Performance Evaluation of Eventing Web Services in Real-Time Applications

Hossein Saiedian, ITTC, University of Kansas
Shawn Mulkey, University of Kansas

ABSTRACT

The objective of this research is to study the application of Web services technology in distributed real-time data delivery systems, as well as to determine the appropriate contexts in which such a design can be considered. We focus on distributed real-time systems and more specifically, on distributed soft real-time systems, which stand to benefit most from the use of Web services technology. We provide a means to evaluate the inclusion of Web services-based middleware in real-time system design. The decision to use the standardized data representation and communications protocols of Web services can bring tremendous value and benefit to both the service provider and the end user of a real-time system; however, the temporal performance of such systems is a critical factor. This research examines the most significant general performance considerations applicable to such systems and more specifically, provides a model to be used in the determination of whether a given system configuration can meet a specific soft real-time performance target.

REAL-TIME SYSTEMS AND OPEN STANDARDS

As the information technology field has grown in maturity and sophistication, so has the complexity of the systems on which it depends. A key enabler of interoperability and agility in modern distributed systems is the use of open standards for data definition and software interoperation. Open standards play a large role overall in software design, providing benefits such as vendor neutrality and community development in disciplines ranging from graphical display definitions to standardized network protocols. By leveraging standardized definitions of data and behavioral interfaces, systems designers were able to decouple software components from long-lived dependence on implementation details. Unfortunately, such flexibility can come at an unacceptable cost in performance. This performance cost is among the greatest risks in maintaining reliable real-time performance and represents a real dilemma to the designers of strict real-time systems. The focus of this research is on soft real-time services and the emerging Web services standards for message delivery, data definition, and integration standards.

A soft real-time system is one which conforms to an average or best-case response time requirement, as opposed to guaranteeing a minimum or worst case requirement as in hard real-time systems [1]. A soft real-time service guarantee can be defined more formally as a statistical confidence that some maximal delivery time can be maintained between a system stimulus event and any given end user of that system [2]. This maximum occasionally can be exceeded, but with a statistically bounded frequency.

Real-time systems, in general, and particularly, soft real-time systems, stand to gain substantial interoperability and extensibility with the inclusion of abstract middleware in their design and implementation. Middleware in any system provides a means to separate high-level software components from lower-level implementation details. This separation of concerns promotes extensibility and reuse, as well as simplification of maintenance, all of which would be more difficult in a tightly integrated system. A key enabler of interoperability and agility in modern non-real-time systems is the use of open standards for data definition and software interoperation. The use of standards-based design has been practiced for several decades. Early efforts included the Object Management Group's common object request broker architecture (CORBA) specification, the component object model supported by Microsoft, and boutique efforts such as [3] and [4]. More recently, the specifications developed by W3C and Oasis represent some of the most ambitious work to date in common interface and data definition standardization. All of these efforts were undertaken with the common goal of providing a method for software components to intercommunicate without the requirement of a common language, operating system, or programming model.

By leveraging these standard definitions, systems designers can decouple software components from long-lived dependence on implementation details. In turn, this provides for flexibility and interoperability that might otherwise be impossible. Unfortunately, such flexibility can come at an unacceptable cost in performance. This performance cost is among the greatest risks in maintaining reliable real-time performance and represents a real dilemma to the designers of such systems. The use of such standards would promote component reuse and both internal and external interoperation, thereby, reducing both development and maintenance; but designers of real-time systems have as their highest priority to maintain established and bounded system performance for the system consumers.

The standardized protocols examined in this research include established and emerging Web services standards that were approved or are under consideration by the W3C or similar bodies. The foundational standards analyzed include the XML for data description, Simple Object Access Protocol (SOAP) for message transport, HTTP for client-server interaction, and well-established protocols such as TCP/IP to provide lower-level support functions. All of these protocols serve to normalize communication across heterogeneous computer systems and network topologies. The popularity of Web services in distributed systems largely is due to the ubiquitous support for the standards on which they are based. Nearly every operating system and programming framework provides low-level support for the protocols used in standardized Web services (e.g., TCP/IP sockets, XML parsing, etc.). Unfortunately, this ubiquity comes at the cost of performance as the most common and typically least optimized standards typically are employed in the most open systems.

The cost of performance, and particularly performance variability, must be considered seriously when Web services are included in any system design [5]. In addition to these foundational protocols, several extended protocols were developed to support real-time functionality and other associated features. The protocols most applicable to real-time systems are the ones that enable asynchronous communication and include the WS-Notification system proposed by IBM *et al.* in [6] and the WS-Eventing protocol proposed by Microsoft *et al.* in [7]. The use of asynchronous messaging is required both for reasons of performance and for the ability to use the interrupt-driven or publish-subscribe [8] pattern of behavior common in real-time systems. The WS-Notification and WS-Eventing protocols are relatively new but so far have shown the most promise toward establishing a general method of providing asynchronous messaging using standardized host and client interfaces [9]. Note that the term [[eventing Web services]] as used in this analysis refers to both protocols and to the general class of asynchronous Web service-based messaging that they represent.

THE ANALYSIS MODEL

A central goal of this research was to develop a model considering quantifiable factors that could affect the performance of a real-time system using eventing Web services as its notification mechanism. Eventing Web services are those that asynchronously pass events from a publisher to a subscriber as described in the WS-Notification and WS-Eventing specifications. Enumeration and analysis of factors affecting performance in these systems is a key step in the adoption of any technology in a software system but is especially important in real-time applications, where the overall correctness of the system is determined in part by the timeliness and stability of system responses. The three general classifications of performance factors used in this model are listed in Table 1.

Model component	Description
Transport protocol	Communications overhead including transmission time and communication protocol handling
External processing elements	Extra processing required for meta-data functionality (e.g., security)
Data model	Size and detail of the data model including meta-data constructs (e.g., internationalization)

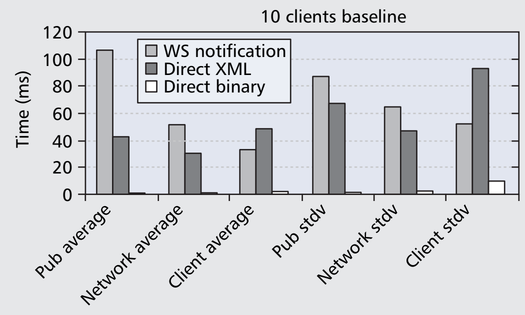
Table 1. *Performance analysis model components.*

The first category is the network transport system used to deliver data through the distribution network. For most Web services, the most common transport is HTTP, which is typically based on the TCP/IP protocol, which itself forms the foundation of most Internet technologies. The second category of performance classification is the data model used to represent data and the context in which it is consumed. The sustained performance of any Web services-based system is particularly sensitive to the data model used because the underlying XML encoding used to represent the data model grows quickly as a function of its size [10]. The final category for organizing performance factors are the external processing elements (EPEs), which are not concerned strictly with transporting data, but with transformation or other interaction with the data. The search for EPEs is facilitated by an examination of the SOAP message structure, as individual SOAP header elements typically map directly onto these extra capabilities. Other external processing must be derived from a deeper study of the system or from requirements analysis. The performance factors of many components of Web services-based systems have been studied individually, and the EPE provides a way to bundle each one as an element to the performance evaluation of eventing Web services in real-time systems. These classifications of eventing Web services performance factors were compiled into a set of formulas that are organized using these high-level principles. In addition to message latency, a method for classifying performance variability was introduced that builds on the statistical models for confidence and tolerance intervals [11]. These intervals can be used to reason about expected system performance and therefore structure a service level agreement between a provider and a consumer.

EXPERIMENT RESULTS AND ANALYSIS

EXPERIMENT OVERVIEW

The general concepts and specific performance model previously described were implemented and analyzed via a series of controlled tests conducted over several weeks. The purpose of these experiments was to deepen the general understanding of performance constraints in Web services-based real-time systems, as well as to test the viability of the formulas developed in this



Event time system	Upper confidence	Lower confidence	Total tolerance
Direct binary	3.18	3.76	16.52
Direct XML	117.96	124.74	283.63
WS-notification	194.14	201.18	350.66

Figure 1. *Baseline performance and variability.*

research. Therefore, the outcome of the analysis is a set of qualitative discoveries regarding the systems under study, as well as a quantitative framework for reasoning about the feasibility of such systems in real-world applications. The purpose of these experiments and the subsequent analysis was not to determine the raw limits of acceptable real-time performance, but rather to determine the factors that are the most relevant to the performance of a given system, and in which contexts eventing Web services-based real-time systems can be applied reliably.

EXPERIMENTAL APPLICATION

The application used in the experiments is a simplified version of a soft real-time system actually deployed in the financial data services industry. In the original system, trade and quote events occur in various stock exchanges, and these events are transmitted via a proprietary, managed distribution, wide-area network to each client's local network. The distribution system typically is rigidly controlled and rigorously provisioned so that factors such as latency and bandwidth are virtually always kept within acceptable means. Therefore, the latency introduced by the wide-area component of the system is considered constant in this research and not considered further. The client LAN may not be similarly provisioned and almost certainly is not dedicated to the application providing this particular data. The local network also is likely to be utilizing mainly commodity hardware and software protocols that may not be optimized for real-time usage. It is in this uncontrolled and heterogeneous environment in which these experiments take place and indeed, in which Web services are most often prescribed.

To provide input for the simulation, a set of data was captured from an actual real-time feed over random one-hour intervals (while the markets were open) over the course of several weeks. Several one-hour intervals were selected from these random traces and from these, either three- or ten-minute slices were selected from each and used as input to the simulation. The simulation consisted of a server that could replay these events at roughly the same intervals as the original stream. The server software that replayed the exchange events would broadcast into the local network using either a direct TCP/IP protocol or the higher level, but more widely consumed, HTTP transport. The data would be encoded either in a direct binary format or as XML and then packaged according to the appropriate messaging standards (i.e., WS-Notification and WS-Eventing). The direct TCP/IP mode of transmission either would send the bytes directly in a binary stream using proprietary encoding, which the clients would then decode, or using the XML-based standard in WS-Notification, but without the connection overhead of HTTP.

EXPERIMENT APPLICATION EXTENSIONS

To demonstrate the technique of EPE analysis, several simulated external functions were developed for this research. The first was a simulation of authentication by performing a series of modular arithmetic operations on a character string of constant length that the publisher placed in a message header element. The client then parsed this string and reversed the operation to perform the authentication. The second extension required the publisher to place a URL string in a message header element that the client read when consuming the message. The client then made an HTTP request for this URL and consumed whatever data was returned (which was of a random length for every call). This feature represents a data reference or routing type of function where the client is required to consult more than one network source before consuming the data. Each of these components emulate the application of Web services extensibility in the form of metadata headers that indicate extra processing required by the client in order to consume the message.

BASELINE EXPERIMENTS

The first set of experiments established a baseline for performance across the three primary system configurations (i.e., direct binary, direct XML, and Web services over HTTP). A single server published notification data to ten clients distributed among three different PCs.

The illustration in Fig. 1 visually depicts the drastic differences in both the absolute performance figures of direct binary versus XML-based encoding demonstrated in the experiments and also shows the relatively high degree of variability in the XML-based system. This variability could be attributed to a variety of factors, but one important indication is that the process of XML serialization is resource intensive (i.e., CPU and memory) for both the clients and the publishers in the system, and the non-determinism in those resources can contribute a great deal to the performance of the application. A similar broad conclusion is that growth in message size as a result of XML encoding also contributes to the performance volatility by increasing resource requirements required for multiple message packets, as well as increasing the likelihood that message resends will be required.

Performance comparisons between the direct binary methods and the XML alternatives present a strong case for the more compact binary format in real-time systems; however, there is a broad class of soft real-time applications that require bounded real-time performance but may tolerate a relatively high delay and some variabil-

Eventing system	Mean publisher time	Mean network time	Mean client time	System throughput	Upper confidence	Lower confidence	Total tolerance
Direct binary	0.60	1.33	2.91	5.77	4.262978	5.455196	33.3905
Direct XML	41.30	25.45	51.66	5.30	115.1768	121.6695	273.8026
WS-notification	153.50	91.19	21.74	2.01	261.975	270.9157	460.7882

■ **Table 2.** *Heavy load performance and tolerance.*

Eventing system	Mean publisher time	Mean network time	Mean client time	System throughput	Upper confidence	Lower confidence	Total tolerance
Direct binary	27.83	37.95	1.62	5.38	63.64	71.17	247.63
Direct XML	48.62	157.15	74.20	0.41	272.32	287.64	613.11
WS-notification	153.50	91.19	21.74	2.01	261.975	270.9157	460.7882

■ **Table 3.** *Wide area performance and tolerance.*

ity in mean event delivery time. The data associated with Fig. 1 also lists the confidence interval and tolerance levels for the baseline experiment as calculated via the performance model.

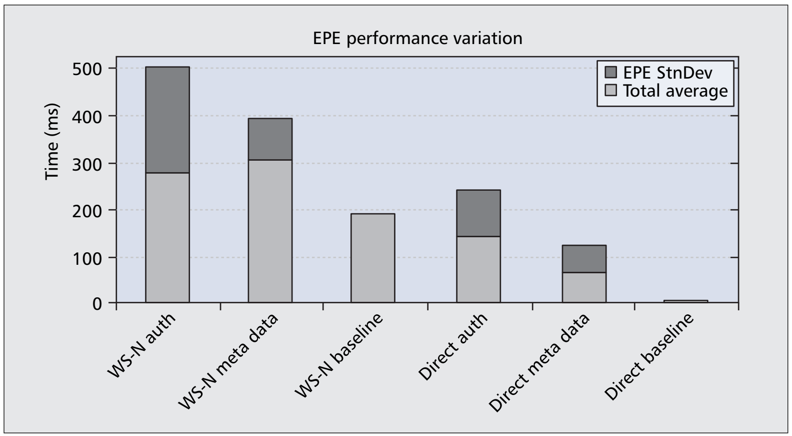
The confidence interval holds close to the original cumulative average, which is principally due to the large sample size. The tolerance interval exhibits a wide latitude from the mean, which appropriately captures the volatility introduced by the binary XML and Web services-based systems. Note that every confidence interval presented in this section is given at the 95 percent level, and the tolerance is given at 99 percent confidence for 90 percent one-sided coverage. Figure 1 also indicates the overall variation and volatility in the baseline. Recall that the tolerance interval provides a statistical view of performance given the data set provided. This implies that a practical application of this process should be conducted over several experiment repetitions to ensure a sufficient level of accuracy. For the purpose of this research, these tolerances provide the first point in a trend of performance that highlights the variability introduced by the Web services-based protocols.

Heavy Load

The second experiment set used the same parameters as the first in each case, but the number of clients was doubled from 10 to 20. Such a change in a distributed application often meets with unexpected results as the requirements for resources such as network bandwidth and server CPU cycles increase dramatically. Table 2 presents the basic performance statistics for the heavy load experiment. The raw data shows a relatively sharp change in throughput for the eventing Web services-based system, but little change for the other two cases at this load level. The increase in this last case is localized mainly to the publisher, which speaks partly to the queuing of notifications over additional HTTP connections and also to the extra processing required to prepare each message. Table 2 also provides the confidence and tolerance intervals for the heavy load case. The tolerance intervals are relatively unchanged except in the Web services case in which the extra publisher processing requires an incremental increase in total event delivery time.

Wide Area Network

For the wide-area network test, the baseline experiments were conducted as before, except that the server was in an office 230 miles away and was connected to the clients via a private T1 communications line. Note that due to limited server availability, this test was conducted only for the direct binary and Web services eventing-based test cases. Because of the networking hardware between the sites and the unpredictable traffic encountered during the test, this experiment had a higher degree of variability in the results. The basic performance statistics are given in Table 3, and the tolerance intervals for the wide-area case also are provided. The propagation time over a network of this distance is much higher relative to the local network baseline, but propagation time of the various message sizes alone does not account for the sharp increase in network processing time for both cases studied. This leaves the network processing nodes (routers, gateways, etc.) and connection management (including packet loss) as factors affecting network performance. The overhead in managing an HTTP connection also is evident in the network processing time and total throughput for the Web services case. The extreme performance variation illustrated by the tolerance times underscores the concept that a Web services-based eventing system likely is best deployed at the network edge (i.e., on the client LAN) and that compact proprietary encoding and protocols should be used within the greater distribution network. This arrangement maintains the benefits of standards-based communication and data representation between providers and external consumers but increases the cost of change within a provider's distribution system.



■ **Figure 2.** *EPE performance deviations.*

Eventing system	Mean publisher time	Mean network time	Mean client time	System throughput
Direct Binary	0.89	2.01	6.31	2.38
Direct XML	33.77	21.46	70.71	2.38
WS-Notification	156.10	92.29	55.58	1.23

■ **Table 4.** *Performance statistics using full record model.*

External Processing Experiments

The analysis formulas described in the performance model provide a means to incorporate EPEs into the overall calculation of performance boundaries and variability. The application of the formula also includes two standard EPEs that represent the publisher and client overhead specific to translating data to and from the appropriate eventing Web services format. Beyond these two persistent processing elements, any number of extra behaviors can be applied. The formula represents these as a simple additive increase in performance overhead and variability; however, more complicated interactions can be represented by extending the formula or by consolidating several EPEs into one composite factor that can be added to the others.

The tests discussed in this section performed two separate EPEs that emulate message authentication and metadata inclusion as discussed in the application overview section. In each case, an extra header element was inserted into the SOAP message (or raw stream in the direct case) that provides the relevant context for the given process. In the authentication case, the server would mask a string representing a name and password and then perform a set of modular operations that emulate encryption. The resulting string would be placed in the authentication header. The client would detect this header, reverse the operation, and verify the string. This EPE clearly has both a server and client component, and both were combined into a single performance factor. The reference data EPE was constructed by requiring the server to add a SOAP or raw header containing a URL, and then, upon detection of the header, the client would retrieve the data associated with the URL via a simple HTTP GET call. The data returned from the URL was a random length string between four and 4096 bytes in length. The client would verify that four or more bytes were returned from the call and then continue processing.

The two actions tested required heavy system resources in the form of CPU and memory for the authentication case and network communication for the metadata case. Furthermore, the process of capturing these shared physical resources generated a certain amount of uncertainty as did the context switching that occurred as a result. The associated uncertainty in these cases is clearly demonstrated by examining the standard deviation around the mean for these processes as illustrated in Fig. 2.

In all cases, the measured tolerances indicate a dramatic increase in variability that can be problematic for a real-time system. A system required to include such capabilities would require a broad service level agreement relative to the raw performance capabilities of the basic message delivery system. Not every extra service creates such overhead and instability; however, the common practice of including these elements in non-real-time Web services contexts indicates the need to understand these factors and to consider them before casually including such features.

Data Model Modification

The final set of experiments considers the impact of data model selection on eventing Web service performance. The organization and representation of data is important in a distributed computing system, but in the case of eventing Web services, the incremental increase in XML-related performance cost is potentially much higher than a comparable change in a simple binary system, and therefore, the selection of data model is a more serious exercise when considering the use of Web services. These experiments manipulated the data model by changing the update pattern from only the changed values to all fields in the record. Such a representation is common in document-style Web services in which communication of data is performed using a static set of fields passed from one service to the next [12].

Table 4 illustrates the basic performance differences in each distribution system using the full record data model. Each component in the distribution requires extra time to process the extra data and in the XML-based systems, has the added burden of encoding and decoding the data between native representation and the inefficient text-based representations. The direct XML and true Web services system also are constrained by the fact that the message text must be fully parsed and interpreted using string compare and copy operations. This performance data can be compared with that illustrated in Fig. 2, which represents the standard data model (transferring only the changed fields).

Conclusions and Further Research

Conclusions

Web services are one of the latest evolutions of abstract middleware systems and have been established using the commonplace SOAP and HTTP protocols with data encoding utilizing the XML syntax and structure. This set of ubiquitous and open technologies give Web services the capability of functioning over heterogeneous network, hardware, and software topologies. However, this adaptability comes at a significant performance cost as the data and communication protocols must be translated from compact proprietary formats to the often inefficient text-based standards used by Web services. The analysis and experimental results presented throughout this research are intended to provide a functional guide for practitioners who may wish to utilize real-time Web services in addition to promoting further research as the underlying technology evolves. The experiments conducted in this research highlight some key findings with regard to the use of a Web service platform in real-time systems:

- The performance of Web services-based systems is chiefly constrained by the processing overhead of the XML-based message and data encoding scheme and the inefficiency of HTTP as a real-time transport.
- Web services systems are best used at the edge of a network and not as a part of the core distribution.
- Usage of eventing Web services standards in real-time systems is viable, as long as the extended behaviors and data model remain reasonably constrained.
- The tolerance level for a Web services-based system grows rapidly compared to proprietary alternatives as a result of extra system load or other processing requirements.

The purpose in considering Web services as application middleware in a given real-time system is to extend the lifespan or usefulness of that system by basing its interactions on standardized interfaces and data definitions. This goal can be achieved in varying degrees by implementing all or part of the standards embodied by the Web services specifications. The system tolerance guarantees can be reduced to a more acceptable level by replacing key elements of the architecture with proprietary alternatives while retaining other standardized components. For instance, the inefficient HTTP protocol could be replaced by a direct TCP/IP variant, while the message and data encoding is still based on SOAP and XML. This would reduce the overall value of the system in terms of adaptability, but can be a viable consideration if the performance of HTTP is simply too poor to consider in the context of a given service level agreement (SLA). An SLA typically defines not just performance expectations, but penalties to the provider if those expectations are not met. This consideration may require compromises in the adoption of standardized protocols. In any event, the move towards ubiquitous data and interface standards for real-time systems undoubtedly will serve to extend their reach in today's heterogeneous distributed applications.

Future Work

The specifications that allow for the use of Web services technology in real-time systems are relatively new and therefore, so is the study of their performance capabilities. This research presents a framework for reasoning about the capacity of such systems in various application contexts, but it is kept open-ended so that more refined models can be plugged into the analysis. The existing research into low-level aspects of distributed system performance, as well as analysis of higher level constructs could be used to refine the model and provide specific insights into the performance constraints of such systems. The protocols on which Web services are based are themselves subject to change and evolution as application contexts are better understood. In addition to the underlying transport protocols such as HTTP, the high-level messaging protocols such as SOAP and WS-Notification would benefit from extension to directly support real-time systems. Indeed, if eventing Web services are to become the preferred future platform for all distributed applications, issues such as quality of service negotiation and scheduling priorities must be addressed.

If eventing Web services are to become the preferred future platform for all distributed applications, issues such as quality of service negotiation and scheduling priorities must be addressed.

References

[1] M. Moore and A. Pruitt, *Principles of Real-Time Software Engineering*, Wall & Emerson, 1998.

[2] G.C. Buttazzo, *Soft Real-Time Systems: Predictability vs. Efficiency*, Springer, 2005.

[3] K. Balasubramanian *et al.*, "A Platform-Independent Component Modeling Language for Distributed Real-Time and Embedded Systems," *Proc. 11th IEEE Real-Time and Embedded Technology and Apps. Symp.*, San Francisco, CA, 2005.

[4] A. Kanevsky, A. Skjellum, and J.Watts, "Standardization of a Communication Middleware for High-Performance Real-Time Systems," *Proc. Wksp. Middleware for Distrib. Real-Time Sys. and Svcs*,. San Francisco, CA, 1997, pp. 206–13.

[5] A. Arsanjani *et al.*, "Web Services: Promises and Compromises," *ACM Queue*, vol. 1, no. 1, 2003, pp. 48–58.

[6] S. Graham and B. Murray, Web Services Base Notification 1.2; http://docs.oasisopen.org/wsn/2004/06/wsnWSBaseNotification-1.2-draft-03.pdf, June 2004

[7] D. Box *et al.*, WS-Eventing; http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf, Aug. 2004

[8] D. Schmidt and C. O'Ryan, "Patterns and Performance of Distributed Real-Time and Embedded Publisher/Subscriber Architectures," *J. Sys. and Software*, vol. 66, no. 3, 2003, pp. 213–23.

[9] S. Vinoski, "More Web Services Notifications," *Internet Computing*, vol. 8, no. 3, 2004, pp. 90–93.

[10] S. Vaughan-Nichols, "XML Raises Concerns as It Gains Prominence," *IEEE Comp.*, vol. 36, no. 5, 2003, pp. 14–16.

[11] R. C. H. Cheng and W. Holland, "Calculation of Confidence Intervals for Simulation Output," *ACM Trans. Modeling and Comp. Simulation*, vol. 14, no. 4, 2004, pp. 344–62.

[12] E. Newcomer, *Understanding Web Services*, Addison-Wesley, 2002.

Biographies

Shawn Mulkey received his M.S. in computer science from the University of Kansas in 2006. He is currently director of information technology at myFreightWorld.com, a freight brokerage tools and service company, and the co-owner of Melete Web Solutions, an online development company. He has previously worked as a software engineer and architect in the financial services and telecommunications industries.

Hossein Saiedian [SM] (saiedian@ku.edu) received his Ph.D. from Kansas State University in 1989. He is currently a professor of software engineering in the Department of Electrical Engineering and Computer Science at the University of Kansas (KU), and a member of the KU Information and Telecommunication Technology Center (ITTC). His primary area of research is software engineering. He has over 100 publications on a variety of topics in software engineering and computer science. His research in the past has been supported by the NSF, as well as regional foundations.