

Using An XML Database To Coordinate Communication Between Mobile Computations On The Internet

Sarah Monisha Pulimood
The College of New Jersey
2000 Pennington Rd
Ewing, NJ 08628
1-609-771-2788
pulimood@tcnj.edu

ABSTRACT

One approach for sharing resources on the Internet is to use mobile computations (programs that can commence execution at one site, discover a need for a resource at a different site, halt execution, migrate, and continue execution at the new site). Migrating an executing process is a non-trivial task that is exacerbated by the heterogeneity and security concerns of the computing infrastructure. Native support for strongly mobile computations is an essential prerequisite for a user-friendly programming environment that can produce efficient and secure grid / Internet applications. Our prototype of a mobile computational model enables creation and execution of mobile computations. Communication between executing mobile computations within the model is a significant issue that we address in this paper. We discuss the motivation for, and design of, our solution of using an XML database to facilitate communication between computations that may be located on geographically widespread hosts.

Keywords

Internet programming, mobile computation, strong mobility, XML database, communication.

1. Introduction

Internet Programming envisions the sharing, selection, and aggregation of resources across a large network like the Internet based on their availability, capability, performance, cost, and ability to meet quality-of-service requirements. Mobile computations are a powerful concept that can facilitate sharing of resources on the Internet. A mobile computation is a program that can commence execution at one site, discover a need for a resource at a different site, halt execution, migrate, and continue execution at the new site [7]. In order to continue execution (without being forced to start over) the current state of execution must migrate along with the computation. A software artifact that can carry its closure with it on migration is considered *strongly mobile*, while one that merely carries its code is considered *weakly mobile*. Migrating a running process is a non-trivial task that is exacerbated by the heterogeneity of the computing infrastructure.

Grid technology is fast revolutionizing business models and scientific research methods in areas as diverse as bioinformatics and financial services industries. The heterogeneous nature of the Internet coupled with security concerns pose many challenges for internet-based or grid application developers. The Globus Toolkit and the Open Grid Services Architecture (OGSA) [10] have emerged as de facto standards, providing a framework for secure grid computing. The Grid Application Toolkit (GAT) [11], for example, provides an interface between the application level and the grid middleware. However, building a grid system or application to meet the specific needs of a user currently requires the integration of many tools and services that are available “off the shelf” or from the Grid community. Systems like D'Agents, MobileML, and Mobile UNITY are extensions of languages that were originally designed for non-mobile environments [17]. Java is considered a ubiquitous language that supports mobility. However, the Java Virtual Machine (JVM) does not allow capture of closure of running processes. There have been some efforts to modify or extend the JVM to enable closure capture. While there are advantages to be gained by utilizing features already available, the constraints, disadvantages, and overheads of circumventing this shortcoming, are detrimental to the efficient and secure execution of mobile computations.

Native support for strongly mobile computations is an essential prerequisite for a user-friendly programming language that can produce efficient and secure Internet applications. As a proof-of-concept, we designed and implemented a prototype of a Mobile Computational Model that includes the Mobile Computational Language (MCL), for developing applications with mobile computations, and a RISC-based Virtual Machine (RVM), that performs the actual execution of the computations. Preliminary performance testing of the RVM has yielded promising results [16, 17]. This paper describes faculty directed undergraduate research that explores the feasibility of utilizing an XML database to manage communication between mobile computations on a large heterogeneous network like the Internet. The bulk of literature in this area refers to autonomous mobile software artifacts as ‘agents’. In this paper, when referring to research conducted by others we adhere to their terminology. However, we refer to the software artifacts in our model as *mobile computations* to emphasize their strongly mobile nature.

In the Section 2, we provide a brief overview of the mobile computational model focusing on the communication aspects. In Section 3, we analyze the primary requirements and present our design for a database to facilitate communication between mobile computations. We also discuss the motivation behind our decision to use XML for the purpose. Finally, in Section 4 we present some concluding thoughts.

2. The Mobile Computational Model

An application developer uses the Mobile Computation Language (MCL) to generate a mobile computation. The language provides constructs for the developer to explicitly request resources or to initiate communication if desired. A computation within the Mobile Computation Model is accorded permission to execute on specific hosts that form a ‘computational subnet’ of the Internet. These hosts are in the ‘domain’ of the mobile computation. The Mobile Computation Manager (MCM) present on every host in the computational subnet verifies the ‘credentials’ of a mobile computation – its authenticity, domain, etc. It also runs some checks, based on the closure map and ‘maximum number of hops’, to minimize the possibility of malicious code attempting to tie up resources, before the computation is allowed to execute at the host. The RISC-based Virtual Machine (RVM) executes the mobile computation, while the MCM facilitates management of requests for resources unavailable at the current host and communication with other computations.

Communication between computations and migration of the computations are essential features of Internet and mobile applications. However, these are complicated by the uncertain and changing locations of the computations, security concerns, and the uncertainty of network conditions. Performance of the computation is influenced by code and data relocation techniques, micro level of optimization, network load generated by a single migration between two agencies, etc. Due to the lack of space, we restrict our discussion in this section to the Aglet [4], NOMADS [2], and Tracy [3] mobile agent toolkits.

2.1 Agent Communication

The two major categories of communication models in mobile agent systems are message passing and information space [3]. The message passing model, as the name implies, involves the sending and receiving of messages by mobile agents. Agents that transfer messages to other agents need to be able to identify the receiving agent as well as the current location of the agent [3]. The information space model involves providing agents with a single space where they can exchange information, data, and knowledge with one another [3]. Aglets, being Java-based, support weak mobility, i.e. when they migrate to another host, they carry their code as well as the objects in the program and the states of those objects, but they do not completely capture the state of execution (closure) when they move. Aglets use two forms of communication: message passing, for peer-to-peer communication between aglets, and the Java Remote Method Invocation (RMI) protocol [4, 10]. NOMADS is a Java-based mobile agent toolkit that supports strong mobility [2] by using the Aroma virtual machine (VM) to manage execution and capture of closure. Aroma and Oasis, the NOMADS agent execution environment, can be freely distributed but the main disadvantage is that the VM must be completely Java-compatible and efficient [2]. Agents communicate with each other using message passing [2]. The Tracy Mobile Agent toolkit [3] is another Java-based toolkit that uses the message passing protocol for communication.

2.2 Agent Migration

During the migration process, an aglet is cloned and a copy with the same data as the original one is sent to the remote host multiple times, leading to multiple instances of the same aglet [13]. This is not very efficient, but necessary since aglets only support weak mobility and Java does not allow explicit passing by reference. NOMADS agents are moved from one destination to another by calling the ‘go’ method [2].

The two main migration strategies in use are push and pull. In the push strategy the class closure (code of the agent and its objects) and the object closure (the serialized agent state) are transmitted along with the agent. In the pull strategy classes are searched for and loaded remotely and dynamically when required. Aglets adhere to the Mobile Agent System Interoperability Facility (MASIF) standard, so the code the aglet needs is loaded dynamically at runtime and / or during execution [13]. Agent

transfer in the NOMADS mobile toolkit is handled through the Oasis execution environment [2]. The default Oasis protocol handler supports its own custom-made agent transfer protocol, but new protocol handlers may be created to support other standard transfer protocols [2]. The Tracy Mobile Agent toolkit allows the programmer to specify whether they wish to use the push or pull migration strategy [3]. Aglets are transmitted over a network using Agent Transfer Protocol (ATP) and Remote Method Invocation (RMI). ATP is very similar to HTTP, and is platform independent [13]. The NOMADS mobile toolkit uses the sockets protocol to transfer agents from one location to another [2]. The Tracy Mobile Agent toolkit allows programmers to specify their own network transfer protocols to use for transmitting agents, although Transmission Control Protocol (TCP) is the default protocol [3].

2.3 Communication Within the Mobile Computational Model

All the mobile agent toolkits discussed in this section use message passing. Being Java-based, a major concern is the capture of closure. The Aglets and Tracy toolkits support only weak mobility, which is a drawback for our purposes. However, they do not require modification of the JVM. The NOMADS toolkit uses a modified version of the JVM, the Aroma VM, to support strong mobility. The result is poor performance as compared to the other Java-based mobile agent toolkits. We find that the communication models of existing toolkits require trade-offs that are not acceptable in the domain of mobile computations, where strong mobility is a necessity. While designing the communication model for our mobile computational system, the key issues we considered were the heterogeneity of the Internet and the computer architectures, security concerns, and communication needs of the computations.

When a mobile computation is created, it is given a unique computation identifier. A computation B may communicate with other live mobile computations by exchanging ‘parcels’ under closely controlled circumstances to avoid unsafe data being passed to a verified computation in an attempt to subvert it or its host. In order to prevent unauthorized accesses to data, a parcel within this model can only be accessed by the mobile computation it is explicitly addressed to. A powerful concept like mobile computation has a very high potential for malicious use, so security and prevention of such misuse must be a crucial aspect of any Internet Programming model.

During execution, a mobile computation A may encounter an instruction to communicate information to another mobile computation B . This data is packaged into a parcel and addressed to the recipient computation. Since the computation has been verified to be trustworthy, it is assumed that the parcel is safe. At some point computation B , will encounter a requirement to obtain the data from A . The MCM attempts to locate the parcel from A addressed to B on the same host. If the parcel is not at the current host, the MCM checks for messages indicating that B has a parcel waiting for it. If such a message is found, the parcel is tracked and transferred to the current site. Once the parcel is found and received at the site, its contents are verified for security purposes and then handed over to B for its use. If the parcel is not found, B is informed so that it can either wait for the parcel or continue execution without it. If the computation decides to wait for the parcel, the MCM frees up all resources being used by it and moves it into a wait state.

Communication between computations is accomplished through message passing. One issue to consider was whether to implement the synchronous or asynchronous protocol. Asynchronous protocols generally perform better than synchronous ones, but offer less reliability than the latter [3]. At first glance, it would appear that the synchronous socket protocol is better because reliability is of high priority. However, the primary goal of our system is efficiency so we implement the asynchronous protocol with additional checks to assure reliability. Although there are user-friendly APIs available with the RMI protocol, we decided against using it for two reasons. First, this protocol is specifically for Java-based systems, which ours is not. Second, the performance of RMI is poor when compared to that of sockets. Sockets have the additional advantage that they are platform and language independent. Another issue to consider is whether the push or pull migration strategy is to be used, and whether any of them works well in every situation. The fact is that neither method is particularly efficient in all situations. In general, the pull strategy is more efficient than the push strategy, but in certain scenarios, the push strategy may be more efficient [3]. We use the push strategy thus avoiding the additional network traffic generated by searching for and downloading required code. In a later stage of the project, we will consider a combination of the two strategies to improve efficiency. The primary issue, however, is tracking down the current location of the parcel addressed from A to B rather than the actual mechanism used for transmitting the data. In addition, the geographical distance between hosts on the network and the instability of the network connections make communication using traditional methods impractical for mobile computations. In the pre-web days, research efforts in distributed computing focused primarily on local area networks with a limited number of nodes, making communication between computations or processes manageable. For example, REV [20] nodes did not share memory but communicated through messages. In Orca [1], processes could communicate through shared data even if the hosts on which they ran did not have physical shared memory. Distributed Oz [19] used a two-level addressing scheme with local and global addresses to refer to nodes. In Obliq [6], communication between two independent servers was mediated by a shared global name server, which allowed the servers to export and import local values.

Our model uses a lightweight database to manage relevant data for computations and parcels to facilitate communication.

3. Modeling The Database

3.1 Analyzing the Requirements for the Database

The design of such a database is complicated by the unusual and demanding nature of the environment. Each mobile computation is a potential client for data. It is conceivable that the mobile computational system can grow to include millions of computations on millions of hosts. At any given time, there may be several thousand clients requesting data and update operations. These clients may be operating on a diverse set of platforms and all this data would need to be transferred over a network with limited bandwidth and uncertain connections. Aside from handling a large volume of data, the system must also be able to perform updates and process queries. For example, a host's location may change; a host's status may change from 'secure' to 'insecure' or vice versa; computations are created, executed, suspended, killed, or terminated; network conditions may change; parcels are created and need to be picked up, etc. Efficient access to data about potential hosts, actual hosts, computations, parcels, and network connections for the purpose of coordinating a system of mobile computations requires a data management system tailored to the specific requirements of the mobile computational model. Primarily the database will need to store data about hosts, computations, computation history, network connections, and parcels [16].

- Host data includes host identification, IP address, resources available, security level, whether the host is accepting mobile computations or not, and location stability. Hosts may be 'potential', i.e. they do not currently have any resident mobile computations, but may in the future. Hosts are 'actual' if they have one or more resident mobile computations.
- Computation data includes fields such as a unique computation id (CID), current location, resources required, maximum number of hops, hops used, originator, and closure map. (The closure map is unique to the particular state of execution.)
- Parcels and history are associated with mobile computations – creator and recipient.
- Significant events, like initiation, migration, and termination, are associated with each computation.

Other issues to be considered include performance measures for queries, updates, and recovery; operability across diverse platforms; security; and scalability. The database should be both fast and powerful while being secure, available to potential hosts, available to a large number of computations, capable of handling large volumes of simultaneous queries, and capable of representing and handling imprecision in data and queries. Based on these requirements, we developed the high-level entity-relationship diagram as shown in Figure 1.

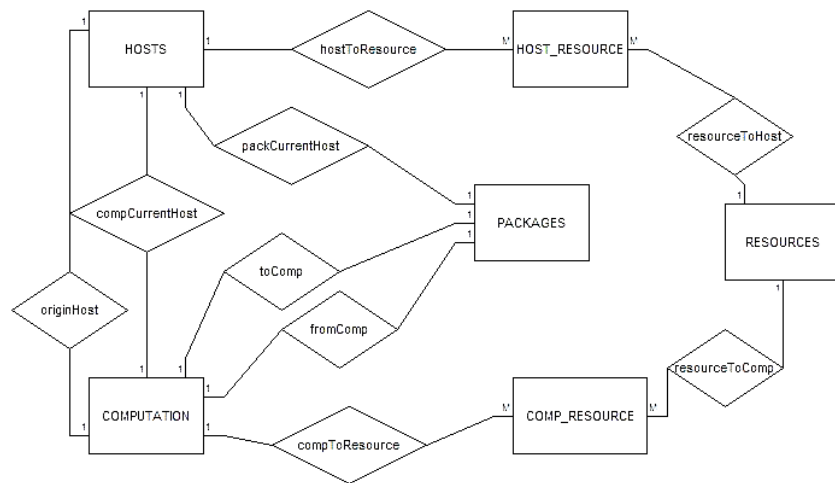


Figure 1: Entity-Relationship Diagram for Mobile Computational Model Database

3.2 Evaluating the Database Models

We now review some current database models in light of the requirements of our mobile computational model.

3.2.1 Relational Databases

The typical implementation of the relational database model entails a client-server architecture, where all the data resides on a centralized file server and software on each client machine allows queries and other transactions to be submitted to the central database by operators on each client machine. The model has been modified, upgraded, debugged, and optimized for decades now in response to experience and user demand. Security is generally good, partly due to information hiding and centralized control. Recovery is fast, especially if there is a backup server, because the concurrency issues are relatively straightforward

and all necessary data, including transaction logs, can be found at a limited number of locations. These databases have been shown to work well for a broad variety of tasks. Moreover, the relative simplicity of the theory, as compared with other currently used models, is an attractive feature. A major disadvantage of the relational model for a mobile computational environment is that it can handle only well structured, certain, and precise data. Poorly structured input, such as images or long text strings, and missing or unavailable data are not well supported [9]. Processing speeds for various individual transactions tend to be high but significant performance degradation can occur with a large volume of transactions. Individual updates, like insertion or deletion, tend to be straightforward. A delay occurs due to the transmission time between the client and the server. However, if the server is available these updates can take constant time since the records are generally accessed via hash tables. Updating index structures, which are usually trees, take longer ($\log N$) time. The most computation intensive queries, with joins, can be optimized to $N\log N$ time [9], although if the network transmission speeds are slow then the overall transaction times will be large.

3.2.2 Object-Oriented Databases

Object-oriented database management systems (DBMS) offer several advantages over the traditional relational DBMSs, although many are typically implemented as 'post-relational' rather than purely object-oriented. Advantages include information hiding, encapsulation, and ability to manage complex structures. These systems offer flexibility in the representation of data and a clear distinction between abstraction and implementation. Like the relational model, most object-oriented models have a client-server architecture and therefore have many of the same advantages and disadvantages as the relational model. Perhaps the biggest advantage of the object-oriented model in the context of the mobile computational model is the ability to handle semi-structured and unstructured data, as well as structured data. The total number of bytes stored and handled can be enormous (of the order of petabytes are possible) because each individual object can be large [8]. Object-oriented databases lack a secure theoretical foundation and standardization. The model is more complex and suffers from some of the performance degradation issues seen with relational databases. Although claims have been made that object-oriented databases can exceed the performance of relational database in completing queries and performing other transactions, in actual practice query optimization is very complex and negatively impacts performance.

3.2.3 Distributed Databases

Distributed databases may be relational, object-oriented, or based on some other model. However, they differ fundamentally from the traditional client-server architectures since they store data at more than one site. Distributed databases require more complex algorithms to preserve the traditional advantages of consistency and integrity [18]. Recovery techniques are also more complicated but faster than with centralized databases if there is a high degree of data replication. The performance of distributed databases depends greatly on the nature of the transaction, the location of the needed data, network delays, and the tolerance for temporary inconsistency. Queries requiring joins of tables at several sites can be optimized but are still generally slow since large amounts of data must be transferred over a network. Other transactions, such as modification, insertion, and deletion are very fast only if all the copies of the necessary data are available locally. Replicated data and strict consistency means that a fair amount of network traffic is necessary to enforce locking protocols. Kossman [15] observes that both replication and data caching can greatly speed up the processing of queries. Since future queries are likely to be similar to recent queries, cached data will often be relevant.

3.2.4 XML Databases

XML, or Extended Markup Language, is a meta-language used to format documents that can contain varying data formats, and it is well suited for operating across a heterogeneous collection of platforms. This gives it a significant advantage over standard relational formats. Although XML is not designed for efficient indexing and querying, most major database vendors now offer facilities for efficient interfaces with XML query languages and translation of data to and from XML. Direct mapping between the relations of a traditional relational database and XML provides reasonable performance and preserves the traditional advantages of a relational DBMS, including stability, portability, and scalability [14]. Implementing queries in XML tends to be complicated and large XML documents can take considerable time to load into memory. Query tools are not currently well adapted to large documents. This is an area of active research however. For example, Ives et al [12], have demonstrated a system for optimizing XML queries that can efficiently handle large documents, with performance roughly equivalent to that achieved by traditional relational database systems.

3.2.5 Fuzzy Databases

The fuzzy database model provides good support for handling uncertain and imprecise data and is able to generate responses to fuzzy or imprecise queries [5]. Fuzzy databases are based on fuzzy logic, which allows attributes to have degrees of membership in a class [22]. For example, a query may request an "excellent" network connection but a "good" connection may be acceptable as a result if no "excellent" connections are available. Currently fuzzy database models are mainly relational and centralized, although some theoretical work on object-oriented fuzzy databases has been done. Performance, especially with large data sets and complex queries, tends to be poor since they are computationally intensive.

3.2.6 Analysis

Silberschatz et al [21] propose that some database assumptions should be re-evaluated, particularly for databases with high demands for data access operating over the Internet. They suggest that a distributed model may be necessary and that the traditional database strictures regarding timeliness, consistency, and completeness be relaxed. The cost of moving data over the Internet can be extremely high. Consequently, this movement must be minimized as much as possible by maintaining frequently requested data on local nodes. At the same time, ensuring that all copies of data values be consistent and current slows performance by requiring too many update operations, too much traffic, and too many blocks on accessing and modifying data. Indeed, the most formidable issue to be resolved for our database is the performance penalty that can result from a large volume of simultaneous transactions. Thus, the database should be fragmented, with replication and caching on hosts within the subnet of the mobile computation. Some discretion will be required to determine when consistency must be maintained to minimize network traffic. Changes in status of hosts or network connections must be disseminated quickly. In general, most hosts will contain a snapshot of a small subset of the data. These can be synchronized at periodic intervals with the master site(s). Since much of the data about the Internet and the mobile computational environment is uncertain and imprecise, the database must be capable of handling unstructured and imprecise data. Currently, the fuzzy database model shows the most promise in this regard. However, performance issues, particularly with respect to scalability, will need to be overcome before we can seriously consider this as a viable option.

A system of mobile computations will operate over the Internet and across heterogeneous platforms. Given the scalability of current fuzzy databases, the requirement to manage imprecision and uncertainty needs to be addressed in another way. We considered implementing fuzzy queries on a distributed relational database but rejected this option due to the performance constraints and lack of efficient support. Hence, we determined that the ideal database to coordinate the mobile computations in our system would be distributed, with files organized according to some common standard, such as XML. XML is already the standard for sending such data over the Internet and a great deal of software is being written for it. It offers the added advantage of the ability to model semi-structured as well as structured data. In addition, it would have good performance characteristics and excellent scalability. The Globus Toolkit has migrated to implementing grid functions as web services [10]. While the web service itself may be written in the developer's language of choice, the web service deployment descriptor (WSD) that tells the server how to publish the service is written in the Web Service Description Language (WSDL). The fact that WSDL is an extension of XML provides a further motivation to use an XML database.

3.3 The XML Database for the Mobile Computational Model

The structure of XML documents is hierarchical, so the computation is represented as an element that contains other elements such as the ID of the computation and number of hops. The challenges of designing an XML database surface when designing the actual structure that the XML document must follow. In order to ensure that the XML database is platform-independent and can be accessed by any application, it follows an XML Schema Definition (XSD) that defines the actual structure that an XML document must adhere to. It limits what elements can appear within it, along with how many times they can appear, what type of data they can contain, etc. Constructing the XSD is more complicated than building the XML document since the structure of the XML must be translated, allowing for necessary flexibility, and restrictions on data types. Fortunately, XSD files are actually XML files, so there is no need to learn any new syntax to build them. Figure 2a shows a part of the XML Schema Definition we use, while Figure 2b shows a part of a sample XML document.

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xsd:element name="meta">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="computation"
          type="Computation" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  ...
</xsd:schema>
```

Figure 2a: XML Schema Definition

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<meta
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="meta.xsd">
  <computation>
    <cid>15</cid>
    ...
    <currentHops>2</currentHops>
  </closure>
  ...
  </closure>
  </computation>
</meta>
```

Figure 2b: Sample XML Document

4. Conclusions

Mobile computations are powerful tools that can enable more efficient, reliable, and secure utilization of idle resources on large networks like the Internet. The prototype of our mobile computational system has been demonstrated to prove that native support for mobility is a key ingredient for achieving these goals. In this paper, we discussed our approach for facilitating communication by using an XML database to keep track of the location of computations and parcels. XML is now a standard format for providing and transmitting content on the Internet, making it well suited for the task. The overheads of maintaining and updating the database are offset by the enormous benefits proffered by secure and efficient resource sharing and utilization.

5. Acknowledgments

This paper draws upon background research and initial design by John Gaskins and William Kurzius, under my guidance, during their course of undergraduate study.

6. References

- [1] Bal, H., Kaashoek, F. and Tannenbaum, A. S. 1992. Orca: A language for parallel programming of distributed systems. *IEEE Trans. Softwre Engg.* 18, 3, pp. 190-205.
- [2] Bradshaw, J., Breedy, M., Groth, P., Hill, G., Jeffers, R., Mitrovich, T., and Suri, N. An Overview of the NOMADS Mobile Agent System. *2nd International Symposium on Agent Systems and Applications, ASA/MA2000*, Zurich, Switzerland, September 2000.
- [3] Braun, P. and Rossak, W. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Elsevier, Inc., 2005.
- [4] Bubendorfer, K. and Lu, F. A RMI protocol for Aglets. *In Proceedings of the 27th Conference on Australasian Computer Science*, Dunedin, New Zealand). 2004. Estivill-Castro, Ed. ACM International Conference Proceeding Series, vol. 56. Australian Computer Society, Darlinghurst, Australia, pp. 249-253.
- [5] Buckles, B. and Petry, F. E. 1995. Fuzzy Databases in the New Era. *Proceedings of the 1995 ACM Symposium on Applied Computing*. pp. 497-502.
- [6] Cardelli, L. 1995. A Language with Distributed Scope. *Computing Systems, USENIX*, 8, Jan. 1995.
- [7] Cardelli, L. Global Computation. *ACM SIGPLAN Notices*, 32 (1997), No. 1, pp. 66 – 68. Also available from *ACM Computing Surveys*, vol 28, Issue 4es, December 1996, Article 163.
- [8] Connolly, T. and Begg, C. 2005. *Database Systems: A Practical Approach to Design, Implementation, and Management*. 4th ed., Addison-Wesley.
- [9] Elmasri, N. and Navathe, S. 2004. *Fundamentals of Database Management Systems*. Addison-Wesley.
- [10] The Globus Alliance, <<http://www.globus.org/>>.
- [11] GridLab <<http://www.gridlab.org/>>.
- [12] Ives Z. G., Halevy, A. Y., and Weld, D. S. 2002. An XML Query Engine for Network-bound Data. *The International Journal on Very Large Data Bases*, 11, 4, 380-402.
- [13] Karjoth, G., Ono, K., and Oshima, M. Aglets Specification 1.1 Draft. Tech. Report. IBM Tokyo Research Laboratory. October 2005. <<http://www.trl.ibm.com/aglets/spec11.htm>>.
- [14] Khan, L. and Rao, Y. 2001. A Performance Evaluation of Storing XML data in relational Database Management Systems. *Proceedings of the 3rd International Workshop in Web Information and Data Management*, pp. 31-38.
- [15] Kossman, D. 2000. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, 32, 4, pp. 422-469.
- [16] Pulimood, S. M. 2003. A Mobile Computational Model for Internet Programming. Ph.D. Thesis, Tulane University, New Orleans, March 2003.
- [17] Pulimood, S. M. and Belkhouche, B. 2004. A Mobile Computational Model for Internet programming. *In Proceedings of the 42nd Annual Southeast Regional Conference, ACM-SE 42*, Huntsville, Alabama, April 02 - 03, 2004. ACM Press, New York, pp. 347-352.
- [18] Ramkrishnan, R. and Gehrke, J. 2003. *Database Systems: An Application-Oriented Approach*. 3rd ed., McGraw-Hill.
- [19] Roy P. v. et al. 1997. Mobile Objects in Distributed Oz. *ACM Trans. Program. Lang. Syst.* 19, 5, pp. 804-851.
- [20] Stamos, J. W. and Gifford, A. K. 1990. Remote Evaluation. *ACM Trans. Program. Lang. Syst.* 12, 4, pp. 537-565.
- [21] Silberschatz, A. and Zdonik, S. 1997. Database Systems – breaking out of the box. *ACM SIGMOD Record*, 26, 3, pp. 36-50.
- [22] Yazici, A. and George, R. 1999. *Fuzzy Database Modeling: Studies in Fuzziness and Soft Computing*. Physica-Verlag.