a high performance/low power SA-1110 (206 MHz) Compaq iPAQ H 3600 and 3700 series, with 32 MB of SD RAM and 32MB of Flash ROM. We had three 3600 series and seven 3700 series iPAQ running middleware and calendar applications connected through a wireless network using a 2.4GHz wireless router. The operating system was Windows CE. We used JDK version 1.3 to code our programs and JVM for iPAQ was Jeode EVM Version 1.9. The DBMS of the directory server was Oracle 8i.

In Section 3 we have shown that SyD middleware enables structured, streamlined and rapid application development on mobile devices backed with theoretical and proven case study implementations of the calendar application. However, in a mobile setting, it is also significant that the applications developed scale well in terms of bandwidth, memory storage and response time parameters, as these resources are scarce for mobile devices. The motivations for considering aforementioned parameters are as follows: 1) Mobile devices cannot afford large amounts of message transfers, as the network bandwidth is limited; hence, we measured message size transferred. 2) Storage size on iPAQ is scarce and larger storage size for applications is not desired; hence, we measured storage requirements; 3) Response time for executing method calls on mobile devices is critical, as higher response times are possible when applications (a) consume more storage space, (b) transfer larger message sizes, and (c) require higher memory; hence we measured response time. We carried out experiments on calendar application for three scenarios: set up meeting, cancel meeting, and reschedule meeting. Our experiment results have been encouraging, as the application has shown to scale well in terms of all the parameters.

4.2. Setup Meeting Scenario

A constant message size of 50 bytes is transferred for each participant in a meeting consisting of meeting details. The storage size for group sizes of 2, 3, and 4 are: 120, 146, and 170 bytes respectively. For group sizes of more than 3, the storage size does not increase linearly as we associated a meeting id for each meeting, which avoids repetitive information such as start times, end times and comments.

Response time: Response time is the time required to execute set up meeting method call. A set up meeting method call includes time required to execute a get available time method returning the available times of all the participants, time required to execute the set up meeting for all involved meeting participants, and time to write the meeting details of all the participants to a file. It should also be noted that any method call must go through SyD middleware components. More specifically, it includes time required for (i) SyDEngine to contact SyDDirectory to get other user url information, (ii) SyDDoc to create a request document, and (iii) SyDEngine to invoke SyDListener remotely and get back the results.

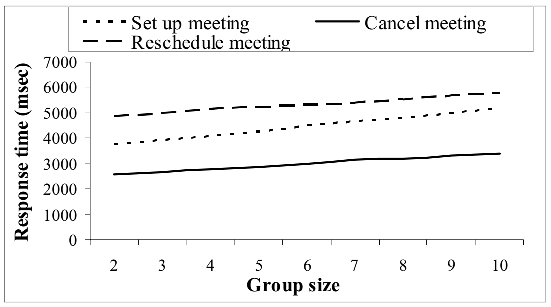

Figure 4: Response time for three scenarios

In Figure 4, we show the response time for all three scenarios based on varying number of group sizes. We observe that response time scales well (does not increase rapidly) for increasing group size through parallelism in processing and this behavior can be explained by analyzing different middleware component timings that make up response time as can be seen from Figure 5. The different components and their timing analysis are given below:

The "Engine to Directory Service" takes around 47-60 msec for group sizes of 2-10, which is less than 1% of total time. The "Create SyDDoc" value ranges from 13-90 msec for group sizes of 2-10, which is again less than 1% of response time. Now, we go in details on the components that make up a large share of the total response time.

Engine to Remote Listener: SyDEngine invokes remote listener for executing method call on remote devices by using the request document generated from the above step. This involves sending the request document to the remote listener, parsing the request document at the remote listener end, invoking the method call on the remote listener and writing the meeting details of each individual participant to a file. For increased group sizes, we achieve some concurrency as multiple remote listener calls are made to participant devices and results are collected. This value ranges from 1725-2900 msec for group sizes of 2-10 (takes around 48% of total time).

Server Processing: This refers to all other miscellaneous processing times such as, opening, writing and closing of file at initiator side, initializations for middleware components (SyDEngine, client side RMI registry components of directory server), and different application specific objects such as vectors. Here, we achieve concurrency for increased group sizes. This value ranges from 1995-2100 msec for group sizes of 2-10 (takes around 50 % of total time).

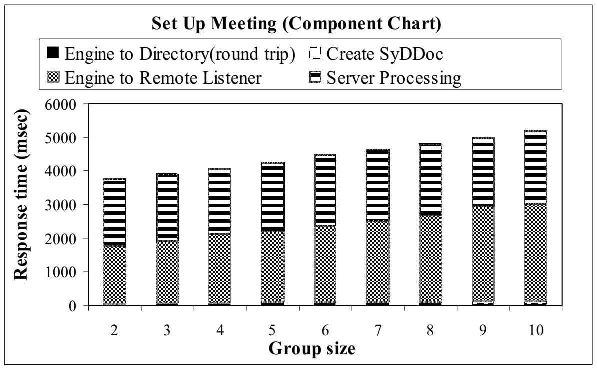

Figure 5: Set up meeting response time for components

Other Meeting Scenarios: In a reschedule meeting scenario, from the initiator point of view, size of message transferred is the message size transferred to convey the information that meeting has been cancelled to the other participants, and another message to send a confirmation of the meeting set up that has been tentative so far. The initiator does not have to wait on any acknowledgements in either case as one corresponds to cancel and for the tentative meeting the timings have been already agreed as tentative. We assume that only an initiator can cancel the meeting as he alone knows all the participant details and the tentative meeting participant details. This yields in a very small amount of data to be transferred, two messages containing initiator name, start time, end time and date (around 20 bytes each). Similarly, cancel meeting takes also takes around 20 bytes of data transfer.

5. Conclusions

We have described the high-level programming and deployment methodology of System on Mobile Devices (SyD) middleware which is the first working middleware prototype supporting an efficient collaborative application development environment for deployment on a collection of mobile devices. One of the main advantages of SyD is a modular architecture that hides inherent heterogeneity among devices, data stores, and networks 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 for collaborative applications composed over mobile web objects. We detailed this process for the design of a system of calendars, a representative application. We also presented implementation details and performance metrics for this particular application. Specifically, we measured the bandwidth required, the storage requirements, and the response timings. The results we obtained show that the application scales well as we increase the group size and fits well within the framework of mobile devices. Therefore, SyD objects, their interactions, and the underlying techniques discussed in this paper provide a direct benefit to web services and their compositions and coordination.

6. References

[1] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha. "Service Composition for Mobile Environments", J. on Mobile Networking and Applications, Feb., 2004.

[2] Gianpaolo Cugola, Gian Pietro Picco. "Peer-to-Peer for Collaborative Applications", 22nd Intl. Conf. on Distributed Computing Systems Workshops (ICDCSW), July, 2002.

[3] Chef (2004). Chef Project Website available from http://www.chefproject.org/index.htm

[4] W. Keith Edwards, Mark W. Newman, Jana Sedivy, Trevor Smith, Shahram Izadi, "Recombinant computing and Speakeasy Approach," in Proceedings of MobiCom 2002, Atlanta, GA. USA, September 23-28, pp. 279-286.

[5] Aarthi Hariharan, Sushil K. Prasad, et al. "A Framework for Constraint-Based Collaborative Web Service Applications and a Travel Application Case Study", Proc. Intl.. Symp. on Web Services and Applns., Nevada, 2004.

[6] E. Kirda, P. Fenkam, G. Reif, and H. Gall. "A service architecture for mobile teamwork", Intl. Conf. on Software engineering and knowledge engineering, Ischia, Italy,2002.

[7] Allan Meng Krebs, Mihail F. Ionescu, Bogdan Dorohonceanu, Ivan Marsic: The DISCIPLE System for Collaboration over the Heterogeneous Web. HICSS 2003.

[8] Gerd Kortuem. Proem: A Middleware Platform for Mobile Peer-to-Peer Computing. ACM SIGMOBILE Mobile Computing and Communications Review (MC2R), Vol. 6, No 4, October 2002.

[9] Oliver Krone, Fabrice Chantemargue, Thierry Dagaeff, Michael Schumacher, Béat Hirsbrunner, "Coordinating autonomous entities," The Applied Computing Review, Special issue on Coordination Models Languages and Applications (SAC),1998.

[10] S. Lee, S. Ko, G. Fox, K. Kim, S. Oh. "A Web Service Approach to Universal Accessibility in Collaboration Services", ICWS03, Las Vegas.

[11] Craig Neable."The .NET Compact Framework", IEEE Pervasive Computing Magazine, October-December 2002.

[12] Thomas Phan, Lloyd Huang, Chirs Dulan , "Integrating Mobile Wireless Devices Into the Computational Grid", in Proc. of MobiCom , Atlanta, September 2002.

[13] Sushil K. Prasad and Janaka Balasoorya, "Web Coordination Bonds: A Simple Enhancement to Web Services Infrastructure for Effective Collaboration", Proc. 37th HICSS, 2004.

[14] Sushil K. Prasad and J. Balasooriya, "Fundamental Capabilities of Web Coordination Bonds: Modeling Petri Nets and Expressing Workflow and Communication Patterns over Web Services", 38th HICSS, 2005.

[15] Sushil K. Prasad, V. Madisetti, S. Navathe, et al. "System on Mobile Devices (SyD): A Middleware Testbed for collaborative Applications over Small Heterogeneous Devices and Data Stores", Proc. ACM/IFIP/USENIX 5th International Middleware Conference, Toronto, Oct. 2004.

[16] Sushil K. Prasad, Anu Bourgeois, et al. "Implementation of a Calendar Application Based on SyD Coordination Links," Proc. 3rd Intl. Workshop Internet Computing and E-Commerce in conjunction with IPDPS, April, 2003.

[17] R.S. Pressman, Software engineering: A practitioner's approach, 4th ed., New York: McGraw-Hill, 1997.

[18] Microsoft ActiveSync Version. www.microsoft.com/windowsmobile/downloads/pocketpc.mspx

[19] Martin Fowler, Kendall Scott, UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition), Addison - Wisley publication

[20] Wenjun Wu, Ahmet Uyar, Hasan Bulut, Geoffrey Fox. Integration of SIP VoIP and Messaging Systems with AccessGrid and H.323, ICWS 2003.

[21] Adenauer Yamin, et al. "Collaborative Multilevel Adaptation in Distributed Mobile Applications", XII International Conference of the Chilean Computer Science Society (SCCC'02), November 2002.