Governance and Control Challenges and Approaches for Service Oriented Architectures

A. B. K. Madurai [a,b]

March 2011

[a] **Kingston University, Penrhyn Road, Kingston Upon Thames, KT1 2EE**

(b) CSC, Royal Pavilion, Wellesley Road, Aldershot, Hampshire, GU11 1PZ | bmadurai@csc.com | www.csc.com |

Abstract

Service Oriented Architecture (SOA) has been advocated to deliver the promise of better integration in order to help organisations to be adaptable and agile to changes in the market, competition, regulation, compliance and more importantly to become sustainable. Modelling and formalisms can play a major role in achieving true service orientation. The IT industry for some time now has recognized the need for understanding and modelling Enterprise Architectures and Enterprise Integration – yet in practice the problem is often approached using methods and tools that are more art than science. The service-oriented architecture (SOA) paradigm provides a promising way to address these challenges at the level of the company's IT infrastructure. These challenges and the management of the introduced complexity and heterogeneity are targeted by SOA Governance approaches. Though the basic structure of IT Governance frameworks is applicable to SOA implementations, they lack applicability concerning some SOA-specific challenges.

In this paper, I discuss deficiencies of current methods and provide insights of how these challenges can be potentially addressed through SOA Governance approaches that are underpinned by modelling and simulation techniques and processes.

Introduction

The SOA paradigm provides for a holistic approach for the execution of end to end business processes within enterprise architectures (EA). Services represent a central aspect and foundation of the architectural paradigm SOA. Services can be defined as "atomic invariable building blocks that can be combined flexibly over open communication mechanisms". Their functionality scope is small and they appear in multiplicity. Generally, services are designed to support reusability in different scenarios by simple reconfiguration. Service functionalities can be automatically discovered via service brokers or registries that are centrally placed in an implementation landscape

Services are centrally registered either on a database or through a file, providing information about the services upon request. While interacting, services are loosely coupled and the mutual association is via discrete messages. Dependencies are minimized to mere awareness, facilitating a number of operations, e.g., their replacement by other services during runtime. Service operations always involve several parties or stakeholders. Services therefore adhere to a communications contract, a Service Level Agreement (SLA), defined by one or more service descriptions and related documents in order to regulate and control service execution [**TErl05**].

These characteristics make an SOA a powerful, flexible, easy to operate candidate for a company's enterprise architecture. The SOA paradigm offers a number of advantages. However, these advantages imply challenges emerging from the large numbers of services as scalability requirements and their heterogeneous nature increases. There are various methods and frameworks currently in use for SOA, developed either by organic processes or propagated by product vendors. Most of these frameworks lack the formalisms required to

accurately describe integration requirements between participating applications. The following are some the key pitfalls of SOA that are currently observed.

Pitfalls	Reality / Challenges
SOA is seen as a better version of EAI that finally improves middleware weaknesses by providing better interoperability, more reliability, wider adopted standards etc,	SOA is an architecture on higher-level than EAI. It is architecture of loosely coupled independent services communicating to each other. EAI is appropriate for small scale integration projects
Architects expect loose coupling comes for free with using web services toolkits or with using standards like SOAP, WSDL, XML Schema and so on	Loose coupling is a result of very careful API Design. Standards and tools might (or might not) contribute loose coupling
Service APIs are often designed from particular application perspective	Service APIs should conform to constraints defined by central body. This process is usually SOA Governance
RESTful services are considered insufficient for enterprise use cases because there is no standard way how to describe all kinds of application-level protocols. This makes design, development, management and monitoring of such services more difficult	Design and Development of RESTful services are indeed more difficult, especially for architects that are used to working with the traditional API Model. At the same time it is much easier to integrate such services together
It is easier to implement new business logic in the middle tier because it does not require changes in existing applications	Any additional business logic should be pushed out to the endpoints as much as possible. This is more difficult but keeps the SOA system much cleaner from an architecture perspective.
SOA Governance addresses both design time and run time activities for service development by default	In reality SOA Governance is a involves a substantially bigger people / process and technology problem and needs appropriate focus on these areas in order to get maximum returns

Thus, a critical aspect for success in the adoption and the operation of SOA is governance. The primary goal of SOA Governance is compliance, i.e. compliance to intra-company, normative, or legal standards (required by, e.g., the Sarbanes Oxley Act, Basel II etc.). Following specific guidelines in a top-down approach, an SOA is directed through a number of maturity levels or development steps – it is adopted, commissioned, operated, and continuously monitored and checked for adherence to regulations

The Problem Statement

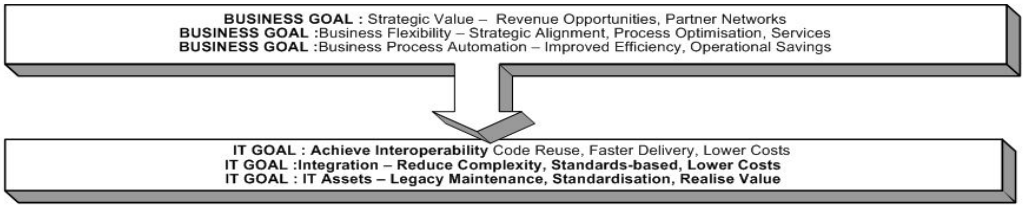
The problem statement hence for this study is to address the issue of ambiguity, testability of the design and run time artefacts and, the prevalent lack of proper due diligence / governance when creating and deploying services using currently available service orientated architecture methods and tools.

SOA Reasoning from a business justification perspective

The fundamental goal of Web Services and SOA is to improve Return on investment (ROI) and Total Cost of Ownership (TCO) over traditionally implemented accidental architectures through point to point interfaces which have caused issues of ambiguity, scale and cost of change and large number of redundant interfaces for every small change. However, the use of Web Services doesn't guarantee results. There are a number of challenges, many of which are organizational and not strictly technical, that needs to be addressed proactively to achieve measurable benefit. Narrowing down the set of acceptable ways to use standards from all the available options needs to be checked with policies and procedures. Of course, many of these policies and procedures will naturally need to change over time as regulatory or other compliance requirements change, so when thinking about TCO the inevitable cost of change needs to be considered.

A second challenge is determining who pays for a service shared by many applications. With traditional line of business applications, figuring out who pays is easy - since one team owns everything. For a shared service, ideally each line of business should pay proportional to their use. Those who use it most should pay the most. Essentially this is a transfer-pricing model. It is important to consider how to track usage by each line of business accurately - if one can't measure usage, one can't charge for it!

A third challenge is ensuring that service levels are met. To end users, the customers of the IT infrastructure, the order management application is still the order management application whether it's built as a monolith or it leverages shared services. In either case it must meet the same expectations of performance, availability and functionality. Conversely, if the same user uses two different applications, he may have different expectations of each - even if both applications leverage the same set of shared services. Figure [1] below introduces an aligned framework to detail the critical success factors (CSFs) and key performance indicators (KPIs) that need to be considered at when adopting SOA (see diagram below)



PROCESS	OBJECTIVES	CSF	KPI
BUSINESS AGILITY	Increase Agility and Flexibility of existing IT assets Define and maintain an Integration Architecture that links business functions to IT Service Delivery capabilities	Loosely-coupled, self-describing, independent services Services aligned to business functions	Percentage increase in deployment of independent Services Percentage increase alignment of Business Processes and IT Services
IT AGILITY	Enforce IT Enterprise Architectural Governance Deploy Web Services to enable Partner Links Speed Up Application / Service Deployment within IT	Ensure deployed services are based upon the defined and managed Enterprise Architectural Vision Build Services on IT common, interoperable standards	Adherence to EA and SOA Governance displayed Percentage Increase in Interoperable Solutions
	Define and Maintain Standards Create Centralised Services Infrastructure Reduce IT Operational Investment	Enable re-use of common services across multiple business functions	Percentage Increase in return of IT Investment
SERVICE QUALITY	Ensure Consistent Agreement and Monitoring of Target Response Times For Services	Deliver Services as agreed within Service Profile	Percentage Increase in SLA's agreed for Services Percentage Increase in SLA's against Operational Services run

Figure 1: Enterprise Architecture for Boundaryless Information Flow

While the core Web Services standards successfully address the mechanics of letting applications to talk to one another, successful SOA implementations through delivery of technologies like Integration Buses mean addressing the challenges that lie beyond the pure mechanics of communication. The complexities are numerous: stakeholders with different agendas, policies with cross-functional implications, service levels that must be maintained at all costs, complex interrelationships between services and no lines painted on the data centre floor to connect any of the dots. Left to grow on its own, a network of Web Services will quickly degenerate into a tangled spaghetti of brittle, single-use integrations and fail to achieve the economies of scale or the cost and flexibility benefits of SOA.

These challenges call for a new breed of solution - SOA Command and Control - that addresses the various technical, business, and organizational requirements and unites all pertinent knowledge in the SOA into a form that's understandable and actionable. There are five key imperatives of SOA Command and Control: Align, comply, observe, respond, optimize. These five imperatives encapsulate the required, and the associated value, of SOA Command and Control.

Align
IT is successful only if the business is successful, so IT must always be aligned with the business. SOA Command and Control must allow to measure SOA activity against business objectives to understand its current impact on the business, to determine how it's trending, and to predict where it will go in future.

Comply
True SOA Command and Control empowers stakeholders to move from a passive role to an active role of driving policy changes immediately and automatically across the organization. In addition, stakeholders gain visibility as to where and when the policies and procedures are being applied and/or violated.

Observe
By definition, SOA Command and Control provides both detailed and at-a-glance visibility into the inner workings of SOA at any point in time - automatically, without expensive, time-consuming manual configuration. This facility lets us understand SOA-wide patterns and trends that would never be uncovered with solutions that provide simple statistics and only threshold, rule-based or predictive alerting.

Respond
Root cause analysis (RCA) is important in an SOA because symptoms rarely appear at the location of the root cause - and the root cause may be a service owned by a different group. With SOA Command and Control allow us to accurately and automatically determine the root cause of problems, without expensive, time-consuming manual configuration of rules or relationships. And, once the root cause is determined, the business process can respond in one of many different ways such as notifying administrators, black-listing users, rolling back service changes, rationing capacity, or modifying documents in transit. These responses can be triggered manually, fully automated or even manually overridden when automated responses don't produce the desired result.

Optimize
As with any IT infrastructure, services have a finite capacity to process consumer requests. Determining the capacity requirements of services is especially complicated because each consumer has a different pattern to use with different kinds of requests, and different peak usage periods. And, as new consumers come online, they consume capacity from the service and potentially affect the service level of everybody else. SOA Command and Control lets the business both proactively and reactively optimize the allocation of scarce service resources.

With an effective SOA Command and Control infrastructure, policies can not only be defined once, centrally, but also automatically enforced in the fabric of the network itself. The capabilities of an effective SOA Command and Control platform lets organizations bypass the knowledge gap and successfully achieve the economies of scale as well as the critical cost, time and flexibility benefits of SOA. With effect to this need, typical organizations intend to leverage the power of standards based, metric driven SOA by implementing the Enterprise Service Bus (ESB) product suites for application integration using a phased approach. However it is important to consider that early implementations of ESB based architectures have not focused on governance issues where services have been built without any conformance checks to the design constraints or their models have not been tested through formal methods there by resulting in the implementation not recognizing the intended benefits of service orientation.

Governance Findings

In the area of IT Governance, a number of existing frameworks provide structures, action scope, guidelines, reference processes, and best practices. However, the basic structure of IT Governance frameworks often exceeds the needs of SOA, while lacking applicability concerning SOA specific challenges, e.g., cross-company cooperation issues like inter-company service deployment. Hence, in order to meet SOA Governance requirements, existing IT Governance frameworks need to be extended [**Wool06**].

Diligent SOA Governance has been recognized in recent years as a major requirement for successful adaptation and operation of an SOA, especially for large systems. Governance in general, be it political governance, Corporate or IT Governance, deals with the successful governing of organizations or projects. SOA Governance elaborates guidelines and rules that need to be adopted and realized by the affected management processes. It provides a means to effectively exploit the capabilities of SOA [**Mar06**]. The achievement of governance goals is supported by SOA Maturity Models and respective governance mechanisms.

SOA Governance focuses on the smooth adoption and successful operation of an SOA. By providing guidelines, responsibilities, and reference processes, it ensures its integrity and adaptability to business and administration processes. Governance tools support the monitoring and control of services concerning the alignment to business processes. A best practices catalogue serves as a repository of implementation recommendations that are continuously supplemented, supporting all of the mentioned procedures. Besides the achievement of IT goals and the realization of business-IT alignment, a further goal of SOA Governance is to realize system adherence to regulations and standards, such as ISO norms or internal regulations. Existing approaches to governance frameworks do not fully cover special SOA Governance requirements.

There are few scientific contributions that have dealt with SOA Governance so far. Nevertheless, a definition is important. There are numerous definitions of SOA Governance, all diverging in focus. In the context of this paper, based on [20], we understand SOA Governance *as a holistic long-term management model. It guarantees sufficient adaptability and integrity of an SOA system as well as the ability to check services concerning capability, reusability, security, and strategic business alignment. Overall goals are SOA compliance and the guarantee of reusability and standardization throughout the system.*

Numerous frameworks have been specified for IT Governance, e.g., CObIT, ITIL, ISO 17799, and many more. The ISO 17799 standard primarily targets security management [ISO], the IT Infrastructure Library (ITIL) mainly deals with IT process definition [**OGC**]. CObIT defines 34 reference processes as control framework, more tightly aligned with the business objectives of the organization than with operational issues [**ITGI**]. Comparing all these frameworks discloses that they complement each other and, as a matter of fact, CObIT represents a frame integrating all other frameworks – it has, so far, become a de facto standard for IT control globally.

In the area of SOA Governance frameworks, there are a few research contributions. Existing concepts are mostly motivated by software providers that offer SOA business solutions and closely align their SOA Governance perspectives with their products ("the fairly narrow view" [**Allen08**]). An overview of the different understandings of this topic is given by [**AMCIS08**], providing a survey and analysis of ten approaches to SOA Governance. Ten typical components of SOA Governance have been identified by times included in existing approaches (cf. Table 1). Detailed descriptions can be found in [**AMCIS08**].

Examination of the comparison shows that components that are covered by IT Governance frameworks are considered less important for SOA Governance. Role and accountabilities, metric models, and impact on behaviour are all not taken into account at the first place and, besides the first, they are part of standard IT Governance frameworks [**HBR04**]. This illustrates one difference between the approaches – some aim at completely covering SOA Governance challenges, others, not considering maturity models and behavioural impact, build on the additional implementation of parts of IT Governance approaches. Implementing parts of an IT Governance approach and additionally following an SOA Governance model increases cost and time efforts in IT departments – a diligent SOA Governance model should cover not less than all SOA related regularization aspects. Few approaches emphasize mechanisms to impact behaviour of employees or people working with the system, as well as SOA Maturity Models - although these aspects seem important for the operation of an SOA. Common IT Governance frameworks like CObIT address these issues. Although there are plenty of maturity models, e.g. they did not yet establish as a prevailing element of SOA Governance approaches, although they are part of, e.g., CObIT.

This supports the need mentioned at the beginning to bring together best practices from IT Governance and additional SOA-related regulation requirements – an extension of either SOA or IT Governance frameworks to allow for the respective needs, as already stated by [**Woolf**]. Overall, organizational changes, roles/accountabilities, policy catalogues, and service lifecycle, integrated in the majority of approaches, can be considered the most important elements. The governance model below Figure[2] gives a clear separation of concerns and the granularity of services.

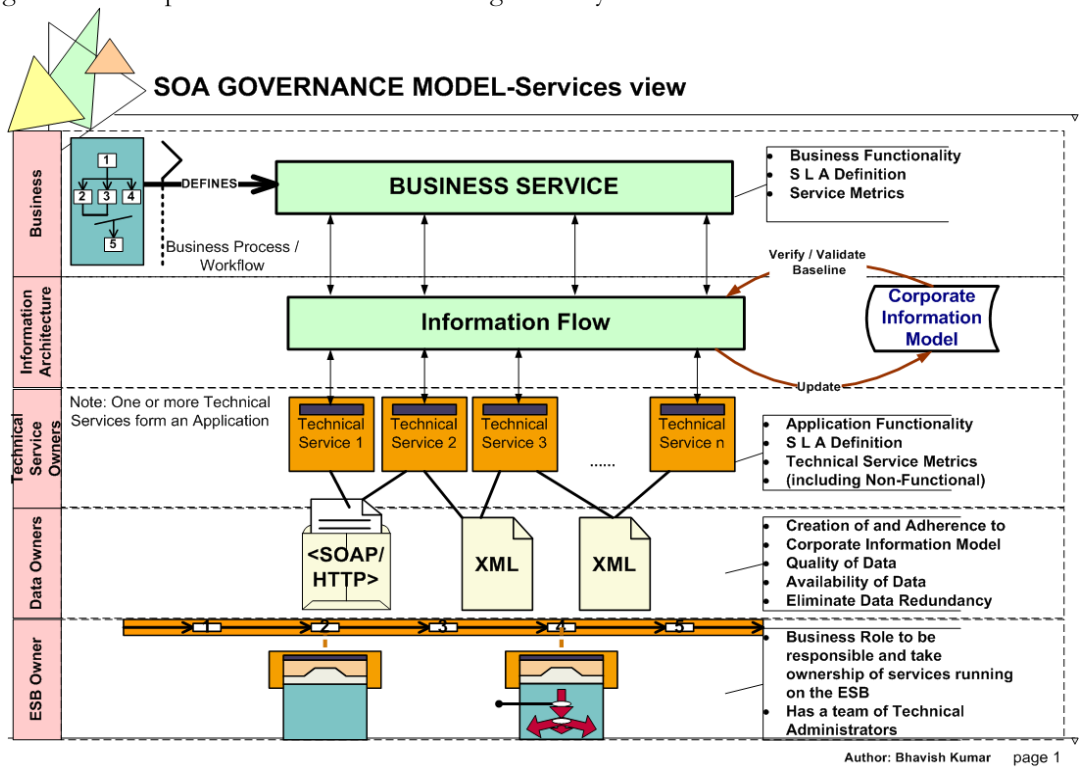


Figure 2: SOA Governance Reference Model

Steps to take when establishing SOA Governance:

Thinking of SOA Governance as a layered model helps to clearly articulate the vision and the deliverables and ownership of actions thereby leading to helping to build the justification case/ business case with tangible benefits identified at each layer. Each aspect of SOA governance requires defined goals which could be around SOA processes and policies must be defined and documented, SOA processes must have clear accountabilities, there should be strong support/commitment from senior management, all SOA processes should be agile, that facilitates continuous change and feedback. For effective SOA Governance, it is necessary to have workable organizational structures to control and support all governance activities. People within the SOA Governance Model must be empowered to make and enforce decisions. Each enterprise will have differing structure requirements and should transcend the organization chart. SOA Board and SOA Compliance Team should have global, regional or business line scope

This requirement warrants for an enterprise class coherent SOA framework to help deliver this governance and control through formal methods bringing together the required rigor and control for SOA

A Complete Coherent Framework for SOA Governance

There is a need to define a complete, coherent governance framework for SOA and here the fundamentals of such a framework are introduced. Each of the layers defined in the diagram below addresses key concerns and considerations that we have brought to light in the sections above. This framework also highlights clear responsibilities between vendor / suppliers and customer organisations who engage in outsourcing partnerships to deliver service orientation. Such a comprehensive framework ensures credibility, adaptability and flexibility in approach when using service orientation.

However to make this framework really come to life, early research indicates that we must use complimentary formal modelling methods within the context of the following framework to ensure that control and due diligence is applied both in Design Time and Run Time of service development by introducing formal modelling methods and techniques for service orientation in relation to the overarching framework given in Figure [3] below

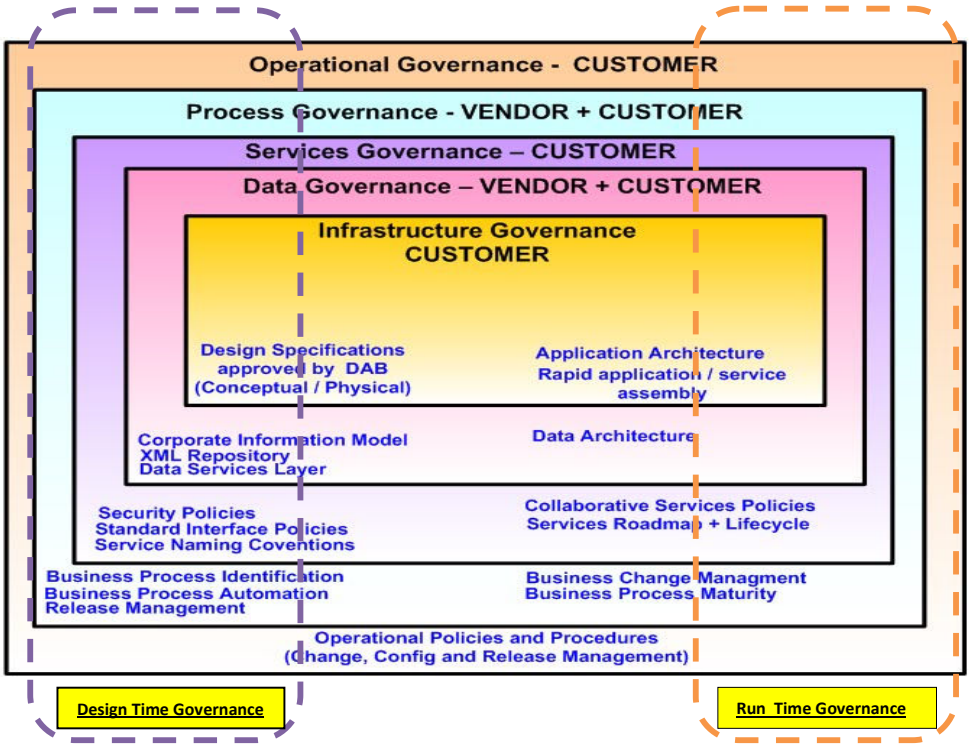


Figure 3: A Coherent and Complete SOA Governance Framework

It is very important to consider that the governance is implemented not just in TOP DOWN or BOTTOM UP fashion but using a MIDDLE-OUT / Blended (i.e. focussing both on **design time** and **run time** activities) approach where in there are equal considerations made to business justification, roles and responsibilities, processes at the TOP while also realistically implementing key formal techniques at the BOTTOM to ensure that services developed are in conjunction with the processes and guidelines defined.

While TOP DOWN SOA Governance requirements can be addressed through best practice from industry and reference implementations, the BOTTOM UP requirements warrants formal methods and approaches.

IEEE STD 1471-2000 defines Enterprise Architecture as "the systems fundamental organisation, embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution"

One of the other leading Enterprise Architecture Frameworks from The Open Group (TOGAF 9) defines Enterprise Architecture as

- A formal description of a system, or a detailed plan of the system at the component level, to guide its implementation (source: ISO/IEC 42010:2007)
- The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time.

Other major definitions detail Enterprise Architecture is a set of principles, practices and processes, that defines the structure as well as operations of the enterprise and its systems for effective realisation of enterprise goals to enable an enterprise performance to be predictable, measurable and manageable
The key factor in the above definitions for enterprise architecture is the focus on principles, components and more importantly formal inter-relationships between components.

Much of the architecture we see today do not emphasize on formal relationships between participating components which is brings the main problem of ambiguity and error within various architectural layers. The problem domain around failed programmes and effort lost in extensive and often repeated testing lifecycles is primarily because of ambiguity in requirements (capture, analysis or engineering) and then ambiguity between architecture and requirements and finally the cascading effect of ambiguity between implementation and architecture. There is ambiguity because requirements are divorced from architecture and architecture is divorced from implementation. As architects we write a lot of documentation and create a lot of great diagrams.

However, how many of us have really proven that what we have written in terms of architecture is actually what is built finally? If proven, is the proof empirical or derived or formal?

While empirical or derived proofs (through various kinds of testing) are okay for simple projects with straight forward architectures, they do not water on large programmes and end up in extensive testing cycles which are often repeated and involve huge efforts and wastage of time.
As a result of ambiguity we end up with:

- Poor Alignment of IT to business goals and objectives
- High Cost in managing complexity
- High cost of testing
- Lack of transparency and control in delivery and change management issues in large programmes
- Poor re-use of key IT assets
- Lack of Business agility hindered by inefficient IT Architectures

So removing ambiguity by joining up things, moves us from "art to engineering" leading to the industrialisation of IT through efficient use of architecture methods. Testable Architectures described in the next chapter is a formalism that addresses governance issues of ambiguity through modelling and simulation for service oriented architectures (SOA)

A formal methodology for Service Orientation

Testable architectures are the foundation of removing this ambiguity. Testable Architecture enables the architecture of a system to be described unambiguously using Choreography Description Language (CDL) and BPMN2, such that it may be tested against requirements and is used to generate implementation artefacts for delivery thereby improving governance and control across large system integration programmes.
If we can deliver a solution that connects requirements to architecture and to implementation, we shall change the nature of complex systems delivery, reducing costs, mitigating delivery risks and improving time to market of key business functionality

Testable architecture methodology uses a unique combination of abstraction, modelling and simulation to the architecture definition process and the ordered interactions between participating components coupled with any constraints on their implementations and behaviour. Testable architecture is formal hence reduces defect injection across a programme lifecycle. Testable architecture is formally grounded and with strong type definition and has its foundations in "pi-calculus" which is a formal communication framework developed by Prof. Robin Milner – Professor Emeritus of Computer Science at the University of Cambridge and Turing award recipient: (**Miln80a, Miln93, Miln99)**

Key benefits of Testable architectures:

- Improved delivery assurance
- Reduced cost of implementation and testing
- Increased quality of overall solution
- Increased agility of overall solution

Testable Architecture is a unique method to unify both types modelling techniques in order to capture the several facets of the distributed communication systems and demonstrate the power of modelling to develop software artefacts of high quality. The development of distributed messaging systems is a complex activity with a large number of quality factors involved in defining success. Despite the fact that inductive modelling is scientifically thorough for analysing and building quality engineered systems, it brings additional cost into the development life cycle. Hence, a development process should be able to blend inductive and deductive modelling techniques, to adjust the equilibrium between cost (time resource) and quality. As a result, the field of software process simulation has received substantial attention over the last twenty years. The aims have been to better understand the software development process and to mitigate the problems that continue to occur in the software industry which require a process modelling framework.

When it comes to modelling the interaction and communication of Distributed System, Choreography Description Language (CDL) is one of the most efficient and robust tool. CDL forms part of Testable Architecture, hereafter TA, and is based on pi calculus which is a formal language to define the act of communicating.

TA abstracts the given set of constructs using pi-calculus and provides a language through a series of methods and logical sequences that are presented in a unified toolset. The latter facilitates the modelling and simulation of communication in concurrent systems. In order to achieve rigour in the power of modelling, TA exploits the capability of CDL as a framework to model global message flows and the subsequent impact of communication on local behaviours, which is defined by pi-calculus. Formal description in the global calculus, has a precise representation in the local calculus (**Carb06**). As a result, unlike other modelling

frameworks, TA is not limited to deductive and static modelling techniques, as it uses pi calculus based on non-deterministic models, that are well known within the academic world, but not yet of a common use within industry. In fact TA acts as a natural "glue" to blend the various modelling approaches providing a framework with the primary objective of removing the ambiguity within the modelling process

Conclusion

Governance frameworks address aspects of an SOA that need to be regulated to guarantee business-IT alignment and successful long-term operation. However, for SOA-specific regulation requirements beyond current IT Governance scope, current literature and industry efforts lack applicability in some aspects. In this paper, I have discussed particular aspects of service-oriented systems that exceed coverage by existing frameworks.

Two major areas were identified where future SOA Governance approaches to provide steering and control support in order to operate an SOA successfully: service lifecycle management addressing service-specific phases and stakeholder integration. Concerning service lifecycle management, the precise definition and the regulation of particular phases are the most important aspects. In particular, service granularity is an important detail whose impact will increase with growing adoption of service-oriented systems and service marketplaces.

Accurate service deployment regulations and usage of Testable architectures ensure consistent operation and well-maintained registries and cover tight integration of the service customer. A major challenge is to define and provide a reliable service change process. Using an adequate service lifecycle, service lifecycle management can be deployed as a powerful governance instrument addresses these challenges.

Existing respective governance models do not address all outlined aspects to an extent which relates to their importance in the operation of an SOA. Service lifecycle management and the field of open service marketplaces give rise to a number of crucial tasks to be regulated by governance that have not been integrated in according frameworks yet. Findings till date indicate a lack of standardization in advocacy of service orientation and governance methods and practices. In fact most of the current SOA frameworks are driven by organic needs rather than based upon the use of formal methods which give the much needed rigour, proof and repeatable success required to make SOA deliver upon its promise as an architecture paradigm

With development budgets getting tighter and the need for agility becoming more important, there is simply no need for architectural errors to still be present in the testing stage of IT projects. They're expensive and time consuming to fix and, crucial business requirements fall through the gaps. By bringing in a high level of testing rigour, measurement and formalism to SOA and the software development lifecycle, Testable architecture will deliver real returns for customers, reducing the cost of ongoing projects, and freeing up budget for further, revenue-generating initiatives

In conclusion, Testable Architecture' ensures that artefacts defined in each phase of the software development lifecycle (e.g. business requirements, architectural models, service designs, code, etc.) can be verified for conformance for truly realisable SOA Governance. For example, architectural models can be verified against requirements, service designs against architectural models and code against service designs. This guarantees that the deployed systems can be shown to implement the originating business requirements. Future work covers further investigation of proposed models around Testable architectures for SOA Governance and their validation concerning the requirements and challenges that an SOA really makes.

Reference

(Allen08)	Paul Allen. SOA Governance: Challenge or Opportunity? CBDI Journal, pages 20–31, April 2008. Retrieved July 20, 2008 from http://www.cbdiforum.com/secure/interact/2008-04/challenge_opportunity_br.php.
(TErl05)	Thomas Erl. Service-Oriented Architecture - Concepts, Technology, and Design. Prentice Hall Professional Technical Reference, Boston, 2005.
(Wool06)	Bobby Woolf. Introduction to SOA Governance. IBM Developerworks, June 2006. Retrieved July 22, 2007 from http://www.ibm.com/developerworks/library/ar-servgov/.
(Mar06)	Eric Marks and Michael Bell. SOA: A Planning and Implementation Guide for Business and Technology. John Wiley & Sons, Inc., New Jersey, USA, 2006
(ISO)	International Organization for Standardization (ISO). ISO 17799. http://www.17799central.com/.
(OGC)	Office of Governance Commerce (OGC). IT Infrastructure Library, 2007. http://www.itil.org.
(ITGI)	IT Governance Institute (ITGI). CobiT 4.1, 2007. http://www.itgi.org/cobit.
(AMCIS08)	Michael Niemann, Julian Eckert, Nicolas Repp, and Ralf Steinmetz. Towards a Generic Governance Model for Service-oriented Architectures. In Proceedings of the Fourteenth Americas Conference on Information Systems (AMCIS 2008), Toronto, ON, Canada, 2008.
(HBR04)	Peter Weill and Jeanne W. Ross. IT Governance - How Top Performers Manage IT Decision Rights for Superiour Results. Harvard Business School Press, Cambridge, MA, 2004.
(Carb06)	Carbone M, Honda K, Yoshida N, Milner R, Brown G, Ross-Talbot S , "A Theoretical Basis of Communication-Centred concurrent Programming", PhD thesis, Imperial College, London UK, 2006
(Miln80a)	Milner R, "A Calculus of Communicating Systems", Lecture Notes in Computer Science, volume 92, Springer-Verlag, 1980
(Miln93)	Milner R, "The Polyadic pi-Calculus: A Tutorial", L. Hamer, W. Brauer and H. Schwichtenberg, editors, Logic and Algebra of Specification, Springer-Verlag, 1993
(Miln99)	Milner R, "Communicating and Mobile Systems", Cambridge Press, June 1999