Abstract

Today’s collaborative applications over the web span domains as diverse as enterprise e-commerce applications and biomedical applications. We have previously developed and formally investigated “Web Coordination Bonds,” a theoretically powerful artifact for interconnecting autonomous web objects for control and data flows, and for enforcing dependencies. Here, we describe BondFlow, a system that enables the configuration and execution of workflows using web bonds over heterogeneous web objects. The system automatically generates lightweight proxy wrapper objects to enable encapsulated web services to be interconnected through web bonds. The small footprint wrappers (less than 10 KB) can reside on Java-enabled handheld devices. We describe the design, architecture and performance of the BondFlow system. Our initial performance results indicate that both wrapper creation time and web bond related execution time are small. In contrast to our BondFlow system, existing systems need expert programming to create workflows, do not easily allow distributed coordination of workflows, and usually do not support heterogeneity of device, data and network. The system is preliminary, and further research is ongoing to extend its configuration and execution platforms.

1. Introduction

“Software as a Service” (SaaS) is the recent notable development in software engineering. These software services will be running on heterogeneous platforms and distributed information networks, providing services to other entities in the network [13]. Web service infrastructure is arguably the most important realization of the SaaS concept. Web service (WS) is defined as a “self-contained modular application that can be described, published, located and invoked over the net” [18]. It encapsulates the computational complexity and device heterogeneity, and the client entities interact through the interface. Therefore, in the SaaS model, WSs as service providers become the building blocks out of which new applications are created. Thus, one can harness the true potential of WS infrastructure by composing different web services together to create sophisticated useful applications and workflows [24]. Such composition enables inter-organizational collaboration and coordination (workflows and collaborative applications). Hence, integration/coordination technologies play an important role in WS infrastructure [13].

However, there are several limitations with both the current web infrastructures and the application development and integration technologies.

- In [27], authors illustrate how existing WSs are tailored to develop business processes (composite web processes) over the Internet. Such applications are long running rather than mere invoke/response interactions. Current state of the art in WS composition is to model the composite web service as a separate state preserving web process, as WS are stateless, and not capable of actively participating in such application scenarios. The composite web process encapsulates the coordination logic that makes it a central coordinating agent. This is a classic example of 2-tier application development [15], which is not desirable for complex, long-lived interactions. Two-tier systems have scalability and heterogeneity issues and also legacy object integration difficulties [15]. Ironically, one of the main objectives of the web is to solve these problems.
- Even though web services technology is one of the most highly talked about technologies, it is not widely used. This is mainly due to the lack of easy to use, reliable, and robust development environment for web service application development. In [39], authors have pointed out the difficulty of using BPEL and WSDL
especially for non-experts. In most cases, extensive programming is required [26, 44].

- Currently, there are many languages and standards for web services composition [13]. However, as things stand today, these languages propose different techniques to perform WS composition and there is no fundamental, theoretically sound WS composition/choreography framework [3, 44].

- As opposed to the prevailing centralized coordination, distributed coordination has two categories of advantage: (i) Due to security, privacy, or licensing imperatives, some web-based objects will only allow direct pair-wise interactions without any coordinating third-party entity; and (ii) Centralized coordination/workflows suffer from issues such as scalability, performance, fault tolerance, etc.

We have recently proposed web coordination bonds as a set of core artifacts for web Service coordination/choreography, and shown that they have the modeling power of extended Petri nets, and that they can express all established workflow and communication patterns [33, 34]. Web bond concept has evolved from our experience with developing several collaborative peer-to-peer distributed web applications using our System on Mobile Devices (SyD) middleware [36]. Section 2 briefly explains web bond artifacts and the SyD middleware system.

Based on web coordination bonds and SyD middleware, we have developed a prototype system, namely, BondFlow system, which is an environment to develop and execute collaborative applications including workflows over heterogeneous web services. The contributions of this paper are to present the BondFlow system and illustrate its effectiveness. The key features distinguishing our system from others are (i) programming ease of workflow configuration, (ii) automatic encapsulation of heterogeneous web services through lightweight proxy wrapper objects, providing capability of persistently embedding high-level workflow logic and executing web bonds, and (iii) distributed coordination of workflows. The lightweight wrappers enable these objects and the workflows to be hosted on small handhelds as well. The system is preliminary, and further research is needed to (i) extend its limited configuration capabilities, (ii) implement the full coordination capabilities of web bonds, and (iii) improve its execution platform.

Section 3 presents the BondFlow system from developer’s perspective, addressing both workflow configuration through proxy wrappers and web bonds, and their deployment and distributed coordination. The heterogeneous web services are given a uniform object view and made capable of web-bonding by automatically generated proxy wrapper objects. The wrapper encapsulates logic for bond management along with web service method calls. The wrapper objects communicate with each other and help enforce the flows and interdependencies. Section 4 describes the software architecture of the BondFlow system. Performance measurements for the system operations and sample workflows are presented in Section 5. Section 6 presents related work. Section 7 contains our conclusions and future research directions.

2. Background Concepts

Our BondFlow system provides an environment for configuring and executing workflows on the fly over heterogeneous web objects. The system is based on the concept of web coordination bonds [34, 33] and SyD middleware system. Here, we briefly explain the web bond concept and the SyD middleware system.

Web coordination bonds: Web bonds enable applications to create data and control flows between entities, enforce interdependencies and constraints, and carry out atomic transactions spanning over a group of web entities/processes. An entity may be part of a data-store or an object or a software component, and here we specifically consider web services. There are two types of web bonds: subscription bonds and negotiation bonds. Subscription bonds allow automatic flow of information and control 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. If there is a subscription bond from activity A to activity B, it implies that once A completes its execution, B will be invoked with suitable control and data as specified by the subscription bond. Negotiation bonds enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction. If there is a negotiation bond from activity A to activity B, then A must ensure that B can be also executed as well.

SyD middleware: SyD is a recently prototyped middleware for a novel Internet programming methodology and a framework for developing collaborative e-Service and other applications executing over a collection of independent, possibly mobile, and heterogeneous data stores, leveraging off the best in object-oriented design and middleware technologies. SyD middleware system is useful in enabling web object view of heterogeneous entities as well as in linking web objects through coordination bonds. Each data store is encapsulated as a SyD object (even on a small mobile device employing a tiny SyD listener), and multiple SyD objects can collaborate in a p2p fashion using XML-based messages. SyD uses the simple yet powerful idea of separating 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 per-
sistent 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 worldview to the SyD application to be developed and executed on. All objects hosted by each device are published with the SyD groupware directory service that enables SyD applications to dynamically form groups of objects hosted by devices, and operate on them in a manner independent of devices, data, and underlying networks. The SyD groupware hosts the application and other middleware objects, and provides a powerful set of services for directory and group management, and for performing group communication and other functionalities across multiple devices. Web bond is one of the key enabling components of the SyD framework. SyD coordination bonds enable applications to create and enforce interdependencies and constraints, and carry out atomic transactions spanning over a group of SyD and non-SyD entities/processes.

3. Developer’s View of BondFlow System

BondFlow system provides an easy to use platform to develop and deploy collaborative applications including workflows over web services. Users of the BondFlow system can configure their workflow by selecting suitable web services and bonding them using web coordination bonds to enforce dependencies. Workflow coordination architecture of the BondFlow system is inherently distributed. Following sections discuss the workflow configuration, execution, and coordination of the BondFlow system in developer’s perspective.

3.1. Configuring and Executing Workflows

The BondFlow system initiates its operation by web service lookup and discovery (Fig. 1). The web service interface module contains a WS locator module and a WSDL parser module. The WS locator helps in discovering the services of interest. The WSDL Parser parses the WSDL and allows the service components to be viewed in the form of summary of methods and parameter lists. Users can choose to save the viewed services for future reference. An instance of java-enabled web service wrapper object is created when the user wishes to save the web service. Web coordination bonds can be created among the chosen services at any point of time to reflect data and control flow (using “Subscription bonds”), and other dependencies (using “Negotiation bonds”). The most important information provided at the bond creation time is the type of the bond to be created. Dependency enforcement and entire operation of bond execution depends on the type of the bond that has been created. Bond related information is stored in an XML storage file. The wrapper encompasses all the coordination capabilities of the web bond artifacts (Fig. 2). As shown in Fig. 2, each web service method call is encapsulated by negotiation and subscription bond check. This logic makes sure that data and control dependencies are met before and after making the actual WS invocation. It hides the heterogeneity of various objects including legacy web services distributed among the network by enabling them to coordinate using the BondFlow system. Once the wrapper is created and bonded, the basic skeleton of web service composition for BondFlow system is ready. The wrapper generation process can be centralized or distributed. However, the bond execution process, though, is distributed. The footprint of the wrapper is small and can reside on a mobile device easily. The wrappers can communicate with each other using the SyDListener [35] component of SyD middleware. The BondFlow system makes the collaborative application development very easy for small devices running SyD [30]. Once any of the wrappers is invoked, the presence of the web bond is initially checked and depending upon the presence and type of the bond, coordination among components is carried out by enforcing the specified constraints and dependencies.
3.2. Workflow Coordination Architecture

At present, the web bond artifacts enable two different ways of coordination. In the traditional centralized version, a composed web service is created using web bond artifacts (Fig. 3). The composed service contains all the coordination logic and will invoke participating web services as required. This is what most competing workflow development systems are capable of (without the high-level web bond programming). Alternatively, one can leverage the distributed computing and communication capabilities of the devices and networks with all the inherent advantages over a centralized coordination (concurrency, fault tolerance, availability, performance, to name a few). The BondFlow system enables both centralized and distributed coordination. For distributed coordination, we create a web bond wrapper around each web service and instantiate the wrapped web services distributed across the network (Fig. 4). The proxy wrappers embed the workflow coordination logic so that instances of wrapped web services become stateful self-coordinating web objects. However, the proxy wrappers need to interact with the actual web service to complete each method invocation. These two implementations provide a comparative test-bed to analyze quantitative (performance measurements) as well as qualitative (coding and architectural) aspects of BondFlow system.

4. BondFlow System Architecture

The BondFlow system consists of web service interface module, wrapper generator module, workflow configuration module, and execution module (Fig. 5). The WS Interface module uses SOAP and UDDI to locate the web service as shown. Web Bond Manager that deals with bond creation and execution is a part of the workflow configuration module and runtime module. Here, we briefly describe each component of the system.
4.1. Web Service Interface Module

This module contains two components, Web Service Locator and WSDL Parser, as shown in Fig. 5. The WS Interface module is the system’s interface to the web services infrastructure. It deals with locating the web services of interest for the user and parsing those web services for desired data. It interacts with the Wrapper Generator module, which generates wrappers if a particular WS is selected for the workflow. Following subsections briefly discuss components of web service interface module.

Web Service Locator: When a user wishes to browse through the services provided by a provider, he supplies the WSDL url of that service. The web service Locator module locates the service by contacting UDDI, gets the WSDL and passes it to the WSDL Parser module. We have used Apache-Axis implementation of the web services.

WSDL Parser: The WSDL parser uses WSDL4J API for WSDL parsing. It parses the WSDL file for required components. It stores the result in the XML storage for persistence, if the user opts to save the web service. Some of the entities saved to the storage are the name of the service, locationURI, namespaceURI, methods and parameter list. NamespaceURI and locationURI are used for making a call to the actual web service from the wrapper body. XML data is stored in the persistent storage in a predefined format.

4.2. Wrapper Object Generator Module

This forms the heart of the system. It deals with the creation of the proxy wrapper objects corresponding to the web services. Upon selection of a particular WS for the workflow a wrapper is generated (Fig. 6). Wrapper object code is generated based on the parsed WSDL file of the selected WS and the wrapper generator template (Fig. 2). The wrapper has the intelligence of managing the bonds dynamically and enforcing dependencies to cater to the need of a particular workflow. The wrapper possesses a signature identical to the original web service. It encapsulates the original web service methods along with bond management capability. The entire operation of bond management is transparent to the user.

4.3. Workflow Configuration Module

Once suitable web services have been selected, the next step is to use web bond artifacts to hook wrapper objects to other wrappers in the workflow to reflect data and control flow (using “Subscription bonds”), and other dependencies (using “Negotiation bonds”). The web bond configuration manager deals with all the bond related operations, such as creation, deletion and updating of the web bonds. When the bond is created, bond related information is stored in a persistent XML storage for future reference. Bond parameters are specified while creating the bonds. Once wrappers are created and bonded, the basic skeleton of the workflow is ready. The BondFlow system has capabilities to view selected web services and their methods/parameters concurrently and to bond them together to enforce dependencies. After invoking a wrapper method, the dependencies are checked and web bonds, if any, are executed according to defined criteria. Fig. 7 shows a code snippet that gives an overview of data that is stored at bond creation and Fig. 8 gives a pictorial representation of a workflow configuration module.
As shown above, the important parameters stored in the persistent storage are source and destination web service name, method name, type of bond (S- Subscription and N- Negotiation) and presence of the trigger in this bond.

4.4. Workflow Execution Module

Upon wrapper invocation, the wrapper consults the workflow execution environment and carries out series of operations depending upon the bond parameters specified at the bond creation time. Checking of the type of bonds, getting bond parameters and executing the actual bond are some of the major operations by the Bonds Management System. The final call to the original web service is made by the wrapper using SOAP. The BondFlow execution platform (Fig. 9) enforces dependencies by invoking the bonds.

At runtime wrapper objects can reside in a central location or they can be distributed over the network. If they reside in a central location, then the communication is local. If the wrappers are distributed, then they can use SOAP or other RPC communication protocol to send/receive data and control. We have implemented SOAP based communication. The footprint of wrapper is small (Table 1), and a workflow thus constructed can reside in a small handheld device easily. If the wrappers reside in small handheld devices where even SOAP server/client components are usually not available, our SyD listener module enables them to communicate [35]. SyD listener is a lightweight module in our SyD middleware framework for enabling mobile devices to host server objects [36].

4.5. Enabling Technologies

In this section we will briefly describe the various technologies/products used to create a prototype of the BondFlow system.

- **JDK1.4.1**: The Wrapper Generator system is implemented using Java. This makes the system platform independent and provides all the advantages of a typical Java-built system.
- **WSDL4J API** [6]: The WSDL parser has been built using WSDL4J API. WSDL4J API is an IBM reference implementation of the JSR-110 specification (Java API’s for WSDL). It facilitates efficient creation, representation and manipulation of WSDL documents that describe the services. The web services Description Language (WSDL) which is an XML-based language for describing web services allows developers to describe the inputs and outputs to an operation, the set of operations that make up a service, the transport and protocol information needed to access the service and the endpoints via which the service is accessible. WSDL4J is a very efficient API for parsing and representing a WSDL document in Java.
- **NanoXML 2.2.1** [8]: Data persistency is achieved in our system by using XML storage mechanism and a lot of data manipulation is required during bond creation and execution. A lightweight parser for XML was needed which is exactly what NanoXML provides. A tree-based parser is not efficient to use with small devices due to memory constraints. Event driven parsers like NanoXML are best to use. NanoXML is a small (about 6K), reasonably fast, non-validating XML parser for JAVA that provides a set of APIs.
- **Axis 1.1** [35]: Locating the web services and subsequent communication with the web services is implemented using Apache Axis, which is essentially a SOAP engine. It provides many other important features along with extensive support for WSDL and compatibility with Tomcat. Axis offers improved speed, flexibility and stability. Axis, along with apache web server, provides an efficient web service interface for various applications. Other software that are used are Oracle 8i, Apache 2.0.52, and Jakarta Tomcat 5.0.

5. Performance of the System

Here, we report results from preliminary experiments and performance metrics obtained based on a few workflows that we have configured and executed using BondFlow system.

**Hardware and Software Setup**: We ran our experiments on a high performance SunOS 5.8 server. We built wrappers using JDK 1.4.2. The WSDL parser has been built using WSDL4J API. WSJD4J API is an IBM reference implementation of the JSR-110 specification (Java API’s for WSDL). NanoXML 2.2.1 is used as the XML parser for JAVA. Xmethod’s SOAP based web services (eg: http://www.xmethods.net/sd/2001/TemperatureService.wsdl) have been used for experiments. For our experiments, we
have used these web services to create simple Workflows comprising subscription and negotiation bonds (Fig. 10, Fig. 11). As shown in Fig. 11, there are subscription bonds from Barnes and Noble web service to eBay web service and eBay to Amazon and Amazon to Currency web service. This chain of subscription bonds enable them to exchange book price data and control. By having negotiation bonds in reverse direction makes sure they activate sequentially. For example, Amazon can only be invoked if eBay has finished its activity. Similarly, Fig. 11 enforces control and dataflow using subscription bonds.

Size of WSDL (number of methods) vs. Wrapper creation time: As web bond wrapper is central to our system it is important to analyze wrapper creation time and to investigate how wrapper creation time varies with different size (number of methods) of web services. Table 1 shows that wrapper creation time is very small and wrapper size is less than 10 KB even for a web service with 17 methods. This is an advantage as these wrappers can easily be placed in memory constrained small handheld devices. The bond creation time for both types of bonds is less than 25ms. Also, note that once wrappers are created and bonded, the basic skeleton of the workflow is ready. Developers can add more logic into it if needed. This will reduce the programming effort considerably.

Workflow execution time: Any workflow activity checks for bonds and verifies that all the dependencies are met before executing it. Table 2 shows the bond related timings vs. workflow execution time. Note that the total execution time consists of time taken to invoke remote web service and get the result. The bond execution time is only a small fraction of the total workflow execution time.

6. Related Work

In [27], authors illustrate how existing WSs are tailored to develop business processes over the Internet.

## Table 1. Number of WS methods vs. Wrapper creation time

<table>
<thead>
<tr>
<th>#of Methods</th>
<th>WSDL size (KB)</th>
<th>Creation Time (ms)</th>
<th>Footprint (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
<td>2.2</td>
</tr>
<tr>
<td>2</td>
<td>2.3</td>
<td>23</td>
<td>2.3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>32</td>
<td>2.6</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>40</td>
<td>2.5</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>46</td>
<td>3.5</td>
</tr>
<tr>
<td>17</td>
<td>32</td>
<td>76</td>
<td>5.7</td>
</tr>
</tbody>
</table>

## Table 2. Workflow(WF) execution time, SB= Subscription bond, NB= Negotiation bond.

<table>
<thead>
<tr>
<th>WF Des.</th>
<th>Check Bonds</th>
<th>SB (ms)</th>
<th>NB (ms)</th>
<th>Total (ms)</th>
<th>Bond Related time (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB’s=3, NB’s=2</td>
<td>48</td>
<td>17</td>
<td>17</td>
<td>5577</td>
<td>1.47%</td>
</tr>
<tr>
<td>SB’s=3</td>
<td>46</td>
<td>0</td>
<td>21</td>
<td>6406</td>
<td>1.07%</td>
</tr>
</tbody>
</table>

## Figure 10. BookPrice workflow.

## Figure 11. Weather and traffic condition workflow.
tion Middleware (WSTMW) [37], is one such platform developed by IBM to carry out transaction over web services. WSTMW resides in both the web service side as well as the client (mediator) side. They have employed WS-Transaction, WS-Policy and BPEL4WS to prototype the system. Similar approach can be found in [17], where they have extended the BPEL4WS to support transactions. However, such developments have lacked scalability due to the use of centralized coordination. Net Traveler [14] is another middleware platform for distributed coordination among web services. In their architecture QSB (Query service broker) together with RS (Registration server) residing in the web services side provides query handing capabilities and distributed coordination. However, such developments are in very early stage and warrant further research. Development of web service composition techniques has added a new dimension to the workflow management systems over the Internet [21, 45]. In [20], authors have pointed out the importance of integrating web services in to workflow management systems. [20] describes possible workflow application domains over the Internet. Application of workflow management systems (WFMS’s) spans large number of application domains including business process models [21], scientific applications [28], and health care systems [46]. Among WFMSs, FlowManager [25] is an open source system based on Petri nets. OpenFlow [1] is another open source WFMS. IBM’s FlowMaker [7], Xerox’s InConcert and Fujitsu’s Regatta are commercially successful WFMSs [11]. More extensive treatment of classifying WFMSs can be found in [66]. A drawback of most WFMSs is that they are based on database concepts [11, 12] using extended/advanced transaction principles. They have the luxury of utilizing rich database functionalities to provide execution environment, correction criteria and communication. As a result, they do not have required functionality to cater to today’s heterogeneous, autonomous, distributed computing paradigm. Another drawback of most workflow management systems is their inability to adjust according to runtime environment and QoS parameters.

In [39], authors have pointed out the difficulty of using BPEL and WSDL especially for non-programmers (main beneficiaries of web services!). They propose alternative UML-based visual development platform called UML-WSC. The major WS workflow languages, BPEL, XLANG, WSFL, BPML, and WSCI, are XML based textual languages [16, 41]. Even though there are many development platforms available, none of them have any sound theoretical underpinning. Also, they do not provide enough easy to use tools to develop, modify (re-program) and deploy workflows, long-lived transactions and virtual organizations over web. Extensive programming is required in most cases [26, 44]. Therefore, it is necessary to build robust and reliable environment /deployment platform so that non-programmers such as natural scientists, business executives and everyday users can develop such collaborative applications. Our BondFlow system is an effort in that direction.

7. Conclusions and Future work

The next generation of Internet applications will involve various kinds of collaborative applications among heterogeneous, autonomous entities. The users will be scientists, engineers or financial and trade experts that are essentially untrained in programming and application development. Web Services are offering an important paradigm that is making a variety of services available for shared use in the development of collaborative applications. However, the research prototypes and the industrial middleware products or the workflow products have not yet addressed the issue of configuring and executing complex workflows that arise in the challenging applications such as e-commerce or bioinformatics. In this paper, we have presented the BondFlow system as a framework to develop and deploy such collaborative applications and workflows. We have used and demonstrated the concept of “web bonds” which allow us to develop bonding among web-services and enable workflow creation and execution without much programming. The paper has presented the architecture of the BondFlow system and the workings of its components. We have implemented a lightweight web bond wrapper object, which encompasses all the coordination capabilities of web bond artifacts. Also, we have demonstrated how these wrapper objects residing in heterogeneous devices including on small handheld devices coordinate among themselves. Our performance results show that web bond operations take only a small fraction of the total execution time of the workflow.

In the future, we would like to build a complete IDE for BondFlow system. It will have capabilities of web coordination bonds [34, 33] so that developers can enforce all the workflow control flow dependencies and distributed communication patterns [34]. Also, current menu driven system need to be enhanced towards drag and drop kind of IDE. Further, we need to enhance the runtime engine of our systems so that users can monitor the execution and make changes while workflow is active. Finally, we would like to further investigate the power consumption of workflow execution on small handhelds.

References


Biographies

Janaka Balasooriya is a Ph.D candidate in Computer Science department at Georgia State University working under the supervision of Prof. Sushil. K. Prasad with Prof. Shamkant B. Navathe as a committee member. His dissertation work, “web coordination bonds”, is geared toward finding a fundamental framework for web service coordination and composition and has already published several papers. He received his B.S.in Computer Science and Engineering from university of Moratuwa, Colombo, Sri Lanka in 1998. From 1998 to 2000 he worked as an Engineer at Sri Lanka Telecom. He joined the Ph.D program in Fall 2001. Since 2001 Fall, he is serving as an executive committee member at GSU ACM student chapter. His current research interests are web services, mobile computing, networks and databases. Contact Janaka at jbalasooriya@ieee.org.

Mohini Padhye was a M.S student in Computer Science department at Georgia State University and did her theis under the supervision of Prof. Sushil K. Prasad. Her M.S theis "Coordinating Heterogeneous web Services through Handhelds using SyD’s Wrapper Framework” is based on web coordination bonds and SyD middleware system.

Sushil K. Prasad is a Professor of Computer Science at Georgia State University (GSU) and the Director of Yamacraw/GEDC Distributed, Mobile and Embedded Systems Program. Prasad has participated in external grants and contracts with funding of over $4M. He has carried out theoretical as well as experimental research in parallel and distributed computing, with over 60 publications, and made 3 utility patent applications and over a dozen provisional patent applications. His recent professional activities include serving on NSF review panel for Networking Infrastructure program, in the program committees of several international conferences including IPDPS-05, Colorado, ICA3PP-05, Melbourne, HiPC-04, Bangalore, ICA3PP-02, China, HIPC-01, India, DS-RT-2001, Cincinnati, PDCAT-2001, Taiwan, as publicity chair for PADS-2001, LA, and as proceeding chair of HiPC 2003-05. Recently, as PI of the Yamacraw/GEDC Embedded Systems research contracts, he has been leading a GSU team of seven faculty and 20 Ph.D./M.S. students on developing System on Mobile De-
vices (SyD) middleware for collaborative computing over heterogeneous mobile devices and data sources. His current research interests are Parallel Algorithms and Data Structures, Parallel Discrete Event Simulation, Web-based Distributed and Collaborative Computing, and Middlewares and Collaborative Applications for Handheld Devices.

Prasad received his B. Tech. in Computer Science and Engineering from Indian Institute of Technology, Kharagpur, in 1985, M.S. in Computer Science from Washington State University, Pullman, in 1986, and Ph.D. in Computer Science from University of Central Florida, Orlando, in 1990.

Shamkant B. Navathe is a professor at the College of Computing, Georgia Institute of Technology, Atlanta, USA. He has published over 130 refereed papers in database research; his important contributions are in database modeling, database conversion, database design, conceptual clustering, distributed database allocation, and database integration. His recent work has been addressing issues of mobility, scalability, interoperability, and personalization. Current projects include transaction models in P2P and web applications, text mining of medical literature databases, and data mining for better understanding of genomic/proteomic and medical data. Navathe is an author of the book, Fundamentals of Database Systems, with R. Elmasri (Addison Wesley, Edition 4, 2004) which is currently the leading database text-book worldwide. He also co-authored the book "Conceptual Design: An Entity Relationship Approach" (Addison Wesley, 1992) with Carlo Batini and Stefano Ceri. He has worked with IBM and Siemens in their research divisions and has been a consultant to various companies including Digital, CCA, HP and Equifax. He was the General Co-chairman of the 1996 International VLDB (Very Large Data Base) conference in Bombay, India. He was also program co-chair of SIGMOD 1985 and General Co-chair of the IFIP WG 2.6 Data Semantics Workshop in 1995. He has been an associate editor of ACM Computing Surveys, and IEEE Transactions on Knowledge and Data Engineering. He is also on the editorial boards of Information Systems (Pergamon Press) and Distributed and Parallel Databases (Kluwer Academic Publishers). His research interests span human genome data management, intelligent information retrieval, text mining, security risk modeling in information systems, engineering data management, and mobile as well as peer-to-peer information processing.

Navathe holds a Ph.D. from the University of Michigan in 1976.