Architecture Patterns for Enterprise-wide SOA

What do we actually use SOA for?

Borjan Cace

Service Oriented Architecture has entered the mainstream of business applications and articles about SOA continue to proliferate. However, texts that share people’s real-life experiences with SOA are scarce. Partly this can be attributed to difficulties in sharing architectural knowledge in a structured way. This article calls for more effort to be put into sharing knowledge through architectural patterns. That is done by describing five concrete patterns that have emerged in the SOA practice of information-intensive enterprises. Fellow architects are invited to join the effort of ‘Via Nova Architectura’ and share their experiences through patterns.

Introduction

There are thousands of articles offering advice on SOA; there are “do’s and don’ts” all over the Internet. Consultants, integrators, product vendors … all those people want to tell the rest of the world how to aim the new silver bullet in the right direction. How come so many people know what others should do with SOA? How do we know what can be trusted?

I do not feel that there is much to be gained by analyzing all that is being written from the perspective of the parties selling products or services. It would be much more effective to look at SOA from the perspective of the enterprise and ask ‘what solutions have worked and what have failed? What real-life problems have been successfully tackled by service orientation?’

One way to do this is to describe the recurring problems and the associated, well-proven architectural solutions that involve service orientation. In other words, let’s look for the viable architectural patterns that are based on SOA. For that purpose, I will describe five patterns common to information-intensive enterprises such as insurers, banks and government agencies.

The problem domain addressed by these patterns is associated with the delivery of business services to customers/citizens. The services we are concerned with are delivered by information systems. For example, local authorities issue a variety of permits; tax offices make decisions on tax returns; government agencies process requests for welfare support and decide on eligibility; insurance companies process insurance applications and claims and so on. All these services are information-intensive and the service fulfillment has been automated to a large extent. What architecture helps those information-intensive organizations to really deserve the label e-Business or e-Government?
Please note:

a) The solutions I am describing in this article are not my invention - my contribution is to describe those solutions as patterns¹;

b) The patterns are strongly related to service orientation but we should not label them as ‘SOA patterns’. Service orientation is one essential aspect of these solutions but not the only one.

c) By saying ‘Architecture Patterns for the Enterprise’, the broad scope of these patterns is emphasized, namely, the entire enterprise. This does not imply that I am writing about ‘Enterprise Architecture’ as a theme.

d) It is not claimed that the patterns described here are the most popular ones;

There are several standard ways to write patterns - one of these has been agreed upon for the ‘Via Nova Architectura’ site. However, for the time being and for practical reasons, I am pursuing another way and use the original ‘Alexandrian’ form [Alexander, 1977]; patterns written in that way (more narrative than structured) can be embedded more easily in an article. Eventually, I will use the wiki of ‘Via Nova Architectura’ and structure the patterns accordingly.

In the following paragraph, I will summarize the essence of SOA as an architectural style. That paragraph can then be used as the reference point for the claim that SOA also contributes to solving real-life problems, being more than just hype. Subsequently, the choice of five patterns is described. Some other pattern sources are referred to, this in order to properly re-use the work done elsewhere and put the patterns described here into context.

**What Makes a Solution Service Oriented?**

> "If, as I believe, the conceptual structures we construct today are too complicated to be accurately specified in advance, and too complex to be built faultlessly, then we must take a radically different approach. Let us turn to nature and study complexity in living things, instead of just the dead works of man. Here we find constructs whose complexities thrill us with awe. The brain alone in intricate beyond mapping, powerful beyond imitation, rich in diversity, self-protecting, and self-renewing. The secret is that it is grown, not built. So it must be with our software systems"


If we want to attribute any solution to Service Oriented Architecture, we need to agree what that architecture is, and what makes it different. I have elaborated on this in another Via Nova article: ‘SOA Terminology and the SOA Reference Model of OASIS’ [Cace, 2008], so I will quote from there and start with the definition of OASIS [OASIS, 2006]:

> "Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."

1. ‘**Distributed capabilities**’ means that SOA is concerned with software components². Those components have independent life-cycles and interact in a certain way which we call

---

¹ “...The point of patterns is not to describe tricks any of us have invented individually, but solutions many of us have come up with, independently, to solve a wide range of problems. In academic or research paradigms, originality is key. With patterns, originality is not important; in fact, the best validation one can give a pattern is to say it’s not original, that it’s ubiquitous...” [Cunningham, 2006]

² The term software components used here is by no means related to “Component Based Development” (CBD). The term is used in a general sense to designate a piece of software, a programmatic entity; an application is considered a ‘component’ too. Also it needs to be stressed here that this definition excludes interactions that are not programmatic, i.e. those that do not take place between programmatic entities. This is not in line with the definition of the Reference
'service-oriented': some components offer services (sets of individual functions called operations) and others (service consumers or clients) consume those services. SOA is an architectural style that focuses on leveraging those service-oriented interactions.

2. The interacting software components “\textit{may be under the control of different ownership domains}”. This means that the components have independent life-cycles and the level of coupling should be minimized, i.e. the components in SOA must be ‘loosely coupled’. The fundamental aspect of ‘loose coupling’ in SOA is that the service interfaces are defined as separate entities and that the usage of these interfaces is further formalized in contracts.

3. SOA is a solution in the context of ‘business applications’, i.e. the software that is part of information systems in businesses. In that context, the central issue being addressed is the ability to expose specific \textit{business functionality} as a set of software functions and to have that business functionality consumed by other software components.

To summarize, SOA makes interactions between software components with independent life-cycles viable, something which could make the IT architecture truly adaptive on the enterprise scale\(^3\). The promise of this new architectural style is that the software components can behave like business units and by collaborating can grow together into a ‘system-of-systems’.

\section*{Choice of Architecture Patterns}

\begin{quote}
\textit{...no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that it is supported by other patterns: the larger patterns in which it is embedded, the patterns of the same size that surround it, and the smaller patterns which are embedded in it.}

\end{quote}

There is no doubt that patterns cannot exist in isolation. If we believe that patterns are a means to describe knowledge, we also have to recognize that those chunks of knowledge are interdependent. A pattern supports some larger patterns and is supported by patterns on a lower conceptual level. That is what makes it difficult to start writing patterns: where to start from and where to end? How can we connect to the existing body of knowledge?

The choice of patterns described in this text is pragmatic and focuses on the most common SOA solutions for ‘multi-channeling’\(^4\), a business approach that is common in information-intensive organizations like government agencies, insurance companies and banks. Five patterns\(^5\) often used to tackle the multi-channeling issues are elaborated. The patterns deal with the application architecture. The one on the highest level, MULTI-CHANNEL BRIDGING, sets the conceptual direction of the solution for the business problem. The other four describe the supporting choices that make the bridging pattern work.

Not all patterns are equally proven in practice. Some are the recognized, best practice solutions (“true invariants”, as Christopher Alexander says), like MULTI-CHANNEL BRIDGING, BUSINESS PROCESS COMPOSITION and ENTERPRISE DATA RETRIEVAL SERVICES. The titles of the paragraphs describing these three patterns have been marked with two asterisks\(^6\).

---

\(^3\) Please note here that significant numbers of integration solutions were service oriented much before the term ‘Service Oriented Architecture’ has been introduced.

\(^4\) Multi-channeling is an overloaded term. Here, it refers to the usage of different channels to fulfill different business service deliveries within a certain customer (citizen) population.

\(^5\) Those patterns are obviously not at the highest level of the system of architecture patterns we should envisage. However, the problem it solves can be understood in isolation so we may assume that the pattern that solves it can be used as a provisional starting point in describing a limited number of patterns.

\(^6\) Marking pattern titles with no, one or two asterisks is the convention used by C. Alexander in his patterns book [Alexander, 1977].
REPLICATION IN THE MIDDLE on the other hand, appears less often in publicly available sources. However, it is known to work well in practice, so it has been marked with one asterisk. COMPOSITE RETRIEVAL SERVICES has been proven in practice, but other solutions are also possible.

**Multi-channel bridging**

![Multi-channel concept in the Dutch Government Reference Architecture (NORA)](image)

**Multi-channel behavior is essential for modern, information-intensive enterprises. If business service fulfillment is not shared over channels, the enterprise will not be able to maintain consistent business service delivery.**

Nowadays profit-seeking and non-profit organizations use multiple interaction channels to communicate with customers/citizens (business-to-consumer, B2C\(^7\)). This includes Call Centers, Internet portals, Smart phones and PDA’s, written correspondence (supported by standardized paper-forms), front-office counters (points-of-sale) and partner channels. In addition, this multi-channel behavior is also a vehicle for channel migration: the customer interactions are gradually moved to the channels the service provider prefers, namely the ones that require a minimum or no human assistance on the service provider side (channel substitution).

Multi-channeling requires that customer requests are processed uniformly: the processing of an inquiry may not depend on the channel used. It is an absolute requirement to have consistency of information and services across channels.

This business requirement is incompatible with the mature ‘stove-pipe’ information systems. The applications of these ‘classic’ systems tightly couple user interactions with the processing (the business logic and the data) which implies a monolithic user-interaction solution for all channels. There are problems associated with this issue.

Firstly, it is technically difficult to provide multiple user interfaces (mobile phones, Internet portals, call centers and so on) in a single application.

Secondly, the functionality of one such application cannot be properly integrated with the functionality of other applications. Consequently, users need to simultaneously access multiple enterprise applications. In some cases integration on the presentation level may suffice (this

---

\(^7\) “user-to-business” in the terminology of IBM’s Patterns for e-Business [Adams, 2001], [IBM, 2008]
The separation between the front-end and the back-end presents a major change in the critical business activities of the enterprise. So we need to look more thoroughly at what is required to make our ‘service bridge’ meet the challenge. What constructs do we need to make this solution viable? What sort of ‘traffic’ should we expect to see on the ‘bridge’?

In order to understand the implications of this pattern, let us look at the functionality of the back-end services. That functionality can be logically clustered into three groups:

- In the first place, we need services that provide business functions that correspond to customer inquiries.
- Secondly, there will be services that expose some business logic that is common to multiple channel applications, to augment the first group of services. This includes access to diverse supporting functions like business rules, server-side data validations, inventory checks and the like.
- Finally, there will be services providing access to all other enterprise content that are not directly associated with any individual customer.

The first group can be further subdivided into services that:

- a) Process requests for a business service delivery;
- b) Retrieve customers’ personal data such as address, contract information, previous purchases and so on;
- c) Update customers’ personal data such as e-mail address and telephone number;
- d) Provide access to the status of a process initiated earlier.

These services correspond strongly to the use cases that capture the most common customer interaction with the enterprise. That is to be expected, as enterprises intentionally push the implementation of business functionality into the back-end as much as possible.

To make this clear via an example, let us look into the self-service portal that enables citizens to acquire permits from their local authority.

---

6 This should not be understood as a statement against the enterprise mash-ups: those provide a great solution when we actually do want to expose business services separately.

9 Or channels, when that is feasible and opportune to do - for example Internet self-service users and call center agents may use the same web application. This should be seen only as an optimization afterwards – the principle is to support each channel separately.
The functionality associated with these use cases can only be implemented in the back-end systems, which means that we effectively propagate the user’s requests into those systems.

That imposes some challenges regarding the synchronicity of the services. In SOA, we typically prefer all services to be asynchronous – this to allow for better utilization of resources and to keep implementation costs low. However, asynchronous interactions are inappropriate in the case of services that are exposed to channels; the functionality of the services is part of the interaction with the end-user. To match the customer expectations, those services must be synchronous, available 24/7 and high-performing. This applies not only to information retrievals; even some transactions need to be functionally synchronous, like updating a telephone number, an email address or a bank-account number.

On the other hand, service fulfillment is almost always asynchronous by nature (and customers do not expect anything else). The process may involve human interaction or it may be dependent on some external system which imposes unavoidable delays. In other words, the service fulfillment mostly commonly requires a long-lived process.

Based on this analysis we can group all operations as follows:

1. Non intrusive operations (information retrievals), all synchronous (retrieval of customer data, retrieval of other enterprise content and diverse supporting functions such as data validations)
2. Transactions that require functionally straight-through processing - all synchronous (simple updates)
3. Long-lived transactions – asynchronous (the most business service fulfillments)

This grouping of operations imposes constraints that need to be considered when constructing solutions based on this pattern. How to ensure 24/7 availability? How to achieve the best response times? These have to be addressed by smaller patterns, some of which are suggested later in this paragraph.

To summarize, disregarding these considerations, we can conclude that there should be no substantial obstacles to realize this pattern.
Therefore:

Provide a bridge between the customer interaction channels and the common portion of the service delivery process by exposing a uniform set of services.

Source: Gartner, 2007

Figure 3 Multi-channeling in "Applied SOA: Transforming Fundamental Principles into Best Practices" [Gartner, 2007]

Ensure that synchronous services are performing well. The asynchronous, long-lived transactions should be modeled as automated processes - consider the BUSINESS PROCESS COMPOSITION pattern. Typically, customers/citizens require access to the information that is related to them – use ENTERPRISE DATA RETRIEVAL SERVICES. If combined views over multiple data sources are required in multiple channels, consider COMPOSITE RETRIEVAL SERVICES. Performance issues can be addressed by REPLICATION IN THE MIDDLE.

In the resulting context, examine the implications for security and transactional integrity arising from the use of services. Select patterns that are best matching your technology choices (J2EE, .NET etc)

This pattern appears to match the “thin mid-office” pattern as described in ‘Structures to Effectively Share Architectural Knowledge’ [Ponisio, 2007]
**Enterprise data retrieval services**

... at the heart of many organizations, especially those that are administrative in nature such as government agencies and insurers, there needs to be a central, shared data store. That data (for example, customer or citizen records) is accessed in real-time by most of the organization’s business processes including the automated process execution of BUSINESS PROCESS COMPOSITION and the channel applications supported by MULTI-CHANNEL BRIDGING. This pattern ensures that the data is consistently used across the organization’s processes.

✧ ✧ ✧

Data replicated in various information systems of the enterprise is difficult to keep consistent and this hampers the agility of the organization.

Data replication is most commonly a legacy of ‘stovepipe’ solutions, typically implemented in 4GL. Let’s take a government agency as an example. The citizens’ data maintained by the agency is stored in multiple databases that together form a (logical) central registry of the relevant citizens’ records. These records are needed in other databases that belong to multiple information systems, mostly implemented in some 4GL. Most commonly, 4GL environments require all databases to be integral parts of a single, tightly coupled system. Updates, either batch bulks or individual transactions are preferably applied only to the central register and then periodically propagated to other databases.

This solution has several inherent problems:

- All of the organization’s major systems are logically tightly coupled by the data model.
- Data that is dynamic in nature needs to be kept synchronized across all systems which is technically challenging\(^\text{11}\).
- It is error prone – technical problems easily lead to data inconsistency that sometimes goes unnoticed.

---

\(^{10}\) This scheme has been translated from an official site of The City of Amsterdam. It shows the so called “basisregistraties”, six centralized data stores (information registrations) that are maintained by the municipality. [http://www.gvi.amsterdam.nl/producten/basisregistraties]

\(^{11}\) A replication is not a problem if the replicated data is not time-critical; that is typically the case with data that is changed only periodically.
Therefore:

**Exposé the data that is shared across the organization as data retrieval services.**
Cluster the data in a way that ensures maximum service coherence.

In this pattern, a central register allows real-time access to its data records. An organization may have multiple central registers - data coherence determines the boundaries between the registers. Define individual data retrieval operations in such a way that the number of requests will be minimized (for more on enterprise data access issues refer to [Lublinsky, 2006]). This creates layers that enable enterprise agility – regarding which an enterprise-wide data layer is a good place to start.

When exposing data outside the organization (EXTENDED ENTERPRISE business pattern [Adams, 2001]) consider first using Web Services and a solution like WEB SERVICE GATEWAY [IBM,2004] or similar.

If data from multiple registers and other systems needs to be combined, use the COMPOSITE RETRIEVAL SERVICES, eventually combined with the REPLICATION IN THE MIDDLE pattern.

Consider using an ODS solution while migrating from the legacy data sources.

Please note that ENTERPRISE DATA RETRIEVAL SERVICES is an anti-pattern when used outside of context, and that happens in practice too. Retrieving enterprise data via services may cause serious performance problems in batch processing - services are not meant for that purpose. On the other hand, accessing enterprise data via services works perfectly well in all kinds of interactive applications and process orchestrations (see BUSINESS PROCESS COMPOSITION).

**Composite retrieval services**

... Suppose you have organized your data in coherent data sets and have exposed it for retrieval through services like in the pattern ENTERPRISE DATA RETRIEVAL SERVICES. However, the need to create views over multiple sets will not be eliminated and may exist in multiple, service-consuming applications. In this case it is helpful to create centralized combined data views through composite services.

Creating the same combined view over data from multiple sources may be required in multiple applications. How can we minimize the maintenance burden of accessing multiple data sources? How can we ensure that all applications use the same business rules if those are required while creating a combined view?

Some logically related data may span multiple systems. For example, some customer data would reside in a central data store, some in a corporate CRM system and the telephone contact history would be recorded in a separately developed call-center application. Multiple enterprise information systems may also contain their own, specific information about customers. Applications using this information often need combined views and, by the nature of these applications, most of these views are reusable. The self-service enterprise portal would typically require a set of views over customer data that is very similar or identical to that required by other channels, like the partner channel and the call center. Moreover, from the business perspective, each view must provide the same information, regardless of the channel being used; if a customer can access the same logical view through the enterprise self service portal and through the portal of a business intermediate (a partner) the information
needs to be the same. The simplest service-oriented solution would be that all consuming applications access the data by invoking multiple operations of separate services.

However, this solution has several drawbacks:
- The consuming applications use more contracts than ideally needed;
- Changes in the way data is organized (like retiring legacy systems) need to be coordinated with all consuming applications;
- If any business rules are needed while creating a view, those rules would be implemented in more than one application;
- The number of dependencies may cause governance problems, especially if some service consumers are external to the enterprise (partners).
Therefore:

**Expose the combined data views as composite retrieval services.**

![Diagram of composite retrieval services](image)

*Figure 6 Using the composite retrieval over multiple information sources*

When exposing services to the outside world, use EXTENDED ENTERPRISE [IBM, 2008] (External Services gateway in Figure 6)

**Replication in the middle**

... e-business and e-government require 24/7 availability. How can we ensure that ENTERPRISE DATA RETRIEVAL SERVICES needed in MULTI-CHANNEL BRIDGING also match that requirement? This pattern answers that question, and provides a low cost option in constructing part of the inner layout of MULTI-CHANNEL BRIDGING.

Some data that needs to be accessed from the Internet or a partner channel resides in non-24/7 systems. It is costly to improve the availability and the performance of those systems.

The availability and the performance of services exposed to channel applications directly impact those applications. Some services are scalable and can provide high-availability and high-performance as required by service-consuming applications. However, practice often shows that this does not apply to data retrieval services exposed by enterprise systems. Typically, those systems are built only to be used internally and cannot be easily adapted. Even modern systems, custom built on Java or .NET technology, or implemented in CRM packages may be hard to scale appropriately. The load generated by self-service portal users is hard to predict and may come in heavy bursts. Therefore, major upgrades may be required to keep system response times at a satisfactory level. 24/7 availability is also hard to ensure, especially when the systems were not initially designed to provide it.
This problem can be solved by replicating data in a separate data-store component and make that replicated data (technically an Operational Data Store, ODS) available to channel applications. This replication may integrate data from multiple sources. The updates are never performed directly on the replica but on the original data sources. The updates can then be propagated to the replica in real-time or near-real-time by using some common ODS techniques.

The viability and the costs of this solution depend on the update frequency - too many updates can cause problems for ODS solutions. Practice shows that the administrative, information-intensive organizations deal with large volumes of relatively stable data. In addition, the nature of the data is such that often updates are not time-critical in OLTP terms. For example, an address change is a real-life event and a sub-second capture of that event in a database is pointless. Additionally, administrative data often changes periodically, and those changes take place in processes that are batched by nature (for example, payment of social benefits is a monthly process). It is even the case that some data is so stable that regular bulk loads (for example, nightly) suffice as the update mechanism.

Such ODS-based applications are very simple to develop and maintain. Performance and 24/7 availability are easy to achieve due to the inherent functional simplicity.

Therefore:

**Replicate a selected set of data from multiple sources in the middle of the organization, in between the channel applications and the corporate databases.**

This pattern can be used instead of COMPOSITE RETRIEVAL SERVICES. Alternatively, these two patterns may be combined in one solution.
**Business Process Composition**

![Diagram of insurance claim process](image)

Figure 8 Sample insurance claim process depicted in ArchiMate [Lankhorst, 2004]

... Flexibility in modifying an organization’s processes is a tenet of business agility. This can be best achieved by separating the process flow from the business logic. This pattern also completes the MULTI-CHANNEL BRIDGING by providing entry-points to business service delivery – automated processes are exposed as services to channel applications.

✧ ✧ ✧

**Automating a business process without a means to keep the process modifiable hampers the agility of the business.**

There is a continuous evolution in automating information-intensive business processes. The classical, monolithic information systems provide the necessary functionality but are not adaptive to changes. New business requirements typically require programming changes in software which consequently lead to a variety of software maintenance problems. The time-to-market is unsatisfactory and the risks involved may become unjustifiably high. A way to deal with this problem is to identify and separate the stable processing elements from those that are predisposed to change. The assumption is that the changes in business requirements are mostly related to just a limited number of services and the process flow.

Following this line of thought, business processes should be automated in a way that separates services (that, in general, tend to remain stable) from the process flow which needs to be adaptable. The implementation of a process flow then consists of a number of service execution steps invoked according to the pre-defined sequential process logic.

The idea of automating business processing in explicit processing steps is neither new nor directly related to some new, ‘SOA’ way of looking at enterprise architecture; document workflow has existed for many years, although focused on organizing and coordinating human work. However, the continuous trend towards automating routine-cognitive human work has
opened new possibilities. Some well defined units of information-processing can be exposed as services and those services can be orchestrated into a complete business process. This entire ‘orchestration’ is also a service; most typically processes automated in this way are initiated by invoking an operation of the service.

Although this ‘process composition’ appears conceptually simple, its implementation requires a professional product (for example based on the open standard WS-BPEL). Orchestrating services means integrating independent services. This involves issues like waiting for currently unavailable services, compensating transactions, waiting for long-lived transactions and so on. All these issues have been addressed in the so called BPM-engines and no enterprise should attempt custom-made solutions. There are multiple additional benefits of the appropriate tooling. The operational business intelligence and other Business Process Management facilities are commonly provided ‘out of the box’ as a part of the product.

The human workflow\(^{12}\) must also be considered in this solution. Few of the processes are entirely automated for a variety of reasons. It may be that there are special cases that cannot be automated or there may be legal constraints that demand human involvement.

There are two available options:

- The human involvement may be supported in the same BPM engine as the rest of the orchestration
- A (possibly already existing) workflow solution can be exposed as a service and invoked by top level orchestration.

It has to be noted that this BUSINESS PROCESS COMPOSITION pattern is far from simple to apply. Gartner [Gartner, 2007] says about this:

“*The BPM-style pattern of SOA use is an advanced case of SOA. It requires an advanced level of maturity in the organization regarding:*
- The management of service registries and metadata repositories
- The underlying middleware and quality of services
- The coordination of service releases and version control. ”

Moreover, some business process composition attempts have led to disastrous pseudo SOA implementations, probably due to gross, but unfortunately widespread misunderstandings of service orientation. Those situations apparently reiterate and are also described as anti-patterns. The so called POA (Process Oriented Architecture) is described in Steve Jones’ ‘SOA anti-patterns’ [Jones, 2006]. Some ‘dos and don’ts’ of POA related to the ignorance of the elementary aspects of architecture and SOA in particular are summarized here:

- ‘Grand designs’ do not work; enterprise architecture should envisage an adaptive, self-growing ‘system-of-systems’.
- Business process composition is about *business*; BPM tooling can only solve a recognized business problem. If a business problem is only recognized by IT people, then to introduce SOA and BPM on a large scale could create huge problems.
- If analysis shows that a business problem should be solved without BPM and/or SOA, do not misuse those; just leave out what’s not applicable.
- Operation of a service should expose well defined and coarse business functionality (CRUD services are usually a bad idea [Microsoft,2005])

However, the benefits of composing services into business processing are obvious. Gartner [Gartner, 2007a] states: “*SOA and flow management go hand-in-hand: flow management is an enabler for SOA, and SOA facilitates process integration through flow management.***

\(^{12}\) In this I refer to any human contribution to process execution, disregarding the implementation option: a workflow package, a portal solution or a 3GL application.
Therefore:

**Improve business agility by automating information-intensive business processes via the orchestration of services. Use the appropriate tooling to implement these orchestrations.**

Expose process orchestrations as services too. Implement these orchestrations in a professional, ‘off-the-shelf’ tool.

When invoking transactional operations\(^{13}\) from orchestrations, use reliable-messaging. Alternatively, consider modeling those operations to be idempotent (see Pattern #2: Idempotent Message in [Microsoft, 2005]).

Access to enterprise data can be improved by ENTERPRISE DATA RETRIEVAL SERVICES and possibly by REPLICATION IN THE MIDDLE and COMPOSITE RETRIEVAL SERVICES.

**Some Other Patterns and Pattern Sources**

_I have often been troubled by a lot of the mystique that people summon up around patterns. One of the common areas to kick up this kind of dust is in the issue of patterns and pattern languages. This is often accompanied by ‘this isn’t a pattern language, it’s merely a catalog of patterns’._

**Martin Fowler in ‘Writing Software Patterns’, 2006 [Fowler, 2006]**

Patterns should be more valuable as part of a larger collection of patterns, disregarding the distinction between a ‘mere catalogue’ and a real ‘pattern language’. A collection of patterns

---

\(^{13}\) Transactional operation is an operation of a service that may result in a database update on the service provider side. In SOA, one should not use a distributed transaction monitor to synchronize this transaction with eventual other transactions of the service consumer. Instead, a compensating approach should be used.
provides a framework that guides the user into a certain way of thinking when analyzing the problem domain. For that reason, an isolated pattern or a small collection such as the one presented in this article may be harder to use.

In my search for a catalogue (or language) that would be suitable to ‘host’ the patterns described here I have only identified one publicly accessible pattern source that appeared coherent and well thought out, namely the collection of IBM’s Patterns for e-business [Adams, 2001], [IBM,2008].

**IBM Patterns for e-business**

This collection constitutes a system of patterns that aims to help enterprises ‘to develop e-business solutions. IBM also defines the process that has to be followed in using the patterns: development starts by identifying business patterns from the requirements, continues via two layers of ‘smaller’ patterns and ends with product mappings.

In the IBM process, the step after selecting a business pattern is to select the appropriate Application pattern. Those patterns present the user with choices about how to partition the application logic between the logical tiers and to select the styles of interaction between the logic tiers. Subsequently, a Run Time pattern which would provide you with a grouping of functional requirements into nodes has to be selected.

Perhaps it’s needless to say that the product mappings of the IBM literature lead you to IBM products. However, this does not disqualify their system; other, non-IBM products can also be used.

Of the four business patterns defined by IBM, two can be directly related to the patterns described here:

- **EXTENDED ENTERPRISE**
  
  “The Extended Enterprise business pattern (aka Business-to-Business or B2B) addresses the interactions and collaborations between business processes in separate enterprises. This pattern can be observed in solutions that implement programmatic interfaces to connect inter-enterprise applications.” [IBM, 2008]

- **SELF-SERVICE**
  
  "Also known as the User-to-Business pattern, Self-Service addresses the general case of internal and external users interacting with enterprise transactions and data." [IBM, 2008]

What place would the patterns described in this article have? MULTI-CHANNEL BRIDGING would definitely belong to the layer of application patterns [IBM, 2008], despite the fact that the IBM system does not list multi-channeling as a business pattern. The issues of multi-channeling have been recognized as such and discussed in the process of selecting an application pattern suitable for SELF-SERVICE; therefore MULTI-CHANNEL BRIDGING can be placed in the context of IBM’s business patterns.

The other four patterns would also belong to the IBM category of application patterns. The pattern BUSINESS PROCESS COMPOSITION is essentially the same pattern as IBM’s ‘EXPOSED SERIAL PROCESS APPLICATION PATTERN’ [IBM, 2008]. Other patterns would complement the IBM collection, but, as the IBM pattern catalogue is described in multiple sources (and appears mainly written for IBM professionals) it is difficult for an outsider to extend that catalogue. Moreover, the waste multitude of documentation is a problem for anyone intending to use IBM’s patterns.

Please also note here that, unlike in the architecture that deals with cities and buildings, in our field it is not always clear which pattern comes first, which pattern is ‘larger’ and which is

---

14 This is simplified by ignoring the Integration and Composite patterns. For better explanations, please refer to IBM’s sources or to the original book.

15 There is at least one implementation based on Microsoft technology.
‘smaller’. For example, EXTENDED ENTERPRISE [IBM, 2008] is a pattern to be used in opening an SOA-based partner channel. From the data centric point of view, it is a lower pattern than ENTERPRISE DATA RETRIEVAL SERVICES. Contrary to that, thinking from the service delivery point of view, it is an even higher pattern than MULTI-CHANNEL BRIDGING. However, further discussion would go beyond the scope of this article.

Other sources

The popular, established pattern sources are typically concerned with software architecture and software design. The best starting point for looking into software architecture and design patterns is probably the Hillside group site (www.hillside.net).

I also have to mention popular and often-quoted books such as ’Pattern-oriented software architecture’ [Buschmann, 1996], ’Patterns of Enterprise Application Architecture’ [Fowler, 2002], ’Enterprise Integration Patterns’ [Hohpe, 2003], and, of course, the book of the gang-of-four (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides), ’Design Patterns: Elements of Reusable Object-Oriented Software’ [Gamma, 1994]. However, there is a substantial problem with patterns published in a book: those patterns cannot be actively maintained. As Rebecca Wirfs-Brock said in her blog [Wirfs-Brock, 2007]: “... I don’t expect patterns to be fixed and unchangeable. They should wiggle around a bit. But I like thoughtful discussions and reasonable examples ... Currently, most patterns are copyrighted by authors and are locked up in relatively static media - books or conference proceedings or magazine articles - or static online versions of the same. There’s no central source, no common repository for a growing body of pattern wisdom gained from experience. So when pattern interpretations shift, as they invariably do, it is in a quirky ad hoc manner ...”

Microsoft’s ’Enterprise Architecture, Patterns and Practices’ site is a large and impressive source of information regarding software architecture (of course, restricted to Microsoft technology). A catalogue of patterns for building Enterprise Applications in .NET technology has been also published in book form [Microsoft, 2003].

Some sources list SOA-specific patterns. One of these is A. Arsanjani’s article ’Toward a pattern language for Service-Oriented Architecture and Integration, Part 1: Build a service ecosystem’ [Arsanjani, 2005]. A source that may be of interest is Thomas Erl’s ’SOA Patterns’ site (www.soapatterns.org). It is actually the companion site to a new book, not yet available at the moment of writing.

Final Thoughts and Conclusions

“...We hope, of course, that many of the people who read, and use this language, will try to improve these patterns—will put their energy to work, in this task of finding more true, more profound invariants—and we hope that gradually these more true patterns, which are slowly discovered, as time goes on, will enter a common language, which all of us can share.

You see then that the patterns are very much alive and evolving. In fact, if you like, each pattern may be looked upon as a hypothesis like one of the hypotheses of science. In this sense, each pattern represents our current best guess as to what arrangement of the physical environment will work to solve the problem presented. The empirical questions center on the problem—does it occur and is it felt in the way we have described it?—and the solution—does the arrangement we propose in fact resolve the problem?”


In this article I have elaborated on SOA’s role in resolving the issues associated with multi-channeling. My goal has been twofold:

1. to describe some proven SOA solutions for concrete problems;
2. to describe the solutions as patterns.
Regarding the first goal:

a) There is no need to prove that Service Oriented Architecture significantly contributes towards resolving issues associated with e-business and e-government - there is broad consensus on that. However, browsing through publicly available sources I have been surprised how little has been written by way of concrete examples. This is in stark contrast to the numerous articles elaborating on things of secondary importance. But then, one could attribute this situation to the viewpoint taken by most writers; most publicly available SOA articles are not written from the perspective of an enterprise.

b) One should not view SOA as something that comes first, as a kind of precondition. Instead, SOA should be well understood but used only when its use is appropriate. The patterns described in this article depend on SOA, but not because I am pushing SOA solutions. Rather, I have selected the problem domain (multi-channeling) for which I knew beforehand that it could best be solved by introducing service interactions.

Regarding the second goal, writing patterns;

a) I have followed the advice of Martin Fowler [Fowler, 2006] and used the pattern form which I believed to be the most suitable for me. Writing narrative text appeared easier at the time than structuring the information in a more detailed pattern template. However, describing patterns in an article can only be the first step. Like all other instructions, patterns must be unambiguous, clear and perfectly up-to-date with technological advances. To achieve that, one must continually update pattern descriptions as necessary. The form of an article is definitely unsuitable for that, while wiki’s, for example, are meant just for that purpose. Moreover, a wiki collection of patterns can grow and patterns published can easily be connected with other pattern sources. Therefore, these five patterns will be placed into the wiki of Via Nova Architectura soon.

b) Nowadays few architecture practitioners would question that explicitly described patterns are useful in capturing chunks of architectural knowledge. However, one should also be aware of the difference between architecture that deals with information systems and IT components and that which deals with cities, streets, promenades, houses and so on. Urban planning and the architecture of buildings, which constitute the domain of Alexander’s patterns, have existed for thousands of years. Our field is very different - we are in the middle of an overheated, run-away technological evolution. The patterns of enterprise-wide IT solutions have all emerged in recent years and some will probably disappear in less than a decade. However, perhaps that in itself makes the case for the patterns in our field even stronger. We deal with knowledge that needs to be used now, and patterns open the possibility to communicate that knowledge in a structured way, yet without unnecessary layers of abstractions. In other words: the patterns for enterprise we can recognize now are perhaps already destined to retire, but can still be extremely valuable at this moment.

Last but not least – where is the hard evidence that the five patterns described in this article really resolve the problems in the way it is described?

Perhaps to the disappointment of some readers, that evidence is not provided in this article. Instead, in coming months, I will join the initiatives of Via Nova Architectura and, together with colleagues, start systematically building a practical repository of patterns and anti-patterns. We believe it will be of great value to enterprises and we hope that more practitioners will contribute to a stronger community at Via Nova Architectura.

Borjan Cace
Cace Consult
borjan.cace@ieee.org
References


[Cunningham, 2006] Cunningham&Cunningham, Portland Patterns Repository’s Wiki, “Have This Pattern”, http://c2.com/cgi/wiki?HaveThisPattern


