Patterns for Enterprise-wide SOA

Borjan Cace

CaceConsult
Jan de Beijerhof 14
1191ep Ouderkerk aan de Amstel
The Netherlands
borjan.cace@ieee.org

Abstract: 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 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' (www.via-nova-architectura.org) and share their experiences through patterns.

1 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, in this article I will describe two 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 patterns.
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.

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 [Al77]; patterns written in that way (more narrative than structured) can be embedded more easily in an article.

In the following section, I have summarized 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, two patterns are 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.

2 What Makes a Solution Service Oriented?

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' [Ca08], so I will quote from there and start with the definition of OASIS [OA06]:

"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."

We can use this definition as an anchor point to emphasize the three main characteristics of service orientation:

1. '*Distributed capabilities*' means that SOA is concerned with software components[1]. Those components have independent life-cycles and interact in a certain way which we call 'service-oriented': some components offer services (sets of individual functions called operations) and others (service consumers or clients) consume those

[1] 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 Model of OASIS [OASIS, 2006] which extends SOA to services that are accessed not only programmatically, but also through human interactions. I consider that wrong. There is no point in extending a relatively new concept (SOA) over concepts that have existed for hundreds of years.

services. SOA is an architectural style that focuses on leveraging those service-oriented interactions.

2. The interacting software components "*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 *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 (please note here that significant numbers of integration solutions were service oriented much before the term 'Service Oriented Architecture' has been introduced) 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'.

3 Choice of Architecture Patterns

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', a business approach to deliver services to customers or citizens by using multiple interaction channels. Multi-channeling is common in information-intensive organizations like government agencies, insurance companies and banks. Two patterns often used to tackle the multi-channeling issues are elaborated. The patterns deal with the application architecture.

Figure 1 depicts this relationship between patterns (arrows are directing from 'larger' patterns to 'smaller' ones). The diagram is intended to be more indicative than exhaustive and definite. However, it shows how we could build a cohesive pattern language to provide the optimal solution for multi-channeling.

Only two of the patterns shown in the Figure 1 are described in this article; others can be found in the wiki of the online magazine 'Via Nova Architectura' [SK09].

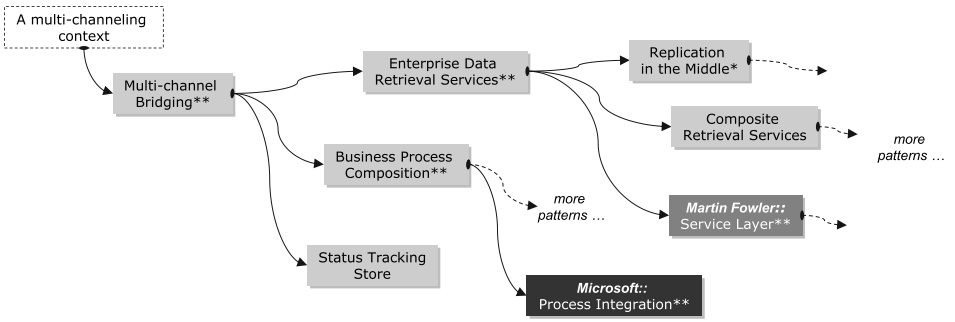


Figure 1 Multi-channeling related patterns as a 'pattern language'

The pattern ‘Multi-channel bridging’ is obviously not at the highest level of the system of architecture patterns we could envisage. However, the problem it solves can be understood in isolation so we may assume that this pattern can be used as a provisional starting point for this pattern language.

3.1 Multi-channel bridging**

Dutch government facilitates governmental organization by providing the 'reference architecture' called NORA. This architecture promotes service orientation and one of its focuses is on providing better service to citizens through Internet.

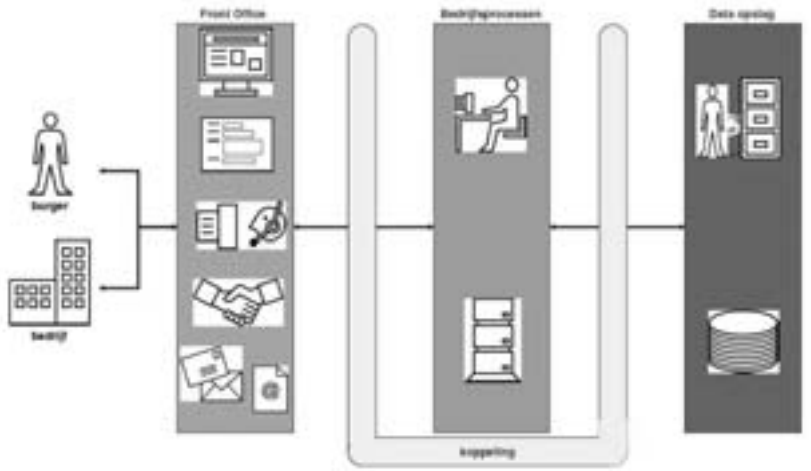


Figure 2 Multi-channel concept in the Dutch Government Reference Architecture (NORA)

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 organizations use multiple interaction channels to communicate with customers/citizens (business-to-consumer, B2C). 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.

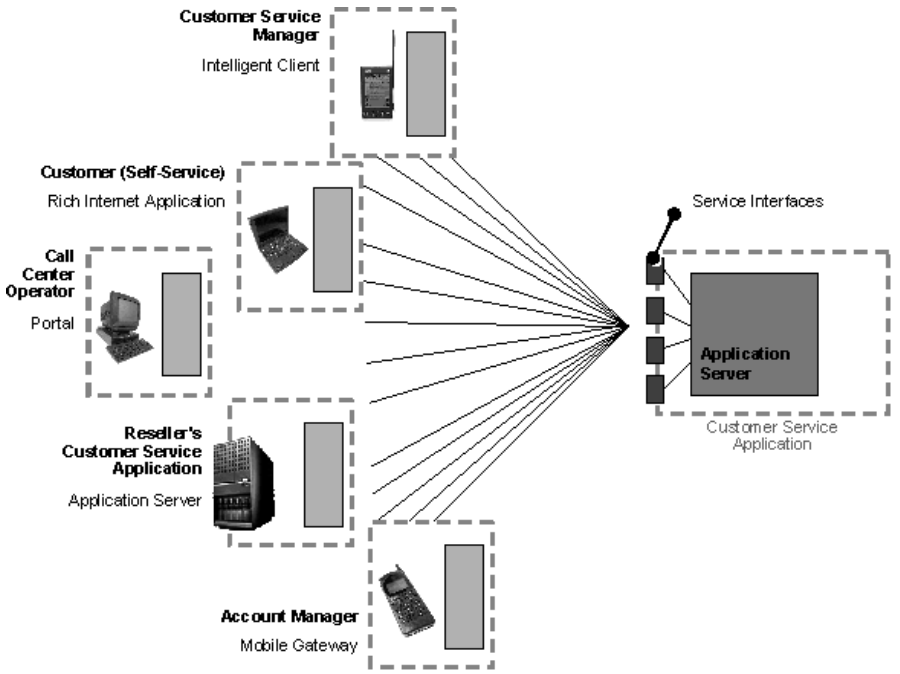
This business requirement is incompatible with the mature 'stove-pipe' information systems. The applications of these 'classic' systems tightly couple user interactions with the processing (the business logic and the data) which implies a monolithic user-interaction solution for all channels. There are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

- It is technically difficult to provide multiple user interfaces (mobile phones, Internet portals, call centers and so on) in a single application
- 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 varies from the old-day 'screen scraping' to the modern 'enterprise mash-ups'[2]). However, this does not address the core of the business issue: information systems that are tuned towards efficient fulfillment of the services may not hamper customer interactions.

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.

[2] 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.



Source: Gartner, 2007

Figure 3 Multi-channeling in "Applied SOA: Transforming Fundamental Principles into Best Practices" [Na07]

The 'classic', SOA based solution is to use services to uncouple the customer interactions from the back-end fulfillment. The interactions are supported by 'front-end applications' while the service fulfillment is provided by 'back-end applications'. The front-end applications are specifically made to provide the most appropriate customer experience for the given channel (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 back-end applications expose a number of services that are typically re-used by all front-end applications.

Yefim Natis of Gartner writes [Na07]: *"SOA is a perfect fit for multichannel applications. Documented separation of the back-end business logic, independent of the nature of the requester, from the requester-specific, front-end logic potentially enables application projects to deliver full application functionality to a maximum number of users in a minimal amount of time. ..."*

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:

- We need services that provide business functions that correspond to customer inquiries.
- Also, 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.

These use cases, narrowed down to the context of a local authority self-service portal are shown in the following figure. The portal enables citizens to acquire permits from their local authority.

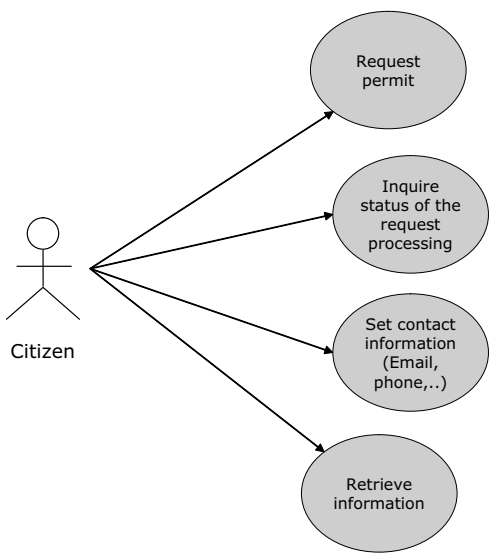


Figure 4: Example use cases for an e-Government self-service: requesting a local authority permit

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.

❖ ❖ ❖

Ensure that synchronous services are performing well - use the patterns available from your technology vendor (Java or .NET, for example).

The asynchronous, long-lived transactions should be modeled as automated processes - consider the BUSINESS PROCESS COMPOSITION pattern.

Use ENTERPRISE DATA RETRIEVAL SERVICES [SK09]. If combined views over multiple data sources are required in multiple channels, consider COMPOSITE RETRIEVAL SERVICES [SK09]. Performance issues can be addressed by REPLICATION IN THE MIDDLE [SK09].

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).

4 Some Other Patterns and Pattern Sources

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 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 [Ad01], [IBM08].

4.1 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[3]. 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 (There is at least one implementation based on Microsoft technology).

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*." [IBM08]
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*." [IBM08]

What place would the patterns described in this article have? MULTI-CHANNEL BRIDGING would definitely belong to the layer of *application patterns* [IBM08], 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

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

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' [IBM08]. 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 'smaller'. For example, EXTENDED ENTERPRISE [IBM08] 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 [SK09]. 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.

4.2 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' [Bu96], 'Patterns of Enterprise Application Architecture' [Fo02], 'Enterprise Integration Patterns' [Ho03], 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' [Ga94]. 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 [Wi06]: "… *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 [MS03].

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 eco-system' [Ar05]. 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.

5 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."

Christopher Alexander in 'A Pattern language', 1977 [Al77]

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

- to describe some proven *SOA* solutions for concrete problems;
- to describe the solutions as patterns.

Regarding the first goal:

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.

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;

I have followed the advice of Martin Fowler [Fo06] 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.

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 how to improve the patterns described in this article? These patterns are already available in the wiki of the online magazine 'Via Nova Architectura' [SK09], an open repository of patterns and anti-patterns. The colleague practitioners are invited to join the community and contribute to the free exchange of architecture knowledge.

References

[Ad01] J. Adams et al: "Patterns for E-Business: A Strategy for Reuse", IBM Press, Oct. 31, 2001, ISBN-1931182027 (the most of the contents also available in [IBM08] and the so called "Red Books" of IBM)

[Al77] C. Alexander et al, "A Pattern Language", Oxford University Press, 1977

[Ar05] A. Arsanjani, "Toward a pattern language for Service-Oriented Architecture and Integration, Part 1: Build a service eco-system", IBM, http://www.ibm.com/developerworks/webservices/library/ws-soa-soi/

[Bu96] F. Buschmann, R. Meunier, H. Rohnert, P.Sommerlad, M. Stal, "Pattern-oriented software architecture, Volume 1" Wiley, 1996, ISBN-0471958697.

[Ca08] B. Cace, "SOA Terminology and the 'SOA Reference Model' of OASIS", Via Nova Architectura, http://www.via-nova-architectura.org/magazine/reviewed/soa-terminology-and-the-soa-reference-model-of-oasis.html

[Fo02] M. Fowler, "Patterns of Enterprise Application Architecture", Addison-Wesley, 2002, ISBN-0321127420

[Fo06] M. Fowler, "Writing Software Patterns", http://www.martinfowler.com/articles/writingPatterns.html

[Ga94] E. Gamma et al, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, ISBN-0201633612

[Ho03] G. Hohpe and B. Woolf, "Enterprise Integration Patterns", Addison-Wesley, 2003, ISBN-0321200683

[IBM08] IBM's developerWorks site, IBM Patterns for e-business, http://www-128.ibm.com/developerworks/patterns/index.html and http://www-128.ibm.com/developerworks/patterns/library/index.html

[MS03] "Enterprise Solution Patterns Using Microsoft .Net: Version 2.0 : Patterns & Practices", Microsoft, ISBN-0735618399 (also available for download:http://www.microsoft.com/downloads/details.aspx?familyid=3C81C38E-ABFC-484F-A076-CF99B3485754&displaylang=en)

[Na07] Y. Natis, "Applied SOA: Transforming Fundamental Principles Into Best Practices", Gartner Research Note Id G00147098, 4 April 2007, http://www.gartner.com

[OA06] "Reference Model for Service Oriented Architecture V1.0" OASIS, Committee Specification 1, 2 August 2006, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[SK09] "Sharing Knowledge of Enterprise Architecture through Patterns" site (link to 'Via Nova Architectura' wiki: http://www.skeap.nl/Browser/Patterns/P4/Multi-channel.htm)

[Wi06] R.Wirfs-Brock, "Rebecca's blog: Pattern Drift", http://www.wirfs-brock.com/2006/01/pattern-drift.html