Some Business Concerns around Service-Oriented Architectures

Eleni Stroulia

Department of Computing Science
University of Alberta
stroulia@cs.ualberta.ca

Abstract. The objective of this paper is to explore the relationship between the engineering of Service-Oriented Applications and some strategic and economic concerns of the organizations that (consider to) adopt this architecture style for the development of their software systems. To that end, (a) we discuss some pragmatic observations regarding the potential role of SOAs in the current economy, (b) we identify some distinct types of applications envisioned to be developed in the SOA style, (c) we correlate some strategic business decisions with different types of SOA evolution scenarios, and (d) we outline a novel model for estimating the ROI of such evolution scenarios. This work rests squarely within the newly articulated area of "Service Science, Management, and Engineering (SSME)" [15,19] as an "interdisciplinary approach to the study, design, and implementation of service systems, i.e., complex systems in which specific arrangements of people and technologies take actions that provide value for others".

1 The role of SOAs in a Services Economy

It is becoming increasingly apparent that the services sector accounts for a substantial percentage of the worldwide economic activity and that this percentage has been steadily increasing. According to IBM [24] "*The world is becoming one global service system. Five of the top ten nations ranked by labor force size have more than half of their labor working in services (versus agriculture or goods production). China and India are not yet in the top five – but with roughly 40% of the combined worldwide labor force, as these "sleeping giants" migrate towards services the implications are profound*." Even more importantly, the percentage of workers engaged by the services sector increases at the expense of the corresponding percentages of workers in agriculture and goods production. These statistics underline the increasing importance of the services sector in the world economy and give rise to the need for clarifying which exactly activities are included in this sector.

According to Zysman [25], there are four different types of economic activities that are accounted under the general label of "services".

1. The first is an artifact of financial engineering: activities outsourced from manufacturing are relabeled as services because they are conducted by different organizations, even when they remain essentially the same.
2. The second category involves an increasing number of the traditional activities

involved in business-consumer interactions and between-business interactions.
3. The third service category accounts for the conversion of unpaid domestic work (traditionally the responsibility of women) into commercial services, as women increasingly work outside the home.
4. The fourth category represents a fundamentally new and innovative class of service activities, facilitated by software systems, which are increasingly used to formalize, codify and drive the execution of business processes.

This last category, characterized as the "fourth service transformation" by Zysman, is the focus of this paper. Zysman points out that "*tools and technologies based on algorithmic decomposition of service processes may have the power to revolutionize business models, by displacing and by complementing the people (involved in these processes)*" thus implying a tight inter-dependence between the evolution of today's software systems and the business processes they support.

Service-oriented architecture is a new software-engineering paradigm, which is being increasingly adopted [38,39,40], and which advocates the design and development of complex software systems through the composition of (independently designed and developed) "services". The term "services" in this context refers to software components, which, although implemented in a variety of platforms and programming languages, are interoperable through open XML-based specifications of their interfaces.

A quite sophisticated stack of open standards (based on XML syntax) is currently under development to support the specification of various aspects of service-oriented applications. For example, in addition to the services' interfaces represented in WSDL [32], the run-time information exchange among services is governed by the SOAP protocol [30], and the composition of services (including data and control flow among them) is represented in BPEL [26]. These three specifications are quite mature, as evidenced by the variety of existing implementations and tool support. A set of less mature specifications are also under development to elaborate the nature of transactional support required for these complex systems with long-running workflows [37], the agreements between the collaborating services [35], and the relevant security requirements [36].

The use of the term "service" in these two contexts, namely as (a) a value-producing process between an organization and its customers (business context), and (b) a reusable software component delivering a coherent set of functionalities (software-engineering context), is an indication of the importance of the abstraction it captures: "software services" can be recursively composed to develop complex processes, whose execution, in turn, drives "business services". The declarative composition specification (in terms of WSDL and BPEL) constitutes an effective formalization and codification (as per Zysman's fourth definition) of the business processes.

The specification process changes, and in some cases it even revolutionizes, the current business processes since it explicitly considers which activities can and should be automated, which should be mediated by organization employees (acting in specific roles), which might be driven directly by the interaction between the customer and the SOA application, and which should be outsourced to SOAs of the organization's business partners (B2B interactions).

This phenomenon signifies a "phase transition" in the traditional interface between

business-process modeling and engineering and software engineering, the most significant aspects of which are as follows.

- Software-architecture descriptions are now visible to CEOs, who can consider specific investments in IT adoption and evolution in the context of their strategic business planning. On the opposite side, the economic and strategic business concerns can now more explicitly influence software-architecture decisions.
- We are increasingly witnessing the development of domain-specific repositories of software services, based on which complex compositions can be developed across individual organizations. Although the global UDDI [31] repositories envisioned by OASIS do not seem to gain sufficient momentum, business consortia develop their own standardized vocabularies [2,10] to describe their products, information and activities, thus paving the way to the development of shareable services automating these activities and the information exchange among them.
- The SOA stack of standards supports a rich set of metadata, about sector-specific vocabularies and software, which can be potentially considered in the context of business decisions.
  - Service-Language Agreements can be used to formalize different service variants, corresponding to different classes of workflows subject to different regulatory and performance constraints.
  - As application sectors define standard vocabularies and services, test-beds can be specified to define acceptable and desired performance targets, against which new service implementations can be measured.
  - Finally, as multiple services offering similar functionalities become available, their user programmability, as reported in developers' forums, becomes yet another criterion for their adoption.

2 Innovations in the Service-Oriented Computing Paradigm

Having reviewed the general economic environment to which SOAs are envisioned to contribute, let us now discuss the essential properties of this software design and development paradigm.

The SOA paradigm is not fundamentally new; in fact, the field has already made several attempts to develop standards for the composition of complex applications based on independently developed components available in shareable repositories. However, there is a set of important innovations that this new paradigm encapsulates, which makes its widespread adoption more likely.

Services are network-accessible components, which may have been developed in any of a broad range of programming languages. Their broad accessibility and utilization is enabled by their WSDL-specified interfaces and the fact that at run time they can be invoked through SOAP, an XML-based protocol. Never in the past have interface-description languages been agnostic of programming languages and platforms: CORBA has been partial to C++, COM/DCOM has depended on the Microsoft platform, and, more recently, we have seen a variety of Java-specific protocols (such as Jini for example) as part of "Java as middleware" trend [23]. The SOA standards, based on XML, are open in a way none of these precursors were and we are already seeing a variety of commercial and open-source IDEs supporting the

development of services in a variety of programming languages. Furthermore, SOAP, the run-time invocation is also XML based and can be implemented on HTTP (as well as on a variety of platform-specific protocols) which makes it less sensitive to firewalls and other organization boundaries. The choice of XML as the syntax underlying the SOA stack of standards constitutes an innovation of great potential importance for their widespread adoption.

The second important innovation is the flexibility of the service-discovery scenarios supported by the discovery-related standards. Originally, the UDDI [31] registry structure and discovery protocol envisioned globally accessible repositories of general services, organized under multiple overlapping taxonomies. However, there were not enough rich taxonomies to precisely classify organizations and their services, and no clear business motivation was articulated for why businesses should advertise their software services in these registries, which were too public and did not offer much support for the developers (beyond browsing). Consortia-specific and private UDDI registries are seen some adoption and the Web-Services Inspection Language (WSIL) [33] offers an alternative, lightweight scheme for the advertisement and discovery of services, which should be increasingly usable as specific sectors and consortia develop their own taxonomies for specifying the information they exchange and their interactions [2,10].

Finally, the most interesting and powerful innovation of the new standards is the fact that they enable recursive composition of services in increasingly complex services specified in BPEL, whose interfaces are again specified in WSDL. Traditional distributed middleware has aimed at "transparency": through syntactic extensions to traditional programming-language constructs, the middleware designers have tried to hide from developers the complexities of accessing components outside their system boundaries. By making distributed components seem just like any other component – at the programming-language syntax level- the learning curve of the middleware frameworks is simplified, at the cost, however, of hiding the dependencies across domains of software ownership and control. This approach is eminently inapplicable to today's vision of enabling interoperation across organizations through the Internet. Different organizations want to integrate their software systems and processes in order to increase the effectiveness and efficiency of their partnerships. At the same time, they need to maintain control over how to manage their own software assets and how to grant access to their partners, subject to their own strategic goals and governance rules. To enable the flexible maintenance and possible evolution of processes involving multiple organizations, the rules of composition among components across organization boundaries have to be explicit. This is why BPEL is critically important for accomplishing the envisioned Internet-wide interoperability across programming-languages and organization boundaries.

3 Some Issues in the SOA Research Agenda

We believe that the software-engineering research agenda [11] for supporting the development and management of SOAs should cover the following four broad areas of research.

Defining a taxonomy of SOAs: The first concern is to identify the different types of SOAs, for which there is a pragmatic need. Not all new software systems should be developed in the SOA style and some existing systems should probably be migrated to an SOA style. The question then becomes to develop requirements-analysis methods that can recognize when an organization should migrate to an SOA [26] (whether through reengineering of existing systems or through development of new ones according to best practices associated with each distinct type).

Run-time SOA monitoring and adaptation: Once an SOA is developed and deployed, its run-time performance will have to be monitored to ensure that the orchestration instances are progressing as expected and the various SLAs applicable to the constituent services are respected. When orchestration instances fail or SLAs are (about to be) violated, autonomic, run-time reconfigurations might be possible to address the (potential) violations. Although there exist a variety of autonomic-adaptation algorithms for managing the performance of different types of computations (database query latency, network throughput, web-server request throughput) the task of end-to-end management, especially when the constituent components belong in different administrative domains, arises as a grand challenge in SOA research.

SOA evolution management: Inevitably, SOAs will have to evolve. Evolution is part of most software systems' lifecycles and it is bound to be even more important for SOAs, since evolvability is one of the main motivating factors for adopting the modular SOA style for an application. Given the modular nature of SOAs, the question then becomes to articulate (a) a taxonomy of "canonical" adaptations, (b) the opportunities that may motivate them in the operating environment of the organization that the SOA supports, and (c) the best practices for carrying out these adaptations. A critical element of such a "best practices" list is a method for understanding the economic trade-offs of SOA adaptations, especially when multiple alternative adaptations are possible, in terms of return-on-investment estimation.

Mixed service delivery with SOAs: Finally, we have to study the nature of interactions between SOA systems and users. This question will have to be informed by ergonomic and cognitive-science theories about what tasks people are good at (as opposed to tasks that people need help with) and by a study of the canonical roles that people (employees and customers) fulfill in service processes. The BPEL4People specification [29] is a first attempt at characterizing these roles, the nature of user-system interactions they imply and what the technical solutions for the necessary user interfaces might be. Clearly further study is required to refine our understanding of people-mediated and automated activities so that SOAs can be developed with an appropriate mix of the two types.

3.1 A taxonomy of SOAs

As an initial investigation of the question "what types of applications is an SOA good for" we have examined research publications and white papers to come up with a set of three distinct types of SOAs.

First, it is clear that SOAs, based on the web-services stack of standards, are perceived as the natural evolution of business frameworks that support complex business workflows. To date, we have seen organizations transition from legacy

systems, custom-built to the specifications of the organization, to (b) sector-specific frameworks – like SAP and PeopleSoft – customized to meet the differential requirements of each organization, to (c) J2EE (Java2 Enterprise Edition) frameworks that enable a greater degree of flexibility in system development, enabling the integration of COTS components with custom components, which sometimes may wrap legacy subsystems. This trend has been driven by the need to map the organization's processes onto its software systems (through integration of custom elements) while at the same time enabling (some) integration with partner systems (through reuse of general frameworks and integration of COTS). Both these requirements are extremely well met by the SOA standards, in fact better than all these previous middleware as application sectors standardize their information and interactions. Therefore, we expect to see a third wave of transitions to standard SOAs, especially from J2EE systems (since there is substantial tool support for this type of migration).

We believe however that there is an interesting conceptual difference between J2EE components and services, which should be accounted for in this transition. Although, services are frequently discussed as analogous to java interfaces and classes [7], they are meant to offer a coherent set of related functionalities, which together support an interesting complex capability and would be more appropriately conceived as use cases, for modeling and assembly purposes, which could be reverse engineered from the existing system, in cases of migration to SOA. Then, a WSDL specification can be developed to describe the external interface of the service and a BPEL specification can be developed to define its usage protocol. These two specifications can enable automatic means-ends compositions of services, possibly even at run time [5,6].

A different type of services is exemplified by mash-ups. According to Vint Cerf "mashups represent new ways of applying existing basic infrastructures, not originally intended by their designers" [1]. This ad-hoc and opportunistic development of services is becoming especially interesting when associated with an on-line social network since it supports "long- tail" business activities [3]. Essentially, social-networking sites enable the formation of small niche markets of people with common interests, thus defining target markets for niche products and services, which might not be profitable in a more traditional economies-of-scale model. According to an Amazon employee: "*We sold more books today that didn't sell at all yesterday than we sold today of all the books that did sell yesterday*." [3] SOA as an enabler of efficient, low-cost service composition is well poised to become the de-facto architectural style for developing such niche applications.

The third distinct type of SOAs are motivated by the need to exploit cyberinfrastructure installations. The term cyberinfrastructure denotes a collection of powerful computational clusters, large-capacity storage devices, high-speed wired networks, ad-hoc, mobile networks and sensor networks. Today these installations are being used for special, resource intensive applications developed by researchers' teams, mostly in areas such as E-Science. The grid middleware [34] builds on the web-services standards to provide additional support for this type of applications, including job scheduling, data distribution and load balancing, aiming at the virtualization of the underlying resources and the simplification of application development on them, so as to encourage a wider variety of applications by a larger

community. As the variety of grid applications and the types of underlying hardware resources increase, more middleware for enabling the efficient development of applications needs to be developed. As grid services are increasingly adopted, we expect to see extensions of this type of SOA middleware, designed to enable the utilization of other types of resources, such as sensor and RFID networks. This is a necessary pre-requisite for exploiting the promise of these inexpensive devices that are envisioned to cover our world with "smart dust" feeding us with continuous information about it.

3.2 Run-time SOA Monitoring and Adaptation

Autonomic computing has been, in recent years, a fertile area of research, focusing (a) on predicting trends in the performance of the subject system and (b) configuring the system's operating parameters so that it meets its performance requirements. To our knowledge, most autonomic-computing solutions focus on monitoring and managing homogeneous system resources, at a small temporal granularity.

Two fundamentally new autonomic-configuration problems arise in the context of complex SOA applications. First, as a natural extension to autonomic performance management, research will have to address end-to-end management of complex systems. In this more complex conceptualization of the problem, one has to consider that the subject system utilizes a variety of resources and that, for any given anticipated violation, multiple alternative reconfigurations might be relevant, each one addressing different plausible root causes, having different impact to the system configuration, and implying different costs and risks. The objective then will be to synthesize multiple sources of performance metrics, to infer alternative root causes that could "cover" the collected evidence, and to decide on an economic and effective reconfiguration.

The second autonomic-management task is to recognize orchestration failures due to the independent evolution of the constituent services. Even when the WSDL specifications of these services do not change, their usage protocols may change due to the particular policies and regulations to which they may be subject. For example, suppose that Organization A starts imposing a particular ordering to the invocations of two of its operations that used to be independent. BPEL orchestrations that used to work will now start to fail. If BPEL orchestration middleware becomes explicitly aware of the usage protocols of the constituent services, when they evolve it will be able to automatically reconfigure the failing orchestrations to accommodate the changes.

An initial approach to addressing this problem is exemplified in our WRABBIT project [5,6]. This research examines the avoidance of orchestration failures that result from out-of-sync service-usage models. WRABBIT views service composition and coordination as a conversation among intelligent agents, each of which is responsible for delivering the services of a participating organization. Each such service is characterized by its WSDL interface and the abstract BPEL specifications of its usage protocols. In this context, an agent is a layer wrapping each peer organization, and is able to communicate with the other agents responsible for partner services, recognize mismatches between its own conversation model and the models of other agents (as these are revealed by conversation failures), and adapt the models

as necessary to eliminate these errors. This approach applies to situations where the basic information and operations required for the successful completion of the agents' conversation is available to them and failures can be addressed by renegotiating the conventions of the conversation. Further research is necessary to address failures resulting from actual differences between the required and available ontologies and operations.

3.3 SOA Evolution Management

3.3.1 SOA Modification Operators and Scenarios

Software-architecture research has identified several qualities supported by modular design [19, 12]. Baldwin and Clark [4] examined specific software systems – like the IBM 360 – which owed their significant business success in their modularity. Based on their case-studies analysis, they formulated six modularity operators, in terms of which any change to a modular system can be described:

(a) Splitting - Modules can be separated into multiple independent modules.
(b) Substituting - Modules in a system can be substituted by other modules.
(c) Excluding - Existing Modules can be removed from a system to build a new solution.
(d) Augmenting - New Modules can be added to systems to create new solutions.
(e) Inverting - The hierarchical dependencies between modules can be rearranged.
(f) Porting - Modules can be applied to different contexts.

In this subsection, we discuss different variants of the above operators in the context of SOA evolution scenarios, we identify potential business-strategy contexts that may motivate these scenarios, and we review the tool support that they require.

Splitting: As a particular service becomes increasingly reused in the context of multiple SOAs, a variety of SLAs may be developed to further refine the constraints on its performance. For example, a given service may be invoked to provide just-in-time updates of a real-time variable, like stock prices, or a periodic report of the same variable at a lower cost to the subscriber.

Furthermore, different types of usage protocols may emerge as variants and/or refinements of its original usage protocol, based on the needs of its clients. For example, some clients may use only a subset of the possible combinations of the service operation parameters, i.e., for low-cost items in a product catalog, users never exercise the additional insurance-buying option. According to the "Interface Segregation principle" [16] clients should not depend on interfaces they do not use; therefore as particular variants of the original service become widely adopted, a new service should be developed to meet these specific needs.

To enable the recognition of service variants as candidates for becoming independent services, an automated monitoring capability could be exploited to recognize patterns in the SLA constraints and the usage record of a given service by its various clients. In addition, an automated workflow-refactoring process could potentially support the integration of the discovered variant services as potential alternatives for (some of) the roles that the original general service used to fulfill.

Substituting: Several different types of substitution scenarios exist. First, an organization may decide to provide an automated alternative for an employee-mediated task. For example, while previously a customer had to contact an employee to place an order, an on-line product-order form is now available. This type of substitution does not appear to require any special support, if there already exists a user interface for the employee receiving the order; this user interface can be migrated to a web-accessible platform and can still drive the original order service.

Alternatively, an organization may decide to select an alternative partner for an outsourced service, such as instead of using the FedEx "Saturday delivery" service, they start using a similar UPS service. This scenario requires support for an example-based service discovery API: developers can use the interface and usage protocol of the service they currently employ as an example for discovering similar services that can be used to fulfill the role of the retired service. Neither UDDI [31] nor WSIL [33] offer such an API but a substantial body of research exists in this area, including some of our own integrating lexical, syntactic and semantic information available in WSDL specification to assess similarity [21,18]. This work is not mature enough to support automated discovery, which will eventually rely on semantic-web technologies.

Excluding: At any point in time, the process-monitoring engine may recognize that execution paths, although possible in terms of the BPEL-orchestration specification, are not actually exercised. For example, a path of a "pick" structured activity is never exercised, since users never choose to fill optional questionnaires. Or, alternatively, some condition in a "case" structure activity is never met. In such cases, all redundant services may be excluded from the process, so as that the specification better reflects the actual practice. To support this type of modification, one envisions a BPEL-process transformation utility, which should be able to automatically transform the configuration of the affected BPEL structured activities, ensuring the consistency of the impacted data and control flow.

In cases where some parts of a service are used while others are not, when for example only a subset of the service API is used – when the monitoring engine for example observes that the high-cost stock-price service variant is not used - this modification may necessitate a "splitting" operator first, before excluding the not utilized parts of the service interface.

Augmenting: One can imagine a variety of augmentation scenarios, motivated by the organization's need to differentiate its offerings from those of its competitors and its own current offerings. The first is conceptually the inverse of the "splitting" scenario: through SLA negotiations, the organization may discover that an extended variant for an existing service is required. For example, in addition to the notification that an order is ready for pick-up, a feed for changes to the transit state of the order may be desired by some clients. Another augmentation scenario is conceptually the inverse of the "excluding" scenarios above: a new service may be added as one more option in a "pick" structured activity. For example, a new discounting mechanism is added to enable a new type of special pricing promotions. Both these types of scenarios require some support from the BPEL-process transformation utility.

The above augmentation scenarios have a rather local impact on the BPEL process specification. One can imagine a more challenging augmentation scenario. For example, the organization may decide to offer more features in its process, such as to enhance a product-catalog service with a recommender service. Or, motivated by a

strategic partnership, the organization may decide to integrate its processes with a partner's process. For example, in addition to selling the products in their own product catalog, a business may act as reseller to a partner product catalog, thus necessitating additional interactions with the partner accounting and warehousing services. These types of augmentation modifications will require a sophisticated BPEL-process planning utility.

Inverting: Inversion scenarios are essentially standardization scenarios. For example, in order to enable the introduction of multiple alternative services where only one was invoked, the essential, and common across all alternatives, interface needs to be decided. Consider, for example, the case where a business product catalog enabled books' ordering only and has evolved to also enable ordering CDs. At this point, if the more general "product" concept is specified, any spurious book-related dependencies between the ordering process and the original product-catalog service can be eliminated.

Porting: Two fundamentally different porting scenarios exist. The first involves the migration of an application from a traditional middleware platform to an SOA relying on the web-services tack of standards (e.g. from a J2EE framework to Apache Axis). This scenario can be supported by code-transformation tools. Although to our knowledge no such tools exist yet for this specific transformation, there is a substantial body of research in syntactic code transformations [10], which could provide a substantial basis for this task.

A different type of porting scenario may involve the migration of a service from one domain to another, such as for example, porting a book product-catalog service to the DVD domain. This type of modification would be motivated by the organization's need to expand its offerings and would require support with the alignment of the ontologies underlying the source and target service domains [26].

3.3.2 Return-on-Investment estimation of SOA decisions

We hope that the example scenarios motivating the SOA modification operators in the above subsection sufficiently demonstrate that the gap between investing in SOA development and evolution and enabling strategic business decisions becomes increasingly smaller. This implies the need of a more sophisticated model for estimating the cost and the value of any particular SOA evolution investment.

Traditionally, software-engineering economic models have focused on the prediction and analysis of the costs associated with the development of an application [7]. The implicit assumption is that the value of a software product is known from the outset, based on a contract between the software company and the business client. As the value is constant, profit for the software development company can be maximized simply by reducing operating and development costs.

In our recent work [22], we have been examining present an initial step towards such an integrated model for cost/value estimation of SOA evolution. We adopt COCOMOII [8] as a software-development cost-estimation model because it explicitly models software-development cost as a function of several factors of the reuse-based development process, which SOA represents. In effect, COCOMOII estimates the cost of creating a software-development project as a function of the

complexity of the transformation of the application from an existing state to its envisioned state. In estimating the value of a software-development project, one has to consider both the revenues produced directly by marketing the newly supported services and also the non-tangible value of the potential service that could "easily" be built in the future using the evolved service-oriented application.

The flexibility and reusability of a service composition can be modeled with real option theory [12]. Real-option analysis, in conjunction with traditional service-valuation market research, enables the determination of the value of an SOA project. Combined with the cost analysis of developing and maintaining the system, a complete service-based economic model can be developed.

3.4 Mixed Service-Delivery with SOAs

Today, mixed-contact B2C service-delivery processes, i.e., processes incorporating automated, self-service and employee-mediated activities, are the de-facto standard for service delivery. For example, customers may order goods from the online product catalog of a business, pay through PayPal, have the products shipped to their address, receive them, and then benefit from ancillary services, such as an exchange policy, and a bricks and mortar store near them.

Given this range of interactions among consumers, employees and systems in the context of service delivery, the research question becomes to design and manage high-quality service-delivery environments, with multiple heterogeneous points of contact between the consumers and the environment. To date, SOA standards have been mostly silent regarding the roles that people play in the execution of BPEL-driven processes: any such interactions have to be hidden as asynchronous messages sent to the process. The BPEL4People specification [29] is layered on top of the BPEL language so that its features can be composed with the BPEL core features whenever needed. It represents an effort to define the types of roles that people play and the precise mechanism by which the user interfaces through which they interact with the SOA can be integrated with the core BPEL process.

In this context, the service-delivery environment should be designed to accommodate heterogeneity in how diverse groups of consumers utilize the mixed contact automated and employee-delivered service. In the end, the technical specifications of the automated and human service-service delivery systems must translate into customer-relevant service quality attributes (including service fees) that yield both customer satisfaction and organizational sustainability (i.e., profitability).

4 Concluding Remarks

In this paper, we have attempted to discuss a framework around research activities in the area of SOA engineering and to provide an overview of some important pragmatic concerns for this research. To summarize, we have tried to argue that

1) SOA is well suited to support traditional business processes, offering a huge potential inter-organization system integration. Furthermore, it is also appropriate

for the development of ad-hoc, bottom-up integrations of existing services to deliver functionalities to niche markets of social networks. Finally, it can provide a powerful vehicle for software development for a wide variety of applications on cyberinfrastructure.

2) Run-time SOA management requires solutions to a more extended set of problems than the ones addressed by traditional autonomic-computing work, such as dynamic process workflow reconfiguration and end-to-end SLA management.
3) The evolution of SOA systems closely reflects the evolution of the organization business processes and it can be understood in terms of the Baldwin-and-Clark modularity operators. This type of systematic understanding also enables the cost-effectiveness analysis of each considered evolution scenario, based on software-development cost estimation and the real options created for the organization by the evolved SOA.
4) Finally, we argue that SOA orchestration should explicitly model the role of the users involved in their execution, as most organizations today rely on mixed-service delivery processes, involving multiple contact points among customers, employees and automated software systems.

References

1. Q&A: Vint Cerf on Google's challenges, aspirations http://www.computerworld.com/developmenttopics/development/story/0,10801,106535,00.html, November 25, 2005
2. Allianz Global Risks, Sekhon Associates: Allianz Global Risks: International Claims Reporting Using ACORD XML and Web Services. http://www.acord.org/Standards/pdfs/AllianzSekhon.pdf, February 2005
3. Chris Anderson: "The Long Tail: Why the Future of Business Is Selling Less of More". Hyperion, 2006. http://longtail.typepad.com/the_long_tail/2005/01/definitions_fin.html
4. Carliss Y. Baldwin, Kim B. Clark: "Design Rules, Vol. 1: The Power of Modularity", MIT Press Cambridge, MA, USA, 1999.
5. Warren Blanchet, Eleni Stroulia, Renée Elio: Supporting Adaptive Web-Service Orchestration with an Agent Conversation Framework. ICWS 2005: 541-549
6. Warren Blanchet, Renée Elio, Eleni Stroulia: Conversation Errors in Web Service Coordination: Run-time Detection and Repair. Web Intelligence 2005: 442-449
7. Barry W. Boehm, Kevin J. Sullivan: Software economics: a roadmap. ICSE - Future of SE Track 2000: 319-343
8. Barry W. Boehm, Alexander Egyed, Daniel Port, Archita Shah, Julie Kwan, Raymond J. Madachy: A Stakeholder Win-Win Approach to Software Engineering Education. Annals of Software Engineering 6: 295-321 (1998)
9. William R. Cook, Janel Barfield: Web Services versus Distributed Objects: A Case Study of Performance and Interface Design. ICWS 2006: 419-426
10. James R. Cordy: The TXL source transformation language. Science of Computer Programming 61(3): 190-210 (2006)
11. Philippe Deschênes: B2B Business Models- Case studies, MCETECH2006, http://www.michelleblanc.com/images/Presentation%20of%20various%20B2B%20Business%20models%20case%20study.pdf
12. M. Hakan Erdogmus, Jennifer Vandergraaf: Quantitative Approaches for Assessing the Value of COTS-Centric Development. IEEE METRICS 1999: 279-
13. Rick Kazman, Mark Klein, Paul Clements: ATAM: Method for Architecture Evaluation, http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html
14. Kostas Kontogiannis, Grace A. Lewis, Dennis B. Smith, Marin Litoiu, Hausi A. Müller, Stefan Schuster, and Eleni Stroulia, "The Landscape of Service_Oriented Systems: A Research Perspective," Proceedings International Workshop on Systems Development in SOA Environments (SDSOA 2007); Workshop at 29th IEEE/ACM Int. Conf. on Software Engineering (ICSE 2007), Minneapolis, Minnesota, USA; May 21, 2007, 6 pages, May 2007.
15. Paul P. Maglio, Savitha Srinivasan, Jeffrey T. Kreulen, Jim Spohrer: Service systems, service scientists, SSME, and innovation. Communications of the ACM 49(7): 81-85 (2006)

16. Robert Martin: “The Interface Segregation Principle”, http://www.objectmentor.com/resources/articles/isp.pdf
17. Robert Martin: “The Dependency Inversion Principle”, http://www.objectmentor.com/resources/articles/dip.pdf
18. Rimon Mikhaiel, Eleni Stroulia: Examining Usage Protocols for Service Discovery. ICSOC 2006: 496-502
19. David Lorge Parnas: On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM 15(12): 1053-1058 (1972)
20. Linda Dailey Paulson: Services Science: A New Field for Today's Economy. IEEE Computer 39(8): 18-21 (2006)
21. Eleni Stroulia, Yiqiao Wang: Structural and Semantic Matching for Assessing Web-service Similarity. International Journal of Cooperative Information Systems 14(4): 407-438 (2005)
22. Brendan Tansey, Eleni Stroulia: Valuating Software Service Development: Integrating COCOMO II and Real Options Theory, 29th International Conference on Software Engineering Workshops, 2007: 87-89.
23. Tom Welsh: Has Java become middleware? http://www.middlewarespectra.com/abstracts/2000_11_06.htm
24. John Zuk: “IBM Venture Capital Group Report” 19 Oct 2005, http://www-304.ibm.com/jct03004c/businesscenter/venturedevelopment/us/en/inthenewstemp/!!/gcl_xmlid=35968
25. John Zysman: The algorithmic revolution---the fourth service transformation. Communications of ACM 49(7): 48 (2006)
26. Ontology Alignment Evaluation Initiative, http://oaei.ontologymatching.org/
27. ISIS Approach: Service Migration and Reuse Technique (SMART) http://www.sei.cmu.edu/isis/smart.htm
28. BPEL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
29. BPEL4People http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people
30. SOAP: http://www.w3.org/TR/soap
31. UDDI: http://www.uddi.org/
32. WSDL: http://www.w3.org/TR/wsdl
33. Web Services Inspection Language: http://www.ibm.com/developerworks/library/specification/ws-wsilspec/
34. Open Grid Services Architecture, http://www.globus.org/ogsa/
35. Web Services Agreement Specification, http://www.ogf.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf
36. Web Services Security, http://www.ibm.com/developerworks/library/specification/ws-secure/
37. Web Services Transactions specifications, http://www.ibm.com/developerworks/library/specification/ws-tx/
38. Web Services usage survey, CBDI report http://www.cbdiforum.com/bronze/webserv_usage/webserv_usage.php3
39. Adoption of Web Services and Technology Choices, TechMetrix Research http://www.techmetrix.com/products/products.php?type=rep

40. Gartner Surveys Show Web Services Are Entering the Mainstream
http://www4.gartner.com/DisplayDocument?doc_cd=114570