Conceptualizing Service-Based Information System Evolution as a Complex Adaptive System

Ghada Alaa
British University in Egypt
Misr Ismalia Desert Road, El-Shorouk City, PO Box 11837, Egypt
ghada.alaa@bue.edu.eg

Service-based information systems are considered a cornerstone in architecting modern enterprise applications. Service-oriented architecture (SOA) is designed to enable dynamic integration of heterogeneous application elements, and thus improve enterprise agility. This is achieved by publishing reusable services on a common registry or enterprise service bus to become available to users who will request and invoke them according to their business needs. In this paper it is argued that service-based information systems are different from component-based systems. SOA relies on the concept of contracting services to become invoked by users through service matching and binding, beside a middleware interface that integrates heterogeneous components (as supported by component-based architecture principles). In order to enable service evolution it is suggested to conceptualize service-based information systems as a complex adaptive system. Complexity science seeks to theorize the phenomenon of emergence of new properties and the spontaneous creation of new order, and thus provides elements to realize adaptability and evolution. By mapping complexity principles to SOA features factors that would facilitate information service evolution can be derived. It is concluded that in order to enable sustainable evolution of information services controlling factors, such as contracting, licensing, provenance, reliability and sustainability are paramount beside component-based development principles that include reusability, loose coupling, inter-operability, scalability and platform-independence. New programming discipline practices are also concluded, that include service choreography model, BPEL specifications, meta-data specifications, SLA specifications, service semantics and service tests. These ensure fast development balanced with discipline and quality, beside extensive collaboration and interactions facilitated by conventional agile development practices like JAD sessions and prototyping.

Key words: SOA, service evolution, factors, complex adaptive system

Service-Oriented Architecture (SOA) Standards

Service-based Information Systems and Service-Oriented Architectures (SOAs) are considered as cornerstone framework and standards in architecting modern enterprise applications. SOA applications are composed of reusable services, well-defined and autonomous, that are published on open repositories and interlinked by standards-compliant interfaces (Zimmermann et al., 2004). In this regard, service-based information systems are argued to offer an effective way to realize business agility as they provide a mechanism for integrating existing legacy applications regardless of their platform (Ren & Lyytinen, 2008).

According to Mohan (2002) Webservices provide the technology required realizing SOA, he defines Webservices as **self-contained**, **self-describing** and **modular** Web elements that can be **published**, **located** and **invoked** by users or enterprises across the Internet. Web services standards formulated by W3C have outlined architecture, protocols and language specifications to realize generic SOA implementations. These include according to Weske (2007) and Ren & Lyytinen (2008):

- XML, markup language for data structuring
- SOAP or REST, Web protocol for communicating service calls and messaging over HTTP or SMTP/email protocol
- UDDI (Universal Description, Discovery, and Integration), a universal application programming interface (API) that facilitates service registration and searching
- WSDL (Web Service Description Language) for service description that include information on data structure types returned or invoked by the service, transport protocol used, physical service endpoint, etc. (see Figure 1 for more elaborations)

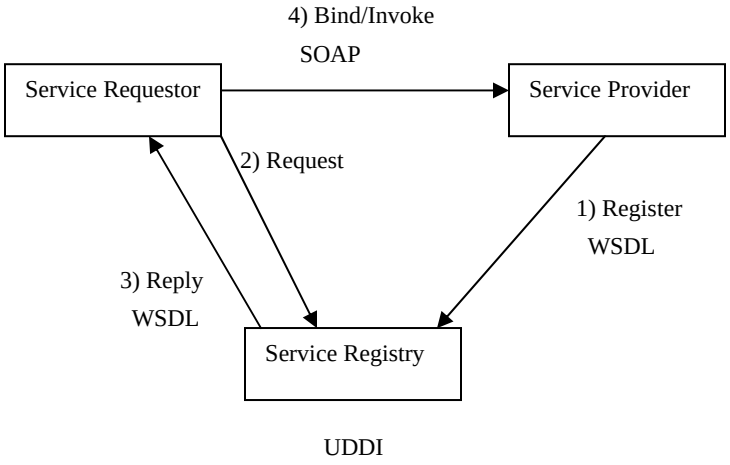

Figure 1: The SOA Framework adopted from Weske (2007)

Business processes typically encompass multiple service invocations and in this regard WS-BPEL (Web Services Business Process Execution Language) introduced by OASIS describes language syntax to compose and execute Web services. These include operation commands, such as invoke, receive, reply, wait, assign, throw and terminate, as well as control flow commands that include sequence, switch, pick, while, flow and link (www.oasis-open.org). Once a system architect finalizes the required service composition using a BPEL editor, the relevant BPEL file is generated. The service can then be made available by storing the respective WSDL file in UDDI. When the service is invoked by a user the BPEL engine reads the stored BPEL specifications in order to execute the sequence of required services (Weske, 2007). The sequencing, selection, and execution of services is termed *service choreography* that evolve as response to a certain business need (Zimmermann et al., 2004).

Issues in Modeling Service-Oriented Architectures (SOAs)

Zimmermann et al. (2004) specify quality attributes for SOA that cover reusable (well-crafted services), loosely coupled, cohesive abstractions, stateless, meaningful to business and standardized to comply with enterprise architecture patterns and underlying technologies. Major modeling activities will concentrate on service discovery, service composition, and service granularity, defining Service Level Agreements (SLAs) or in other words standards-compliant interfaces between the services. In addition, semantic brokering is an important issue in SOA modelling. This refers to semantic interpretation of related service invocation parameters and underlying domain ontology (descriptive domain key words) that are paramount for dynamic service discovery and binding.

According to Ravichandran et al. (2007) IT architectural design features for SOA include reusable components, modular, autonomous, i.e. capable of interaction and adaptability without human intervention, interoperable, and re-configured flexibly in run time through service matching and dynamic binding. Ren & Lyytinen (2008) specifically distinguish between reusability, agility and scalability as quality attributes for IT architectures. They explain that OO concepts support reusability factors, client-server architecture supports agility and scalability, whereas SOA supports reusability, agility and scalability as it combines OO and component-based development features with the client-server architecture advantages.

In addition, Ren & Lyytinen (2008) classify design features for service-based information systems as **system features** (that include platform-independence, loose coupling, re-usability and interoperability), **service features** (that include encapsulation, autonomy, deceivability and designed for contracting) and **business features** (that include meaningful to business, comply with business process and suitable for enterprise integration).

Reflecting on a real-case technical practice of SOA, a Learning Resource Recommendation System for universities, Shabir & Clarke (2009) suggest that key elements in designing service-based information systems are sustainability, provenance, licensing and reliability. By reliability they mean to ensure that service-based information systems will manage cases when linked sources become temporarily or permanently unavailable by providing a feedback message instead of encountering an error, whereas sustainability will find contingency plans to recover data sources that become unavailable or find substitutes if the original host shuts down permanently. Provenance refers to the possibility to trace data published on a linked repository back to its original source which will ensure its provenance/correctness and originality, and licensing is to provide governance to retrieval, processing and storage of data on open linked registries.

Similarly, Lin et al. (2009) emphasize workflow monitoring and management, provenance management and data quality management as core building blocks for SOAs. Provenance module will cover Querying, Exception handling, RDF-to-Relational data mapping, OWL (Web Ontology Language)-to-Relational Schema mapping and Relational Provenance repository. Whereas the data quality module will cover XML-to-Relational data mapping and the workflow

management module will cover workflow scheduling, removing redundancy, orchestration and breakdown into discrete, autonomous task activities.

Development Approaches for Service-Oriented Architectures (SOAs)

According to Zimmermann et al. (2004) service-oriented information systems analysis and design (they refer to as SOAD) have roots in three major existing disciplines; *Object-Oriented Analysis and Design (OOAD), Enterprise Architecture (EA) frameworks*, and *Business Process Modeling (BPM)* techniques. They suggest a hybrid approach that collates suitable elements from OOAD, EA, and BPM to come up with a three layers SOAD approach to include component, software service & business service layers. In the **business service layer** the approach suggests the use of *BPM techniques*, such as workflow diagrams, as well as *UML Sequence and Interaction diagrams* to model the interaction between the different components across the *enterprise service bus*. In the **software service layer** the approach suggests the encapsulation and *granularity of services*. In this regard, integration of existing legacy applications can be decomposed into stateless services, where *reusable business processes* and rules are abstracted into autonomous services managed by a *business choreography model* represented by *BPEL specifications*. In addition the CRUDS (Create, Read, Update, Delete and Search) metaphor would also help in service modeling and abstractions. In the **component layer** the elementary components that constitute the service will be represented as *UML class diagrams.*

Bell (2008) suggests a three phase approach for service-oriented modeling that includes service abstraction, service analysis and design activities. In the **abstraction phase** *service discovery* and *conceptualization* (high-level abstractions of business logic and re-usable processes) will be carried out, in the **analysis phase** *service descriptions* will be carried out along with *business integration*, *enterprise architecture* and *meta-data specifications*, whereas in the **design phase** *component* and *architecture logical and physical designs* will be outlined.

Bitzer & Schumann (2009) interpret the development approach of SOA and the interplay between the Functional & IT department during this process. They emphasize appropriate interactions between both departments in order to overcome the Business/IT gap in modeling service-based information systems. The development process starts with a **business analysis** and *service conceptualizations* by the Functional department. Then both departments collaborate in **designing the SOA** by producing the corresponding *BPM models* and *BPEL specifications* supported by BPEL editors. Then **orchestration** of the different services will be carried out by the IT department in order to form the *service choreography* supported by BPEL editors. Finally **execution** and **governance** of services within the required SOA will be undertaken by the IT department.

Niemann et al. (2008) suggest a generic governance model for SOA based on a survey of several approaches suggested in literature to govern development, provisioning and operation of service-based information systems. They specify an SOA Governance Control Cycle to include four phases; planning, design, realization and operation. The **planning** phase will cover *SOA preliminary specifications* along with *organizational governance* issues such as staffing, competences, *streamlining cross department processes, migration of legacy systems &*

processes, *enterprise-wide consolidation*, as well as *policy and metrics planning*. The **design** phase will address detailed business and technical requirements, *SOA topology* and detailed service specifications. The **realization** phase will target implementation issues, such as realizing the *service registry and semantics, SLA* implementations, *continuous service tests and reviews*. Whereas the **operation** phase will cover the major governance activities that will include *business service registry* management, business service evolution management, *architecture evolution* and management, *SLA management*, etc. The several SOA stakeholders will contribute to outline *Best Practices* for SOA governance that will in turn define *SOA governance policies* and relevant *metrics & SOA maturity measurement*. This is an *iterative* and *continuous* process that will cover several *feedback loops, adjustments* and *improvements*.

Baskerville et al. (2005) investigate development activities acquired by banks in order to set up their SOA topology. They have studied the implementation of SOA at a Scandinavian bank and a Swiss bank. The Scandinavian bank adopted a more agile approach implementing the bank's IT architecture as network-centric, where a service integration layer facilitated the integration of the many legacy systems in the bank. This enhanced the internal enterprise application integration, extensibility to other bank's SOAs and different outside services, as well as made continuous *redevelopment* and re-configurations easier. The implementation has been accomplished as incremental steps while a service integration layer facilitates the integration of the many legacy systems in the bank. On the other hand the Swiss Bank faced problems in aligning the SOA technologies to their business processes and legacy systems. This required extensive training of operations and technical staff in SOA standards, as well as an *enterprise-wide consolidation* in order to streamline *cross-departmental processes*. Both cases had partnerships with external *vendors* that fuelled the *organisational learning process* in acquiring SOA standards and best practices. Collaboration with the vendor encountered extensive *prototyping* activities in order to incrementally review and test parts of the system, and also to discuss and negotiate new requirements. SOA development and deployment life cycle also witnesses extensive collaborative modelling and evaluation activities facilitated by *JAD (Joint Application Design)* sessions as implied by Abraham et al. (2008).

Complex Adaptive Systems Theory (CAS)

Complexity science seeks to explain the process of self-organisation, emergence of new properties and the spontaneous creation of new order. CAS theory originated in the natural sciences and articulates how interacting agents in systems adapt and co-evolve over time in creative and spontaneous ways (Dooley, 1997). According to Kaufman (1993) the behaviour of complex adaptive systems (CAS) is typically unpredictable, but exhibits various forms of order and regulation. Heylighen (2001) defines CAS as a system composed of interacting agents, which undergo constant change, both autonomously and in interaction with their environment. Heterogeneous agents exhibit various agent behaviours that can be defined in terms of "simple rules" where they adapt and evolve through their interactions and by changing their rules through learning as experience accumulates (Holland, 1996).

Complexity principles emphasize that emergence of properties and creation of new order are not explicable from a purely reductionist viewpoint, but the whole is greater than the sum of the parts (Kaufman, 1993). This means the focus of attention shifted from understanding the parts

or entities of which the whole was composed to the interaction of subsystems (agents) to form a system. The emergence of order from heterogeneous local interactions is formed when feedback from environment and interacting agents informs the circular dynamics of the system. Thus local interactions will influence the formation of global structures and the stability of their mutual reproduction(Küppers, 1999) (see Figure 2).

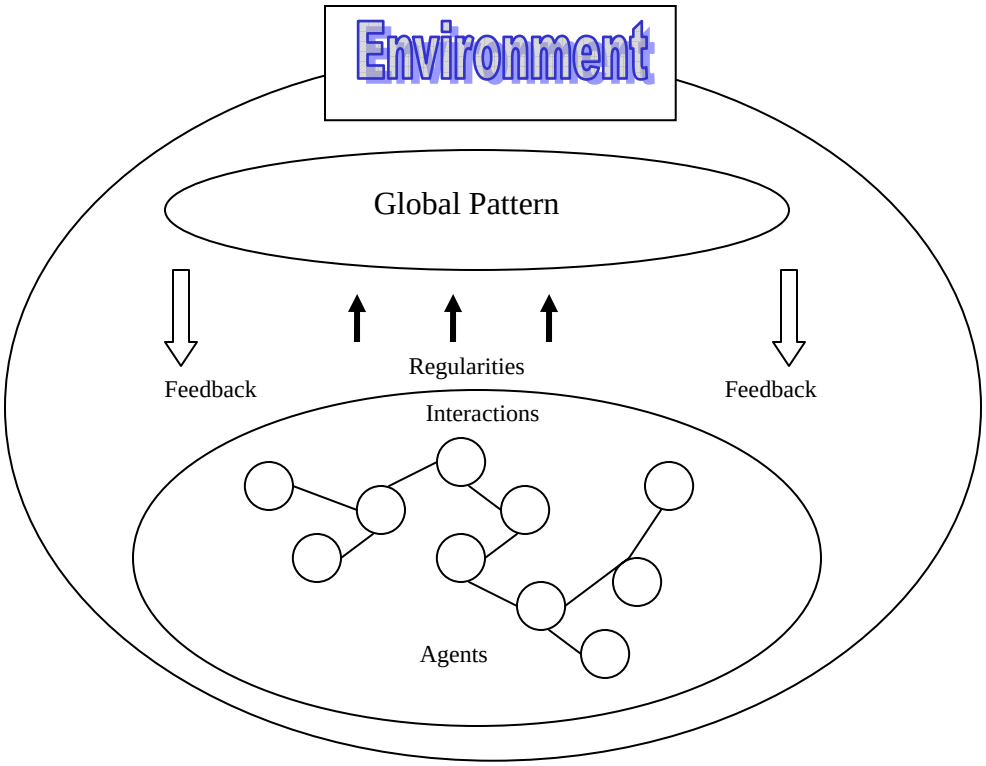

Figure 2: CAS Evolutionary Process

While a uniform description or interpretation of CAS is still not provided, several key aspects that characterise CAS were suggested in the literature. After reviewing several CAS literature Alaa (2009) concluded twelve CAS principles that facilitate emergence and evolution of business ecosystems; these include:

(P1) Large number of components:
CAS consists of a large number of components that undergo continuous change processes and re-arrangements, that in turn will define new identity of the system (Wulf, 1996).

(P2) Variation & diversity:
CASs are made up of heterogeneous agents (Holland, 1995); each agent is different from the others (diversity), and its performance depends on the other agents and the system itself (Benbya & McKelvey, 2006).

(P3) Space of possibilities/ adaptation to environment:

Tendency to adapt to a particular situation depends on the context and the influence of environment; this will determine the possibilities and alternatives available for change (Keller, 1996).

(P4) Connectivity & interdependence of components:
Structural coupling emphasizes the analysis of systems and their evolution in terms of their form, structure & degree of interconnectivity (Küppers, 1999).

(P5) Far-from equilibrium state/edge of chaos:
In order to harness change with no anarchy CAS strives to maintain a balance between the completely ordered, "frozen" regime and the completely disordered, chaotic regime, which is known as operating on "edge of chaos" (McKelvey, 1999).

(P6) Non-linearity:
Nonlinearity principle emphasizes that emergence of properties and new order are not explicable from a purely reductionist viewpoint, but the whole is greater than the sum of the parts (Kaufman, 1993).

(P7) Interactions:
Agents in a CAS undergo constant interactions, both autonomously and with their environment (Heylighen, 2001).

(P8) Feedback loops:
Inter-relations between the system parts result in feedback loops where components in the output stage inform components in the input stage (Andriani, 2003).

(P9) Pattern recognition & learning:
As a result of feedback loops adjustment in CAS takes place through the learning experience exhibited by its agents. This will change the agents' effect to realise the required outcome (Webb & Lettice, 2005).

(P10) Historicity & path dependence:
The behavior of the system in one period of time feeds back and informs to determine behavior in the next time; this gives the system 'historical dimension' (Stacey et al., 2000).

(P11) Self-organization:
Kant (1970) introduced the notion of self-organization as a mechanism to explain the emergence of order in CAS where external influences, e.g. natural forces or social contracts do not govern the internal dynamic of an entity/organization.

(P12) Co-evolution:
Agents rarely are partitioned into non-overlapping groups; they rather participate in multiple neighborhoods/undertakings simultaneously, where their various activities co-evolve (Anderson, 1999).

Information Systems Evolution from Complex Adaptive Systems Theory (CAS) Perspective

There is great interest in understanding modern organizations and systems in terms of theories of complexity (McKelevey, 1997, Stacey et al., 2000 and Mitleton-Kelly, 2003). This would provide a new way of thinking and reasoning on how adaptability and emergence can be realized in organizations, and similarly information systems development and evolution can be interpreted using complexity principles.

In order to operationalize CAS principles, Alaa (2009) provides a classification of underlying concepts and theoretical representation of CAS evolutionary process by categorizing CAS factors of emergence into (a) *dynamics of emergence*, i.e. factors that realise emergent properties such as flexibility of subcomponents, diversity, simplicity, high level abstractions, short-term orientation and rapidity in response and operation, (b) *enabling infrastructure*, i.e. factors that enable the dynamic properties to become effective, such as systems architecture with re-usable and loosely coupled components, organization structure and management style & culture, etc. and (c) *controlling factors*, i.e. factors that will balance excessive change with stability and thus sustain the business ecosystem to operate at the edge of chaos without descent into anarchy; these include feedback loops, continuous reflection and adjustment, non-restrictive/generic rules and discipline for operations and management.

It is found that social construction elements, such as communication, collaboration, interaction, etc. are argued to be critical drivers of human empowerment and self-organisation, whereas mechanistic, adaptive dynamics like flexibility, short-term orientation, small scale approaches, simplicity and rapidity will ensure fast response and quick adaptation to the problem situation. However, evolutionary properties cannot be fully realised without the necessary enabling infrastructure that will allow the dynamics of emergence to become effective. Also appropriate control mechanisms, such as feedback, reflection, learning, discipline frameworks and methodologies for management and operation, as well as embracing quality control factors need to be in place in order to ensure emergence to happen without descent into anarchy (edge of chaos). The elements or factors in each category have been identified and related in a framework, to help understand and analyse the phenomenon of emergence and facilitating evolutionary properties in social organisations in general, and information systems development in particular (Table 1).

	Dynamics	**Enabling Infrastructure**	**Controls**
Intangibles	Communication Collaboration Interaction	Management style Culture	Reflection Learning
Tangibles	Short-term orientation Small-scale Rapidity Flexibility Simplicity	Organisation structure Technical architecture	Feedback Continuous re-adjustment Quality controls Discipline in programming Minimal development methods

	Supports the principles of degree of interdependence, connectivity, quick mechanistic adaptation, interactions, diversity, self-organization and non-linearity. CAS P2, P4, P6, P7, P11	These enable or allow the dynamics of emergence to either be effective or inhibited, as well as adaptation to the environment & co-evolution CAS P1, P3. P12	These operationalize feedback & learning to ensure a balance between excessive change and stability and thus sustaining the edge of chaos and also giving rise to historical dimension while enterprise knowledge accumulates. CAS P5, P8, P9. P10

Table 1: Facilitating Evolutionary Properties in Information Systems Development Supported by CAS

Service-Based Information Systems Evolution from Complex Adaptive Systems Theory (CAS) Perspective

By mapping factors facilitating ISD evolution as implied by CAS (Table 1) to elements and features of service-based information systems development, factors of service systems evolution can be derived. In previous sections specific characteristics of SOA were outlined as compared to component-based systems, along with issues in modelling SOA and development methodologies targeted at successful development, operation and governance of SOA. In doing so, several key elements characterizing SOA implementation have been identified (represented in italic). Mapping such elements to different aspects of the evolutionary process (dynamics, enabling infrastructure, controls) yields the suggested framework for service-based information system evolution represented in Table 2.

For the enabling infrastructure factors like reusable components, loose coupling, interoperability (based on WSDL and application programming interfaces/APIs), platform independence, enterprise service bus and linked-open registries (repositories) have been identified under technical architecture. Under organization structure elements such as enterprise-wide consolidation, streamlining cross departmental processes, migration of legacy processes, etc.

For the dynamics of service evolution attributes like autonomous, encapsulation (service discovery & conceptualization), abstraction, stateless, meaningful and standardized to business (Business/IT alignment), designed for contracting, service matching, linking and binding are suggested. Rapidity is also important in SOA implementations in order to cope with the rapid change in current business environments, and increase the enterprise competitive advantage. Collaborations and interactions are also paramount represented in interactions with the vendors, users and between the different departments (for example during JAD sessions).

Controlling factors for service-based information systems evolution is different from component-based development, due to the fact that SOA relies on the concept of contracting and brokering. Therefore, quality control factors such as licensing, provenance, reliability and sustainability become necessary to govern the process of contracting and service matching & binding. This is because a licence should be provided before contracting, as well as a provenance process is required to trace back sources of data for correctness, and a process to ensure that the link to be invoked is reliable (not to disappear or shut down) and sustainable, i.e. other relevant links are provided in case of shut down of the primary link. This is beside the usual quality control factors for component-based development that include security and redundancy removal.

In order to ensure the successful implementation of SOAs several development methodologies/approaches have been suggested in literature as discussed before. These also provide controlling elements for SOA evolution, as they govern the process of service development and evolution life cycle. But these approaches need to be not restrictive in order to facilitate rapidity and quick adaptation; i.e. provide minimal guiding instructions without being cumbersome as supported by CAS principles. Agile development rationale to emphasize light weight development is rush into coding but governed with discipline in programming that will ensure quality of programming outcome. In case of SOA new programming disciplines can be concluded; service choreography model, BPEL specifications, meta-data specifications, SLA specifications, service semantics and service tests.

According to Baskerville et al. (2005) IS development in modern enterprises requires efforts of application integration to incorporate new functionalities in existing legacy systems, as well as improving the strategic value of enterprise by including new innovative, value-added services. Both application integration and value added services improve enterprise agility and thus require an agile development approach. They witnessed in their field analysis of SOA implementations agile practices, such as prototyping, break down of concerns, and extensive collaboration and interactions with vendors and users. **But** they also noted a possible **conflict** between the conventional mindset of enterprise IT development weighted with regulation and security concerns, as well as tendencies to long-term orientations that would counteract agile development principles. This might give an explanation why there was no mention in surveyed literature about short-term orientation and small scale development for SOA, in contrast to pure agile development projects.

In this regard, Zimmermann et al. (2004) emphasize that the Rational Unified Process (RUP) should be suitable for SOA development as it supports iterative development, but with emphasis on architecture design. Thus RUP will balance the pure agile development approach. Though for RUP the system architecture is the structure of its components interacting via defined interfaces, but for SOA the architecture compromises of stateless, self-describing services that satisfy a generic business use. Another difference is that RUP is use-case oriented and grounded in the UML approach, but BPM compromises event-driven process models, thus SOA implementations will emphasize in first stages BPM techniques such as workflow diagrams and BPEL representations.

	Dynamics	Enabling Infrastructure	Controls
Intangibles	**Collaboration & Interaction for SOA** o Cross-company o IT vendors o Users o JAD (Joint Application Design) sessions	**Management style & Culture for SOA** o SOA Best Practices o SOA Governance Policies o SOA Metrics & Maturity Measurement	**Reflection & Learning for SOA** o Continuous refection & governance
Tangibles	**Flexibility for SOA** o Encapsulation & Abstraction (Service discovery, conceptualization & abstractions) o Meaningful and standardized to business (Business/IT alignment) o Stateless o Designed for contracting, service matching, linking and binding o Autonomous **Rapidity for SOA** o Faster time to service ***Short-term orientation, Small-scale & Simplicity for SOA*** They have not been addressed directly in current literature. Short-term orientation and small scale development counteracts enterprise integration principles, but still need to be embraced in a balanced way due to that fact of rapid technological obsolescence. Simplicity is an important agile principle that needs to be more exploited in SOA implementations.	**Evolutionary Technical Architecture for SOA** o Reusable components o Loose coupling o Interoperability (based on WSDL and application programming interfaces/APIs) o Platform independence o SOA topology & Enterprise service bus o Linked-open registries (repositories) **Organization structure for SOA** o Enterprise-wide consolidation o Cross department process streamlining o Migration of legacy processes	**Feedback & Continuous re-adjustment for SOA** o Extensive prototyping o Continuous improvement **Quality controls for SOA** o Licensing o Provenance o Reliability o Sustainability o Security o Remove redundancy **Discipline in programming for SOA** o Service choreography model o BPEL specifications o Enterprise Integration & meta data specifications o Service registry and semantics o Service tests o Review of SLA for semantic correctness **Minimal development methods for SOA** o SOAD (Zimmermann et al., 2004) o Service-oriented Modelling Framework (Bell, 2008) o Generic Governance Model for SOA (Niemann et al., 2008)

Table 2: Facilitating Evolutionary Properties in Service-Based Information Systems Development Supported by CAS

Conclusion

Complex adaptive systems theory conceptualizes the phenomenon of emergence, self-organization and spontaneous creation of order. Several generic complexity principles have been suggested in literature, such as diversity, large number of agents, interconnectivity, interactions, feedback, edge of chaos, etc. that refer to evolutionary characteristics. In this paper it is suggested to conceptualize service-based information systems as a complex adaptive system and in that way derive factors that would facilitate information system service evolution.

After surveying several elements related to service-based information system development, such as specific characteristics of SOA as compared to component-based development, issues in modeling SOA, as well as development approaches suggested in literature to govern service life cycle and evolution, it is concluded that service-based information systems are different from component-based systems. As implied by Zimmermann et al. (2004), although SOA put forth reusable software architecture principles represented in information hiding, modularization, and separation of concerns, it also embraces new concepts such as service choreography, service repositories, and the service bus middleware in enterprise integration. Thus SOA relies on the concept of brokering or registry of open services to become available for enterprises to search and invoke (bind or link), as well as provide a middleware interface to integrate heterogeneous components.

By mapping service-based information system development elements to CAS principles several factors that would facilitate evolutionary properties of SOA have been derived. It is concluded that in order to enable sustainable evolution of information system services licensing, provenance, reliability and sustainability are paramount beside object-oriented and component-based development principles that include reusability, loose coupling, inter-operability, scalability and platform-independence. New programming discipline practices are also concluded, such as service choreography model, BPEL specifications, meta-data specifications, SLA specifications, service semantics and service tests. JAD and prototyping are also appropriate for SOA development, BUT more attention should be paid to the architecture design as implied by RUP, and also maintenance and governance process is of great importance, as SOAs have more longevity and deployment element than simple information systems that are small scale and short-term oriented.

It is suggested that the introduced framework guides system and IT architects, as well as management teams with factors or strategies that will enhance service-based information system evolution supported by CAS principles. These are preliminary findings and future work will focus on a more detailed mapping of the different factors, as well as application on a real-case study in order to evaluate the proposed framework and put forth metrics for SOA evolution based on identified factors.

References

Abraham C., Junglas I., Willis M., 2008, Enabling an Agile Information Supply Chain in Service Oriented Architectures with Web Services, AMCIS 2008 (American Conference on Information Systems)

Alaa G., 2009, Derivation of Factors Facilitating Organizational Emergence based on Complex Adaptive Systems & Social Autopoiesis Theories, Emergence: Complexity & Organization Journal, Vol 11 No. 1, pp. 19-34

Anderson, P., 1999, "Complexity theory and organization science", Organization Science, Vol. 10 No. 3, pp. 216-33

Andriani P., 2003, Evolutionary Dynamics of Industrial Clusters, in Mitleton-Kelly (ed.), Complex Systems and Evolutionary Perspectives on Organisations: The Application of Complexity Theory to Organizations, Elsevier, 2003

Baskerville R. et al., 2005, Extensible Architectures: The Strategic Value of Service-Oriented Architecture in Banking, Proceedings of ECIS 2005 (European Conference on Information Systems)

Bell, M., 2008, Service-Oriented Modeling: Service Analysis, Design, and Architecture, John Wiley and Sons, New York

Benbya H. & McKelvey B., 2006, Toward a Complexity Theory of Information Systems Development, Information Technology & People, Vol. 19 No. 1, 2006, pp. 12-34

Bitzer, S., Schumann, M., 2009, Mashups: An Approach to Overcoming the Business/IT Gap in Service-Oriented Architectures, Proceedings of AMCIS 2009 (Americas Conference on Information Systems)

Dooley, K. (1997), "A complex adaptive systems model of organization change", Nonlinear Dynamics, Psychology, & Life Science, Vol. 1 No. 1, pp. 69-97.Goldman, Steven. L., Roger N. Nagel and Kenneth Preiss, 1995, Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer, New York: Van Nostrand Reinhold, 1995

Heylighen F., 2001, "The Science of Self-organization and Adaptivity", in: Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems (EOLSS), Publishers Co. Ltd, pp. 253--280

Holland, J.H., 1996, "Hidden Order: How Adaptation Build Complexity", Basic Books, ISBN**:** 0201442302, 978-0201442304

Kant, I., 1970, Universal natural history and theory of the heavens. In: Kant: Cosmogony (original: Allgemeine Naturgeschichte und Theorie des Himmels (1755)).

Kauffman, S.A., 1993, The Origins of Order, Oxford University Press, New York, NY

Keller K., 1996, Socio-technical systems and self-organization, ACM SIGOIS Bulletin Volume 17, Issue 1 (April 1996)

Küppers. G., 1999, Self-organisation - the emergence of order from local interactions to global structures. University of Bielefeld- SEIN (Simulating Self-organizing Innovation Networks) project, Germany, July 1999, http://www.uni-bielefeld.de/iwt/sein/paperno2.pdf

Lin et al., 2009, A Reference Architecture fpr Scientific Workflow management Systems and the View SOA Solution, IEEE Transactions on Service Computing, Vol. 2, No. 1.

McKelvey, 1997, Quasi-natural Organization Science," Organization Science, 8, 1997, 351–381.

Mitleton-Kelly E., 2003, Ten Principles of Complexity and Enabling Infrastructures, in Mitleton-Kelly (ed.), Complex Systems and Evolutionary Perspectives on Organisations: The Application of Complexity Theory to Organizations, Elsevier, 2003

Mohan C., 2002, Dynamic e-business: Trends in Webservices. In Buchmann et al. (2002), pp. 1-5

Niemann M., Eckert J., Repp N., Steinmetz R., 2008, Towards a Generic Governance Model for Service-Oriented Architectures, Proceedings of AMCIS 2008 (Americas Conference on Information Systems)

Ravichandran T., Leong Y., Teo H., Oh L., 2007, Service-Oriented Architecture and Organizational Integration: An Empirical Study of IT-Enabled Sustained Competitive Advantage, Proceedings of ICIS 2007 (International Conference on Information Systems)

Ren M., Lyytinen K., 2008, Building Enterprise Architecture Agility and Sustenenace with SOA, The Communications of the Association for Information Systems (CAIS), Volume 22, Article 4, pp. 75-86, January 2008

Shabir N., Clarke C., 2009, A Resource List Management Tool for Undergraduate Students Based on Linked Open Data Principles, Proceedings of the European Conference on Technology Enhanced Learning (ECTEC 2009)

Stacey R., Griffin D. & Shaw P., 2000, Complexity and Management, FAD or Radical Challenge to Systems Thinking?, Routledge

Webb C., Lettice F., 2005, Facilitating Learning & Sense-Making with Complexity Science Principles in Organisations by Means of a Complexity Starter Kit, Proceedings of the Complexity, Science and Society Conference, 11th-14th September, 2005, Centre for Complexity Research, University of Liverpool, UK

Weske M., 2007, Business Process Management: Concepts, Languages, Architectures, Springer-Verlag

Wulf V., 1996, The autopoietic turn in organization science and its relevance for CSCW, ACM SIGOIS Bulletin Volume 17, Issue 1 (April 1996)

Zimmerman, O., Krogdahl, P, and C. Gee (2004) Elements of Service-Oriented Analysis and Design: An interdisciplinary modeling approach for SOA projects, www.ibm.com/devloperworks/library/ws-soad1/index.html