

Issues in the Design of Flexible and Dynamic Service-Oriented Systems

Gerald C. Gannod*[†] and Janet E. Burge
Dept. of Computer Science and Systems Analysis
Miami University
Oxford, OH 45056
{gannodg,burgeje}@muohio.edu

Susan D. Urban
Dept. of Computer Science & Engineering
Arizona State University - Tempe
Box 878809, Tempe, AZ 85287-8809
Susan.Urban@asu.edu

Abstract

Due to the use of the concepts embedded in Service-Oriented Architecture (SOA), software design, now more than ever, involves the use of incomplete information. Applications that utilize Web Services are also highly impacted by the problem of deployment and subsequent undeployment of services. Specifically, there is a level of uncertainty caused by the potential for services to become unavailable (either temporarily or permanently). In a scenario where an application must switch from one service to another due to the undeployment problem, the client application may require that new or different handlers be used to cope with the properties of the alternative service. In this current development climate, the design issues become clear: there is a need to reason about how a design is impacted by discovered services, design analysis must consider the transaction and event properties of discovered services, and design of systems must incorporate fault tolerance and high-integrity issues to cope with the dynamic landscape caused by the uncertainty associated with using services.

1 Introduction

Web Services have ushered in a new era of software design that focuses on implicit and explicit collaboration between organizations. The level of adoption of Web Services in the industry is immense, especially within Fortune 500 businesses. Due to the use of the concepts embedded in Service-Oriented Architecture (SOA), software design, now more than ever, involves the use of incomplete information. While Web Service implementations that use the Web Service Description Language (WSDL) and SOAP (formerly Simple Object Access Protocol) standards provide information about syntactic structure (and thus an ability to automatically generate glue code), these descriptions lack se-

mantics that allow a client application to reason about the overall impact of using such services. The situation becomes worse with Representational State Transfer (REST)-based services [7] where client interfacing to service APIs must be manually created. So, while the so-called “traditional” Web service focuses primarily on the Remote Procedure Call (RPC) client-server interaction style, REST utilizes other transaction primitives. In addition to the above issues, applications that utilize Web Services are also highly impacted by the problem of deployment and subsequent undeployment of services. Specifically, there is a level of uncertainty caused by the potential for services to become unavailable (either temporarily or permanently). In a scenario where an application must switch from one service to another due to the undeployment problem, the client application may require that new or different handlers be used to cope with the properties of the alternative service. In this current development climate, the design issues become clear: there is a need to reason about how a design is impacted by discovered services, design analysis must consider the transaction and event properties of discovered services, and design of systems must incorporate fault tolerance and high-integrity issues to cope with the dynamic landscape caused by the uncertainty associated with using services.

In this paper, we identify a number of design issues that are related to developing service-oriented systems. Specifically, we raise issues that impact design of flexible and dynamic systems, especially in the context of systems that must consider transaction and event properties of discovered services. This paper is organized as follows. Section 2 described background and related work. Section 3 identifies issues related to the design of flexible and dynamic service-oriented systems. Finally, Section 4 draws conclusions and identifies future investigations.

2 Background and Related Work

This section describes background and related work in the areas of service-oriented architecture, event driven-

*This author supported by National Science Foundation CAREER grant No. CCR-0133956.

[†]Contact Author.

workflow, distributed transactions, and rationale.

2.1 Service-Oriented Architecture

Recent work has integrated the use of ontologies with Web Services to create the notion of Semantic Web Services [17]. Semantic Web Services are capable of communicating information about the meaning and purpose of a service, including semantic descriptions of the input to a service, the output from a service, as well as any constraints, pre-conditions, and post-conditions associated with use of a service. The addition of these semantic descriptions for web services lays the foundation for automated reasoning about the design of a distributed application, providing more advanced approaches for locating services and for understanding the manner in which they can participate in an orchestrated business process. In this context, *design* of SOA systems involves the description of atomic and composite processes (services) and the orchestration of those processes to form a distributed application. Most research has primarily focused on enhancing Web Services with ontological descriptions and has just scratched the surface on reasoning about semantic Web services beyond examining behavior and conditions [11]. Furthermore, advancements in reasoning about semantic descriptions in the design process may reveal the need for a more extensive meta model for semantic Web services that addresses the myriad of quality attributes associated with web services [19].

2.2 Event-Driven Workflow and Service Composition

A research area related to the process of designing for dynamics is the area of event-driven workflow and service composition. Much of this work originated with research on active database technology [24] and the use of Event-Condition-Action (ECA) rules. More recent work has investigated semantic, rule-based, and event-driven service-oriented architectures to provide seamless interoperable integration, dynamic composition, and semantic support among enterprise and business partners [14, 20]. The use of events and rules enhance semantic descriptions with behavioral knowledge, constraints, and policies to produce automatic and dynamic service composition and flow, such as the work with the Integration Rules project [22]. Moreover, current web service composition languages, such as BPML, WSBPEL, and OWL-S, provide solutions only at the descriptive level and not at the analysis and verification levels [20]. To achieve business process management in an SOA, management and consistency of the process [20] must be considered. The management of the process needs to be adaptable, with the ability to dynamically adjust either the process or the services which are functions of the process. Events and rules can also be used to check consistency of a process, either in a static or dynamic mode, or by analyzing event log files to compare actual behavior to the designed

process.

2.3 Distributed Transactions and Failure Recovery

The traditional notion of transactions with atomicity, consistency, isolation, and durability (i.e., ACID) properties is too restrictive for the types of complex transactional activities that occur in distributed applications, primarily because locking resources during the entire execution period is not applicable for Long Running Transactions (LRTs). The Web Services platform offers a distributed environment where autonomous applications can collaborate using a standard interface based on Internet technology. WS-Coordination [5], WS-Transaction [4], and Business Transaction Protocol (BTP) are specifications that use atomic business transactions to achieve ACID transactions and long-running business processes for non-ACID transactions. Different techniques have also been proposed to implement ACID transactions on composite Web Services [12, 1]. In BPEL, scoping is used to specify compensation handlers and/or a set of fault handlers for various fault messages, but there is no distributed coordination regarding an agreed-upon outcome among multiple-participant services.

2.4 Rationale

The rationale for a system, whether software or hardware, is different from standard documentation. Rather than being a static description, the rationale describes the decisions that were required, the alternatives considered, and the reasons for and against these alternatives. Much of the work on rationale has been focused on design rationale, particularly in the fields of engineering design [15] and Human Computer Interaction (HCI) [18]. There has also been significant work done using rationale for software development [6].

The SEURAT (Software Engineering Using Rationale) system [3] uses an Argument Ontology to support reasoning over the rationale. This ontology contains a hierarchy of reusable reasons for selecting between alternatives in software development. The initial ontology was built based on the “ilities” [8] and filled in with information from other quality and NFR taxonomies [13]. SEURAT captures traceability from NFRs to the software itself by associating the NFRs with alternatives that support or deny them.

3 Design Issues in SOA

To date, many of the advances in Web services have been driven by standards bodies. Specifically, SOAP, WSDL, and the plethora of WS-* [2] standards have been developed by consortiums that have been interested in facilitating inter-organizational collaboration of processes. In amongst the standards efforts have been the definitions of “standards” for semantic web services [17]. As these many differing

standards arise (both for traditional notions of web services and semantic web services), the problem of designing distributed applications has become increasingly challenging due to the need for these applications to address application dynamics, flexibility, fault tolerance, and transactional semantics.

The frameworks and technologies for Semantic Service Oriented Architectures (e.g., S²OA) systems include languages and tools for specifying ontologies, services within those ontological contexts, and behavior of services [17]. The efforts in the definition of different web service standards has been varied (e.g., the so-called WS-* standards) including WS-Policy, WS-Security, WS-Transaction, etc. In order to take advantage of all or even some of these capabilities, it is necessary to understand and cope with the challenges associated with their use including:

1. Identification of design processes, design (architectural) styles, and design patterns for S²OA systems. Many different approaches for component-based software development have been suggested in the literature [10]. SOA has ushered in a new era of development that includes programming-in-the-large. The process for this form of development is highly iterative and requires reasoning about uncertainty and dynamic availability of components. WSDL and SOAP-based Web services utilize a remote procedure call (RPC) style of interaction while emerging Representational State Transfer (REST) based services [7] utilize an interaction style using four basic primitives (e.g., GET, POST, PUT, and DELETE). As the use of techniques for creating flexible S²OA systems evolves, they must cope with both RPC-based and REST-based Web services. In addition, new styles of interaction at the architectural level may also emerge.
2. Design for flexibility and dynamics. While semantic web services are meant to facilitate specification, discovery, and execution of services, research in the area has focused more on the development of that technology than on the emerging issues of dynamics. With a stated intention of such semantic web service technologies to support late and dynamic binding of services, it is natural to investigate how the mechanisms of the semantic web can facilitate 1) event and rule driven service composition and exception handling, 2) ontology-based descriptions of web services and their relevant constraints, pre-conditions, and post-conditions, 3) declarative transactional semantics that support a mix of traditional ACID transactions with non-ACID transactions and user-defined correctness and recovery procedures, and 4) dynamic service location and substitution for dealing with variability and the unknown. Moreover, as the specification, discovery, and execution technologies become more mature, de-

velopment of techniques that address the problem of how to effectively design systems to take advantage of these technologies becomes paramount.

3. Identification of methodologies that utilize rationale as a major driver for making and reasoning about design decisions at a number of levels of abstraction ranging from architecture formation to selection of compensation and contingency operations at run-time.

3.1 Design Processes for S²OA.

Our philosophy for designing SOA systems is grounded in design of component-based systems and product-line architectures. The process for designing such systems relies on the use of a baseline or product-line architecture that defines the structure and semantics of a family of software systems. In the definition of such an architecture, domain analysis drives the definition of commonalities and variabilities for systems in the product family. Often, components are used to implement variabilities at interface boundaries. We propose to use a similar approach for the development of dynamic S²OA systems, where discovery of services acts as the primary driver for the identification of new product instances.

Figure 1(a) shows a diagram depicting the development process which is based on an incremental product line development process [21] where the architecture is analyzed to determine impact of new product requests. The entry point into the process is the identification of new product requests (or requirements). The decision point of whether to increase core assets (e.g., increase the number of implemented commonalities) is followed by the process of updating existing products and creating a new product based on the desired requirements. With respect to updating existing products, the process dictates that if new commonalities are deemed necessary, existing products should be modified to reflect the new assets. In the service discovery perspective (as shown in Figure 1(b)), discovery of new services (rather than new product requests) drives the identification of “new” product instances. As such, construction of new (or modification of existing) commonalities is a result of service discovery.

In a context that includes discovered services, software architectures must be analyzed to determine the impact of using those services. Specifically, a new process model (perhaps based on Figure 1(b)), is necessary that also identifies the potential impact of a new service upon potential instantiations of the architecture. Several challenges arise including how to specify requirements (including relevant quality attributes) in order to facilitate reasoning about the reactive and dynamic behaviors of the relevant systems.

Software architectures are hierarchical constructs involving many different levels of abstraction. While a software architecture may describe the high-level interactions (in-

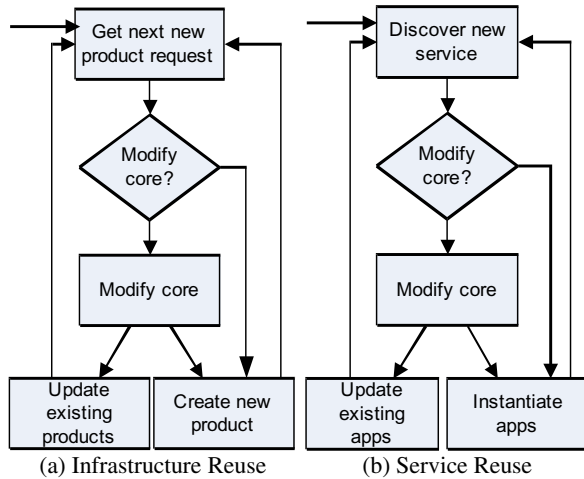


Figure 1. Reuse Process Flowcharts

cluding their composition into some workflow) between components using many of the well-known architectural styles in the literature, a service oriented architecture must cope with a level of uncertainty caused by the lack of knowledge about yet to be discovered services. As different quality attributes, such as flexibility, become of higher concern, new patterns of interaction must be developed that describe how different transactional and event models affect service compositions.

The basic SOA architecture involves a *service provider*, *service client*, and *service registry*. In this architectural style, providers publish descriptions in registries that are subsequently discovered by service clients. Service clients then utilize the description to invoke the service hosted by the provider. In the current services environment, the interaction between the client and provider generally follow either an RPC-based style or a REST-based style. While languages have been developed to model web service flow [16], explicit help on design issues in SOA is still a work in progress. In our research, one of the objectives is to identify, characterize, and utilize new styles of interaction that account for different forms of events, the use of contingent and compensating operations, and potentially, service replacement as a means for providing flexibility in a S²OA environment. Specifically, given the dynamic nature of service-oriented software systems, there is a large amount of uncertainty regarding the design and implementation of systems. This uncertainty results in a need to develop design techniques that utilize different modes of interaction between components. Furthermore, it is necessary to understand the nature of transactions, especially since S²OA systems require a high degree of integrity while being flexible enough to cope with changes in the environment. In our previous investigations, we developed a *Model-Driven Architecture (MDA)*-based approach for specifying semantic Web services [9]. While the approach facilitates the act

of creating a specification, we did not provide explicit assistance for identifying and analyzing different models of interaction or their impact on the resulting service composition.

3.2 Design for Dynamics, Flexibility, Fault Tolerance, and Transactional Semantics

Events can be broadly categorized as application-oriented events, application exceptions, and system faults associated with service execution. Since our goal is to design with dynamics and flexibility in mind, the ability to specify and use events in a meaningful way is critical. To fully support flexibility, however, events must be coupled with the use of rules to reason about execution alternatives, especially in the context of a volatile, distributed service composition environment in which services can change, fail, become temporarily unavailable, or disappear. Our own past work has addressed the use of event-condition-action (ECA) rules for the integration of EJB components [23]. The use of events and rules must also be integrated with checkpointing techniques so that the reasoning process can consider the state of the execution when determining how to respond to application events, exceptions, and faults.

In addition to providing an event-driven architecture, applications that are composed of services must also address the semantic correctness of concurrently executing Web transactions and the unpredictable nature of a service-oriented execution environment. Traditional transactions use serializability to define the correctness of concurrent executions that access the same data, where the concurrent execution of a set of transactions must produce a result that is equivalent to some serial execution of each transaction. Furthermore, transactions usually conform to ACID properties (i.e., atomicity, consistency, isolation, and durability). Supporting serializability and the ACID properties, however, is no longer valid for Web transactions. As shown in Figure 2, Process 1 and Process 2 are executing concurrently and accessing the same services, that possibly access the same data. In a traditional transaction, all of the data accessed by Process 1, for example, would be locked until it is known that Process 1 will commit, with each subcomponent coordinating the commit process using the two-phase commit protocol. As a Web transaction, two-phase commit is not a realistic option due to the autonomy of each service. Furthermore, there is no global log file to support rollback and recovery or to support data dependency analysis. If Process 1 fails in the execution of service operation 5, services that have committed prior to the execution of operation 5 may have already released modified data to Process 2.

Typical solutions from past work with advanced transaction models and transactional workflows use compensating procedures to logically undo (or partially undo) the affects

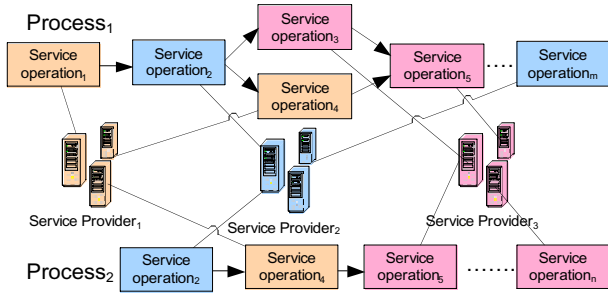


Figure 2. Transaction Example

of the failed process (Process 1), with contingent procedures used as a forward recovery mechanism to continue the execution. In either case, Process 2 may now be executing with incorrect data due to the lack of support for isolation. As a result, user-defined correctness, in the form of constraints, pre-conditions, and post-conditions, is critical to the specification of a web service composition. Although existing standards and languages for service composition provide features for specifying and executing compensating actions, these frameworks are still too rigid and must ultimately be integrated into a more extensive design process that supports an array of transactional features, with the appropriate rationale that will support the ability to respond in an acceptable manner to failure in the environment. Examples of some of the dynamic capabilities that must be considered are:

1. If a service is unavailable, service retry may be an option. The design should capture the execution alternatives that are available when service re-try reaches its limits. It is also important to understand how the local database recovery facilities of a service interact with the global process execution and recovery facilities.
2. Based on the application requirements, there may be subcomponents of a design where a two-phase commit process is needed, thus requiring the use of services that support this capability. This rationale should be captured in the design. Furthermore, if a network failure prevents access to the needed services and similar services are available that do not support two-phase commit, the design should specify how to dynamically address correctness issues for this alternate path of execution.
3. The nature of autonomous services may require relaxed support for isolation, with different transaction models, such as sagas or open nested transactions that support compensation. For locating appropriate services, it is important to capture the conditions for semantic correctness of the logical undo process of the compensating procedure. It may also be possible (or required) to provide multiple options for compensating or contingent procedures. If so, the rationale for choosing one procedure over the other is relevant for

providing more dynamic ways of responding to failure.

Capturing the rationale for the use of different web transaction semantics in the composition process is critical to planning design alternatives that support fault tolerance. The rationale for specific transaction capabilities is also needed to provide intelligent support for the dynamic location and substitution of services at execution time. These issues have an impact of the type of information that must be provided in the semantic description of services.

3.3 Using Rationale to Design SOA Systems

There is a need to reason about how discovered services impact system design. This requires knowing not only the purpose of the service, which indicates which requirements the service may address, but also its impact on system quality attributes. It is also important to know any dependencies or conflicts the service may have with other services that may be used by the system.

One way to capture the impact of quality attributes and dependencies on the design are through its rationale. The rationale for a software system describes the decisions that were made, the alternatives considered, and the reasons, or arguments, for and against each of the alternatives. There are many different types of arguments that should be considered: a) *Does the alternative address or violate any of the functional requirements?*, b) *Does the alternative support or deny quality attributes such as flexibility or fault-tolerance?*, c) *Does the alternative affect any assumptions made about the system?*, and d) *Does the alternative have any dependencies on or conflicts with other alternatives?*

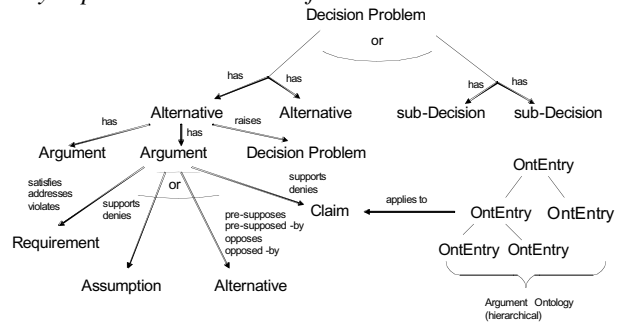


Figure 3. Argument Ontology

Figure 3 shows the structure of the rationale as defined for the Software Engineering Using RAtionale (SEURAT) system [3]. Arguments referring to non-functional requirements can be expressed in the form of “Claims” that describe the relationship between the alternative and a quality attribute. The quality attributes themselves can be obtained from an Argument Ontology that gives a hierarchy of NFRs at different levels of abstraction. The different levels allow for more detailed reasons to be expressed at different levels of design while the ontology itself provides a common

vocabulary that can be used to support inference over the rationale. The arguments for and against each alternative can be used to calculate its level of support relative to other alternatives for that decision.

Composing a system from a collection of Web Services involves making a series of decisions where alternative service implementations are considered based on the functionality they provide and how well they support the non-functional requirements of the system being created. The advantages and disadvantages of the alternative services comprise its rationale.

4 Conclusions and Future Investigations

The *habitat* of applications in a SOA context is an environment that is highly dynamic. Applications must be flexible to be able to cope with the uncertainty inherent in this habitat. Applications must also be able to cope with and recover from situations that would otherwise cause inconsistency in application behavior and data. In this paper, we outlined several challenges that arise as a result of the dynamic nature of SOA. We are in the process of developing a three-pronged approach that addresses the design of flexible and dynamic systems that utilizes design rationale as a driver for making decisions about both design-time and run-time alternatives. The approach is based on the use of semantic Web services, and when complete, will facilitate multiple levels of analysis.

References

- [1] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In *Proceedings of the 14th int. conference on World Wide Web*, 2005.
- [2] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture (w3c working group note). Online [available <http://www.w3.org/TR/ws-arch/>], February 2004. Last accessed Feb 3 2007.
- [3] J. Burge and D. C. Brown. Rationale-based support for software maintenance. *Rationale Management in Software Engineering*, 2006.
- [4] L. F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. *Web Services Transaction (WS-Transaction)*. <http://www.ibm.com/developerworks/library/ws-transpec/>, 2002.
- [5] L. F. Cabrera, G. Copeland, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, and T. Storey. *Web Services Coordination (WS-Coordination)*. <http://www-106.ibm.com/developerworks/library/ws-coor/>, 2003.
- [6] A. Dutoit, R. McCall, I. Mistrik, and B. Paech. *Rationale Management in Software Engineering*. Springer-Verlang, 2006.
- [7] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UC Irvine, 2000.
- [8] R. E. Filman. Achievingilities. In *Proceedings of the Workshop on Compositional Software Architectures*, Monterey CA, 1998.
- [9] G. C. Gannod, J. T. E. Timm, and R. J. Brodie. Facilitating the specification of semantic web services using model-driven development. *International Journal of Web Services Research*, 3(3):61–81, 2006.
- [10] G. T. Heineman and W. T. Councill, editors. *Component Based Software Engineering: Putting the Pieces Together*. Addison Wesley, 2001.
- [11] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. <http://www.w3.org/Submission/SWRL>, 2004.
- [12] T. Jin and S. Goschnick. Utilizing web services in an agent based transaction model (abt). In *Proceedings of the 1st Int. Workshop on Web Services and Agent-based Engineering (WSABE' 2003) held in conjunction with the 2nd Int. Joint Conference on Autonomous Agents and Multi-Agent Systems, at Melbourne, Australia*, 2003.
- [13] H. Jung, S. Kim, and C. Chung. Measuring software product quality: A survey of iso/iec 9126. *IEEE Software*, 21(5):88–92, 2004.
- [14] Z. Laliwalla. Event-driven dynamic web services composition and automation of business processes. In *International Conference on Services Computing*, 2006.
- [15] J. Lee. Design rationale systems: Understanding the issues. *IEEE Expert*, Vol. 12, No. 3:78–85, 1997.
- [16] F. Leymann. Web services flow language. Technical report, IBM Software Group, 2001.
- [17] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of SWSWPC 2004*, volume LNCS 3387, pages 26–42. Springer-Verlag, 2005.
- [18] T. Moran and J. Carroll, editors. *Design Rationale Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, 1995.
- [19] L. O'Brien, L. Bass, and P. Merson. Quality attributes and service-oriented architectures. Technical report, Carnegie Mellon University, 2005.
- [20] M. Rouached, O. Perrin, and C. Godart. Towards formal verification of web service composition. In *Proceedings of the International Conference on Business Process Management*, 2006.
- [21] K. Schmid and M. Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19(4):50–57, 2002.
- [22] S. D. Urban, S. W. Dietrich, Y. Na, Y. Jin, A. Sundermier, and A. Saxena. The irules project: using active rules for the integration of distributed software components. In *Proceedings of the 9th IFIP 2.6 Working Conference on Database Semantics: Semantic Issues in E-Commerce Systems*, 2001.
- [23] S. D. Urban, S. Kamphampati, S. W. Dietrich, Y. Jin, and A. Sundermier. An event processing system for rule-based component integration. In *Proceedings of the International Conference on Enterprise Information Systems*, 2004.
- [24] J. Widom and S. Ceri. *Active database systems: triggers and rules for advanced database processing*. Morgan Kaufmann, 1996.