

Modeling and Formal Verification of IMPP

Sohel Khan and Abdul Waheed Abdul Sattar
King Fahd University of Petroleum and Minerals, Saudi Arabia

Abstract: *This paper describes the modeling and formal verification of the application layer protocol, Instant Messaging and Presence Protocol (IMPP). Spin is a model checker for the verification of asynchronous, distributed and concurrent finite state systems. It accepts the system specification in a high level language called PROcess MEta LAnguage (PROMELA) and verification claims in temporal logic. We have selected Instant Messaging and Presence Protocol (IMPP) for modeling, simulation and verification as it is characterized by concurrency and distributed computing, which makes it a good candidate to explore the potential of model checking and verification. Further, the important properties of the protocol are verified using Linear Temporal Logic (LTL). One of our aims was also to get an insight into the scope and utility of formal methods based on state space exploration in testing larger and complex software systems which has been achieved to some extent.*

Keywords: *Formal methods, verification of communication protocols, instant messaging systems, verification tools, spin, LTL, PROMELA.*

Received March 14, 2004; accepted July 7, 2004

1. Introduction

Several communication protocols are in use and many still needed. Normally such protocols are designed by experts and checked manually. But with increased demand of reliability and fault tolerance of the protocols and the services they provide, the need for formal approaches for verification of protocols is increasingly being felt. Thus human judgment and consensus can be supplemented by the rigorous analytical power of formal methods. Spin is a model checker for the verification of asynchronous, distributed and concurrent finite state systems. It accepts the system specification in a high level language called PROcess MEta LAnguage (PROMELA) and verification claims in temporal logic.

We have selected Instant Messaging and Presence Protocol (IMPP) for modeling, simulation and verification as it is characterized by concurrency and distributed computing which makes it a good candidate to explore the potential of model checking and verification and it can provide insight into the scope and utility of formal methods based on state space exploration in testing larger software systems.

We will describe the evolution of IMPP and the formal description of its five elements: Service, assumptions, vocabulary, format, and procedure rules. We will then define the level of abstraction and focus of our study. After a brief description of related modeling efforts and the motivation for our work, we will describe our model and its PROMELA code. The paper will end with the simulation and verification results and the conclusion.

1.1. What is Presence and Instant Messaging?

Presence is the instantaneous knowledge that someone is available, online and reachable via instant messaging. Presence Service enables this knowledge to be embedded into any application. Instant Messaging involves short text messages that pop-up immediately, enabling chat sessions where people type back and forth. The most popular Instant Messengers (IM) in use are AOL, MSN, Yahoo, and ICQ.

1.2. Evolution of IMPP

Instant Messaging is evolving through merging with other technologies, such as VoIP, Conferencing, SMS and other applications, such as embedded IM in call centers, e-CRM and generally e-Business. In Jan 2003 AOL acquired its IM patent. But MIT's Zephyr and other instant-messaging systems existed long before Mirabilis (now AOL) applied for the patent. Zephyr was a notice transport and delivery system developed at MIT in 1990 and is still in use there on Unix. In July 1996 Mirabilis (now AOL) came up with ICQ "I seek you", which uses the ICQ Protocol. In June 1998 AOL bought Mirabilis (ICQ), and abandoned the ICQ protocol in favor of OSCAR protocol, the protocol used by AOL's instant messenger. AOL has more than 135 million registered ICQ users and some 180 million on AOL Instant Messenger. Few official documents exist for most of the above protocols. But some reverse engineered versions exist on the Internet, figured out by analyzing traffic generated by their Instant Messenger client [12].

Instant messaging differs from email primarily in that its primary focus is immediate end-user delivery.

Presence information was readily accessible on internet-connected systems years ago; when a user had an open session to a well-known multi-user system, his friends and colleagues could easily tell where he was connected from and whether he was using his computer. Since that time, computing infrastructure has become increasingly distributed and a given user may be consistently available, but has no standard way to make this information known to her peers. Some of the arguments for HTTP as a basis for a presence information protocol like ease of crossing firewalls, reuse of existing technology, use of similar protocol, and the applicability of URLs, do not seem convincing. [5].

2. IMPP Protocol with its Five Elements

The IMPP aims to develop architecture for simple instant messaging and presence awareness/notification. It will specify how authentication, message integrity, encryption and access control are integrated. It may also provide a general notification mechanism for data other than user presence information and instant messages. We briefly present here the five elements of specification particularly covering those areas that we are going to model and validate.

2.1. Service

A presence and instant messaging system allows users to subscribe to each other and be notified of changes in state, and for users to send each other short instant messages. IMPP provides two main services, a presence service and an instant message service. The presence service serves to accept, store and distribute information. The information stored is Presence Information. The Instant Message Service serves to accept and deliver Instant Messages to Instant Inboxes.

2.1.1. Presence Service

The Presence Service has two distinct sets of “clients”. One set of clients, called Presentities, provides presence information to be stored and distributed. The other set of clients, called Watchers, receives presence information from the service [14].

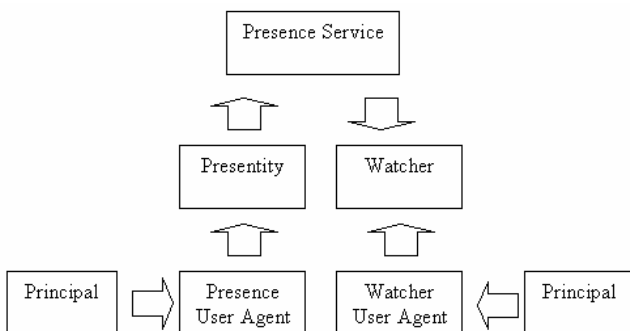


Figure 1. Overview of presence service.

2.1.2. Instant Message Service

The instant message service also has two distinct sets of “clients”: Senders and instant inboxes. A Sender provides instant messages to the instant message service for delivery. Each instant message is addressed to a particular instant inbox address, and the instant message service attempts to deliver the message to a corresponding instant inbox [6].

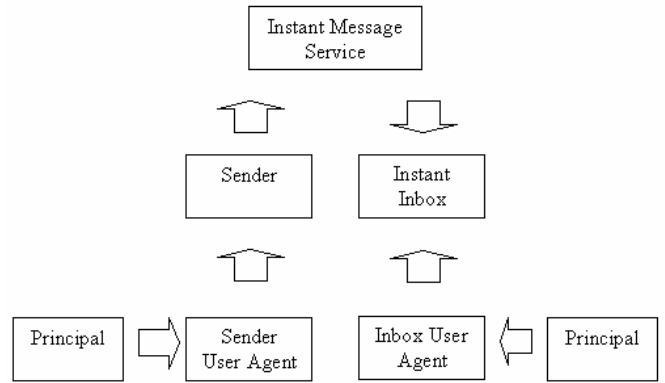


Figure 2. Overview of instant message service.

2.2. Assumptions

The IMPP model is intended to provide a means for understanding, comparing, and describing systems that support the services typically referred to as presence and instant messaging. It consists of a number of named entities that appear, in some form, in existing systems. No actual implementation is likely to have every entity of the model as a distinct part. Instead, there will almost always be parts of the implementation that embody two or more entities of the model. However, different implementations may combine entities in different ways [14].

2.3. Vocabulary

The model defines the Presence Information to consist of an arbitrary number of elements, called Presence Tuples. Each element consists of a Status marker (which might convey information such as online/offline/busy/away/do not disturb), an optional communication address, and optional other Presence markup. A communication Address includes a communication means and a contact address. A Communication means might also indicate some form of telephony, for example, with the corresponding Contact Address containing a telephone number, apart from instant message service.

The Message/CPIM format encapsulates an arbitrary MIME message content, together with message- and content-related metadata. This can optionally be signed or encrypted using MIME security multipart in conjunction with an appropriate security scheme [2].

2.4. Encoding (Format)

An XML-encoded Presence Information Data Format (PIDF) is defined for use with CPIM compliant systems. A presence payload in this format is expected to be produced by the Presentity (the source of the Presence Information) and transported to the watchers by the presence servers or gateways without any interpretation or modification [13].

2.5. Procedure Rules

An entity known as Presentity that wants to make its presence information known to others can register itself with presence service. Those who seek the presence information from the presence service called watchers are of two types, Poller and subscriber. An application can send an instant message to another Presentity through its instant inbox. Access Rules constraint on how a presence service makes presence information available to Watchers. For each Presentity's presence information, the applicable Access Rules are manipulated by the Presence User Agent of a Principal that controls the Presentity [6].

3. Modeling Objective

Modeling is the process of abstracting the functional specifications of a system into a minimal working specimen that enables us to understand and analyze a particular aspect of the system more closely. Hence the two important logical questions that come up are what and how much to abstract and what aspect we want to analyze. We answer them below. This can be understood from the analogy of analyzing an object under lens. The more we magnify the more we can analyze its fine features, but the area of observation also decreases with it.

3.1. Level of Abstraction

We have abstracted the system into the two main services of instant messaging and presence. The presence tuples are limited to a communication channel (IP address), name represented as a short integer, status represented as a short integer and message types which can be one from among the 15 most important message types required in our model.

3.2. Focus of Study

The basic properties we want to study about IMPP are:

- The integrity of access rules related to subscribers. In general the principal controlling Presentity must be able to control: Which watchers can observe that Presentity's presence information, which watchers can have subscriptions to that Presentity's presence information, what presence information a particular watcher will see for that Presentity, regardless of

whether the watcher gets it by fetching or notification, which other Principals, if any, can update the presence information of that Presentity. Similarly the Principal controlling an instant inbox must be able to control: Which other Principals, if any, can send instant messages to that Instant Inbox, which other Principals, if any, can read instant messages from that instant inbox.

- When Presentity changes its Presence information, any subscriber to that information must be notified of the changed information rapidly except when such notification is entirely prevented by access rules.
- The notification system of subscribers and fetchers.
- Liveness property of instant message service (impossibility of message loss) especially in a chat session.
- The multiple login control feature, this basically states that no multiple Presentity can login from two different places at same time.

4. Related Work

IETF IMPP working group is working to develop the architecture and protocol for IMPP. Its work is still under development with two RFC's published [4, 13]. Microsoft has announced that it will soon make the MSN messenger service protocol available to the industry by submitting it to the IETF as a working reference implementation of an interoperable instant messaging protocol. ICGnu is another initiative to create an open protocol for presence notification and instant messaging [6, 8]. But the project seems to have disappeared now. Jabber is another ambitious large-scale open source project to create a unified instant messaging protocol, with connections (known as transports) to other IM services, such as ICQ and AIM. The Simple General Awareness Protocol (SGAP) provides notifications of changes to small data items. SGAP was originally developed as Lotus's contribution to the ongoing process of developing an interoperable protocol for "presence" information. Although SGAP is a small and simple protocol, it supports sophisticated colleague-awareness. The only effort to formally verify a part of IMPP was undertaken by Patrice Godefroidy *et al.* [10]. Although quite interesting, this work was limited to verifying the privacy rules of instant messaging system. Our first model 5.2], the Presence service model can be used to achieve the same purpose. In summary our model is more comprehensive to test the critical properties of IMPP related to concurrency and message loss. Hence our effort to formally model and verify IMPP can be helpful in developing an internet-scale end-user presence awareness, notification and instant messaging system.

5. The Model

Although we started with a comprehensive model of IMPP, the limitations of our tool led us to split this model into three different models. With code size going beyond 500 lines the state explosion problem prevents the reachability analysis from completing. Owing to the limitations of tools (computing power as well as memory) abstraction plays a very important role in Model Checking. We could have used another tool such as Verifsoft, but lack of any implementation of IMPP, precluded that possibility [9].

5.1. Implementation Strategy in PROMELA

In all three models Presence service is common. The instant message service model is only used in the second model where we check the reliable delivery of instant messages. We start with the communication system used in the models.

5.1.1. Communication Channels

All the channels are declared as global objects. The following channels are central to the model.

The channel PSin represents the communication between any Presentity and presence server. Similarly there is an input channel for instant message service.

```
/* for presence service */
chan PSin = [3] of { mtype, short, short, short };

/* for instant message service */
chan INSin = [2] of { mtype, short, short, short }
```

The channels PTSin [3] and SUBin [3] are the input channels for every entity (Presentity and subscriber respectively) through which server communicates with it.

```
/* 10 input (output for the server) channels for ten
presentities */
chan PTSin [3] = [3] of { mtype, short, short, short }

/* 10 input (output for the server) channels for ten
subscribers */
chan SUBin [3] = [3] of { mtype, short, short, short }
```

The inbox of each Presentity is represented by a channel of size 3.

```
chan inbox [3] = [3] of { short }
```

Thus the above system can be imagined as we have four permanent IP addresses through which six different entities (3 Presentity and 3 subscribers) can communicate with the server.

5.1.2. Data Structures

The Presence service stores the names (and status for presentities) of Presentities and subscribers with

respect to communication channel (or IP address)

```
/* global data for presence service */
short pts [3]; /* presentities ip and name as value */
short sts [3]; /* presentities status */
short pt_count; /* Number of presentities logged in */
short sbs [3]; /* subscribers ip and name as value */
short sub_count; /* Number of subscriber logged in */
```

The information about the subscriptions of subscribers to the presentities is stored in following 2d array.

```
/* 2-dimensional array in spin
presentity1 presentity2.
subscriber1 X X
subscriber2 X X */
typedef subs {
short presentities[3];
};
subs subscribers[3];
```

The message format followed in all models is as follows: (msgType, id, ids, num). In this tuple the first is the message type which can be one from following: Pstreg, psfctch, psstatus, psunreg, inssend, insrecv, subreg, subreq, subacp, subref, stschg, regrej, and regacp, the id denotes the communication channel (or IP address) of the entity, ids denotes the communication channel (or IP address) of other entity (but this may be also be don't care for some message types), Num denotes name in some cases and status in other.

5.1.3. Presence Service and Instant Messaging Service

We model both of these services by two PROMELA processes. Channel capacity has been kept fixed to three messages. Both the services continuously wait for a service request. When a request is received the corresponding service procedure code is executed. The instant message service just forwards the instant messages to the relevant Presentity if the Presentity is online. If Presentity is offline then its messages are stored in the instant mailbox. The instant mailbox is modeled by three message channels for three Presentities with capacity of ten messages.

5.1.4. Presentities

We model the set of Presentities using spin processes. These Presentities are used in different ways in three models for testing different use cases of the services. Each Presentity has its own I/O channel for communicating with services. Our Presentity combines the roles of principal and user agent. Lastly we also

model a set of subscribers, which is mostly similar to the Presentity. We show a flow chart-state transition diagram for one of the Presentity, other diagrams are not shown, as they will become clear with this one.

5.2. Presence Service Model

In this model we have two Presentities that log in to the server. As the block diagram of presence service shows the main components of the system, we abstract out them into a single spin process that uses the above-explained input channels for interacting with clients. In this model one of the Presentity continuously changes its status, the other Presentity continuously checks its status, (in IMPP model this entity is called fetcher). Lastly there is a subscriber that subscribes to our constantly changing Presentity (after Presentity grants access to it to be added to its buddy list) and then starts receiving the notifications continuously.

This model will be used to verify the following property from the stated objective.

The validation criterion of interest in this case, particularly integrity of access rules and notification system of subscribers and fetchers, were specified using a watchdog process with assertions included for checking the data.

5.3. Instant Message Model

Here we have a Presentity continuously sending a stream of red, blue and green messages to another Presentity through instant messenger service. Here we use the result found by Wolper [15] that only three types of messages are needed to check the validity of any flow control protocol. Here the main purpose is to test the possibility of message loss in chat session, because as the presentities involved in a chat directly communicate through their IP there are many possibilities to simulate and test for this case.

The Validation criterion of interest in this case, particularly liveness property of instant message service (impossibility of message loss) especially in a chat session, were specified using LTL claims, like the one below which is translated into a never claim by Spin (spin -a ltl). The literals in claim represent Boolean conditions of sending and receiving the messages.

```
/* (! [] (sr -> <> rr)) || (! rr U rb) */
```

5.4. Multiple Login Model

This model is similar to the first model in terms of components, but varies in functionality and validation criterion. Here every Presentity tries to log in with a name and tries another one if fails for the first time, it then logs out and goes on doing the same continuously. The other Presentity does the same. The most important and powerful feature of this model is that

both presentities use the same login name. Hence this seems to be the best model to validate any requirement related to multiple login etc.

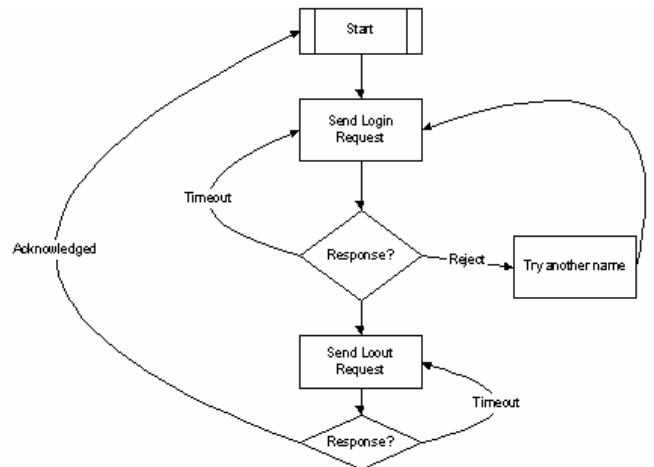


Figure 3. Flow chart-state transition of multiple login model.

The validation criterion of interest in this case, particularly the multiple login control feature, were specified using assertion in a watchdog process to check it in every state:

```

proctype CheckMultipleLogin()
{
do
:: atomic {
assert(!(pts[0] != 0) && (pts [1] != 0) &&
(pts [0] == pts[1])); }
od;
}
    
```

6. Simulation and Verification Results

We now discuss the most interesting part of the work i. e., the verification results. We will consider them in the same sequence as we modeled them and with respect to the properties that we wanted to verify. We state the verification property for each model, then its simulation results and lastly its verification with PAN (PROMELA analyzer).

As mentioned model 1 dealt with the validation properties related to the integrity of access rules related to subscribers and the notification system of subscribers and fetchers.

The simulation of this model didn't show any violation or deadlocks, but helped to implement the model. Initially we were unable to perform exhaustive search on the first model. Hence we tried to increase the search depth (pan1 -m1000000) that resulted in memory shortage.

Table 1. Results of verification and simulation for all models.

Model	States Statistics	Analysis Type	Memory Used (In Megabytes)
M O D E L			
M O D E L 1	State-vector 344 byte depth reached 64002 83328 states stored 93917 states matched 177245 transitions(=stored +matched) 2 atomic steps hash conflicts: 8390 (resolved)	Unable to search completely using any of the strategies (Supertrace Approximation, Bit State Compression)	2.68391e+08 bytes used 102404 bytes more needed 2.68435e+08 bytes limit 29.331 equivalent memory usage for states (stored*(State-vector + overhead)) 27.480 actual memory usage for states (compression: 93.69%) State vector as stored = 322 byte + 8 byte overhead 1.049 memory used for hash-table (-w18) 240.000 memory used for DFS stack (-m10000000) 268.391 total actual memory usage
M O D E L 2	State-vector 404 byte depth reached 2814 606879 states, stored 886066 states, matched 1.49294e+06 transitions (= stored+matched)	Supertrace Approximation and Exhaustive search after increasing the memory size	250.034 equivalent memory usage for states (stored*(State-vector + overhead)) 229.051 actual memory usage for states (compression: 91.61%) State vector as stored = 369 byte+8 byte overhead 1.049 memory used for hash-table (-w18) 24.000 memory used for DFS stack (-m10000000) 253.717 total actual memory usage
M O D E L 3	State-vector 392 byte depth reached 842482, errors: 0 1.8788e+06 states, stored 2.48418e+06 states, matched 4.36299e+06 transitions (= stored+matched) 1 atomic steps hash factor: 2.23243 (best coverage if >100) (max size 2^22 states)	Supertrace Approximation and Exhaustive search after increasing the memory size	Stats on memory usage (in Megabytes): 744.007 equivalent memory usage for states (stored*(State-vector + overhead)) 1.049 memory used for hash-array (-w22) 28.000 memory used for DFS stack (-m10000000) 35.295 total actual memory usage

Next we went to the supertrace approximation, still we were not able to complete the search. But increasing the memory size did the job with the results shown in Table 1. It also showed some unreachable code, but it was because we didn't wanted to stop the process to allow for maximum possible interleaving during simulation. It also showed that timeouts would not occur in normal circumstances, hence unreachable.

Since the hash factor was very low, which shows we were not able to cover a good portion of search space. If we calculate from the output the amount of memory that will be required for exhaustive analysis it is 425.913280 MB. Hence we tried again with an exhaustive search with increased memory size by using the DMECNT label with a value of 29 ($2^{29} = 536870912$).

Similar analysis was performed on all models with results summarized in Table 1.

Although the exhaustive search of the models didn't show any deadlock, live lock or assertion error but we were able to show by simulation that message loss can occur in case of a Presentity involved in chat session going down. Although it is trivial to simulate this case, and IMPP still does not specify this aspect but still the same model can also be used to incorporate the new strategies in future and verify them.

6. Conclusion

We have achieved the primary goal of modeling and verifying IMPP. The set of models we have developed is powerful in that it can be easily extended or used to test all the basic properties of IMPP. The model can also be used as a test bench against which the upgrades of IMPP can be tested as it evolves with time. We demonstrated only a few basic properties, to mention other interesting ones, they may be, Access control rules of offline Presentity, Access control rules of Notifications or Instant Message, Conference of more than two presentities, server or client crash. The Model may also be tried on several other model checkers. As stated in the abstract, we were also able to exploit the formal verification tool, Spin, to its maximum modeling potential.

Acknowledgement

Authors would like to acknowledge the support of King Fahd University of Petroleum and Minerals for this research.

References

- [1] Crocker D. and Peterson J., "Common Profile: Instant Messaging," *Internet-Draft*, Available: <http://www.ietf.org/internet-drafts/draft-ietf-impp-im-01.txt>, December 2002.
- [2] Crocker D. and Peterson J., "Common Profile: Presence," *Internet-Draft*, Available: <http://www.ietf.org/internet-drafts/draft-ietf-impp-pres-01.txt>, December 2002.
- [3] Crocker D., Diacakis A., Mazzoldi F., Huitema C., Klyne G., Rosenberg J., Sparks R., Sugano H., and Peterson J., "Address Resolution for Instant Messaging and Presence," *Internet-Draft*,

- Available: <http://www.ietf.org/internet-drafts/draft-ietf-impp-srv-01.txt>, August 2002.
- [4] Crocker D., Diacakis A., Mazzoldi F., Huitema C., Klyne G., Rosenberg J., Sparks R., Sugano H., and Peterson J., "Common Presence and Instant Messaging (CPIM)," *Internet-Draft*, Available: <http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-03.txt>, August 2002.
- [5] Day M., "HTTP Envy and Presence Information Protocols," *Technical Report*, Available: [http://domino.watson.ibm.com/cambridge/research.nsf/2b4f81291401771785256976004a8d13/56b47d188440d2dd8525661e0051dac1/\\$FILE/http-envy.pdf](http://domino.watson.ibm.com/cambridge/research.nsf/2b4f81291401771785256976004a8d13/56b47d188440d2dd8525661e0051dac1/$FILE/http-envy.pdf), July 1998.
- [6] Day M., "Instant Messaging/Presence Protocol Requirements," *Internet-Draft (RFC 2779)*, Available: <http://www.ietf.org/rfc/rfc2779.txt>, February 2002.
- [7] Day M., "Presence and Instant Messaging via HTTP/1.1: A Coordination Perspective," *Technical Report*, Available: <http://domino.watson.ibm.com/cambridge/research.nsf/2b4f81291401771785256976004a8d13/e512013fe84008bf852568d20050d2a8?OpenDocument>, March 1993.
- [8] Day M., Rosenberg J., and H. Sagano., "A Model for Presence and Instant Messaging," *Internet-Draft (RFC 2778)*, Available: <http://www.ietf.org/rfc/rfc2778.txt>, February 2002.
- [9] Godefroid P., "VeriSoft: A Tool for the Automatic Analysis of Concurrent Reactive Software," in *Proceedings of the 9th Conference on Computer Aided Verification*, Haifa. Lecture Notes in Computer Science, Springer-Verlag, vol. 1254, pp. 476-479, June 1997.
- [10] Godefroid P., Herbsleby J. D., Jagadeesany L. J., and Liz D., "Ensuring Privacy in Presence Awareness Systems: An Automated Verification Approach," *Technical Report*, 2002.
- [11] Hudson G., "The Zephyr Protocol," *Technical Report*, Available: <http://web.mit.edu/zephyr/doc/protocol>, February 2000.
- [12] Isaksson H., "Version 5 of the ICQ Protocol," *Technical Report*, Available: <http://www.algonet.se/~henisak/icq/icqv5.html>, April 2001.
- [13] Klyne Atkins G., "Common Presence and Instant Messaging: Message Format," *Internet-Draft*, Available: <http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-msgfmt-07.txt>, October 2002.
- [14] Sugano H., Fujimoto S., Klyne G., Bateman A., Carr W., and Peterson J., "Presence Information Data Format," *Internet-Draft*, Available: <http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-pidf-07.txt>, December 2002.
- [15] Wolper P., "Expressing Interesting Properties of Programs in Propositional Temporal Logic," in *Proceedings of the 13th ACM Symposium Principles of Programming Languages*, St. Petersburg, pp. 148-193, January 1986.



Soheli Khan received his Bachelor of engineering in electronics and design technology from Shri Ramdeobaba Kamla Nehru Engineering College, Nagpur, Maharashtra, 1999. He has been studying at King Fahad University of Petroleum and Minerals, SA, since 2002, and about to receive his MSc in information and computer science. He is a member of Project Management Institute, Joint Endeavor of Delphi, Global Grid Forum, and IEEE working groups. His research interest includes software engineering and database system.



Abdul Waheed Abdul Sattar is an assistant professor in Computer Engineering Department at KFUPM. Before joining COE, he was working at Inktomi Corporation in Foster City, California, USA as a performance software engineer in network products division. He was a research staff member at NASA Ames Research Center, Moffett Field, California, USA from May 1997 until July 2000. He held a summer position in Concurrent Computing Division at Hewlett-Packard Research Laboratories in Palo Alto, California, USA in 1994. He received the BSc degree with honors in electrical engineering from University of Engineering and Technology, Lahore, Pakistan in 1991. He received the MS degree in 1993 and the PhD degree in 1997, both in electrical engineering from Michigan State University, East Lansing, Michigan, USA. His current research interests include performance evaluation, high-performance computing and networking systems, and multimedia systems. He has written over thirty refereed conference and journal papers on related topics. He is a member of the IEEE Computer Society.