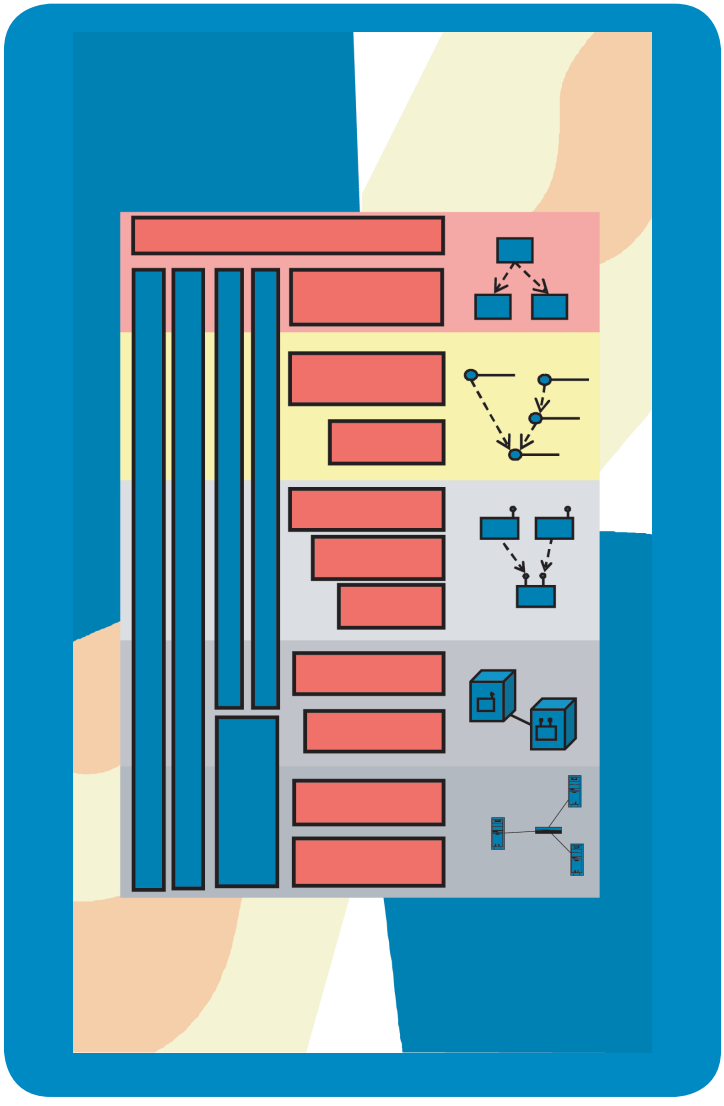
ISSN 1745–1884

CBDI Journal

MARCH 2007



Independent Guidance *for*
Service Architecture and Engineering

Business Change Matters

This month I return to the question of how, when and why to involve the business in SOA. This is a question that surfaces repeatedly in our work and I am prompted yet again that this is far from straightforward issue to resolve.

The basic model that I base our advice upon is that in the early stages of SOA adoption it is highly inadvisable to raise business expectations because there is a significant amount of learning that needs to happen – and that's best done in private, behind closed doors in the IT environment. Creating basic services, breaking up monolithic architectures, improving and rationalizing application integration contracts, creating separation based architectures in project development all provide benefit, but in business terms they are indirect.

What is useful and important is to commence the dialogue with business management as early as possible around the question of what agility the business needs. Of course it's pointless asking business people "what agility do you need?" Generally business people don't know. Even if they are aware of impending change, of M&A based actions for example, they are often unable to communicate this because of business confidentiality requirements.

But in most businesses change is happening constantly, and one approach is simply to analyze change and project requests to classify types of change. I did this, admittedly in a less than formal fashion, a couple of years ago with one of our customers. We assessed types of change on the basis of peoples' experience and developed a simple model that illustrated classes of change and relative incidence. See Table 1.

IT architects need to be providing positive contribution to solving the business problem in a manner that offers a range of choices that puts the responsibility on the business people to choose the appropriate cost, time and adaptability profile.

It's not rocket science to figure out patterns of change (and the need for agility) and to see patterns relate to business maturity or vertical sector. Relatively young businesses will be constantly expanding channels to market. Mature industries will be focused strongly on process improvement.

The interesting question is not just how to establish architecture that has inherently agile characteristics, rather how to define the architecture in a manner that reflects the change characteristics of the specific business and industry sector. It would seem entirely reasonable that the architect should be able to answer the question – how long will it take to make certain types of change? Perhaps each layer of the architecture should be designed specifically to meet a change SLA.

In general terms we may assess that core business and capability services will be relatively slow moving, and that change cycles of 6 months are adequate. In contrast process services are generally regarded as fast moving and we might assess a change cycle time of weeks to be required. However my experience is that this approach is still way too generalized. The layering approach gives a basic model, but what would be really useful is for a set of recommended patterns that can be used dependent on the situation as it's perceived specific to the particular (or cluster of) service. They may also have widely different cost and usability considerations.

In our kitbag we should have a set of patterns that have been developed over time and are well understood, not just in the architect and developer sense, but in terms of life cycle impact on time to market and cost. With this approach, frankly, we should be operating more like an architect in other professions. If you commission a custom build of say a house, the architect will want to know your requirements in both functional and non functional terms. In my own experience the architect will

Business Change	System Change Requests
Differences between Internal/external organization	Visualize differently
Customer required data	Process flow change
Legal differences across geography/industry	New integration requirements
Niche/point solutions	Rules change
Suboptimal original design	Data additions

Table 1: Examples of Classes of Change

come back with positive contribution to the overall design and provide a range of design options – which of course all have varying cost, time, quality and quite probably agility impacts.

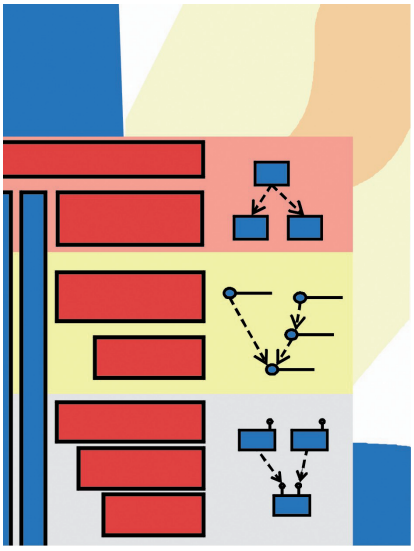
In the same way IT architects need to be providing the same level of service to business customers, providing positive contribution to solving the business problem, but in a manner that offers a range of choices that puts the responsibility on the business people to choose between a solution that will last three months and one that will be able to adapt to changing business circumstances with a well understood cost and time profile.

In so many businesses I work with I find that there is still a huge gulf between the business and IT people, whether they are internal or external providers. The business people are locked into a cycle where they blame "the IT people" for lack of business response while taking no responsibility for helping the systems providers understand the requirements in a manner that they can respond in an appropriate manner.

To crack this problem needs a level of engagement that expands the footprint of the "requirements specification" to inform architectural work. This needs to happen at two levels – in a fairly general manner in order to develop and customize the architectural framework, and at a specific business process level to inform project delivery.

In this month's CBDI Journal I report back on a little exercise that I conducted with some 45 architects at a UK architects conference organized by Microsoft. At this event I ran a little survey to find out where folk are in their SOA maturity development with some interesting results. Also this month we continue our publication of the CBDI SAE™SOA Reference Framework with the Architecture Component. This is highly relevant to the discussion in this editorial – suggesting the architecture is an intersection of separate views with standards, patterns, techniques, deliverables, models and practices that provide the basis for specialization and customization by individual enterprises.

David Sprott, Everware-CBDI, March 2007



SOA Maturity Assessment Survey

Recap and results of the SOA Roadmap Workshop run at Microsoft's Architect Insight conference, March 2007

In early March I ran a 75 minute workshop at the annual Architect Insight conference run by Microsoft in the UK. I introduced the concepts of SOA Adoption Roadmap and worked with the delegates to develop a high level assessment of their current maturity. This article provides a brief introduction to SOA Roadmap and Maturity concepts and documents the results from the workshop assessment session.

By David Sprott

Introduction

This was the second time Microsoft has held their annual architect's "get together" in Wales. For those readers not familiar with European geography, Wales is the small bit sticking out on the left hand side of the United Kingdom – and it always seems to be raining there.

This year Microsoft introduced a number of workshop sessions to complement the tutorials and these were specifically planned to produce some concrete output. In my workshop session I introduced the concepts of SOA Roadmap and Maturity Models and then I walked through a relatively high level Capability Maturity Model, explaining the primary capabilities required at increasing levels of maturity by stream with the delegates commenting on their current status and concurrently recording their own assessment of their maturity. At the end of the session I collected the individual assessments and committed to provide an analysis of the results, which I hope provides an interesting benchmark.

This report therefore summarizes the introductory remarks and presents the aggregate results of the delegates' assessments.

Complex Change Management Problem

We are all constantly involved in many forms of change. We move house, we acquire new technologies and our roles change as our employers' fortunes wax and wane. In a corporate sense change is generally managed to some, but varying degrees. Some enterprises have developed the management of business process change to an art form and specialism using formal methodologies such as Six Sigma[1].

Clearly IT organizations are also accustomed to making change, particularly in response to new technology. In some areas such as IT resource management

Pre SOA	SOA
Project driven	Business/IT convergence
Variable approaches and processes	Contract based services
Point to point integration	High levels of reuse of coarser grained functionality
Low levels of reuse at any level	Manufacturing and assembly environment
Loose coupled technology	Architecture and policy driven
Tight coupled applications	Repeatable processes
Low level of business alignment	Strong governance to maintain architectural integrity

Table 1: A Complex Change Management Problem

orchestration and change of the process is increasingly sophisticated. This is not generally the case throughout all IT functions.

Against that somewhat mixed background SOA adoption represents a complex change management problem. It is not a technology led change rather it is a change in architectural approach that will, over time, have profound impact on many IT and business functions including the way projects are initiated, funded, scoped and governed. It will gradually require the relationship between business and IT departments, providers and suppliers to change. It spans many dimensions and boundaries involving many parts of the organization as summarized in Table 1.

Maturity Models

The idea of maturity Models is not new – they have been used effectively in many different domains. In the IT space the best known maturity model is the SEI's CMMI[2] a widely used framework for measuring and managing IT process improvement in development, service delivery and acquisition.

There have been numerous maturity models developed over the past 18 months to address the SOA space. I reported on several of these in my December 2005 report on the SOA Maturity Model[3]. Most of these models have some superficial relationship to the SEI CMMI framework and are focused on a particular form of maturity, for example the maturity of the service concept as it relates to ESB technology; or the maturity of the integration task.

CBDI developed its SOA Maturity Model in 2003[4] based on earlier research work I and other colleagues carried out in the 1980s that focused specifically on complex, multi-dimensional change problems. The CBDI model illustrated in Figure 1 is distinctive insofar as it focuses specifically on the maturity of the enterprise adoption of SOA.[5] The Maturity Levels identify primary outcomes that characterize the organizational capability.

- Early Learning – the organization is experimenting with SOA. Activity is likely to be characterized as Pilot Projects or Proof of Concept (PoC) projects.
- Applied – SOA is employed within conventional projects to deliver improved structure.
- Integrated – SOA is used to deliver integration between projects and or application silos.
- Enterprise – SOA is optimized at the enterprise level.
- Ecosystem – The SOA is inherently federated supporting virtual business.

Whilst these maturity levels are neither standardized or fixed in stone, we have found these are a useful starting point for most enterprises.

In our early work in this area we recognized that we need efficient ways to break up the problem to facilitate understanding, communications, measurement and management. We introduced the concept of **streams** – broad topic areas that break down the overall task in an organizationally neutral manner, providing the scope for cross organization communities of interest to collaborate and reach an appropriate level of consensus. The streams shown in Figure 2 are provided as an organizing pattern that we have found useful. Many enterprises have altered these slightly to suit their needs, but in general they seem to be widely applicable.

- Management – the focal point for management capabilities spanning visioning, strategy, funding, chartering, governance, measurement and management of the SOA adoption process.

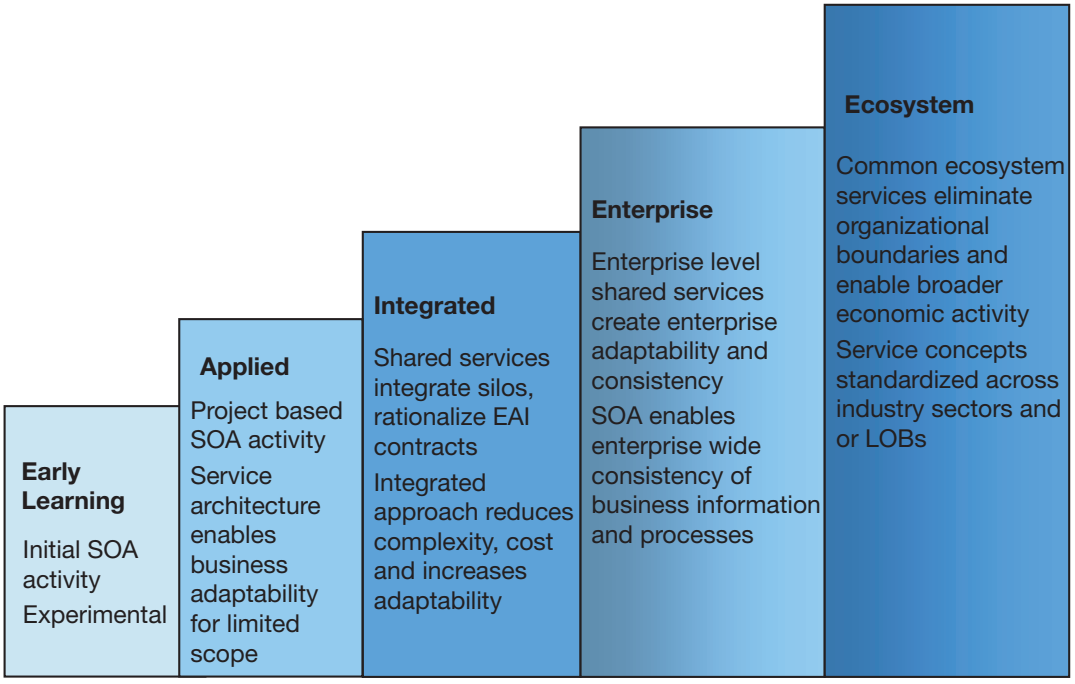


Figure 1: CBDI Capability Maturity Model

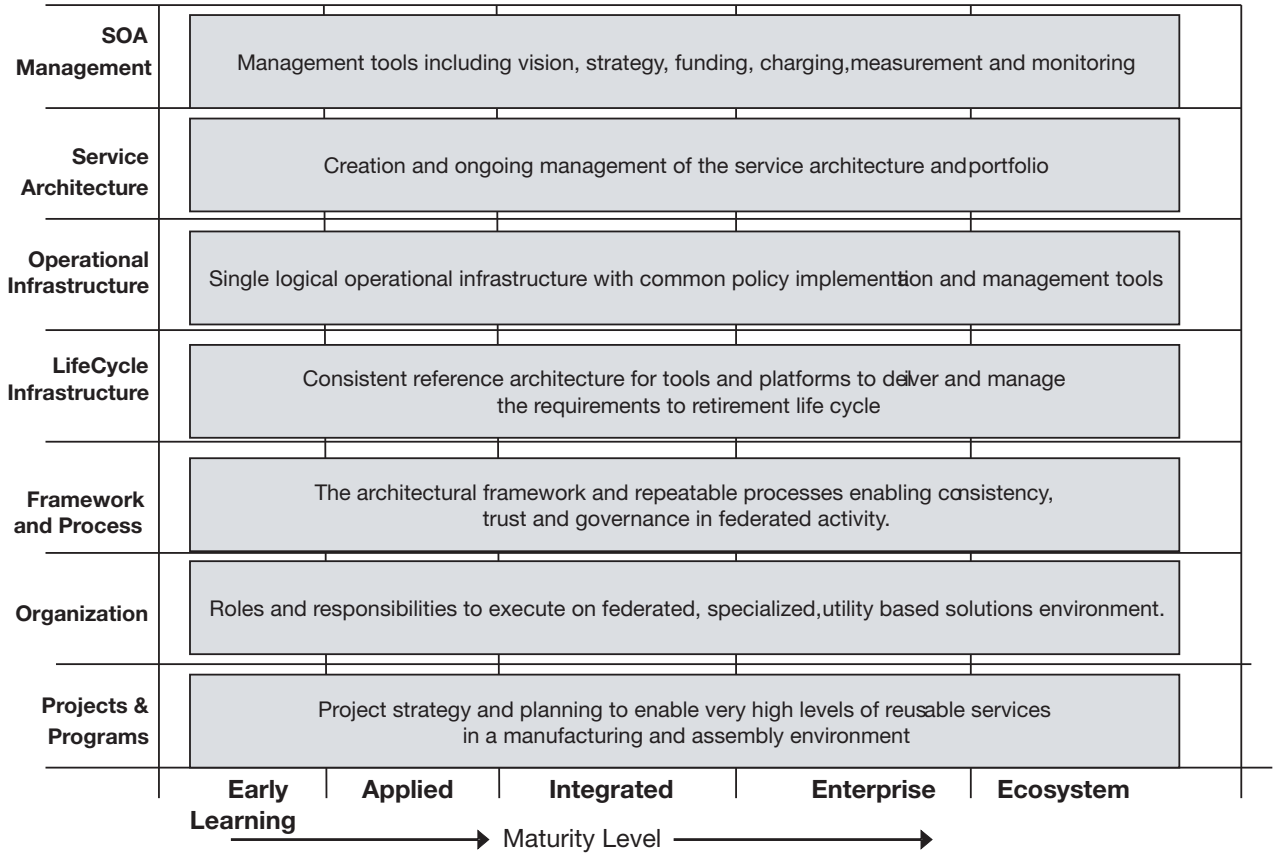


Figure 2: Roadmap Streams

Capability Area	Capability	Maturity Level
Service Governance	Record of Services in Use	Applied
	Monitoring of Service Usage	Applied
	Control over Service Usage	Integrated
	Policy based control over Service Planning and Provisioning	Enterprise

Table 2: Example of Capability/Maturity

- Service Architecture – the creation and maintenance of the service and associated architecture. Note here the architecture is the instance (or enterprise specific) architecture **not** the reference architecture framework that defines the meta objects in the service architecture.
- Operational Infrastructure – the architecture and capabilities to support the run time service environment.
- Life Cycle Infrastructure – the architecture and capabilities to support the entire life cycle of service(s) states, spanning planned to retired and archived.
- Framework and Process – the reference architecture framework detailing the layering, policies, patterns, models, deliverables etc. plus the reference process that facilitates repeatability, governance and quality.
- Organization – the roles and responsibilities required to establish, operate and maintain a service oriented business.
- Projects & Programs – the project capabilities (classes and profiles) necessary to plan, provision, implement and assemble services.

Capability Planning

The intersection between maturity levels and streams is capability – the competence, ability and capacity to perform a specific function, process or task. Basing change management on capability provides a systematic and managed approach to introducing change. An example of capabilities within one capability area at several levels of maturity is shown in Table 2.

The Assessment Exercise

Having discussed the elements of a capability plan we then walked through a high level roadmap populated with selected capabilities shown as Figure 3. Note the capabilities were selected because they are important and illustrate the capabilities at varying levels, but there are of course many other capabilities in a practical plan.

A summary of the results of the assessment exercise are shown in Table 3.

Survey Conclusions

The highest incidence of capability is at the Applied Maturity Level with average maturity at this level at medium maturity (1.8) as opposed to low or high.

The second highest incidence of capability is at the Early Learning level.

This shows a numeric majority of the sample are in the very early stages of SOA and are not yet enabling shared services to any significant extent.

Most organizations are in the early stages and a small minority is making progress to more advanced levels

At both the Early Learning and Applied levels delegates aggregate assessments were just below the mid point.

While the average capability scores are in a fairly narrow range, there is blue sky between architecture (1.3) and all the other streams at the Applied level. This illustrates the limitations of SOA applied to individual projects. While SOA may be effective, particularly for larger projects, we might surmise (this was an audience of Microsoft customers) that the average project size represented may not have been very large.

At the Integrated level there is clear evidence of maturing Operational Infrastructures with an average capability score of 2.4 reflecting a common ESB (or equivalent) environment.

	Early Learn		Applied		Integrated		Enterprise		Ecosystem	
	Average	Sample	Average	Sample	Average	Sample	Average	Sample	Average	Sample
	Maturity		Maturity		Maturity		Maturity		Maturity	
SOA Management	1.8	8	2.1	12	1.8	6	2.5	2	3.0	1
Service Architecture	2.0	7	1.3	2	1.8	6	1.8	5	1.3	3
Operational Infrastructure	1.6	7	1.8	9	2.4	6	2.0	5	1.3	4
Life Cycle Infrastructure	2.0	8	2.0	13	1.4	4	1.3	3		0
Frameworks & Programs	2.1	8	1.7	10	1.9	7	1.7	6	1.3	4
Organization	1.7	9	1.7	12	2.1	7	2.3	3		0
Projects & Programs	2.0	10	2.0	12	1.3	6	3.0	1	3.0	1
Overall	**1.9**	**57**	**2.1**	**70**	**1.8**	**42**	**1.9**	**25**	**1.5**	**13**
Distribution %		27.5		33.8		20.3		12.1		6.3

Table 3: Aggregate Assessment by Stream and Maturity Level

Notes:

On a show of hands the delegates comprised a mix of end user and vendor enterprises, with a significant majority (perhaps 75%) of end users.

Delegates were asked to assess their capability maturity where non zero. It is to be expected that organizations have capabilities at multiple levels.

Sample is the count of the number of assessments for capability/stream.

Average Maturity is on a scale of 1 to 3, where 1 is low and 3 is high. Nil capability was not recorded.

Distribution is the count of recorded capability assessments for a maturity level. Also shown graphically in Figure 4.

	Maturity Level				
Stream	**Early Learning**	**Applied**	**Integrated**	**Enterprise**	**Ecosystem**
SOA Management	Funding for pilot/PoC projects	Services are managed as an IT architecture concept	Funding systems facilitate provisioning of shared services	Services are managed as business assets.	Services facilitate inter business collaborations
Service Architecture	Architecture is fragmentary & experimental	Project architectures are service oriented	There is a standard for rich service specification	The Enterprise has a Service Portfolio Plan	There are agreed business process and data architectures for business collaborations
Operational Infrastructure	ESB pilot or PoC	Project ESB	Common ESB framework	Common framework for enterprise service management and security	Services are managed as federated resources
Life Cycle Infrastructure	Services are not managed assets	Services are project level deliverables	Enterprise level registry and repository provides consistent life cycle governance of the run time service asset	Enterprise registry and repository provides design AND runtime service asset life cycle governance and asset dependency horizon analysis	Ecosystem registries provide governance over collaborative business processes
Framework and Process	Frameworks and practices extended in ad hoc manner	Project specific architecture frameworks	Common SOA reference architecture and process (defined concepts, principles, policies and patterns)	Convergence of business and IT practices around service concept	Collaboration reference architecture (defined points of collaboration)
Organization	IT architects sponsor SOA	SOA is a project level responsibility	There is single point of accountability for integration	Services are owned by the business	Services defined and managed on inter business collaborative basis (vertical, supply chain etc)
Projects & Programs	Pilot and PoC projects	Service delivery and usage integrated into LoB projects	Specialization of Service provisioning, implementation and assembly projects	Service product management	Collaborating parties act as provider and consumer

Figure 3: Selected Capabilities by Stream

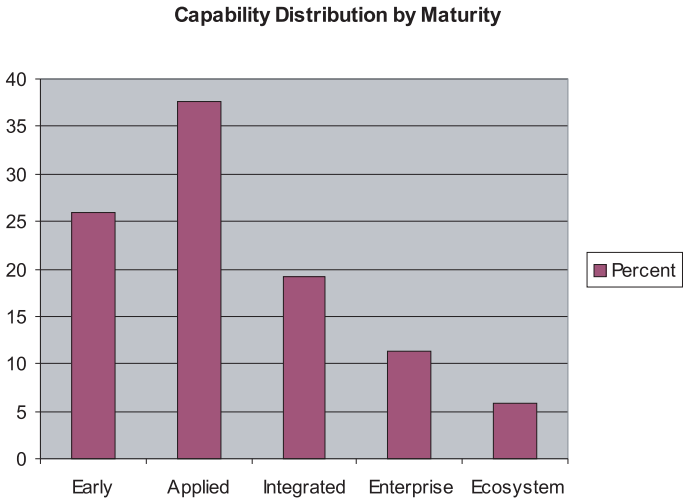


Figure 4: Capability Distribution by Maturity Level

However at the same Integrated maturity level the Life Cycle Infrastructure (1.4) and projects and programs (1.3) are illustrating that change at this level may only be skin deep. While a show of hands might tell us lots of enterprises have installed registries or catalogs, my question (see Figure 3) asked specifically about "consistent life cycle governance of the run time service asset ".

Clearly it is a small minority of the delegates that have significant Enterprise level capability.

At the enterprise level there were clearly one or two organizations that feel they have made considerable progress with project and program organization, although interestingly not architecture. One delegate only indicated his team had created a Service Portfolio Plan. And just two delegates indicated that services are managed as business assets.

The average score for Life Cycle Infrastructure (1.3) also reflects the difficult nature of my question – suggesting that at this level "Enterprise registry and repository provides design AND runtime service asset life cycle governance and asset dependency horizon analysis".

Finally at the Ecosystem level clearly it is a very small minority that have made progress in this area.

However I do find it interesting that there was some evidence of reference architecture governing collaborations in the frameworks and process average score (1.3/4).

Also that again evidence of management of federated resources albeit at a low level of maturity (1.3/4) against operational infrastructure.

Summary Thoughts

I must emphasize that this exercise was carried out in just 75 minutes and is not intended to provide more than a simple snapshot of what organizations are doing. Our methodology and advice recommends a much more comprehensive exercise is needed to really understand current status.

Notwithstanding that caveat, the analysis does confirm anecdotal evidence and random samples that suggest the structured approach to SOA is still very much in its infancy. Most organizations are in the early stages and a small minority is making progress to more advanced levels. The vision of federated service architectures remains for almost everyone, just that, a vision.

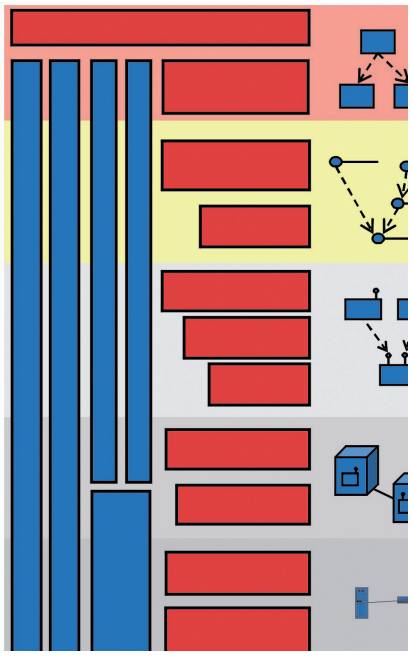
Acknowledgements

Thank you to delegates at the Microsoft Architect Insight conference who participated in this exercise. (A copy of the report has been sent to them, with thanks)

Thank you to Microsoft UK who encouraged and supported the exercise

Notes

1. http://en.wikipedia.org/wiki/Six_Sigma
2. http://www.sei.cmu.edu/cmmi/
3. The SOA Maturity Model. http://www.cbdiforum.com/secure/interact/2005-12/The_SOA_Maturity_Model.php
4. SOA Maturity Model in 2003. http://www.cbdiforum.com/secure/interact/2003-05/maturity.php3
5. Regular readers of CBDI research will note that in the 2007 version of the SOA Maturity Model the maturity levels have been modified. There is one new level – Applied. We have introduced this primarily to reflect widespread practice in which projects commonly adopt SOA at a project level. Whilst this is generally not recommended, and may sub optimize the SOA objectives, it is common practice and therefore necessary to record this in assessment exercises.
   The Integration Level has been renamed Integrated – reflecting the primary characteristic that services are Integrated to some extent, whilst not yet at the enterprise level. The Reengineering Level has been renamed Enterprise, recognizing a primary objective of SOA to optimize at that scope. Cultural Integration renamed Ecosystem recognizing that endpoint maturity will be largely federated.



The Architecture Component of the SAE™ Reference Framework for SOA

By John Butler

There is no "one size fits all" methodology, ours or anyone else's, and so best practice in method development calls for incorporation of a framework of artifacts, tools and techniques that can be tailored to the nuances of each organization that wants to implement the methodology. However, most popular methods don't tend to focus on the needs of service lifecycle instead covering a broad but typically less focused method landscape. CBDI's SAE™ Reference Framework is built to remedy that problem by highlighting aspects of methodology such as process, techniques and artifacts needed to embrace SOA concepts in a structured manner. This article provides an introduction to the Architecture component of the Reference Framework and the rationale that went into its creation.

Introduction

The SAE Reference Framework is designed to provide a comprehensive framework of all the components necessary to support the migration to and subsequent upkeep of a service oriented enterprise. It addresses the three primary perspectives necessary for capturing methodology – Organization, Process, and Architecture of the Artifacts. These perspectives are built upon a firm "foundation" Model that provides the language and principles of SOA. By tailoring these aspects to the development organization's needs, a clear target mode of operation can be established to drive the SOA adoption cycle.

The RF Model Component – Language of the Framework

In order build up a Reference Framework that addresses Organization, Process and Architecture a firm foundation of language and principles must be established to ensure that everyone is on the same page. The Model component plays this role within the SAE™ Reference Framework and comprises four main parts: SOA Metamodel, SOA Principles, Glossary, and Service Lifecycle.

The details of these are outside the scope of this article but suffice it to say that these parts define the underlying SOA concepts and their interrelationships: these form the language that is used to describe the rest of the Reference Framework. In particular, the Architecture component makes extensive use of the SOA Metamodel as the language used to describe the various views and other elements introduced below. CBDI first published this metamodel in 2006[1] and continues to refine it based on feedback from CBDI members and standards efforts that are underway[2].

Likewise, the SOA Principles established in the Model component of the Reference Framework provide the guide for the layout of the Architecture component, both Views and the Best Practices captured there in.

The Reference Framework Triad – Organization, Process and Architecture

The other three main components of the Reference Framework are Organization, Process, and Architecture. These three parts form a triad that describe key aspects of any methodology framework. The Process component of the Reference Framework published in the February 2007 CBDI Journal[3] describes a structure of business processes or activities that a service provisioning organization should follow in order to successfully analyze, plan, design, provision, and run services. The Organization component describes the roles and responsibilities, project profiles, and funding models recommended in order to successfully support the service lifecycle. Finally, the Architecture component, the topic of this article, provides the detailed description of the various views, models and other elements used and created during the execution of the method and how they relate to one another.

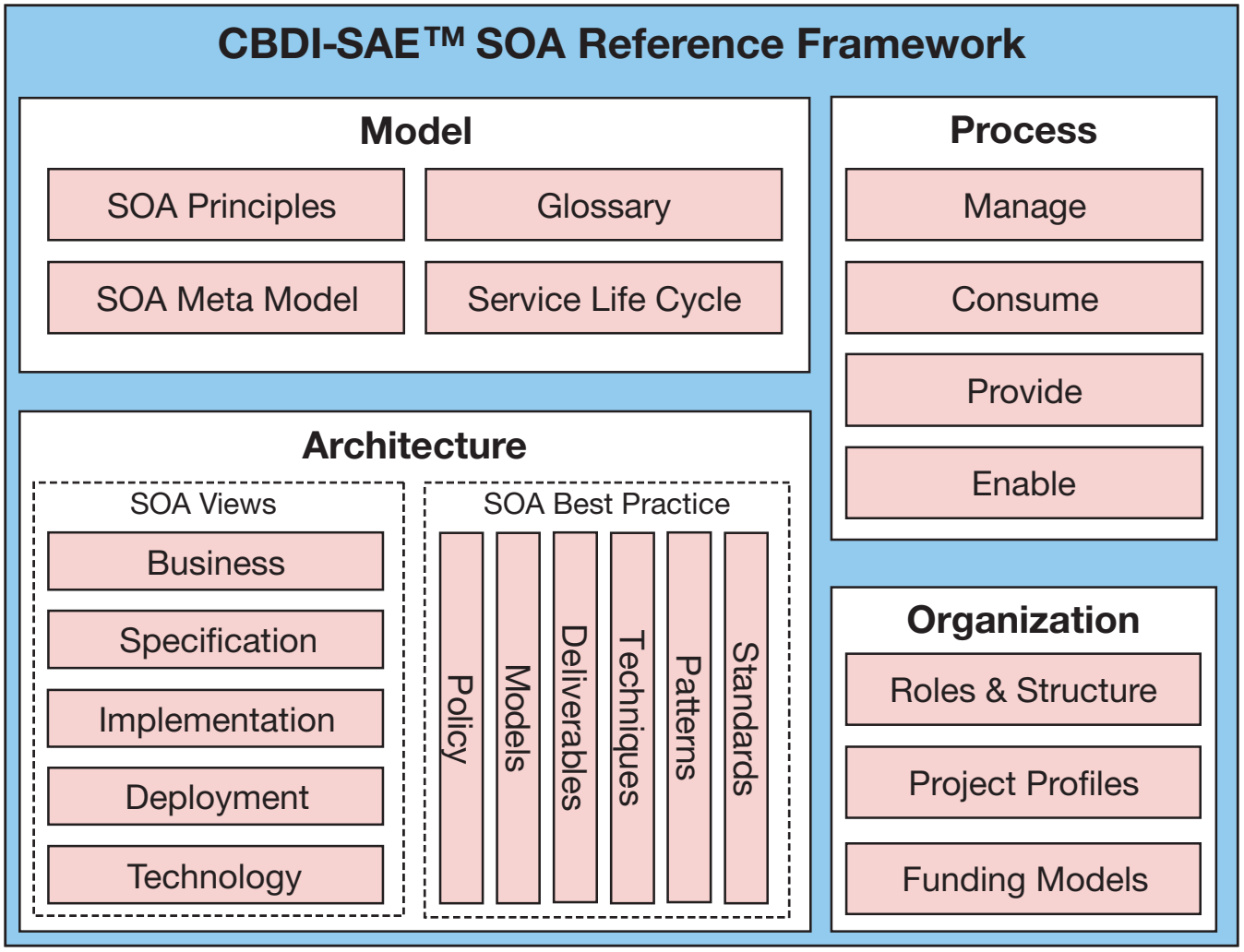


Figure 1: CBDI-SAE™ SOA Reference Framework

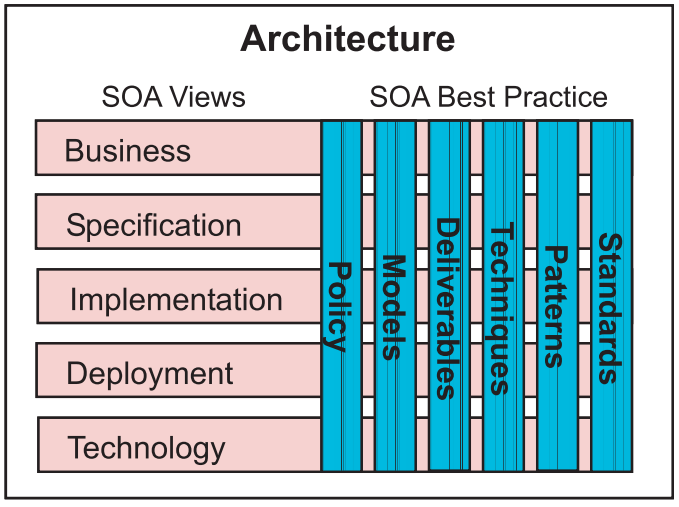


Figure 2: Architecture Component of the SAE Reference Framework

Views – "Slices" of the Service Oriented Enterprise

One of the defining characteristics of any methodology is the structure used to capture the relevant aspects or perspectives of a system, whatever system that may be – business, information system, hardware, or what have you. The CBDI-SAE™ Reference Framework includes five views – Business, Specification, Implementation, Deployment, and Technology. These views comprise a consistent level of abstraction for deliverable artifacts that relate to distinct set of stakeholders. This provides an effective mechanism for grouping related SOA best practices based on a particular part of the enterprise under study. Each View defines and clusters together the standards, patterns, techniques, deliverables, models and policies that apply to appropriate View as illustrated in Figure 2.

Table 1 and Figure 3 provide a first level of detail on each of the five Views highlighting:

- The primary stakeholder roles involved in each level of abstraction
- The mapping to layers commonly used in enterprise architecture.
- The purpose of each View.
- Sample artifacts
- Key perspectives of each layer – the essence of the methodology – showing how the service architecture manages the relationship between conceptual, logical and physical perspectives.

View	Purpose	Primary Role(s)	Enterprise "Layer"
Business	To understand and analyze business needs and how the business operates in terms of goals and objectives, organizational structure, processes, information, etc.	Business Architect	Business
Specification	To plan and specify software services from a platform independent perspective. It provides a means of thinking in depth about logical services and their interrelationships.	Service Architect	Software
Implementation	To package services into automation units, identify dependencies between the automation units, and to determine the implementation constraints that will govern the internal design and deployment of these units.	Service Architect, Software Designer	Software
Deployment	To explore alternative and finally capture deployment choices for run time services. To map implementation view services to deployment units and to construct an optimum configuration on the computing infrastructure.	Infrastructure Architect, Operations Mgt	Software/ Infrastructure
Technology	To ensure technologies are in place to enable the service lifecycle at all levels – from planning through specification, design and execution to retirement.	Infrastructure Architect, Operations Mgt	Infrastructure

Table 1: SOA View Descriptions

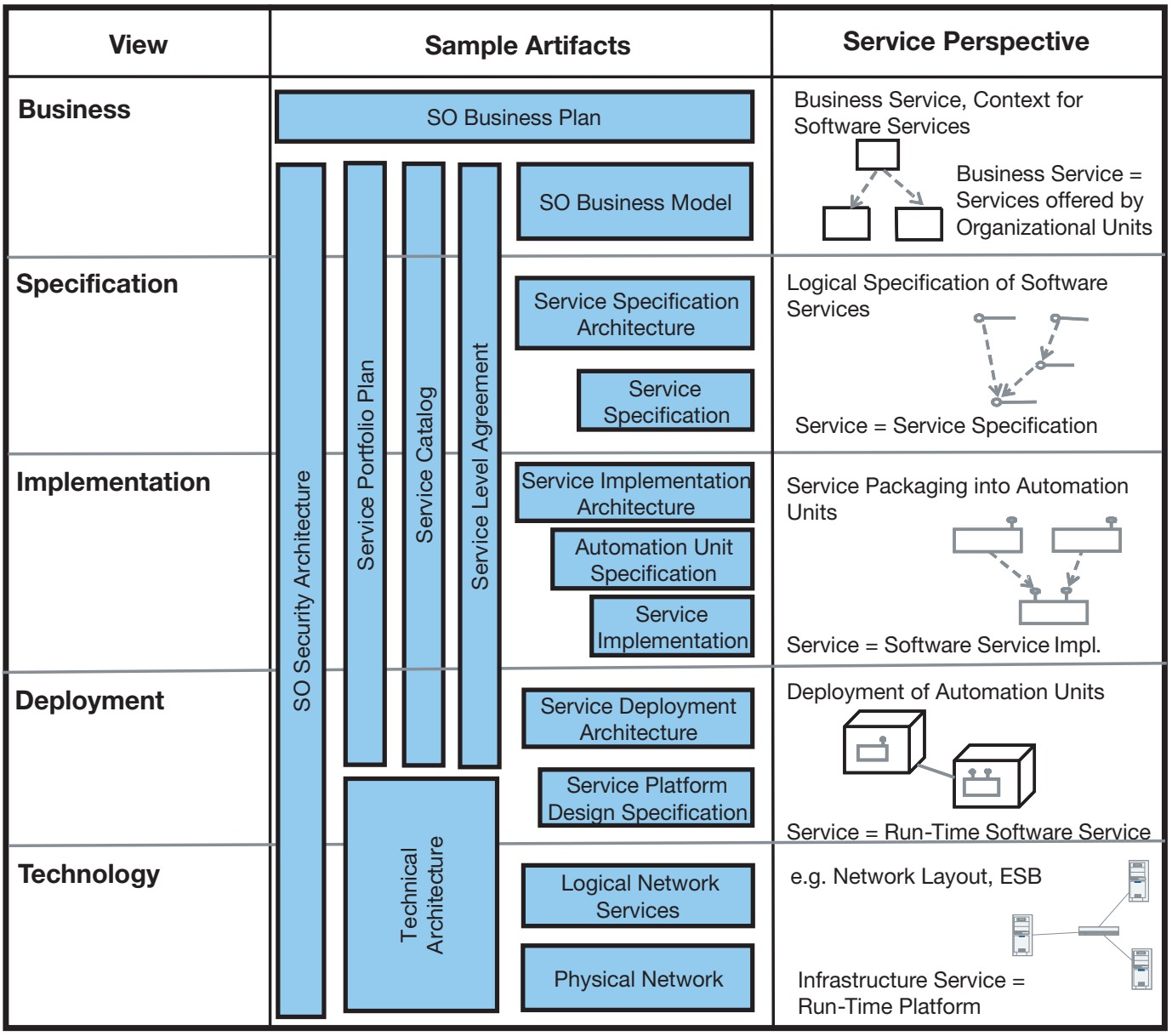


Figure 3: Sample Artifacts and Service Perspectives by View of the SAE Reference Framework Architecture Component

Note: some architecture frameworks break data out as a separate layer however, the Reference Framework captures this within a number of artifacts that reside in the Business, Specification and Implementation views.)

Note Figure 3 depicts only key artifacts of the RF Architecture component by Views. It is not intended to be a complete picture of all the artifacts that would be involved in developing a SOA or a software solution based on services; for example there are many existing models (such as logical data models and business process models) that will also come into play. The February journal article gives a more complete picture of the deliverables involved. In addition we have concentrated here on specifically service oriented artifacts.

The Business View

To fully understand the requirements of software systems and the services that they comprise, we need to understand the business context within which they operate. The Business View provides this context. Further, analysis of business objectives and processes from a "services perspective" often provides a significant return on investment to the business in and of itself for many of the same reasons a service perspective improves

Key Artifact	Focus	Typical Format
Ecosystem/Business Context Model (Part of SO Business Model)	Products or Services offered by a Business or Organizational Unit and the use of those products or services by their customers and suppliers.	BPMN diagrams, UML models including structure diagrams (e.g., package, class, component), behavior diagrams (e.g., Activity or Interaction Diagrams) other proprietary formats
Business Goals (Part of SO Business Model)	High-level goals of the business and the sub-goals they comprise.	UML Object Diagram, other proprietary formats such as Hierarchy Diagram
Event Response (Part of SO Business Model)	Major business events and the organization's response to them.	BPMN Diagrams, UML State or Activity Diagrams, other proprietary formats
Business Process (Part of SO Business Model)	Business processes that realize the services offered by the business.	BPMN Diagrams, UML Activity Diagrams, other proprietary formats
Business Rules	A statement that constrains how the business operates.	A textual table. More formal rule models use UML Class Diagrams with Constraints (in text or in OCL (Object Constraint Language)) or other proprietary formats
Business Type Model (Part of SO Business Model)	High-level information entities that are important at a business level.	UML Class Diagrams, ERDs, other proprietary formats
Organizational Structure (Often part of SO Business Model)	Organizational units and roles therein that comprise a business or enterprise.	Organizational Charts, UML Object Diagrams, other proprietary formats
Business Case for SOA	Justification for migrating to SOA. Key influence over SOA approach and architecture policy. E.g forecast cost and cycle time of delivery and adaptation by class of component and service	Textual documents and spreadsheets
SO Business Improvement Plan	Plan for improving business operations by incorporating services Key driver of architecture decisions that enable agility E.g forecast change cycle time for classes of components and services	Textual documents, project schedules, and spreadsheets
Business Solution Requirements	Solution requirements from a business perspective	Textual documents and requirements models
SO Business Plan	Overall plan for moving the business forward including SO perspectives	Composite artifact including SO Business Models, Business Case for SOA, and Business Solution Requirements
SO Security Policies (part of SO Security Architecture)	Detailed business rules and policies concerning security	Textual document(s)

Table 2 – Key Business View Artifacts

software. It decouples the "what" from the "how" allowing flexibility in the implementation in terms of business processes, whether internal or outsourced. Goals and objectives of the business can be more easily connected with the services the organization provides.

The Business View includes a number of key artifacts and models used to capture and analyze important aspects of the business. These artifacts and models are captured in Table 2.

The Business View also includes other best practices such as policies, patterns, and techniques that provide guidance as to how to capture knowledge about the business. Table 8 includes sample best practices in these areas.

This list of artifacts and models may seem daunting to the neophyte modeler/architect but remember that not all are strictly necessarily. Each project team that uses the Reference Framework will tailor it to their needs using or ignoring artifacts and models as they see fit in order to analyze and address the concerns that they find important. The key is to know how and why to use each one – its pros and cons.

For models the question of notation or "language" comes into play. While business modelers have not found the same level of convergence in terms of modeling language as software modelers have with UML™, there are still aspects that are generally agreed upon such as Organization, Business Process, Policy, Business Objective, Business Rule, and Business Entity. Everware-CBDI has included these salient concepts in our SOA Metamodel[4] and standards from OMG such as Business Process Modeling Notation (BPMN), Semantics of Business Vocabulary and Rules (SBVR) and (hopefully) soon to be approved Business Process Definition Metamodel (BPDM) are a big step in the right direction. Often, multiple languages are used to capture business architectures – swimlane diagrams for business processes, entity relationship diagrams (ERD) for business information models, org charts for organizational structure, and various proprietary notations from tool vendors. UML is increasingly being used to capture business models though some believe it to be too complex for business users to understand. Again, selection of the appropriate language and tools is part of the tailoring process.

Specification View

The Specification View comprises the artifacts and models required by architects to specify the functional and non-functional requirements of *software* solutions and services as well as the architectural dependencies between them. This view is meant to be independent of any particular platform such as an application server, operating system or even Enterprise Service Bus (ESB). The idea is to capture how services behave, allow for refactoring of that behaviour into appropriate "chunks" in order to optimize for reuse and other characteristics that are independent of any particular technology. That said, it may very well be that the choice of deployment platform has already been made and that the services will be required to be implemented on that platform. However, technology churn takes place on different cycles than business requirements and so providing a mechanism for separating these concerns is critical to maintainability of the service architecture.

The primary diagrams of the Specification View is the Service Dependency diagram (part of the Service Specification Architecture) that shows the layers of the Service Architecture, the services in each layer and the dependencies between them (See Figure 4). As shown in the figure, service domains can also be shown in this view.

Though it is certainly one of the more useful, the Service Dependency diagram is not the only view the Service Specification Architecture might contain. As with any system model, additional diagrams that show other characteristics can be captured. Often, the behavioral aspects, such as the actual messages being passed between services in order to realize a request, are very useful in allocating responsibilities to the various services in the architecture. Such diagrams often take the form of UML interaction diagrams (typically sequence or communication diagrams).

The actual diagrams captured will depend on the needs of the architects creating or specifying the services and other stakeholders that will use them. Some organizations capture very detailed structural and behavioral views at the specification level that are used to drive the implementation and deployment design processes down the road. Other organizations only use the dependency diagram for high level organization and portfolio planning.

Typical artifacts and models that are captured as part of the Specification View are described in Table 3 below.

Now the question is how we relate the elements captured in the Specification View back to the Business Model. As stated above, the Business View provides the context and requirements for solutions built using services captured in the Specification View. In order to convince ourselves that each requirement from the Business View has been adequately addressed we need to capture the traceability from elements in the Specification View back to the elements in the Business View.

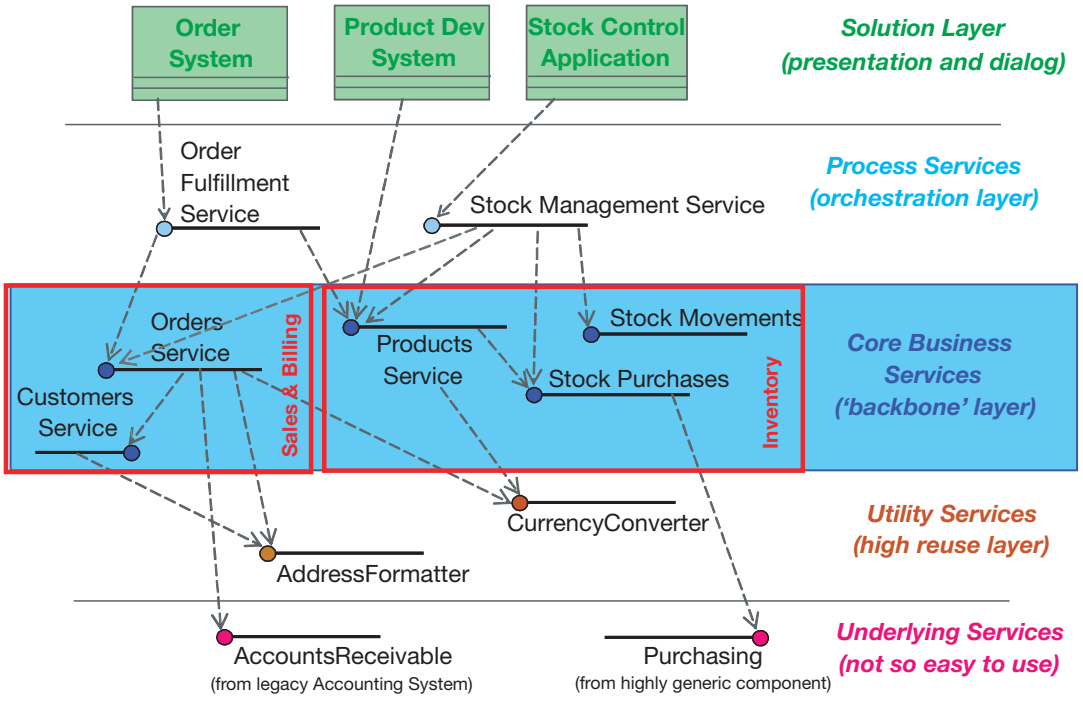


Figure 4: Service Specification Architecture – Layering and Dependency

Key Artifact	Focus	Typical Format
Service Specification Architecture	Complete logical model of the software services and their relationships to solutions, legacy applications and other 3rd party applications.	UML Model including structural diagrams (e.g., package, class, component) and behavioral diagrams (e.g., communication, sequence, state).
Service Dependency Diagram (part of Service Specification Model)	Architectural layers at a logical level and the structural relationships between the services in these layers.	UML Package and Class Diagrams
Service Orchestration Diagram (part of Service Specification Architecture)	Interactions between services that collaborate to provide services at a higher level.	UML Interaction Diagrams (Communication and/or Sequence Diagrams)
Service Information Model (part of Service Specification)	Structure of the information used by services at a logical level.	UML Class Diagrams or ERD Diagrams
Service Description	Overview of a service	Textual document
Service Specification	Detailed specification of a particular service including both functional and non-functional requirements	Textual document and UML models
SO Security Specifications (part of SO Security Architecture)	Specifications for security services/ mechanisms and how they are used by other services in the architecture	Textual documents and UML models

Table 3: Key Specification View Artifacts

A detailed discussion of how traceability is achieved is beyond the scope of this article but at a high level this traceability might be done as follows:

- Business Processes are captured in terms of activity diagrams that include swimlanes representing logical business roles.
- The business roles that are currently or will be automated in software are identified.
- These automated business roles become solutions or services in the Specification Model.
- Lines that cross the swimlane boundaries of automated roles become operations or messages that trigger the activities within the swimlane.
- These activities are the requirements of the solutions or services captured in the Specification View.

Capturing the traceability can take a variety of forms. One mechanism is to use a tool like Rational's RequisitePro to maintain a table of business requirements and the elements from the Specification View that address them. Another mechanism is to create a diagram within the modeling tool that shows the dependency of the Specification View elements to the Business View elements.

Implementation View

Once the Specification View is complete or at least beginning to stabilize depending on the process patterns chosen by the development organization, a model that maps the logical specification onto automation units (things that package or will actually be realized in code) should be created. The mapping may be as simple as one automation unit per logical service or as complex as mapping several logical services into some other number of automation units. Further, the services might be (and often are) provided by legacy applications whose software architecture is very complex and not well understood. In situations such as this, one large automation unit might implement many services.

The primary artifact of the Implementation View is the Service Implementation Architecture that captures the structure of the Automation Units that implement the services identified in the Service Specification Architecture. Figure 5 shows an example Automation Unit Dependency Diagram of the Service Implementation Architecture.

Again, the Implementation View may contain a number of artifacts and models depending on the needs of the project. Table 4 describes key artifacts and models contained in the Implementation View.

The models of the Implementation View are typically captured using UML diagrams. The Service Implementation

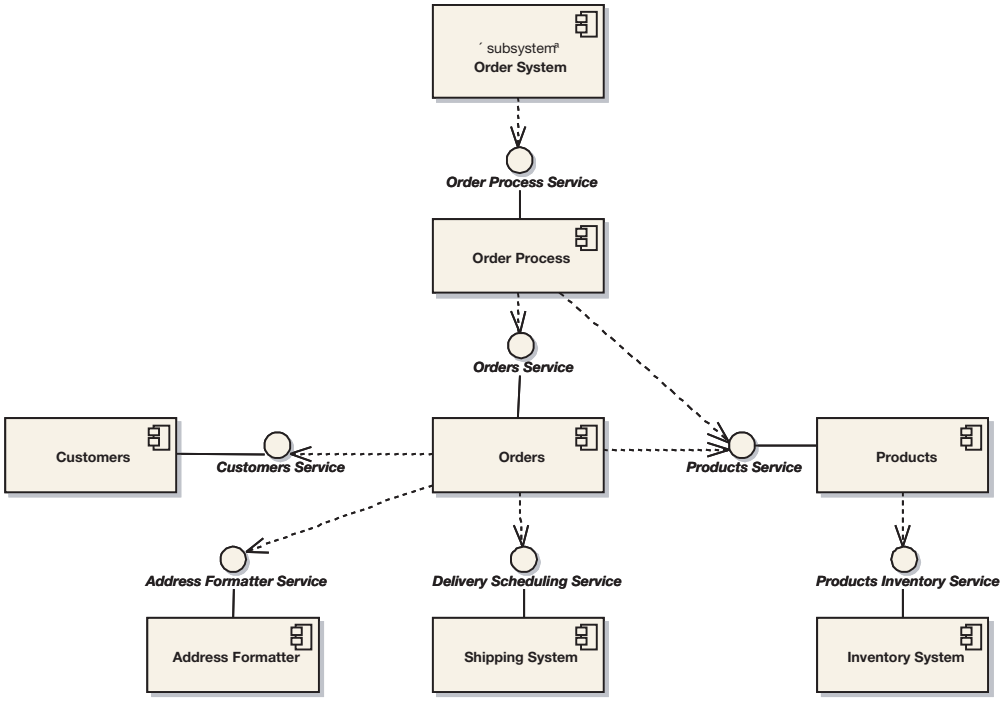


Figure 5: Sample Automation Unit Dependency Diagram (part of Service Implementation Architecture)

Key Artifact	Focus	Typical Format
Service Implementation Architecture	Structure of Automation Units and software modules that realize logic services	UML model containing structural diagrams (e.g., package, component and class) and behavioral diagrams (e.g., communication and sequence)
Solution Implementation Design	Structure and orchestration of services that comprise composite applications	UML model containing package, class and/or component diagrams
Physical Data Model (often part of the Service Implementation Architecture)	Physical structure of the data used by the service or set of services	UML model containing package and class diagrams
Service Message Structure (often part of the Service Implementation Architecture)	Structure of messages transferred back and forth during service interactions	UML model containing package and class diagrams
Service Message Patterns (often part of the Service Implementation Architecture)	Typical patterns of messages exchanged during service interactions	UML interaction diagrams (communication and/or sequence diagrams)
Automation Unit Description	Overview description of a particular Automation Unit	Textual document
Automation Unit Specification	Detailed Specification of an Automation Unit	Textual document and UML models
Solution Implementation	Actual software that implements a solution	Source code
Service Implementation	Actual software that implements a service	Source code

Table 4: Key Implementation View Artifacts

Architecture is typically captured as one or more component diagrams showing the Automation Units and the relationships between them.

As with traceability between the Specification View and the Business View, capturing traceability between the Implementation View and the Specification View can take a number of forms. Traceability Matrices in tools such as RequisitePro are often used as well as UML Class diagrams that include elements from the Implementation View and elements from the Specification View with Dependency relationships between them. This traceability is crucial in order to be able to map all the way from business requirements to the actual code that supports them.

As for the actual mechanism for providing traceability, Everware-CBDI recommends mapping the Specification of the logical Service in the Specification View to the Provided Capabilities of the Automation Units in the Implementation View. Since there isn't necessarily a one-to-one relationship between Services and Automation Units, not all of the operations of a Service will be found on Provided Capabilities of an Automation Unit. In these cases the Service can be traced to the Automation Unit in general.

Deployment View

We've now seen how the Specification and Implementation Views of the solution "layer" of an enterprise work together to separate the logical design of the solutions and services from the implementation design. This is very compatible with Model Driven Architecture™[5] and allows us to separate the logical functionality required of services from the physical packaging and technology thereof. The last piece in this puzzle is the allocation of the service packages or Automation Units to platforms or Nodes on the network (see the Technology View below). This mapping is the focus of the Deployment View and represents a key piece in the methodology puzzle for several reasons. First, it provides the mapping of Automation Units onto Nodes or Service Platforms allowing service or solution

Key Artifact	Focus	Typical Format
Service Deployment Architecture	Static structure and interactions of the Automation Units and their deployment to the Nodes on which they will run	UML model containing deployment diagrams
Runtime Communication Channels (part of the Service Deployment Architecture	Communications Channels between the Nodes on which the Automation Units run	UML model containing deployment diagrams
Service Platform Design Specification (for example ESB)	Detailed specification of the Service Platform including the infrastructure services provided by the platform	Textual document and UML models

Table 5: Key Deployment View Artifacts

architects to communicate with infrastructure architects about how services will run in the production environment. This ensures that services required for runtime will be available on the platforms that will run the Automation Units.

Second, it provides a mechanism for these same service and infrastructure architects to analyze the processing and bandwidth capacity required for each segment of the infrastructure. Often, this type of analysis is left until the last minute or disregarded altogether. The result is generally slow response time and subsequent stakeholder dissatisfaction. Table 5 describes key artifacts and models of the Deployment View.

Ensuring traceability at the Deployment level is relatively easy thing to do since the Deployment View typically includes the Automation Units that come from the Implementation View. This provides direct traceability without any additional work. Alternatively, one might forego creating detailed deployment diagrams and opt for a matrix that shows which Automation Units are deployed to which Nodes or Execution Environments.

Technology View

The Technology View is last piece in the overall enterprise layering. The purpose of this view is to nail down exactly what the network will look like, policies that will govern service operations and to ensure that the technology base required by the services running in the production environment have all the pieces they require.

Table 6 provides a list of the key artifacts and models contained in the Technology View.

Traceability between elements in the Technology View and elements in the Deployment View is often navigated in a direction backward from that of the other layers. For instance, deployments of Automation Units in the service Deployment View need to be traced back to Automation Units in the Service Implementation View. Provisioned Capabilities of Automation Units need to be traced back to Service Interfaces or Operations in the Specification View. Services in the Specification View need to be traced back to roles in Business Process Models. All of these examples go "up" through the Views. Infrastructure-Deployment traceability could go *in either direction*. The only time the service architect is allowed to directly drive the runtime infrastructure is when the project is dealing with a "green field" situation. This might happen when an organization is first being spun up or when there is a planned migration to SOA from a legacy environment that in no way supports SOA. In this situation traceability might run from the Infrastructure View elements to the Deployment View elements.

In the vast majority of situations, however, the infrastructure already exists and must be used with relatively little modification. In these situations, the traceability is navigated from the Deployment View elements to the Infrastructure View elements to ensure that the deployed Automation Units can run on the existing infrastructure.

Multi-View Artifacts

The reader may have noticed in reading the above sections that several of the key artifacts/deliverables described in last month's article on the SO Process and shown in Figure 3 above are conspicuously missing from the Key Artifact tables. This is due to the fact that these artifacts cover a broad range of issues and act to pull together aspects of a number of layers into one place. Table 7 opposite provides a list of key multi-view artifacts.

Key Artifact	Focus	Typical Format
Logical Network and Platform Services Design Model	Logical network layout including processing nodes and network nodes, as well as communication channels between them and the services that run thereon.	UML models containing class and object diagrams, UML deployment diagrams
Technology Dependency	Dependencies between technologies used to implement the SOA	Textual documents, UML models containing class diagrams (showing dependencies), or other proprietary formats
Physical Network Design (part of the Logical Network and Platform Services Design Model)	Physical layout of the network	Network diagrams in Visio or other proprietary notations, UML models containing class and object diagrams

Table 6: Key Technology View Artifacts

Note: For a comprehensive list of the Deliverables created as part of the SAE Reference Framework please see the February journal article on The SO Process.

Best Practices – The Methodology "Toolbox"

Best practices are the tools recommended for use in capturing the various aspects of the Business, Specification, Implementation, Deployment and Technology Views. The Reference Framework groups best practices by type – Standard, Pattern, Technique, Deliverable, Model, or Policy. Attention should be paid to each one of these types when tailoring the Reference Framework to your organization so that all aspects of the Framework are evaluated. Not all practices need to be incorporated into a particular tailoring of the Framework. However, the choice to exclude a particular practice should be a conscious one. Table 8 provides a description of each Best Practice type along with examples.

Concluding Remarks

The Architecture component of the SAE™ Reference Framework is been structured into Views and Best Practices in order to support a number of key architectural principles. Perhaps the most critical of these principles is separation of

Key Artifact	Focus	Typical Format
SO Security Architecture	Comprehensive artifact that captures all policies, procedures and architectural elements related to security	Textual document(s) and UML models.
Service Portfolio Plan	Complete plan used to identify, describe, group and schedule the implementation of services by business domain	Textual document and UML models
Solution Specification	Details specifications for a particular hardware/software solution	Textual document and UML models
Service Catalog	Comprehensive list of Services	Textual document or registry
Service Level Agreement	Contract describing services that a provider will provide and the metrics for ensuring that it is being provided satisfactorily	Textual document

Table 7: Key Multi-View Artifacts

Type	Description	Examples
Standards	Guidelines or requirements for a particular aspect of the service lifecycle.	• UML 2.1 for Service Analysis and Design • All Services will be published in WSDL 1.1 • Service behavior (asynchronous document style, RPC) • Delivery technologies per layer (e.g Process and Capability Services use Web Services, all other classes of service use SCA) • Infrastructure services (e.g logging, monitoring, diagnostics, security etc)
Patterns	A structured description of generic problem and a recommended solution, thus representing reusable best practice knowledge	• Business Service Architecture (BSA) Layering Pattern • Service concurrency patterns • Data access patterns • Agility enabling patterns (e.g differentiated service, tagged values – aka key value pairs, generic domain service, event subscription, service switching, façade, etc) • Automation Unit design
Techniques	A special procedure for performing a task, or group of tasks	• Gap Analysis • Business Type Modeling • Dependency Analysis • Capability decomposition • Event Analysis • Canonical Data Modeling • Identifying Services • Service Information Modeling • Modeling Legacy Applications for Service Integration
Deliverables	A special type of artifact which a project is responsible for producing (see glossary for full definition). A deliverable may (or may not) consist of a model (or set of models)	• Service Description • Service Specification • Service Portfolio Plan • Automation Unit Specification • Service Catalog
Models	An abstract depiction of a problem or solution. In the context of SAE, a model must contain objects defined by the SAE meta model; e.g Business Type Model. A model can optionally also be a deliverable.	• Business Process Model • Event Model • Business Type Model • Service Specification Dependency Diagram • Service Information Model
Policy	Strategies, rules and guidelines that govern a range of SAE related concerns, from service oriented business modeling to SOA technology infrastructure	• Service Classification and Layering • Service Dependency • Change Management • Service Lifecycle • Service Certification • Service sourcing

Table 8: Best Practice Areas

concerns. By dividing the structure into Views, architects can separate business concerns from software concerns, logical concerns from technology concerns and so on. This separation, in addition to allowing the architect to focus on a particular concern without having to remember all the others, also improves the maintainability by "chunking" the architecture into manageable pieces.

The structure is also complementary with industry trends such as the Object Management Group's (OMG) Model Driven Architecture™ (MDA) and the more general model driven development (MDD). By incorporating detailed models at each level supported by rigorous traceability, organizations are able to capture and maintain detailed models of their service architecture and analyze the impact of changes to that architecture in either direction up or down the enterprise "layers" (e.g., business, specification, technology, etc.). As model generation technology evolves, users of the Reference Framework will be able to more easily incorporate these tools and techniques into their methodology as appropriate since the models are already there.

Organizations will have differing needs for an SOA reference framework. The framework will need to integrate with existing architecture practices, techniques and tooling where they exist. We expect variation in modeling languages/notations used (UML, BPMN) and customization of modeling techniques, policy sets, patterns and standards.

Adoption of a reference framework is also an evolutionary process. Techniques, and particularly patterns and policies will evolve with SOA maturity. In the early stages many policies will probably be advisory; but with more experience they may well become strongly recommended or mandatory.

The term framework is used advisedly – it is provided as a basis for customization and specialization. Also in developing the SAE SOA Framework we are very aware that many architects will already have established some form of framework, often using ideas from one or more sources such as Zachman, TOGAF, EA etc. We will follow-up this report with a mapping to a number of the widely used frameworks.

Everware-CBDi is actively evolving the Reference Framework Architecture together with the Model, Process and Organization components. This will be documented in the SAE™ Knowledgebase. Readers' views, experience and feedback would be greatly appreciated.

Notes

1. A Meta Model for Service Architecture and Engineering Dodd, J., *CBDI Journal*, October, 2006. http://www.cbdiforum.com/secure/interact/2006-10/Intro_Meta_Model_for_Serv_Architecture_Engineering.php
2. Everware-CBDI is actively engaged in the Object Management Group's (OMG) UML Profile and Metamodel for Services (UPMS) initiative and is closely tracking work within OASIS to refine their SOA reference model.
3. The Service Oriented Process, Allen, P., *CBDI Journal*, February, 2007. http://www.cbdiforum.com/secure/interact/2007-02/service_oriented_process.php
4. A Meta Model for Service Architecture and Engineering. http://www.cbdiforum.com/secure/interact/2006-10/Intro_Meta_Model_for_Serv_Architecture_Engineering.php
5. OMG Model Driven Architecture. http://www.omg.org/mda/

Independent Guidance *for* Service Architecture and Engineering

CBDI Objectives

CBDI Forum aims to provide independent, action oriented practice guidance on Service Oriented Architecture and Component Based Development for architects, business analysts, project managers, designers and others involved in creating and delivering advanced architectures.

CBDI Delivery Channels

CBDI Forum provides:

- Subscription services – continuous practice guidance published in the *CBDI Journal* every month (with July/August combined into one volume)
- Workshops and Seminars – providing indepth education on architecture, process and practice. Public and In-house classes are available.
- Consulting – specific guidance on adoption roadmap including status assessments, methodology customization, architectural guidance including reference architecture development, governance reviews, business design and strategy development.

CBDI Background

CBDI Forum is the Everware-CBDI research capability and portal providing independent guidance on best practice in service oriented architecture and related delivery processes. Working with F1000 enterprises and governments the CBDI Forum research team is progressively developing structured methodology and reference architectures for all aspects of service oriented architecture.

A CBDI Forum Subscription provides a corporation or government department with access to a unique knowledgebase, ongoing continuous practice research guidance materials and hotline access to CBDI Forum experts. The monthly *CBDI Journal* provides in-depth treatment of key practice issues and guidance for architects, business analysts and managers. Forum Meetings are held periodically in Europe and North America allowing peers to engage and exchange experience and best practices.

Contact Us

For further information on any of our services contact us at: info@cbdiforum.com or +353 28 38073 (International)

Subscribe to the CBDI Forum

The CBDI Journal *is published monthly with a combined July/August edition.*

An annual corporate subscription includes access to all back numbers plus access to Powerpoint Libraries and the CBDI Hot Line Service. In addition Corporate Subscribers are encouraged to participate in Special Interest Groups (SIGs), Reviews and general Forum Meetings.

For more details see: www.cbdiforum.com