Learn the discipline, pursue the art, and contribute ideas at www.ArchitectureJournal.net

Resources you can build on.

THE ARCHITECTURE JOURNAL

Input for Better Outcomes

Journal 6

Strategies for Design

Taking Governance to the Edge

Apply Topic Maps to Applications

Design and Implement a Software Factory

Service-Oriented Business Intelligence

Planning Technical Architecture

Behavioral Software Architecture Language

Microsoft®

Contents

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes
Journal 6

Foreword 1
by Arvindra Sehmi

Taking Governance to the Edge 2

by Philip Boxer and Richard Veryard
Service-oriented architecture (SOA) gives businesses a new way of getting work done, but SOA brings with it the traditional problems of governance as well as new challenges. Discover an approach to asymmetric forms of governance that focuses on how businesses understand their risks.

Apply Topic Maps to Applications 10

by Kal Ahmed and Graham Moore
Topic maps provide a metamodel for representing knowledge models that are integrated with other systems. See what current and potential application areas there are for applying topic maps and how to access topic map information using Web services architectures.

Design and Implement a Software Factory 18

by Mauro Regio and Jack Greenfield
The health care industry offers a representative environment for interoperability among organizations. Take a look at designing and implementing a software factory that is based on the Health Level Seven (HL7) standard that also provides a vision for supporting collaboration in different industries.

Service-Oriented Business Intelligence 23

by Sean Gordon, Robert Grigg, Michael Horne, and Simon Thurman
Business Intelligence (BI) and Service Orientation (SO) are independently developed architectural paradigms that form the synergy of the SoBI framework. Find out the similarities and differences between them that are leveraged by the SoBI architectural framework.

Planning Technical Architecture 33

by Waleed Nema
Managers and operational staff know the value of technical architecture planning and its merits for aligning IT with the business and controlling service levels and expenses. Get insight on how these business drivers can impact creating a tactical plan for an infrastructure architecture.

Behavioral Software Architecture Language 36

by Behzad Karim
A tenet for architecture definition is to design software structure and object interaction before the design phase, and architecture design and approval should precede any software development project. Learn how the Behavioral Software Architecture Language unifies software architecture definition with software implementation.

Foreword

Dear Architect,

In the beginning I wished to publish just six issues of *The Architecture Journal* and gather enough evidence to help Microsoft decide whether or not it was worth keeping. Regular readers know what has happened to *The Architecture Journal*; it's been a marvelous thing for all of us involved, including our superstars—the authors—to see the interest and subscriptions grow dramatically worldwide, month after month, for both print and digital media formats. It has been fantastic!

Now it's time to let a professional team at Microsoft Corporate HQ take over this publication and grow it beyond my initial dreams. As a key component of Microsoft's broad architect outreach program, it makes 100 percent sense for *The Architecture Journal* to be given better resources in terms of people and funding than I can manage with my team in Europe. To this end I am delighted to introduce Simon Guest, group program manager in the Microsoft Architecture Strategy Team, as the new editor for *The Architecture Journal*. Simon and his team have the vision and passion for this publication, and I'm confident they'll remain faithful to its original charter as an independent platform for free thinkers and practitioners of IT architecture.

In this issue we have a number of great articles spanning SOA governance, practical information modeling using topic maps, software factories for collaboration protocols in the health care industry, and a service-oriented perspective on business intelligence. We also have two short papers on software modeling and technical planning. Many of these papers discuss my favorite subject areas.

I'll continue to be involved in *The Architecture Journal* and look foreword to seeing Simon and his team realize its full potential, and I wish them every success in that endeavor.

I'd like to acknowledge all of you for the great support and encouragement you have shown me personally. For the pioneering authors of The *Architecture Journal* I have nothing but the deepest gratitude and respect. Thank you.
With best wishes!

Arvindra Sehmi

Taking Governance to the Edge

by Philip Boxer and Richard Veryard

Summary

Discover the challenges faced by asymmetric forms of governance and an approach to working that focuses particularly on the way a business understands the risks that arise from how it relates to exogenous geometries, in addition to the more familiar risks associated with managing its endogenous geometry.

There are two competing visions of service-oriented architecture (SOA) circulating in the software industry, which we can label as SOA 1.0 and SOA 2.0 (see Table 1). Our approach to governance is targeted at SOA 2.0. One of the central questions raised by Christopher Alexander in his latest four-volume work, *The Nature of Order*, is how to get order without imposing top-down (central) planning, or conversely how to permit and encourage bottom-up innovation without losing order (see Resources). Alexander promotes the concept of structure-preserving transformations and argues that under certain conditions large-scale order can emerge from an evolutionary process of small incremental steps, provided that each step is appropriately structure preserving. However, how (and in whose interests) do we define the *structure* that is being conserved, and at what level?

Note that with this concept of structure preservation, the leadership is not doing the design, but setting the parameters in which orderly design may take place. This function is very much a governance role. To answer the question, we need to distinguish governance from management.

Loose discussion of governance (both IT governance in general and SOA governance in particular) can easily spread until it appears to cover all aspects of management. So where does management stop and governance begin?

If we understand *management* as getting things done, then we can best understand *governance* as something like steering (as in *steering committee*). In the old days, it was common practice for large development projects/programs to have steering committees—often a separate one for each project—governing things like funding, scope, direction, and priorities. The steering committee was the forum to which the project manager was accountable. It was supposed to maintain a proper balance between the interests of the project and the interests of the whole organization, and to resolve conflicts.

However, there were several problems with this approach. First, the steering committee typically met infrequently and often only had a vague idea about what was going on. Second, the steering committees generally didn't talk to each other. Third, there were many important IT functions (such as architecture) and desired outcomes (such as productivity and reuse) that didn't have a steering committee. So the steering committee approach was incomplete, inconsistent, and often produced suboptimal results.

Real-Time Governance

The fashion shifted away from ad hoc steering committees, and many large IT organizations began to pay explicit attention to questions of IT governance. Where significant amounts of IT activity were outsourced, this governance included questions of procurement and relationships with key suppliers.

With SOA, we have a new approach to getting things done. We also have all the old problems of governance plus some new ones. There is a complex tapestry of service-oriented activity going on, all of which needs to be properly coordinated and aligned with business goals, and the timescale has changed. Instead of steering committees meeting every two months, we have real-time governance, covering both design governance and run-time governance.

> "WITH SOA, WE HAVE A NEW APPROACH TO GETTING THINGS DONE. WE ALSO HAVE ALL THE OLD PROBLEMS OF GOVERNANCE PLUS SOME NEW ONES"

We know that this cannot simply be reduced to a management problem. Many organizations experience difficulties in doing SOA properly, not because of technical problems but because of a lack of appropriate governance. How do you fund twin-track? How do you manage twin-track in such a way that the service goals are consistent with the business goals so that the SOA can be managed in an efficient way? And what happens when they are not consistent with each other?

We need to get away from the idea of a committee sitting around a table listening to progress reports and issue logs from project managers. We need to hold on to the idea that steering involves accountability to multiple stakeholders, and this accountability is included in a notion of governance that is critically about addressing questions of value, value for whom, and value to what end. These are essentially ethical questions (in the broadest sense of the word *ethics*—the science of value). Thus, while management is about getting things done, governance is about making sure the right things are done in the right way.

In a hierarchical IT organization in which "right" is defined at the top, this distinction might not seem to matter very much. However, if we are thinking about twin-track development where there are necessary tensions between business goals and service needs, let alone federated/distributed development where these tensions are replicated across multiple business entities, then it matters a great deal. As soon as two parallel activities (for example, service creation and service consumption) aren't accountable vertically upwards to the same management point, governance becomes a question of negotiation between two separate organizations, rather than simple management resolution within a single right framework.

Put another way, hierarchical accountability depends on vertical transparency—what can be seen is what you can be held accountable for. Vertical transparency does not imply horizontal transparency across one or more vertically transparent hierarchies. Without horizontal transparency, how can service providers be held accountable to service needs and/or other business entities?

Governance has to determine how conflicts of interest between stakeholders are represented and contained in the interests of the whole. Put another way, it has to create a structure within which this representation is possible. If we understand stakeholders' interests to be expressed as value from particular points of view (in a horizontal or vertical relationship to what is going on), then this question can be formulated in terms of a structure within which to distribute the benefits, costs, and risks between these interests. For example, SOA governance provides ground rules for interface agreements to be made and enforced. In the real world, we know that all agreements are subject to being reneged and renegotiated as requirements and conditions change. However, who incurs the risk of such changes, and who shall bear the cost of any change? If you decide you need to change the interface, am I forced to respond to this change, and if so, how quickly and who pays?

Structuring Transparency

There is also a conflict of interest between the present (short-term adaptation) and the future (longer-term adaptability). How shall adaptability be supported, how shall large complex solutions be permitted (nay encouraged) to evolve, and how shall this evolution be balanced against the need for order and short-term viability?

These questions can be asked at two levels: first at the management level, where there are a series of trade-offs and controls to be maintained, and second at the governance level, where we need to ask questions about the structure within which to distribute the benefits, costs, and risks on an ongoing basis.

Put another way, it's a question of how we determine responsibilities and accountabilities, and the processes of negotiation between them. Ultimately, it's a question of how to structure transparency: determining who is enabled to know the what, how, and when of things getting done in relation to whom. Governance vests authority by creating responsibilities with their associated accountabilities over the expertise and work mobilized by management (or rather managements), requiring that the appropriate forms of transparency be created.

One of the key principles for the management of complexity in IT and elsewhere is the *separation of concerns*. Separation of concerns implies selective attention: who pays attention to what. There is an important governance role here. Not only must governance make sure that attention is complete (at least in the sense that everything important is being attended to), efficient (without unnecessary duplication of attention), and connected (in the sense that the concerns can be brought together where necessary). It must also make sure that the conditions of transparency exist within which such attention is possible.

Creating these conditions of transparency implies a prior separation of unconcerns. Separating out what needs to be attended to from that which can (safely) be ignored brings us to a key governance concern: the governance of what can be ignored that is prior to the governance of conflicts of interest between concerns. What is anybody permitted to ignore? What forms of ignorance are mandated?

For example, SOA inherits notions of transparency from earlier work on open distributed processing. You can use a service without knowing its location; you can use data without knowing its source (provenance). This "not knowing" is very useful in some ways, but dangerous in others. It implies that there is no need for horizontal transparency.

SOA involves loose coupling (and horizontal coordination) not only between software artifacts, but also between the organization units that are responsible for these artifacts. The organization structure typically (although not always) reflects the software structure.

What happens to governance when we can no longer rely on a belief that somebody else is taking care of this risk?

Table 1 Comparing SOA 1.0 and SOA 2.0

SOA 1.0	SOA 2.
Supply-side oriented	Supply-demand collaboration
Straight-through processing	Complex systems of systems
Single directing mind	Collaborative composition
Controlled reuse	Uncontrolled reuse
Endo-interoperability (within a single enterprise or closed collaborative system)	Exo-interoperability
Cost savings	Improved user experience

Table 2 From Deconfliction to interdependence

Deconfliction	Interdependence
"Bush's gambit—filling the skies with bullets and bread—is also a gamble, Pentagon officials concede. The humanitarian mission will to some degree complicate war planning. What the brass calls deconfliction—making sure warplanes and relief planes don't confuse one another—is now a major focus of Pentagon strategy. 'Trying to fight and trying to feed at the same time is something new for us,' said an Air Force general. 'We're not sure precisely how it's going to work out.'" — "Facing the Fury," *Time Magazine*, October 2001	"We've gone from deconfliction of joint capability to interoperability to actually interdependence where we've become more dependent upon each other's capabilities to give us what we need." — Interview with General Shoomaker, CSA, October 2004

Interoperability

It is sometimes supposed that the SOA agenda is all about decoupling. Requirements models are used to drive decomposition—the identification of separate services that can be used independently of each other when used. These separate services are then designed for maximum reuse, producing low-level economies of scale through satisfying an increased level of demand of any given type than was previously possible, and economies of scope through satisfying demand from a greater variety of contexts.

Figure 1 Relating the supply side and the demand side

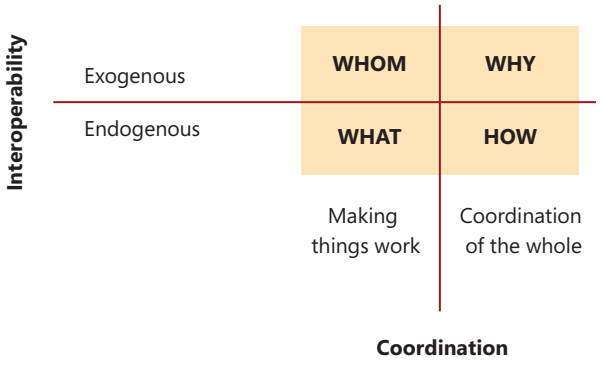
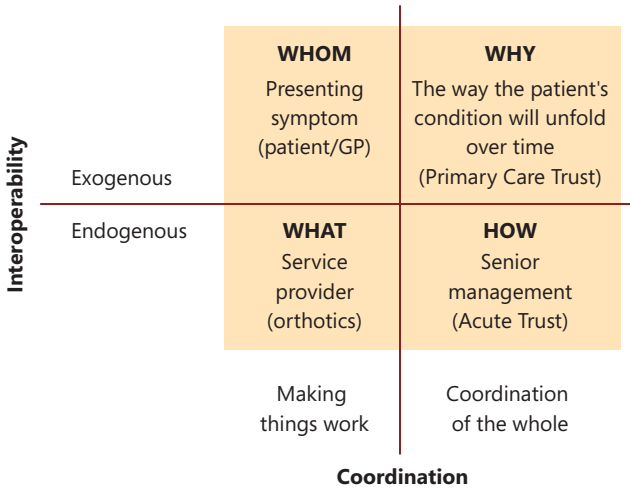


Figure 2 Relating the supply and demand sides for a health care example



Clearly there are some systems that are excessively rigid, and will benefit from a bit of loosening up in such a way that their constituent services can be used independently of the system as a whole. This rigidity is only one side of the story, however. While some systems are excessively rigid, there are many others that are hopelessly fragmented: to get effective use from them, the independent services have to be made to work together. Thus, the full potential of SOA comes from decomposition and recomposition.

An important type of decoupling, as practiced by the military, is known as deconfliction. *Deconfliction* involves taking a whole mission and breaking it down into constituent missions that can be undertaken independently of each other, which produces a hierarchical chain of command in which the mission of any constituent component must depend on the way their mission fits into a larger whole defined by the way the command has imposed deconfliction. Deconfliction relates particularly to the effects that any mission creates, and the side effects of those effects on any other missions. Deconfliction therefore requires not only an understanding of how things work (that is, management), but also an understanding of how composite effects can be achieved from constituent effects with the minimum of conflict between the constituent components and the maximum efficiency in the utilization of resources. Deconfliction is therefore about uncoupling, but crucially about the way this uncoupling is done in relation to the overall mission. (see Table 2).

The military take conflicts between the constituent components of a force very seriously—so-called friendly fire (where we kill our own side by mistake) is clearly a life and death issue. In our terms, friendly fire is an extreme example of interoperability failure. Deconfliction means organizing operations in a way that minimizes the potential risk of this kind of conflict, so that separate units or activities can be operated independently and asynchronously while still contributing to the overall mission.

But deconfliction often also involves a costly trade-off. Resources have to be duplicated, and potentially conflicting operations are deliberately inhibited. The pressures for more efficient use of resources forces specialization capable of delivering economies of scale and scope, combined with increasingly dynamic and political missions, to confront any chosen approach to deconfliction with increasing levels of interdependency.

The response to this issue, as the technologies of communications and control have become more sophisticated and reliable, is to increase the degree of coordination possible between the constituent components of a force, to allow units and activities to be composed in more powerful ways, which is the motivation for network-centric warfare. Rather than depending on a prior plan based on a particular deconfliction, the network enables commanders to coordinate dynamically the relationships between force components as the composite effects they need to produce themselves change. It is this use of the network that makes it possible to take power to the edge of the force where it meets the enemy.

> "IMPLICIT IN ANY FORM OF DECONFLICTION IS AN APPROACH TO COORDINATING THE DECONFLICTED ACTIVITIES THROUGH THE WAY THEY WILL INTEROPERATE"

Interop Risks

Commercial and administrative organizations typically attempt deconfliction through a management hierarchy and through an associated accounting structure of budgets and cost centers that create vertical transparency consistent with the form of deconfliction. This process is known to be inflexible and inefficient, producing *silos* of activity that are relatively impervious to demands for different organizations of their activities. Power to the edge (and, arguably, advanced forms of SOA) is not compatible with traditional budgeting and cost accounting structures.

Deconfliction leads us toward an arms-length notion of interoperability: X and Y are interoperable if they can operate side by side without mutual interference. This operation is the driver behind the uncoupling agendas of SOA, and it does yield improvements in economies of scale and scope.

There is also a positive notion of interoperability: X and Y are interoperable if there can be some active coordination between them. This notion requires us to go beyond the deconfliction *per se*, and not only to consider the assumptions about composition implicit in the form of deconfliction, but also to consider how they can be made explicit and dynamic through coordination. However, this is now network (horizontal) coordination rather than hierarchical top-down planning, which raises the same governance questions we discussed earlier in the relation between the twin tracks of pursuing business and service goals.

Table 3 Governance relationship patterns

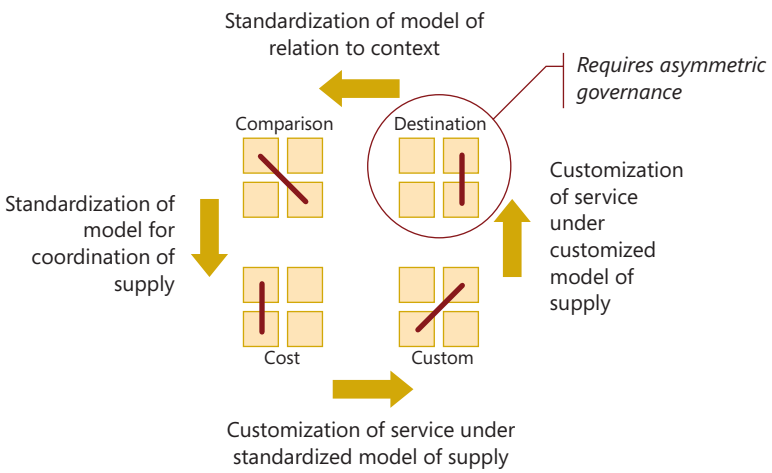
Pattern	Description
Comparison (WHOM-HOW)	The demand is defined in such a way that the context from which it arises is ignored, but the service is coordinated in a way that enables it to respond to that particular demand. This characteristic corresponds to the "comparison" approach for the patient, who is looking for the best solution offered to his or her demand. For suppliers, this form of governance allows them to minimize their exposure to exo-risks by limiting the definition of demand that they will respond to (the user requirement); although, they are still faced by integration risks inside the business.
Cost (WHOM-WHAT)	Not only is the demand defined in such a way that the context from which it arises is ignored, but the response of the service is proceduralized too, and there is no need for an explicit coordinating process. In this "cost" approach both the nature of the demands and the responses to them have become standardized. Now the integration risk is minimized for the supplier and the technology and engineering risks being proscribed.
Custom (WHY-WHAT)	An implicit form of coordination of how things work, often in the form of a particular budgetary regime, constrains the way the service is able to respond to the patient's condition. This characteristic corresponds to the "custom" approach, where the service is standardized but the way it is provided into the patient's context can be varied (mass customization). Here the supplier is exposed to integration risks again, as it builds more variability into the way its service works.
Destination (WHY-HOW)	Each of these three forms treats demand as symmetric to an implicit or explicit form of endogenous coordination. Only in the fourth case do we have asymmetric governance in which the endogenous and exogenous forms of coordination have to be aligned with each other. This characteristic corresponds to the "destination" approach, where the patient goes to that place where he or she can get a treatment exactly aligned to the nature of their condition. It is also only in this case that the supplier takes on the exo-interoperability risks explicitly.

We are particularly focused on the risks associated with interoperability, not only because any given geometry is a particular coordination between its constituent services, but also because these are the ones that emerge as you coordinate across systems and organizations. (We have been following the recent developments around Hurricane Katrina with considerable interest because some of the public criticism of the Federal Emergency Management Agency [FEMA] clearly exposes difficulties in managing some of the interoperability risks.) How are we to think about the nature of these risks?

Governance here involves setting the terms of reference within which management is charged with making certain things interoperable. Interoperability risks may be ranked by severity: in this context this ranking means the extent to which a given risk jeopardizes the way we want to coordinate things.

Implicit in any form of deconfliction is an approach to coordinating the deconflicted activities through the way they will interoperate. A simple view of interoperability is that it introduces a form of deliberate and selective ignorance: if you pay attention to X, you don't have to pay attention to Y. For example, if you adopt this open standard, you don't have to care which of these standard-compliant platforms may be used. This interoperation is a form of specialization (or separation of concerns) as described earlier, which makes this selective ignorance possible. It rests upon an assumption about what X and Y *mean* for those trying to use them to produce a combined effect.

Figure 3 The governance cycle



Let's think about interoperability of management spreadsheets in a large organization. Each manager produces his or her own spreadsheets in an idiosyncratic way to support a particular set of management decisions. Although they all import some data from the corporate database, they have mostly added data from elsewhere, and they have all formatted things differently. A senior manager, Joe, gives a presentation to a board meeting about a major strategic decision, supporting his recommendations with some charts drawn in Microsoft Excel that are derived from a complicated, handcrafted (and completely undocumented) spreadsheet. Joe's colleagues find it impossible to understand his spreadsheet, or to import his analysis into their own spreadsheets for further analysis. Joe's successor is more likely to build a new spreadsheet than try to use the existing one.

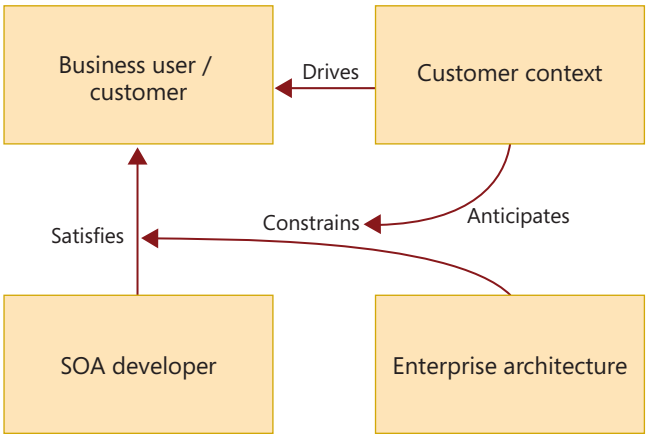
Interoperability fails at two levels here. Not only at the technical level of sharing the spreadsheet as a user-designed artifact, but it also fails at the level of *meaning*. The artifact is an expression of a framework of meaning created by Joe that is not shared by Joe's colleagues and successors. Joe is trying to coordinate data in a way that requires him to make it interoperate in unfamiliar (nonstandard) ways. The very difficulties for managers collaborating on complex strategic decisions also reflect their potential value in creating new ways of acting.

Share the Commitment

What does this have to do with ignorance? It's because of how much you need to know about Joe and his experience of management (that is, his framework of meaning) to make sense of his spreadsheet. The more complex the spreadsheet is, the more it becomes almost an embodiment of Joe himself and his way of paying attention to certain details of management. To use the spreadsheet, you almost need to get inside his skin, see things through his eyes. If Joe is powerful enough within the organization, then he can impose his experience of management on his colleagues and get them to use his spreadsheet, but that will typically result in interoperability problems elsewhere when data starts being used in new and unanticipated ways. A spreadsheet designed for reuse has to assume some level of shared understanding between the user and the originator.

For coordination between X and Y, they clearly have to be able to interoperate in a technical sense—Joe has to be able to send me his spreadsheet, and I have to have the right systems installed to be able to run it. But that is not sufficient. Let's say we have P1 using X and P2 using Y. Coordina-

Figure 4 Role structure



tion has to consider the effects not only of how X and Y interoperate, but also how P1 and P2 affect the meanings of X and Y. With coordination (or its lack) comes the risk that the way use is made of X by P1 and of Y by P2 will not produce the composite behavior expected. We can therefore understand the risks of coordination in the same way that we understand the challenges facing twin-track governance. They are created by a failure to establish a shared framework of meaning within which to act.

With directed composition (central planning, single design authority), the question of shared meaning and permitted ignorance is delegated vertically, but the resultant business geometry has to be endogenous to the interoperations of the activities under the hierarchy. Thus, A is decomposed into (or composed from) B and C; and if there are risks associated with the interoperability between B and C, then these risks are owned by the person in the design hierarchy that owns A. In an organization the requirement for vertical transparency is therefore to be able to enforce this shared commitment to vertical coordination.

In contrast, collaborative composition (planning at the edge between multiple design authorities) requires that shared meanings and permitted ignorances have to be negotiated, and that the resultant business geometry will have exogenous and endogenous elements that are assumed not all to be under the same hierarchy. Horizontal transparency therefore means being able to work out how all the pieces fit together as a whole in order that agreement can be negotiated about how to impose a single, albeit temporary, hierarchy for the particular purposes of collaboration, that is, horizontal coordination.

Ultimately there has to be a shared commitment to a single hierarchy, regardless of whether you are following a top-down (analytic, directed decomposition) process or a bottom-up (synthetic, collaborative composition) process, so long as there can be a shared commitment ultimately to a single hierarchy. The difference in the approaches lies in whether or not the resultant hierarchy is static or dynamic. From the point of view of a business providing services to its customers, static customization will involve agreeing to the hierarchy before entering into a business relationship. In contrast, dynamic customization implies that the very processes of agreeing appropriate hierarchies have to be part of the ongoing business relationship.

Risk Analysis

We are back to Alexander's idea of the level at which we can be structure preserving: we have to be able to decide how much of the geometry has to be variable (underdetermining of the ways in which service users can dynamically customize its uses), and how much of it has to be fixed (overdetermining of the forms of business relationships that can be supported). Whereas symmetric forms of governance can impose vertical coordination, asymmetric forms of governance have to enable horizontal coordination, which raises new issues concerning the way trust is shared. Under vertical coordination it can be guaranteed by the contract with the superior context, whereas under horizontal coordination it has to be negotiated. This issue presents new challenges for distributed leadership under asymmetric forms of governance (see Resources by Huffington et al. and Boxer and Eigen).

In these terms we can see that vertical coordination is dominant when the HOW is dominant, whereas horizontal coordination is dominant when the WHY is dominant (see Figure 1). We can also see that as long as the HOW remains dominant, we are effectively externalizing the exogenous risks.

We use methods of organizational analysis to distinguish the endo-interoperability risks (which come from failures within the organization) from three types of exo-interoperability risk (where the source is outside the organization). These exo-interoperability risks relate to what happens when a supplier's systems and services are combined with third-party systems and services as part of a user's solution. Thus, a supplier's system may not work as expected in the new context, it may not interoperate with other systems as expected, and the whole system of systems may not interact with the user's context of use as expected. These results can be thought of as errors of execution, of planning, and of intention within the user's domain.

> "WHERE IN THE CYCLE THE BUSINESS NEEDS TO BE WILL DEPEND ON HOW IT CHOOSES TO BALANCE RISKS AND REWARDS"

We are aware of various analytical techniques for understanding and managing endo-interoperability risk. We are not aware of analytical techniques for understanding and managing exo-interoperability risk. This situation is not surprising in a world that defines its business as being one of *pushing* solutions, but in a service-driven world these risks begin to become the biggest as suppliers encounter the *pull* of emancipated users (see Hagel and Brown in Resources).

The shift from push to a pull is not only a matter of adopting new forms of governance and horizontal coordination. It also requires that the supplier adopt a *platform* mentality in which the platform is not theirs but the user's—at best they are providing a platform on which users can solve their problem. The general inability to manage exo-interoperability risks is not only a very major problem, but central to supporting a pull relationship with the user. Many of the supplier organizations we talk to are still in denial about this concern, but we are finding more user organizations that recognize the need for their suppliers to adopt a systematic approach to managing their exo-interoperability risks. What is at stake is the old arms-length procurement model that assumes that the user requirement can be separated from the context of use. Instead, users are looking for a collaborative approach to managing the risks attendant upon providing service platforms.

Supply and Demand

The essential characteristic of this platform approach is that it manages both the supply side (that is, the endo-interoperability levels), and the demand side (that is, the exo-interoperability levels). In these terms we will see that the four positions in the cycle we described in "The Metropolis and SOA Governance" (*The Architecture Journal*, Vol. 1, No. 2—see Resources) are four different patterns of governance relationships between the supply and demand sides, only one of which requires asymmetric governance. The other three use variations of vertical coordination (see Table 3).

If we adopt a platform approach, then we have to hold the four areas shown in Figure 2 in relation to each other in a way that is aligned to the WHY—in governance terms, the endogenous coordination imposed on the service by the trust (the HOW) has to be explicitly aligned to the particular form of exogenous coordination imposed on the patient's demand by his or her condition (the WHY).

The governance cycle shows how two different kinds of standardization (designed to reduce certain types of risk) have the effect of rotating a business or system away from asymmetric governance, while two different kinds of customization (which reintroduces certain types of risk) may give the business or system access to the potential rewards of engaging with asymmetry (see Figure 3).

Where are the rewards commensurate with the risks in each case? We have to be able to understand how these different forms of governance are present or excluded within a supplying organization as it changes its relationship to demand, in response to changing competitive and demand circumstances. In Figure 3, the four forms are arranged to show the cycle discussed in our article "The Metropolis and SOA Governance" (see Resources). This cycle makes clear the transitions between each form, two of which require standardization, and two of which require customization:

- The move from destination to product involves reducing the exposure to integration risks by externalizing the exogenous risks.
- The move from product to cost involves reducing the exposure to the technology and engineering risks by standardizing the business model.
- The move from cost to custom involves increasing the exposure to integration risks again, but only the endogenous ones.
- Only the move from custom to destination faces the business with the exo-interoperability risks.

Where in the cycle the business needs to be will depend on how it chooses to balance risks and rewards. As Prahalad and Ramaswamy argue in "The Future of Competition" (see Resources), as demand has become increasingly asymmetric, so it has become increasingly critical for businesses to develop scalable models that can work in the destination quadrant as well. From an understanding of the whole cycle shown in Figure 3, it can be decided what forms of change are needed to capture the demands, and therefore what new forms of risk need to be mitigated in the pursuit of the rewards. While a lot is known about how to manage three of the forms of governance in this cycle, the difficulties of being able to transition through the fourth represents a major obstacle to the continuing growth of a business when faced with growing demand asymmetry.

In the Workshop

Turning to an orthotics case example, a clinic had been located within the Acute Trust to provide it with its own orthotics service (comparison), enabling the demand to be standardized to just those forms of demand that came from the consultants. Over time, the service itself and its budgets were standardized to align them to these forms of demand (cost). As a result, GPs had to refer patients through consultants, even when there was no real necessity to see the consultant, just to get access to the service. As a consequence, limited numbers of referrals were allowed directly to the service where the patient was known to the trust because of previous appointments (custom). The Primary Care Trust was responsible for all the patients in the catchment of the Acute Trust, however, and many of them were not receiving the service they needed. The challenge, therefore, was to enable the service to support these needs directly (destination). The Acute Trust resisted this support because of the need to fund such a service differently; and it was difficult for the Primary Care Trust to initiate because they didn't have the appropriate means for managing the balancing of the costs and risks of such a service. Put another way, new forms of asymmetric governance had to be developed.

> "THE ASYMMETRY OF GOVERNANCE EXPLAINS A LOT OF THE DIFFICULTIES PEOPLE HAVE BEEN EXPERIENCING WITH SOA"

To work with these issues within a supplying organization, we have developed a workshop process that distinguishes four teams: blue, white, red, and black, and is designed to unpack and articulate the different forms of interoperability risk associated with each of the quad-

Table 4 Four workshop roles

Team	Team style	Team focus	Interoperation	Risk	Asymmetry
Blue (WHAT)	We do the business	The capabilities that our side want to use	Endogenous	I *Behavior* of technology	First
		What our side is capable of doing		II *Engineering* of outputs	
White (HOW)	We are the *referees* of ...	... what it is in *our* interests to do		III *Integration* of business	Second
Red (WHOM)	We have the demand	What the demand side is capable of demanding of us	Exogenous	I *Execution* of solution	
		The way the demand side makes use of what it demands		II *Planning* of demands	Third
Black (WHY)	We anticipate ...	... what are all the possible scenarios under which the red team demands might arise		III *Intentions* within context of use	

rants. The workshop process is deliberately based on a military metaphor, but has been adapted for use by commercial/civilian organizations as well as military ones.

The typical objectives of the workshop are to learn collectively how these roles work in relation to one another in this specific organization at this time; to discover the extent to which this organization lacks capability in respect of these roles, and to start to develop this capability; and to obtain a snapshot of the current organization of demand facing this organization.

Possible uses of this workshop process provide a way of approaching a number of issues relating to the impact of asymmetric forms of demand including: business strategy, organizational redesign, SOA design, SOA governance, security analysis, and governance (see Figure 4).

In our example, the blue team was the service, the white team was the Acute Trust, the red team was the patient and his or her GP, and the black team was the Primary Care Trust acting in the interests of the patients within its catchment. Table 4 shows how these teams relate to the different levels and asymmetries. The workshop works this way:

- Facilitating the recognition of the part played by each team, and the particular forms of risk each faced.
- Understanding the consequences of the presence/absence of certain colors for the governance process.
- Facilitating the conversations between all four colors in understanding the system as a whole and their interdependencies in managing risks.
- Using this understanding to develop strategies for managing the risks and agreeing on the basis on which white must determine what is in its interests.

Resources

Boxer Research Limited
www.asymmetricdesign.com

"From Push to Pull: Emerging Models for Mobilizing Resources," John Hagel and John Seely Brown (October 2005)

"Metropolis and SOA Governance," *The Architecture Journal 5*, Richard Veryard and Philip Boxer, Vol. 1, No. 2, (Microsoft Corporation, 2005)

"Special Issue on SOA Governance," CBDI Journal (November 2005)

"Taking power to the edge of the organisation: role as praxis," Philip Boxer and Carole Eigen, ISPSO 2005 Symposium, (2005)

The Future of Competition: Co-Creating Unique Value with Customers, C.K. Prahalad and Vebkat Ramaswamy (Harvard Business School Press, 2004)

The Nature of Order, Christopher Alexander (Center for Environmental Structure, 2003)

The World Is Flat: A Brief History of the Twenty-First Century, Thomas L. Friedman (Farrar, Straus and Giroux, 2005)

"What is the emotional cost of distributed leadership?" Working Below the Surface: The Emotional Life of Contemporary Organizations, Clare Huffington et al., Tavistock Clinic Series, (H. Karnac Ltd., 2004)

Teamwork

The focus of the workshop is ultimately on the particular issues facing the white team in how it exercises governance:

- White constrains the way blue behaves in white's interests. Under conditions of asymmetric demand, white will have to choose to underdetermine blue and allow blue distributed leadership in the way blue responds to red within its particular black contexts.
- White therefore has to understand black to grant appropriate underdetermination to blue to enable blue to satisfy red.
- In these terms, the symmetric governance of white becomes asymmetric as it addresses explicitly the consequences of black for how blue responds to red.

In this case the challenge was to make the asymmetric governance of the relationship between the service and the patients in the Primary Care Trust's catchment feasible, which in turn meant creating horizontal transparency: it had to be possible for the clinic to give an account of the way it aligned its treatment of a particular patient to the particular nature of that patient's condition and its outcomes. Thus, the Primary Care Trust was buying treatments of patients, rather than fixed numbers of treatment episodes, which involved creating a platform that could support this relationship and changing the procurement protocols to reflect the changed axis of accountability. In the article "The Metropolis and SOA Governance" (see Resources) we outlined the role played by this platform in support of a different form of governance. In an upcoming article we will address the analytical challenges involved in designing such a platform so that it is properly aligned to an asymmetric form of governance.

The asymmetry of governance explains a lot of the difficulties people have been experiencing with SOA. Many organizations will have enough short-term stuff to do anyway, without getting into this area, but we are starting to see organizations that are willing to take these challenges on board, and we are enjoying helping them.

Up to the latter part of the twentieth century, it could be assumed that the great majority of time spent by a business would be in the symmetric positions in the cycle. Only the generation of new business propositions needed to take place in the *destination* quadrant. The challenge presented by the twenty-first century is that these proportions are being reversed, so that although the symmetric positions remain important in *harvesting* the value of components within users' platforms, the greatest value will increasingly be created in the asymmetric part of the cycle. Developing an asymmetric capability is therefore of great importance to businesses increasingly competing in a flat world (see Friedman in Resources), in which component activities are outsourced under the pressures of globalization, digitization, and intensifying supply-side competition. •

About the Authors

Philip Boxer is a strategy consultant based in the UK.

Richard Veryard is a writer, management consultant, and technology analyst based in London, UK.

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

Come visit the Journal's new home at **www.ArchitectureJournal.net** live this December. The new site contains a full library of articles from previous Journal issues in addition to upcoming highlights of our next issue. Browse the content today and post comments and letters directly to the editor!

Now live at www.ArchitectureJournal.net!

Microsoft®

ARC

Apply Topic Maps to Applications

by Kal Ahmed and Graham Moore

Summary

Topic maps provide a fairly simple but very powerful metamodel for the representation of knowledge models. In "An Introduction to Topic Maps" (The *Architecture Journal*, No. 5, 2005), we presented how a combination of topic maps and architecture design patterns allow developers to build reusable components for applications. Now we'll take a look at some of the current and potential application areas for topic maps. As topic maps are primarily of use when integrated with other systems, we'll also discuss access to topic map information using a variety of Web services architectures from a traditional RPC-like, client/server interaction to syndication models where either a push model or pull model can be used to interchange updates to a topic map.

The principal purpose of topic maps is to allow the expression of a domain knowledge model and to enable that knowledge model to be connected to related resources. Within this broad remit we can identify several common applications for topic maps in an enterprise.

Most organizations are now publishers of resources to some extent. For some, publishing information is the core business, and for most other organizations the information they publish is a key part of their communication to their customers and partners. Publishers of information face a number of challenges that can be addressed with topic maps.

For a large corpus, the search engine is often the only way for newcomers to find what they are looking for. Traditionally, search has been driven by content keywords or by full-text indexing. Topic maps offer the alternative of indexing and searching against topic names, and then using topic occurrences to present links to all content related to the topics found by the search. Each topic in a topic map represents a single concept but can be assigned multiple names, allowing the topic map to store scientific and common-usage names, common misspellings, or translations for concept names. In addition, a topic map may also store semantic relationships between terms that can inform a search in several ways:

1. This semantic information could be used to provide alternate search term suggestions to a user, or even to transparently expand the entered search term. For example, with the appropriate associations a topic map could expand a search term such as "Georgian" into "17th Century AND England."
2. Topic typing or topic associations could be used to disambiguate distinct concepts with the same term (homonyms) based on their context in the topic map.
3. After executing a search that returns multiple resources, those resources could then be grouped according to their classification in the topic map, allowing users to more clearly see the different ways in which their search phrase has been satisfied.

Benefits for Publishers

One distinct advantage to search optimization is that a search facility that only queries a topic map is significantly easier to tune. For example, if an electrical retail site discovers that a new search term "PVR" has become popular among customers seeking hard disc video recorders, the topic map that drives site search could be modified to add the term "PVR" to the topic for "Hard Disc Video Recorders." The content connected to that topic would not have to be modified at all; the search term "PVR" will now hit the "Hard Disc Video Recorders" topic and result in the related resources being returned.

> "THE KEY ARCHITECTURAL DECISION IN IMPLEMENTING TOPIC MAPS AS PART OF A PUBLISHING SOLUTION IS HOW THE TOPIC MAP SYSTEM IS INTEGRATED WITH THE CONTENT MANAGEMENT SYSTEM"

Link management. One of the principal ways of keeping a user on a site is to present related links that the user can follow to find more information on a subject. Maintaining such links manually is error prone and requires a trade-off between constantly updating the links or accepting that older content will have equally aged links to related content. The nature of topic maps as an index of resources enables these sorts of links to be managed almost automatically.

There are many different approaches to presenting related links using a topic map, but in essence they all consist of traversing from the resource to the point(s) in the topic map where that resource is referenced, and then traversing the topic map in some way and then extracting the list of resources that are referenced from the

end of the traversal. Managing related links in this dynamic manner has the distinct advantage that related links are always up to date; that the list of related links can be generated based on the indexing of previous content; and that the logic for extracting related links can be modified without the need to modify either the resources or their indexing.

Support multiple routes to content. As discussed in our previous article, topic maps can be used to model many of the traditional content indexing and finding aids such as hierarchical and faceted classification. In fact, a single topic map can contain many such indexes. Creative use of multiple indexes can allow a user to find content from many different entry points. For example, on a site publishing financial analysis, a user might find a particular report by a drill down through market sectors to a company and then to the report, or by geographical region to a country related to the story, or even by traversing personalized topics that represent his or her own portfolio or interests.

Serving multiple audiences. Topic maps provide great flexibility to those organizations that need to provide access to different levels of user. In the first instance, the concepts can be given names familiar to each audience. For example, on a site giving information about drugs, the topic map could provide the clinical name of a drug for doctors and the trade name for patients as different names on the same topic. Going further, the topic map can contain both full and simplified domain models that are combined and can optionally overlap. The principal feature of topic maps to support this requirement is the use of scope to specify the context in which a given association between topics is to be presented to a user.

Application Architectures

The key architectural decision in implementing topic maps as part of a publishing solution is how the topic map system is integrated with the content management system. Integration needs to be considered from two aspects: integrating content indexing and content creation, and publishing topic map information with content.

While the creation of a domain knowledge model might be in the purview of a librarian or small set of domain experts, the publishing solution should allow for the classification of content against the domain model to be performed by those responsible for the content, either authors or content editors. Ideally, classifying and indexing the content against the topic map should be made a required part of the content creation/approval life cycle, with classification reviewed as part of the editorial process. A well-designed application will also make use of patterns such as the patterns we discussed in "An Introduction to Topic Maps" (see Resources).

Patterns enable a librarian to make changes to the domain model that reflect changes in the structure or focus of the content without requiring any downstream changes to the management or presentation code. For example, the patterns for hierarchical and faceted classification schemes allow new hierarchies and new classification facets to be introduced in the topic map and to be recognized and displayed automatically by the presentation layer.

Figure 1 shows a simplified block architecture for a Web site that makes use of a topic map. The topic map is managed by a topic map engine component that is populated by the information architect and the content creator roles. When publishing the content there are two possible approaches to the integration of information from the topic map. In the first approach, the content management system (CMS) provides the site structure and one or more regions on a page into which information from the topic map can be added. In the second approach a part of the topic map domain model is used to drive the site.

Figure 1 Sample architecture for a topic maps-driven Web site

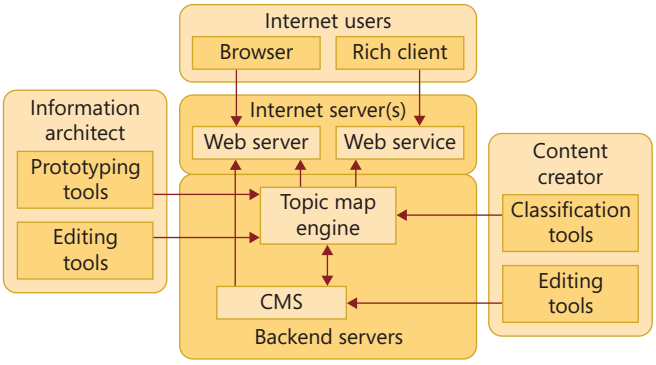
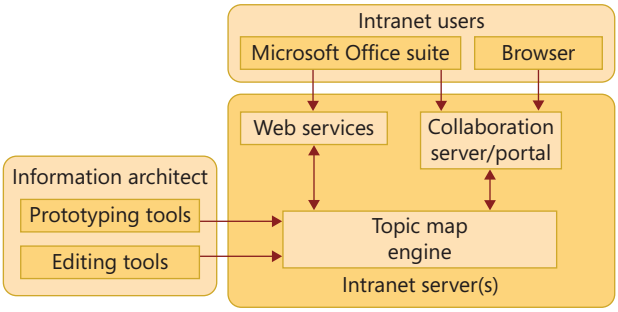


Figure 2 Sample architecture for topic maps on the intranet

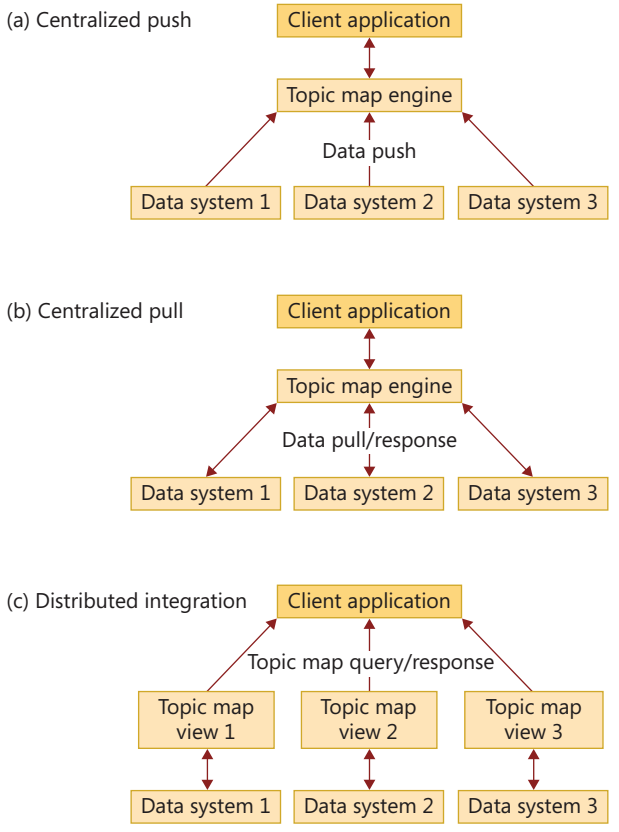


The former approach is most appropriate with CMSs that have strong site-management features or that make use of site structure to implement access controls or other features. The latter approach, however, can be used to create a flexible site structure that can be modified more easily to take account of changes in the way that content indexes are organized. In addition to structuring the content in the CMS, the topic map engine can serve as a useful index of all content available through the Web site. This index can be made available through a Web services interface to rich clients such as an RSS reader or the Research Service in Microsoft Office (see Figure 1).

Topic maps can support a corporate knowledge management application principally by providing a repository for capturing the domain knowledge model. The flexible metamodel provided by topic maps allows new concept types and relationship types to be introduced into the knowledge model with minimal effort, enabling the knowledge model to keep pace with changes in the business.

In addition to maintaining the knowledge model for the domain, the topic map can also be used to index resources on the intranet. In this respect, the topic map provides similar benefits to those described for general resource publishing, but with the development of collaboration systems such as SharePoint, the extent of what is available through the intranet is rapidly increasing. These systems

Figure 3 Three enterprise information integration architectures



make it very easy for users to create new intranet content and to share content related to a project, but this sharing can quickly lead to an intranet overloaded with content in which it is difficult to find relevant information and almost impossible for a newcomer to get acquainted with. An index of the intranet content based on topic maps can not only help experienced users find relevant content regardless of its location, but the high-level domain model embodied by the topic map can also be useful in giving newcomers an overview of the activities of the organization.

Content Connections

A significant part of the problem for corporate knowledge management is that users have to work with many tasks that do not have a one-to-one correlation to a piece of content. For example, a user may work on a case or be part of a project or on a cross-functional team or take part in a meeting. All of these tasks may have related content—case notes, project documentation, meeting minutes, and so on—but very often they have no real identity themselves. While keywords can be used to connect content items to such concepts, a simple keyword tells users nothing about what makes the content relevant to the keyword or about relationships among keywords.

A topic map provides a model for defining noncontent items such as people, projects, and places as topics. Once these topics are created, they can then be connected to content items. Topic maps provide two ways of connecting to content. The first is by creating a second topic that represents the content and then using an association. The second is by using an occurrence that points directly to the content items. The model can also be used to lift important relationships among noncontent items out of content into the topic map to provide an overview of the functions of the organization.

For example, a project may have many participants and may be carried out for a particular customer. By making the participants and the customer into topics in the domain model, it would then be possible to quickly locate other projects for the same customer, or to list the products that have been licensed to the customer, or to find the other project commitments of a team member.

A topic map can be used as a knowledgebase by itself. A simple application would be a Web portal with an interface that allows users to create their own topics and associations and to browse the topics and associations created by others. Users could also add links to internal or external resources as occurrences, or the interface could allow users to create or upload content to the portal itself. Indeed, these are all common features of many generic topic map editing applications. However, the most benefit comes when

> "HAVING CREATED TOPICS AND CATEGORIZED CONTENT, THE FINAL ASPECT TO THE USE OF TOPIC MAPS IN A CORPORATE KNOWLEDGE MANAGEMENT SCENARIO IS ACCESS TO THE TOPIC MAP INFORMATION"

the topic map is integrated into a collaboration system that can provide content management, event management, discussion forums, and other features. As with integration for resource publication, a successful implementation requires that the topic map functionality should be made part of the normal interaction with the collaboration system.

Categorizing content is always going to be seen by some users as an additional burden. This burden can be reduced by using schema-driven classification to minimize the decisions that users need to make about the classification of items, and by making use of the topology of the collaboration system (for example, all documents placed in the folder "Project X" get tagged automatically as being related to the project "Project X"). At the other side of the equation, just a minimal amount of classification can quickly generate benefits for users by bringing otherwise disparate content together, and just a single person working to categorize the content produced by their project or department can result in a better knowledgebase for all users. The inherently flexible nature of ontology based on topic maps makes it very easy to start small, focus on the core ontology needed to address the requirements of just that project or department, and then to subsequently scale up as the project shows a return on investment.

Delivering Knowledge

Having created topics and categorized content, the final aspect to the use of topic maps in a corporate knowledge management scenario is access to the topic map information. This aspect is the area where a topic map can really shine. The topic map can provide a structured, high-level domain model that is easily communicated over a Web ser-

vices interface, giving huge potential for integration into desktop applications. For example, we have recently developed a Web service that implements the Microsoft Office Research Service interface to provide a user with the ability to search and query a topic map from within Internet Explorer and their Office applications.

Figure 2 shows a possible architecture for such a knowledgebase application. The information architect sets up the basic taxonomy and classification schemes for the knowledgebase. The users then work with the collaboration server and populate and extend this taxonomy through browser or rich-client interfaces.

Similar integration possibilities are available using Smart Tags and more detailed service interfaces such as that described later. These integrations allow the information in a topic map to be delivered right to the application in which they are most useful. In addition, the domain model for internal knowledge management can also be used as the basis for integrating data from other enterprise systems as we will describe shortly.

Topic maps are most frequently used as an adjunct to CMSs, which is not surprising given the features that they bring in terms of content organization, indexing, and searching. However, topic maps can also be used as the foundation for integrating all sorts of data sources.

The key to enterprise information integration using topic maps is to define a core ontology and then to map data from each data system into that ontology. For example, an ontology might contain the concepts of a Customer and an Order, but it is the Customer Relationship Management (CRM) system that contains information about the customer's calls to the help desk and the order-tracking system that contains information about the status of the customer's orders. An approach to integrating these systems based on topic maps could be either centralized or distributed. These different approaches are shown in Figure 3.

In a centralized system, each data system provides information about the entities that it manages to a central topic map. The information may be published by a push from the data system (see part a in Figure 3) or updated using a pull from the central topic map management application (see part b in Figure 3). Other applications can then consume that centralized information as topics, associations, and occurrences, thus shielding consumer applications from a need to know the interfaces required to communicate with each external system. In these systems data replication needs to be handled with care to ensure that it is clear where the *master* for any data item resides.

A Simple Topic Map Web Service

While it is possible for applications to generate topic map data, the range of applications that can create or consume this data source is limited. The idea of the Web service is to open up topic maps to more applications regardless of whether they are publishing or querying information. In addition, a well-defined Web service brings a valuable commonality to the application in which knowledge services interact.

The Web service presented here is comprised of a small set of generic methods that ensure that it can be used in a wide variety of topic map solutions. We briefly describe the Web service methods and provide commentary on their applicability and intended use. This interface is implemented by Networked Planet's Topic Map Web Service WSDL (see Resources), which also implements two further methods that allow more direct access to any hierarchies contained in the topic map.

GetTopicMapNames() – A topic map system can store and manage many topic maps. This operation returns a list of names of the topic maps currently available. In addition, it is possible to have this service aggregate or act as a broker over multiple distributed topic maps. This method provides a way to find out which topic maps are available.

GetTopic(TopicMapName, TopicId) – This operation returns a serialization of the specified topic from the named topic map. The serialization provides the caller with enough information that it can traverse to related topics.

GetTopicTypes(TopicMapName) – This operation returns a serialization of a subgraph of the topic map in which each topic that serves as a type of one or more topics in the specified topic map is fully serialized.

GetTopicBySubjectIdentifier(TopicMapName, Identifier) – This operation returns a subgraph of the specified topic map, in which the topic with the specified identifier as its subject identifier is fully serialized.

GetTopicsByType(TopicMapName, TopicTypeId) – This operation allows users to fetch a list of all the topics that are instances of the given type. The return value is a subgraph of the topic map with each instance topic fully serialized.

DeleteTopics(TopicMapName, TopicId[]) – This operation specifies the unique topic identifier of one or more topics to be deleted from the specified topic map.

SaveTopic(TopicMapName, TopicMapFragment) – This operation can be used to both update an existing topic and add new topics. The topic map data contained in the TopicMapFragment parameter acts as the definitive version of topics and associations. To replace an existing topic or association, that construct in the TopicMapFragment parameter must carry the unique identifier of the construct to be replaced. Any topic or association in the TopicMapFragment parameter that does not carry a unique identifier is added as new data to the topic map.

Query(QueryName, QueryParams[]) – The Web service does not allow the execution of arbitrary queries against the topic maps. Instead, the administrator or developer may configure any number of named queries that are accessible through the Web service API. This operation invokes the named query, passing in parameters that are used as direct value replacements in the query string. The structure of the XML returned from this method depends on the results table structure returned by the query.

The key feature of this interface is that it is almost entirely topic centric. All operations are on and return topics or lists of topics. Using the serialization techniques described here, it is possible to implement this Web service using a document-centric approach with XML data that is easily processed using XML data binding or XPath toolsets.

Figure 4 Topic map broker architecture

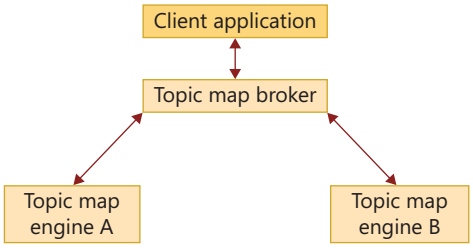
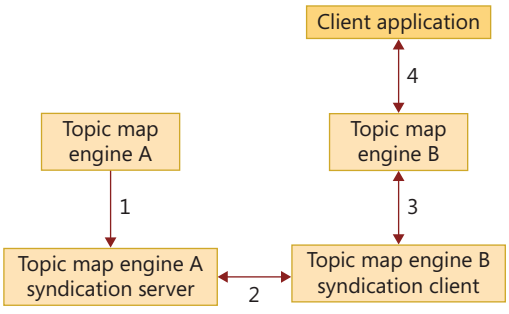


Figure 5 Example syndication architecture for topic maps



In a distributed system, each data system has an integration component that exposes a topic map interface, and consumers contact the systems through that interface (see part c in Figure 3). Again, consumers are only required to understand one interface, but in this case calls to the topic map interface could be translated directly into queries and updates against the underlying information system, which means that there is no data replication issue but can lead to a more demanding integration task.

Identification and URIs

In either a centralized or distributed system, there needs to be a strong emphasis on the identification of the entities that each system manages. It is a simple matter to map entity-unique identifiers such as a customer account number or an order-tracking number into a Uniform Resource Identifier (URI) for the topic that represents that account or order, and with a little care identifiers can be constructed such that there is a common algorithm for converting between the URIs and the entity-specific identifiers.

The principal benefit of using a topic map for this sort of integration exercise is the flexibility that it allows for the modeling of the integrated data. As the model is simply data in a topic map and not a schema for any underlying system, new types of entities can be introduced without any need to alter any existing integration interfaces. Additionally, by representing core business entities as topics in a corporate ontology, the way is made clear to perform and integrate data from enterprise information systems into the knowledge management system or even, where appropriate, out onto the Web.

All applications that make use of topic maps, including all of those presented previously, require some method to access the topic map information. Most applications also require a persistent store for topic map data and in-process API to access and manipulate that data, but these are beyond the scope of this discussion. What is of more interest to the solutions architect is the ability to access topic map information through Web services calls. Topic maps have several properties that make them highly conducive to access through Web services. Let's take a look at them.

Topic addressability. Topics can be assigned server-unique (or if necessary, globally unique) identifiers. The topic addressability property allows us to build client applications that can track the provenance of the topic information they consume. Topic addressability also allows us to traverse association or typing information in the representation of a topic. For example, a query from a client might retrieve topic A with an occurrence that it is typed by topic T. A second query from the client can then retrieve all information about topic T. In the sample Web service described in the sidebar, "A Simple Topic Map Web Service," topics can be retrieved by their unique identifier using the GetTopic() method.

Concept addressability. The concepts that topics represent can be assigned separate unique identifiers, allowing a query against multiple servers for information relating to a specific concept. Concept addressability is the key to supporting the distributed creation and maintenance of topic maps and the subsequent aggregation of the information in those maps. Because a concept (such as Person, Place, Fred Jones, or Birmingham) can be assigned its own URI separate from the system identifier of the topic that represents that concept, multiple systems can provide information about the same concept and use the concept identifier as the key used in aggregation.

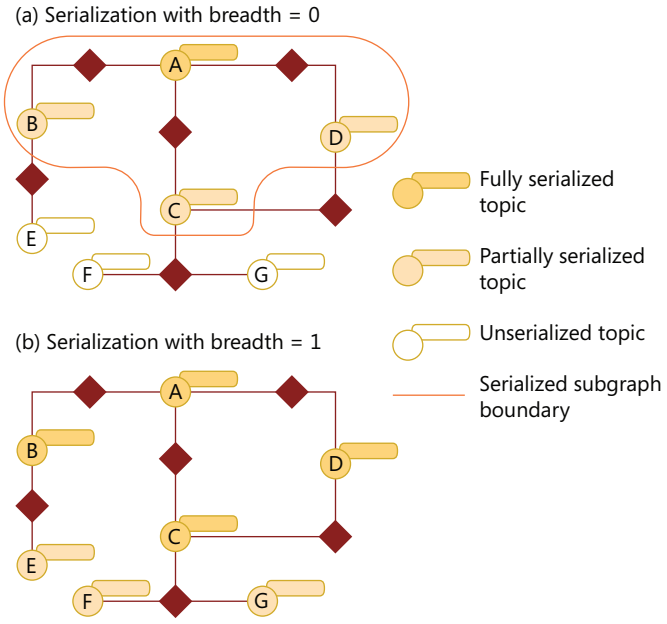
For example, a query from a client might return a topic T with a subject identifier I. The client could then query a second topic map server for any topic with the subject identifier I to expand the amount of information known about the concept represented by that identifier. The advantage of concept addressability is that the client application does not have to know the system-specific identifier of the topic with the subject identifier I. In the sample Web service, this form of addressability is provided by the GetTopicBySubjectIdentifier() method (see the sidebar, "A Simple Topic Map Web Service").

> "THE CONCEPTS THAT TOPICS REPRESENT CAN BE ASSIGNED SEPARATE UNIQUE IDENTIFIERS, ALLOWING A QUERY AGAINST MULTIPLE SERVERS FOR INFORMATION RELATING TO A SPECIFIC CONCEPT"

Document-literal topics. Topics can be rendered easily as data structures that make use of globally unique topic identifiers or hyperlinks to represent the relationships among them. In SOAP terms that means that we can derive a simple document-literal representation of a topic. If Representational State Transfer (REST) is your preferred Web service methodology, it is possible to construct a topic representation as a hyperlinked document sent in response to a REST-ful query. We describe the algorithm for producing such a representation shortly.

Standard merging rules. The merging rules of topic maps can be used to combine topic information received from multiple separate sources into a single functioning topic map. For topic map consumers, the merging rules allow consumers to use concept addressability

Figure 6 Topic map subgraph serialization



to find all information relating to a concept and then combine that information into a single topic that is presented to higher levels of the application. This merging allows a client to aggregate information from multiple topic map sources, and to then present an interface that makes it appear that all of this aggregated information has come from a single source. The merging rules also allow concept addressability to be taken one step further as it allows a topic map to declare two concepts to be equivalent by simply including the address of each concept on a single topic.

For example, a client could query for any information relating to a concept with the identifier I, the topic map returns a topic T that has the identifiers I and I′ as its subject identifiers, which tells the client that the source is claiming that the concept identified by I is the same as the concept identified by I′. If the client trusts the source to make this sort of claim, it may then proceed by querying for any information relating to the concept with the identifier I′ and merging this with the information already received for the concept with the identifier I.

Topic maps can be quite easily used with almost any client access architecture. We will examine three common architectures: client/server, broker, and syndication architectures.

The simplest of the topic map access architectures, the client/server architecture, makes use of a Web service interface that exposes operations to browse, query, and update the topic map. (See the sidebar, "A Simple Topic Map Web Service" for a description of one possible interface that consists of just eight methods.) As topic maps can be easily serialized as XML, there is no issue in using SOAP, REST, or even XML-RPC to implement such an interface.

The broker architecture interposes one or more additional servers between the source(s) of topic map data and the consumers (see Figure 4). A broker aggregates results from several servers performing any necessary merging and responding to a client as if the data had come from a single topic map. The aggregation performed by a broker may vary from simply distributing an operation and aggregating the results to more complex aggregation based on the subject identifiers returned from the various topic map sources.

Syndication Architecture

The syndication architecture makes use of REST to distribute changes to a topic map model. The server that maintains the model simply writes changes as transaction documents containing serialized topic map subgraphs. These transaction documents are then picked up by consumers and applied to their local cache of the model. Syndication architectures work particularly well for distributing ontology topic maps that are relatively stable or for distributing topic maps that index content that is published to a regular schedule (for example, a news Web site's updates). The XML serialization of topic map subgraphs means that this architecture can make use of syndication standards such as ATOM or RSS 2.0 to distribute transaction data.

> "THE TOPIC MAP CAN PROVIDE A STRUCTURED, HIGH-LEVEL DOMAIN MODEL THAT IS EASILY COMMUNICATED OVER A WEB SERVICES INTERFACE, GIVING HUGE POTENTIAL FOR INTEGRATION INTO DESKTOP APPLICATIONS"

Figure 5 shows an example of how a syndication system might work to keep synchronized two topic maps managed by different engines:

1. A change made to topic map engine A is written as an XML document to the syndication server. The syndication server makes a feed of recent transaction documents available to syndication clients.
2. The syndication client checks the feed on the syndication server and requests the transaction(s) it needs to apply.
3. The syndication client processes the transaction documents received from the syndication server and applies the changes to topic map engine B.
4. Clients interact with the updated topic map on topic map engine B, which is now synchronized to the last known state of the topic map on topic map engine A.

Step 2 could also be performed by a push of syndicated transaction information from the syndication server to the client; the mechanism used would depend on the application requirements and the facilities provided by the syndication mechanism used.

Applications that access topic maps frequently require a result set that consists of a single topic (or a list of topics) that match a query. However, the topic map model is essentially a graph model in which topics are connected either through associations or through typing relationships, so when returning a topic it is necessary for the server to provide some context for the client. Essentially a server is required to extract a subgraph from the topic map graph.

Topic Map Serialization

In our experience the best way to go about serialization is to start with the concept of two types of topic serialization. A full serializa-

tion presents all of the information directly connected to a topic: its types, identifiers, names, occurrences, and all of the associations in which it participates. A partial serialization presents a minimal set of information that can be used by a client. Exactly what information is present in a partial serialization may vary from one implementation to another; for some implementations just a unique topic identifier might be sufficient, but other implementations may require all identifiers present on a topic plus a name or occurrence of the topic chosen according to some algorithm. The principal guide in determining what is present in a partially serialized topic is that a partial serialization should not contain any references to other topics; thus, the partially serialized topics form the leaves of the subgraph returned by a serialization.

With these two definitions, subgraph extraction is a relatively straightforward task. To extract a small subgraph centered on a topic, perform a full serialization of that topic. For each topic that the fully-serialized topic references, create a stub serialization of the topic, and replace the reference to it with a reference to the stub.

Larger subgraphs can be extracted by specifying a breadth parameter that defines a maximum number of associations to traverse. Every topic that can be reached by traversing a number of associations up to the breadth parameter from the starting topic should be fully serialized, and all other referenced topics serialized as stubs. Figure 6 shows an example of the serialization of a topic map subgraph using two different breadths of subgraph.

Part a in Figure 6 shows that the serialization of topic A is performed with a breadth of 0, so only topic A is fully serialized. However, to serialize the associations that topic A participates in, each of the topics B, C, and D must be partially serialized. Note that topic C has an association to topic D, but because C is only partially serialized, that association is not part of the serialized subgraph even though its ends are. Part b in Figure 6 shows the serialization of topic A with a breadth of 1. In this serialization, topic A and all topics that can be reached by traversing one association starting from topic A (that is, topics B, C, and D)

> "IN OUR EXPERIENCE THE BEST WAY TO GO ABOUT SERIALIZATION IS TO START WITH THE CONCEPT OF TWO TYPES OF TOPIC SERIALIZATION; FULL SERIALIZATION PRESENTS ALL OF THE INFORMATION DIRECTLY CONNECTED TO A TOPIC"

are all fully serialized. To perform the full serialization of topic B, topic E must be partially serialized, and to perform the full serialization of topic C, topics F and G must be serialized. Topic C is only connected to topics that are fully serialized, so no additional information is required to serialize the associations it participates in.

Having identified the subgraph of the topic map to be extracted, all that remains is to serialize the data in that subgraph as XML. Although the topic maps standard does define an XML interchange syntax, it is a syntax better tuned for authoring and interchange of whole topics and does not provide any syntax for distinguishing between fully and partially serialized topics. It is therefore necessary to define a separate schema for serialization of topic map subgraphs. In designing our own schema (www.networkedplanet.com/2005/01/topicmap/data/TopicMapFragment.xsd) we also took the opportunity to simplify the XTM syntax to remove authoring convenience features, resulting in a schema that can be more easily processed by XML data-binding tools and by XSLT/XPath. •

Resources

"ISO/IEC 13250 Topic Maps," Second Edition (2002)
www.y12.doe.gov/sgml/sc34/document/0322_files/iso13250-2nd-ed-v2.pdf

"ISO/IEC JTC1/SC34 Information Technology—Document Description and Processing Languages"
www.isotopicmaps.org/sam/sam-xtm/

Networked Planet
Topic Map Web Services
www.networkedplanet.com/technology/webservices/intro.html

Topic Map Web Service WSDL www.networkedplanet.com/2005/01/webservices/tmservice/TMService.wsdl

"White Paper: Topic Maps in Web Site Architecture," (Networked Planet Ltd., 2005)
www.networkedplanet.com/download/tm-website-architecture.pdf

Techquilla
"TMShare – Topic Map Fragment Exchange in a Peer-to-Peer Application," Kal Ahmed
www.techquila.com/topicmapster.html

XTM TopicMaps.org
XML Topic Maps (XTM) 1.0
www.topicmaps.org/xtm/1.0/

About the Authors

Kal Ahmed is cofounder of Networked Planet Limited (www.networkedplanet.com), a developer of topic map tools and topic maps-based applications for the .NET platform. He has worked in SGML and XML information management for 10 years, in both software development and consultancy, and on the open source Java topic map toolkit, TM4J, as well as contributed to development of the ISO standard.

Graham Moore is cofounder of Networked Planet Limited, and he has worked for eight years in the areas of information, content, and knowledge management as a developer, researcher, and consultant. He has been CTO of STEP, vice president research and development at empolis GmbH, and chief scientist at Ontopia AS.

Walt Disney World Swan Hotel
May 14-18, 2006

New Date — Same Incredible Content

VSLive! Orlando arrives early this year as weather concerns have moved the event to May. Microsoft insiders and industry veterans will return to Orlando for dynamic sessions, keynotes and product demonstrations on the latest innovations in development.

VSLive! Brings You:

- Informative Keynotes from major industry players
- In-Depth Workshops on essential development topics
- A full slate of Microsoft-led sessions on .NET Day
- ASP Live! - Intensive server-side and Web development techniques for both VB and C# practitioners
- Smart Client Live! - Practical advice on smart-client development in VB and C#
- SQL Live! - Valuable tips on optimizing the performance and reliability of SQL Server
- An insider's tour of Windows Communication Foundation (aka Indigo)
- Virtual Tracks on a range of essential programming topics; and more

Save the Dates - More Upcoming Events

VSLive! Las Vegas
- April

VSLive! Toronto
- April

VSLive! New York
- September

VSLive! Boston
- September

VSLive! Chicago
- September

FTP FAWCETTE TECHNICAL PUBLICATIONS

Register by March 8th and SAVE $300

Call 800-848-5523 today or visit us online at www.vslive.com/orlando

Design and Implement a Software Factory

by Mauro Regio and Jack Greenfield

Summary

In this approach to interoperability we share the experience gathered in designing and implementing a software factory for health care systems based on the Health Level Seven (HL7) standard. We discuss the long-term vision and the scoped-down proof of concept developed so far. We also outline the challenges encountered in our project and the opportunities to widen the scope of the approach to different industries, and, in general, the opportunities to support business-to-business collaboration.

The objective here is to share the experience gathered in designing and implementing a software factory based on Health Level Seven (HL7), a standard for interoperability among health care organizations. We started the work almost one year ago with the high-level specification of the factory, producing a first version of its schema and solution architecture (see Resources).

In its initial phase, the factory targeted the design of HL7 collaboration ports, which are systems designed to be deployed at the edge of IT systems of health care organizations, and enable health care applications to collaborate in conformance with business and technical protocols standardized in HL7 Version 3, using a Web service-based communication infrastructure.

In the second phase, we implemented a first—scoped down—version of the HL7 factory specified in the first phase. The focus of this version is the subset of HL7 collaboration port capabilities necessary to enable communication among health care applications through Web service adapters, in conformance with HL7 Web service profiles (see Resources).

The full scope of the factory, as specified, also included development of enterprise application integration adapters to connect existing applications to collaboration ports and orchestration of business

Figure 1 The production context of an HL7 factory

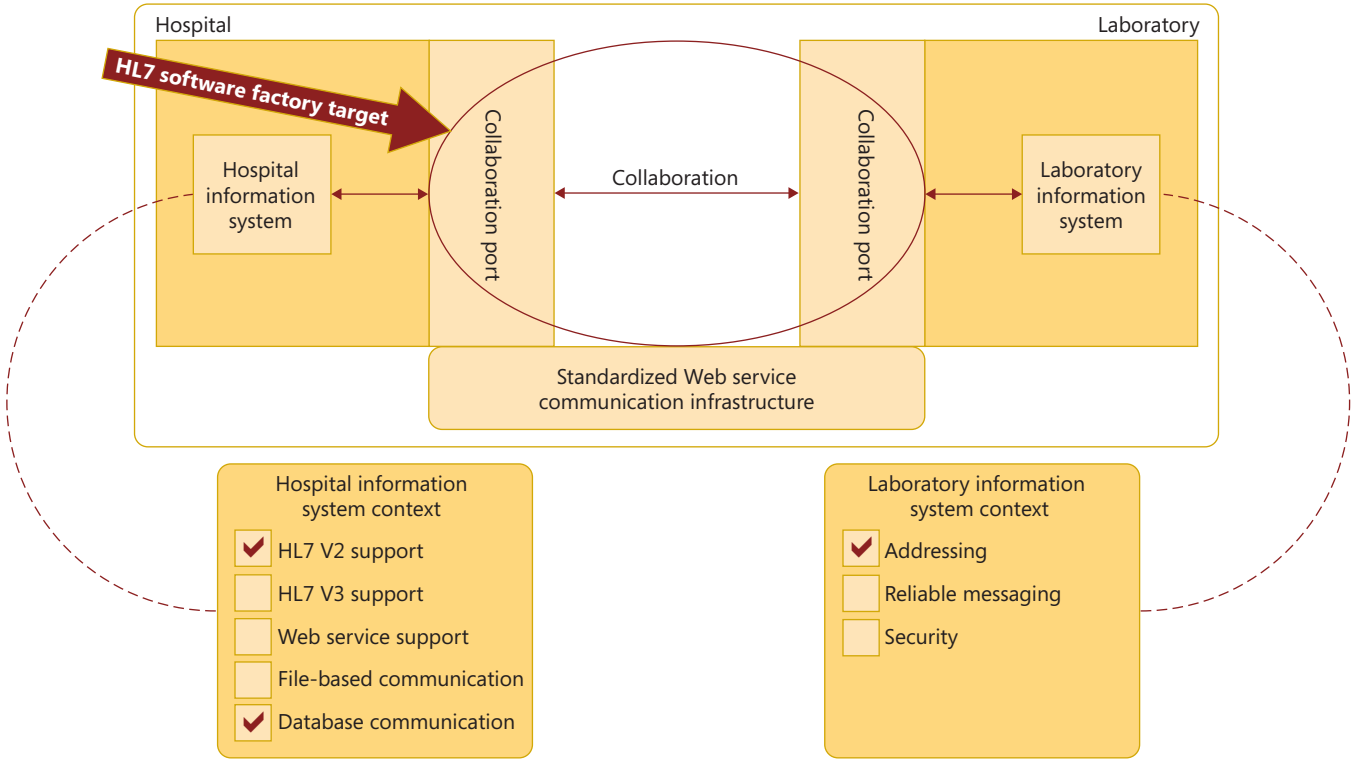
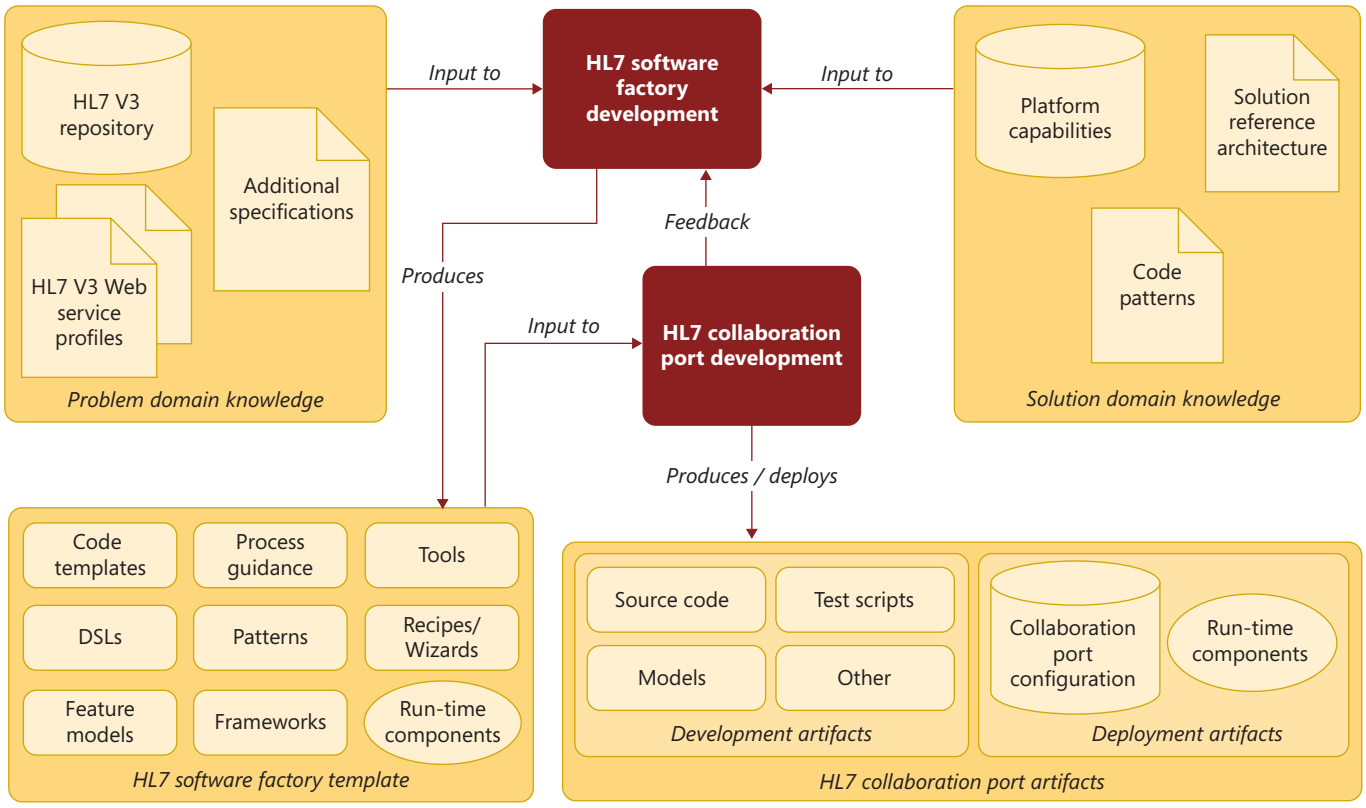


Figure 2 The development context of an HL7 factory



message exchanges realizing a particular collaboration on behalf of line-of-business applications that were not designed to collaborate.

Our experience in designing and developing the HL7 factory has been valuable from two different perspectives. In developing the HL7 factory we encountered some challenges in developing the factory schema, managing the factory configuration, understanding how domain-specific languages would be used, and leveraging the tools available at the time in the development environment.

At the same time, we realized that the factory's scope could be widened from collaboration among health care applications based on HL7 to a more generic notion of collaboration among applications based on standardized (or shared) specifications.

Therefore, we are currently in the process of generalizing the approach proven in the initial implementation of the HL7 factory to design and build what we have called the business collaboration factory.

Software Factories

Software factories use specific domain knowledge, solution architectures, tools, and other reusable assets to help their users produce specific types of software solutions. A software factory is based on three key ideas: a software factory schema, a software factory template, and an extensible development environment.

A software factory configures an extensible development environment, such as Eclipse, Borland JBuilder, or Microsoft Visual Studio Team System (VSTS), using an installable package called a software factory template or guidance package. When configured in this way, the development environment becomes a specialized facility that accelerates the development of a specific type of software solution, such as a user interface or database access layer, or perhaps a whole application in a business domain like health care or homeland security. The software factory template is organized by a model called a software factory schema. The schema defines one or more viewpoints relevant to stakeholders in the production of the target software solutions. Each viewpoint defines the life-cycle artifacts produced or consumed by its stakeholders, the activities they perform against those artifacts, and the reusable assets available to support them in performing those activities.

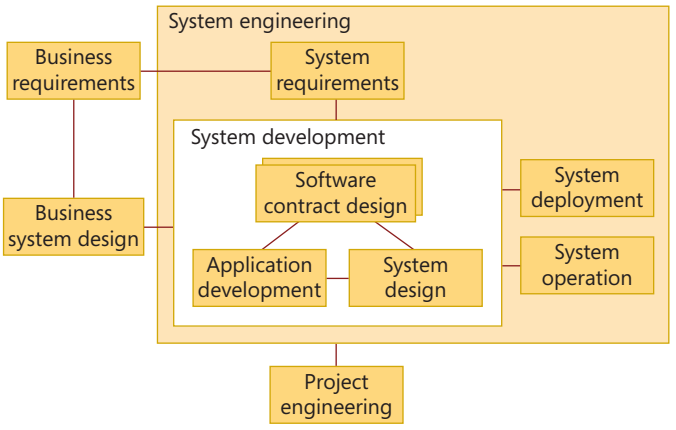
The software factory methodology integrates model-driven development (MDD), component-based development (CBD), and agile development practices, including the use of patterns and pattern languages with models, frameworks and tools (see Resources).

To leverage models effectively for various forms of automation, software factories make heavy use of domain-specific languages (DSLs). DSL technology is much newer than most of the other technologies used in software factories, and relies on families of extensible languages. DSL development tools and frameworks have been under development for some time in academic circles, however, and have recently started to appear in commercial form (see Resources).

The HL7 factory automates the development of systems called collaboration ports, which enable interoperation among systems in the health care domain. Specifically, the solutions produced by the factory aim to:

- Realize interactions defined by the HL7 standard as information exchanges that take place between application roles in response to trigger events. Collectively, these exchanges support the business goals of a specific use case, such as performing a laboratory obser-

Figure 3 System production viewpoint



vation. The factory automates the production of code that implements these interactions by mining information contained in the HL7 Reference Information Model (HL7 RIM).

- Enable application-to-application business collaboration expressed in terms of these interactions over an open standards–based Web service infrastructure that is conformant to a subset of the HL7 V3 Web service profiles, namely the Basic, Addressing, Security, and Reliable Messaging topics (see Resources).
- Enable integration of new or existing applications that were *not* designed: 1) for HL7, version 3; 2) to fulfill a business collaboration; and 3) to communicate over a Web service infrastructure.

It is important to understand the HL7 factory from two different perspectives: production and development. In the production context the factory end products are HL7 collaboration ports. These ports automatically enable disparate health care applications to collaborate behind and across the firewall using Web services, provided that at least one of the applications participating in the business collaboration will deploy an HL7 collaboration port; the other applications may participate through other means, and all of the applications participating in the business collaboration will conform to HL7 V3 standards for message exchange, either natively, or with the help of an HL7collaboration port.

Factory by Template

For a business collaboration between a hospital and a laboratory system, Figure 1 shows where HL7 collaboration ports sit in relation to the systems hosting the interoperating applications. Note that collaboration ports are meant to be highly configurable to enable general dispatching, while also allowing complex message-flow orchestration. Thus, ports like those shown in Figure 1 are configured both in terms of the technical details for a specific implementation and deployment and in terms of HL7 domain definitions and conformance levels.

In the context of development, the purpose of the software factory is to accelerate the specification and implementation of collaboration ports. The factory combines problem domain knowledge supplied by the HL7 Reference Information Model and Web service profiles, with knowledge of the platform technology, solution architecture, and development process supplied by platform documentation and by the factory developers (see Figure 2).

As suggested by the illustration, this knowledge is packaged into numerous assets, which collectively form the factory template. Simply stated, the factory template provides everything required to build an HL7 collaboration port, including reference data and artifacts, such as message schemas; tools, such as adapters generators; and process guidance. The factory template must be installed into an integrated development environment (IDE), namely Microsoft Visual Studio 2005 Team System, before it can be used to produce and deploy HL7 collaboration ports.

As noted previously, the purpose here is to describe what we learned in specifying, designing, and implementing the HL7 factory. We have grouped the information into two categories. The first classifies lessons learned about factory development and usage in general; the second contains insights gained regarding the target domain and with how the factory might be generalized to address a broader set of target domains.

> "WE NEED TO PROVIDE ENTERPRISE ARCHITECTS WITH THE MODELS AND TOOLS NECESSARY TO SUPPORT THE SPECIFICATION PROCESS"

The development and management of the factory schema were the most significant challenges from the inception of the project through to its completion. Producing an initial version of the schema was relatively easy (see Resources). A grid-based approach can be effectively used in this phase, organizing relevant *viewpoints* into a two-dimensional matrix with level of abstraction on the vertical axis and life-cycle phase on the horizontal axis.

However, we quickly realized that the two-dimensional matrix was a fairly inadequate representation of the schema, especially for the fully scoped version of the factory because a) the schema was naturally multidimensional, b) a matrix representation does not capture relationships among nonadjacent viewpoints, and c) the graphs of viewpoints were of different types and depths and unfold into nested graphs of different types and depths.

Nonetheless, working with an amorphous, graph-based representation of the schema required tools that were not available. Therefore, we implemented the factory schema as a set of two-dimensional projections of the relevant viewpoints. Each of these projections—a graph in its own right—details a specific aspect of the factory, effectively projecting it from the multidimensional schema in the same way that a set of tuples is projected from a multidimensional data store to form a two-dimensional view.

Figure 4 Software contract design viewpoint

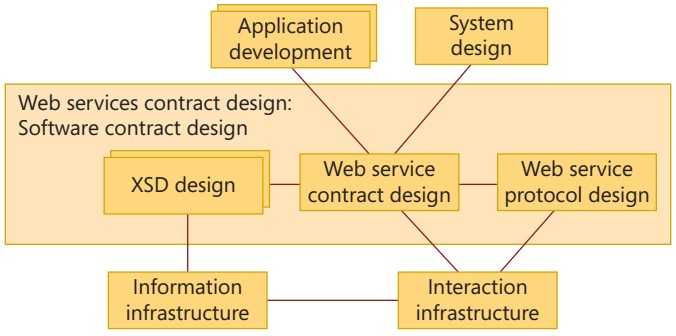
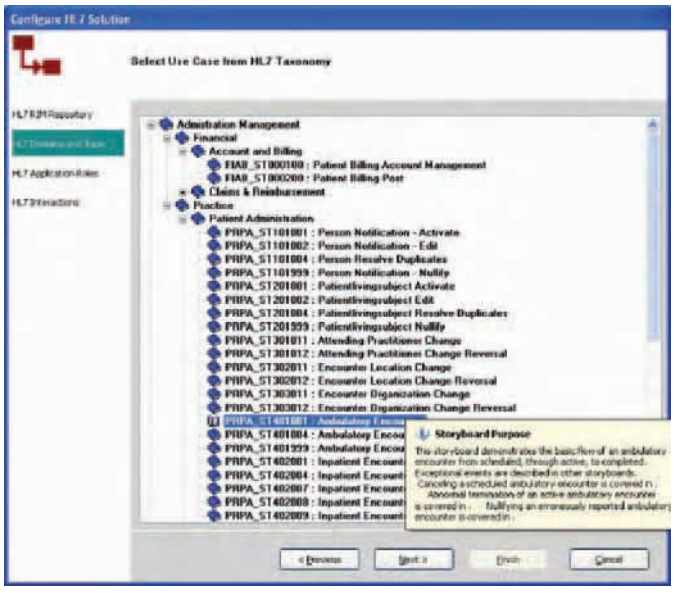


Figure 5 The Configuration wizard's use case selection



For example, Figures 3 and 4 show two viewpoint examples, namely the System Development viewpoint and one particular aspect of it: the Software Contract Design—at a lower abstraction level.

Assigning Tasks

This approach allowed us to produce a version of the factory schema that we deemed complete, that is, it comprehensively specified all the artifacts and tools required in the factory template to produce the products of the factory. However, it left verification of the schema to human inspection and did not allow us to use the schema as metadata to drive the user experience within the IDE.

Configuration management was another challenging aspect of factory development, from two different perspectives.

First, we had to create a configuration XML schema, which would be complete in terms of allowing the expression of all valid combinations of supported features and/or implementation strategies. The XML schema was handcrafted and quickly became a maintenance burden, as it was highly sensitive to changes in the factory schema and template. Second, we had no tools within the target development environment to support the configuration of a specific instance of the factory, or the validation of such a configuration, during product development.

Specification of the product development process was also a challenge in developing the factory. We had no satisfactory way to formally express the process in the factory template, and most importantly no way to inject the process as prescriptive guidance into the development environment.

Although the target development environment did support the creation of tasks, and the assignment of tasks to members of the development team, we had to rely on natural language documents to describe the development process because we had not yet determined how to apply the task-management features to a factory-based, product-development process. In particular, we had not yet determined how to associate tasks with specific assets supplied by the factory template, how to load the tasks from the factory template, or how to configure the tasks for a specific product.

We also found it quite hard to decide whether or not to provide a fully fledged domain-specific language (DSL) for the requirements-gathering phase of product development, especially given that Microsoft has published a set of DSL tools under the umbrella of its Software Factories Initiative.

We knew that a full DSL was not strictly necessary because the relevant use case specifications were already available to us in the HL7 repository. What we really needed was a sophisticated wizard that would help the user choose specific use cases (see Figure 5), application roles, service-interaction patterns (see Figure 6), and other standard data elements and navigate through the repository.

Also, the DSL designer technology from Microsoft was very immature when we started the project, and adopting it would have added significant risk to the project. Although we knew we were missing out on the opportunity to provide more sophisticated automation, and that the alternative—a wizard-based user interface—would be not relevant outside the scope of the factory, we decided not to use the DSL technology in this phase of the project.

Given that we now have a much more consolidated and robust technology preview of DSL tools, we may start to experiment with DSLs in subsequent versions of the factory. Also, even if we did decide to create another wizard-based user interface, we would very likely design and implement it using the DSL tools, instead of developing it from scratch.

The Guidance Automation Toolkit (GAT), another piece of enabling technology under the umbrella of the Software Factories Initiative at Microsoft, proved quite useful.

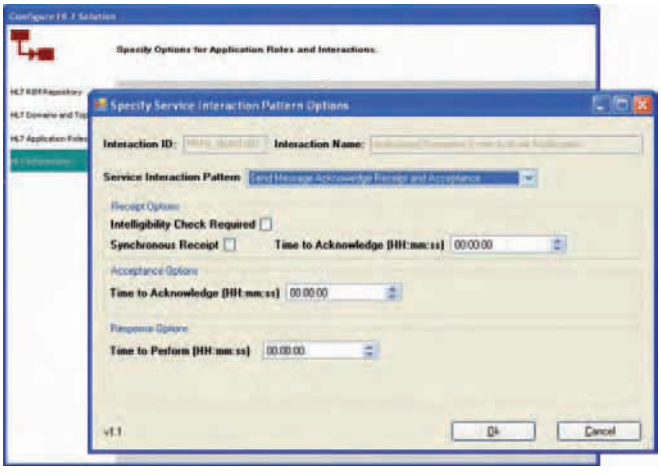
Simply stated, GAT is an extension to the development environment that makes it easy to create rich, integrated user experiences around reusable assets like frameworks, components, and patterns. The resulting guidance packages are composed of templates, wizards, and recipes, which help users build solutions in keeping with predefined architectural guidance.

Broaden the Scope

In our project, we used GAT as the means of packaging and delivering the factory template. It provided an underlying model for the template that was much richer than what the development environment itself had to offer.

Recipes have been provided for activities like HL7 Web Services Adapter creation, execution of the configuration wizard, cre-

Table 6 The Configuration wizard's service-interaction patterns specification



ation of Web services contracts, and automation of the code-creation process.

Unfortunately, the GAT learning curve is quite steep. Its flexibility and customization options are limited and its integration with the IDE could have been better. Nevertheless, we believe that GAT adoption was a key decision that helped ensure project success and significantly reduced the factory development time.

Although we developed the factory to enable business-to-business collaboration among health care applications, it quickly became evident that we could apply such a factory to similar scenarios in other industries. We came to understand that the real scope of the factory should be business collaboration in general, using standardized or predefined specifications of interactions, application roles, events, Web service profiles, and other domain elements, not just business collaboration in the context of HL7.

In retrospect, the reason we initially developed such a factory for HL7 was that the HL7 standard provided a complete set of easily accessible and well-defined domain elements. Of course, many other standards organizations, such as RosettaNet and UNCEFACT have also invested a great deal of effort in specifying domain elements to be used by businesses that want to collaborate using standardized protocols. Interestingly enough, as far as the collaboration protocols and the information they exchange are concerned, these standards look very much alike.

We therefore concluded that it would be feasible not only to build standards-based collaboration systems for other industries, but also to build a collaboration port factory that can be customized using business collaboration specifications for other industries. Of course, there will be a need for various import mechanisms, adapters, and model conversions to work with the different concepts used to describe business-to-business collaboration by different standards bodies. Nevertheless, we think a generic specification could be used to bridge these differences through configuration in a generic business-collaboration factory.

At the same time, we recognize that a more generic factory may also be quite appealing to corporate developers constructing business-collaboration systems inside the firewall, and who want a more formal approach to specifying and implementing those systems to guarantee better alignment between business goals and a portfolio of resulting IT services. However, in this case we would need to provide enterprise architects with the models and tools necessary to support the specification process.

We realize that our factory must support the construction of collaboration ports for various technology platforms. We suspect that this could be accomplished using implementation patterns provided with the factory template to decouple the specification of the collaboration ports from platform-specific implementation details.

Taking the Lead

Our plans are to exploit the opportunity described here to generalize the HL7 factory to form a business-collaboration factory. We think most of the work required to achieve this generalization will reside in these areas:

- Provide models and tools to support the specification of business collaborations, focusing on information schema, business document/messages exchange protocols, and possibly business transactions.
- Define mappings between relevant industry standards and our internal model of business collaboration, and possibly tools for importing the domain elements they define.
- Introduce another level of configuration in the implementation of the collaboration port that will enable factory users to target a wider variety of technology platforms.

Our experience in developing a factory for HL7 collaboration ports has shown that we need to define better frameworks, tools, and processes to specify the factory schema, to manage factory configuration in a flexible and extensible way, and to better understand how and when domain-specific languages should be used. At the same time, initial implementations of extension mechanisms like GAT and DSL have proven their value, filling significant gaps in software factory infrastructure, and pointing to future innovation in that area.

We intend to continue to use and improve these tools in the next version of the HL7 Factory, as well as in the more generic, cross industry, version called the business-collaboration factory. •

Resources

"A Software Factory Approach to HL7 Version 3 Solutions," Mauro Regio, Jack Greenfield, and Bernie Thuman (June 2005)

Domain-Specific Modeling with MetaEdit+
www.metacase.com

Health Level Seven
www.hl7.org

"Advanced Web Service Profiles," Roberto Ruggeri et al.
www.hl7.org/v3ballot/html/welcome/environment/

Institute for Software Integrated Systems
"Model-Integrated Computing"
www.isis.vanderbilt.edu

MSDN
Visual Studio Team System Developer Center
Microsoft Enterprise Framework and Tools Group
"Domain-Specific Language (DSL) Tools"
http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/

Microsoft Patterns and Practices Team
"Guidance Automation Toolkit (GAT)"
http://lab.msdn.microsoft.com/teamsystem/workshop/gat/

Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Jack Greenfield et al. (Wiley, 2004)

"Web Services Enablement for Healthcare HL7 Applications – Web Services Basic Profile Reference Implementation," Mauro Regio (August 2005)

About the Authors

Mauro Regio is an industry solutions architect in Microsoft's developer and partner evangelism architecture strategy team, focusing on large-scale enterprise integration projects and software factories.

Jack Greenfield is software architect, enterprise tools, Microsoft.

Service-Oriented Business Intelligence

by Sean Gordon, Robert Grigg, Michael Horne, and Simon Thurman

Summary

This discussion looks at the similarities and differences between Business Intelligence (BI) and Service Orientation (SO), two architectural paradigms that have developed independently. Here we define an architectural framework that leverages the strengths of BI and SO while defining guiding principles to ensure that the fundamental tenets of each of the component architectures are not broken.

The noun *synergy* means the combined action of discrete entities or conditions such that the total effect is greater than the sum of their individual effects. Service-Oriented Business Intelligence (SoBI) is the synergy of the Business Intelligence (BI) and Service Orientation (SO) paradigms and describes patterns and architecture to accomplish this synergy by providing a best practice implementation framework; the ability to integrate at the most appropriate architectural level; the data modeling of a BI project within the SO strategy of leaving the source systems in place; and a common implementation for data transformations and data logic: data to data, data to service, service to data, and service to service.

BI and SO are broad paradigms. Let's start by defining BI. Data warehousing is a broad discipline that allows for the gathering, consolidating, and storing data to support BI. The components of a successful data warehouse can be summarized as data collection, cleansing, and consolidation (Extract Transform and Load, or ETL) and data storage. BI is the delivery of information to support the decision-making needs of the business. It can be described as the process of enhancing data into information and then into knowledge.

Figure 1 BI and SO views

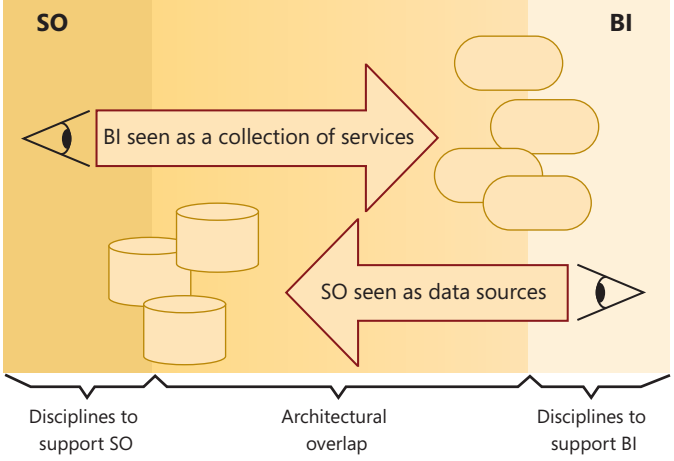


Figure 2 BI view of SO

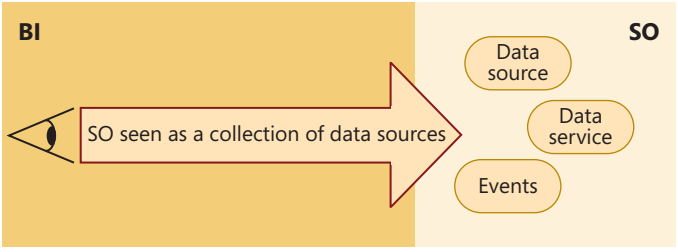
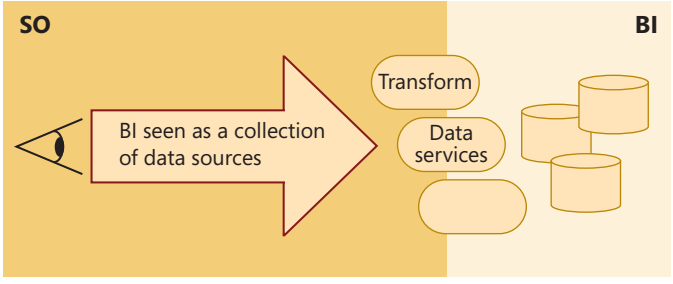


Figure 3 SO view of BI



Every BI system has a specific goal, which is derived from the requirements of the business.

For ease of reading, data warehousing and business intelligence will be consolidated under the single acronym BI throughout this discussion.

SO is a means of building distributed applications; at its most abstract, SO views everything as a service provider: from applications, to companies, to devices. The service providers expose capabilities through interfaces. These interfaces define the contract between the caller of the service and the service itself. The consumer of a service does not care how the service is implemented, only what the service does and how to invoke the service.

The services themselves are the building blocks of service-oriented applications. Services encapsulate the fundamental aspects of service orientation, namely, the separation between interface and implementation. SO is essential to delivering the business agility and IT flexibility promised by Web services. These benefits are delivered not just by viewing service architecture from a technology perspective or by

adopting Web service protocols, but also by requiring the creation of a service-oriented environment that is based on specific key principles.

BI has been around for years; its practices are well established, and people are comfortable with the concepts involved in delivering a BI solution. To many, BI is simply the presentation of information in a timely manner through a sophisticated client interface. To those involved with the delivery of BI solutions, this aspect of the overall BI solution is the tip of the iceberg. Under the covers is a huge exercise in data quality improvement and data integration among disparate corporate applications and systems and the consolidation of that data in a data warehouse. The integration of data is predominantly the biggest cost on a BI project.

EAI and ETL Convergence

Until recently, SO has had little or no part to play in the world of BI, primarily because the SO approach to data integration seemed laborious and overly complex to a community used to moving data of any volume around by connecting directly to the source system at database level. In BI, data integration is accomplished through the ETL process, which is the keystone of every BI solution, and BI solutions tend to look for the most direct and efficient way of accomplishing it.

Enterprise application integration (EAI), like BI, has been around for many years. It is a common problem encountered within an enterprise where systems have been introduced or grown in an organic manner. EAI itself can be defined as the sharing of both process and data between applications within the enterprise. Specifically, when we use the term EAI, we are referring to the integration of systems within the enterprise—for example, application, data, and process integration.

Application-to-application integration refers to the exchange of data and services between applications within the enterprise. Notably, this form of integration is often between applications that reside on different technology platforms, based on differing architectures. EAI is often difficult, and typically it requires the connectivity between heterogeneous technology platforms; involves complex business rules and processes; involves long-running business processes where logical units of work may span days or weeks as they move through different processes within the organization; and is generally driven by the need to extend/enhance an existing automated business and process or introduces an entirely new automated business process.

Solutions to EAI problems can involve solving the application integration problem at a number of different architectural levels such as data, application, process, and so forth. In this way both ETL and SO can form part of an EAI solution. In many ways, SO grew out of the need to find common, open, and interoperable solutions to the EAI problem.

Traditional ETL is a batch-driven process focused on integrating data during business downtime. In today's connected marketplace, business does not have a quiet time for this process to occur. The corporate data pool has the potential to increase significantly as initiatives such as clickstreams, e-commerce, and RFID are increasingly used, and ETL must be flexible enough to emerge from its batch-driven roots to deliver data on an event-driven basis.

Within an SO solution, these events are readily routed, consumed, and integrated as part of an event-driven architecture. BI grew out of the inability of operational systems to efficiently handle analytical capabilities (aggregation, trend, exception, and so on), hence, distinct BI systems were developed to meet this need. This artificial divide is no longer acceptable; today's businesses are increasingly looking to use analytical capabilities to guide their operational decisions—for example, in

Figure 4 SO as a data source to BI

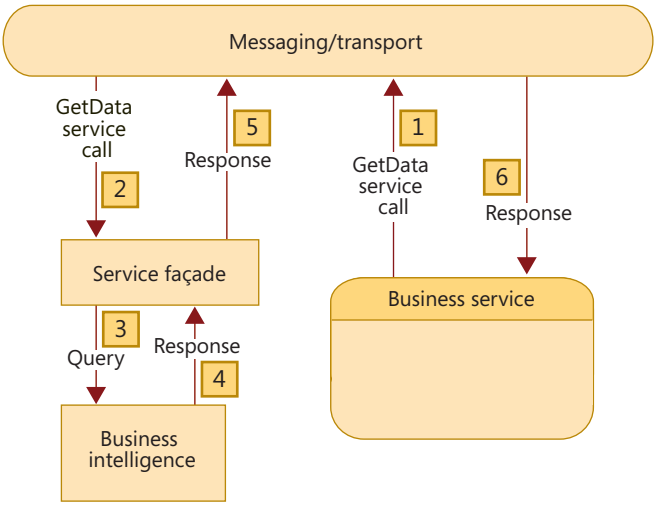
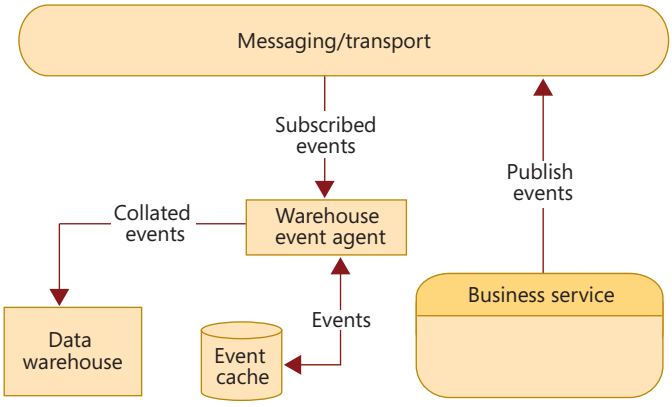


Figure 5 BI consuming SO events



the detection of a suspect credit card swiped at the checkout based on historical data mined for fraudulent patterns.

Organizations are also increasingly keen to unlock this data and make it widely available to other parts of the organization, to a wider audience of users and tools. Traditionally, access to BI information has required access to a specific set of data-manipulation tools. In this current SO environment, the goal must be to open up this organizational data to a wider audience and enable the value of the data to be realized more widely.

A Wider Range

As the variety and number of data sources considered in-scope for BI increases, so does the potential complexity of the required ETL solution. For example, Web services, RSS, and unstructured and semistructured data are sources of data that now fall under the data integration umbrella, but they are not traditionally associated with ETL. With the widespread adoption of service-oriented principles and the associated enabling technologies, access to a greater range of systems has become possible, unlocking a much greater range of business data.

Organizations continue to scrutinize the economics of data integration with every project, affecting the products they select, which is causing a change in the data-integration landscape and an associated

change in the functionality provided by the ETL or EAI software. Vendors have to increase the flexibility and functionality offered by their platforms in response to these new challenges. The net result to the customer is a set of tools with an increasingly merged set of functionality.

Users familiar with Microsoft's product set cannot fail to have noticed this pattern. BizTalk was developed in the EAI and business-to-business (B2B) space but has applicability in some ETL scenarios. Data Transformation Services (DTS), Microsoft's ETL tool that forms part of the SQL Server 2000 platform, grew from an ETL background. It is no coincidence that the SQL Server 2005 version of this product has been re-architected to support a wider range of integration scenarios, and was renamed Integration Services to emphasize this.

Although service-oriented architectures (SOAs) and BI architectures have evolved separately and include technologies and disciplines specific to their own individual architectural aims, many of the technologies that they utilize overlap. There is also a clear mapping between the concepts used, with each able to view the other in their own terms. We have called these "views from the other side," and the overview can be seen in Figure 1. Let's discuss each scenario in more detail.

From a BI perspective, it is possible to view an SO application as a collection of data sources and event sources. There are two primary modes in which a service can operate as a data source in a BI context: service as the provider of data upon request and service as the publisher of events that are of interest (see Figure 2). In both scenarios, the message sizes are small. The solution for large-scale data transfer and transformation will still be through the normal data warehouse import techniques such as ETL. Such physically large messages are not the normal domain of SO.

From the SO viewpoint, BI can be seen as a collection of services. From an SO perspective, a data source can readily be exposed as a service with the introduction of a simple façade layer that receives the service request from the service bus and calls the appropriate query. The façade then transforms the results of the query (if necessary) into the data schema and returns the results to the caller (see Figure 3).

The SoBI architecture makes BI data in the data warehouse available as a service to other applications within the architecture. This availability gives applications a clean way of accessing consolidated data to support the requirements of BI. In this way, the BI architecture becomes an integrated component of SO application architecture. Note that there will be occasions when the type of data that is required from the system is purely of a BI nature, such as large-scale data export. In these scenarios the service interface approach will not be suitable.

Figure 6 Message size versus message volume

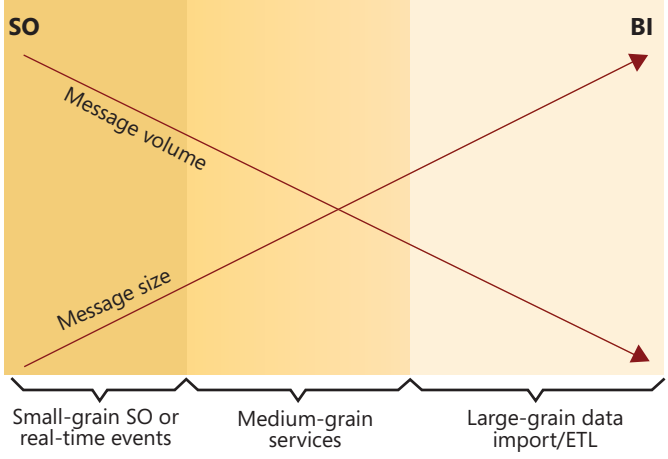


Table 1 SO and BI benefits

Service Orientation (SO)	Business Intelligence (BI)
Best suited to application-to-application integration and well suited to low-volume, high-frequency events	Best suited for data-to-data integration and able to handle large data volumes
Provides an operational platform, tightly defines data formats and structures, and encapsulates and abstracts functionality	Provides a combined model of the enterprise data and provides foundation for business decisions and the ability to ask any question of the data
Supports reuse of enterprise components and allows agile change in business processes	Good tools and mechanisms for transforming data

Service-Oriented Data Provision

From a BI perspective, a service can readily be exposed as a data source with the introduction of a simple façade layer that provides a mapping between the BI interface and the interface exposed by the service. The façade then transforms the results of the call from the data schema used on the service bus to the data format expected by the BI platform and returns the results to the caller (see Figure 4).

Some services expose information through events that are published when an *interesting* change occurs in the service. Other services and applications within the organization can subscribe to the events published by services. Integrating event-publishing services into a BI platform is achieved through the use of an agent that collates the subscribed events and periodically transfers them in bulk to the BI platform (see Figure 5).

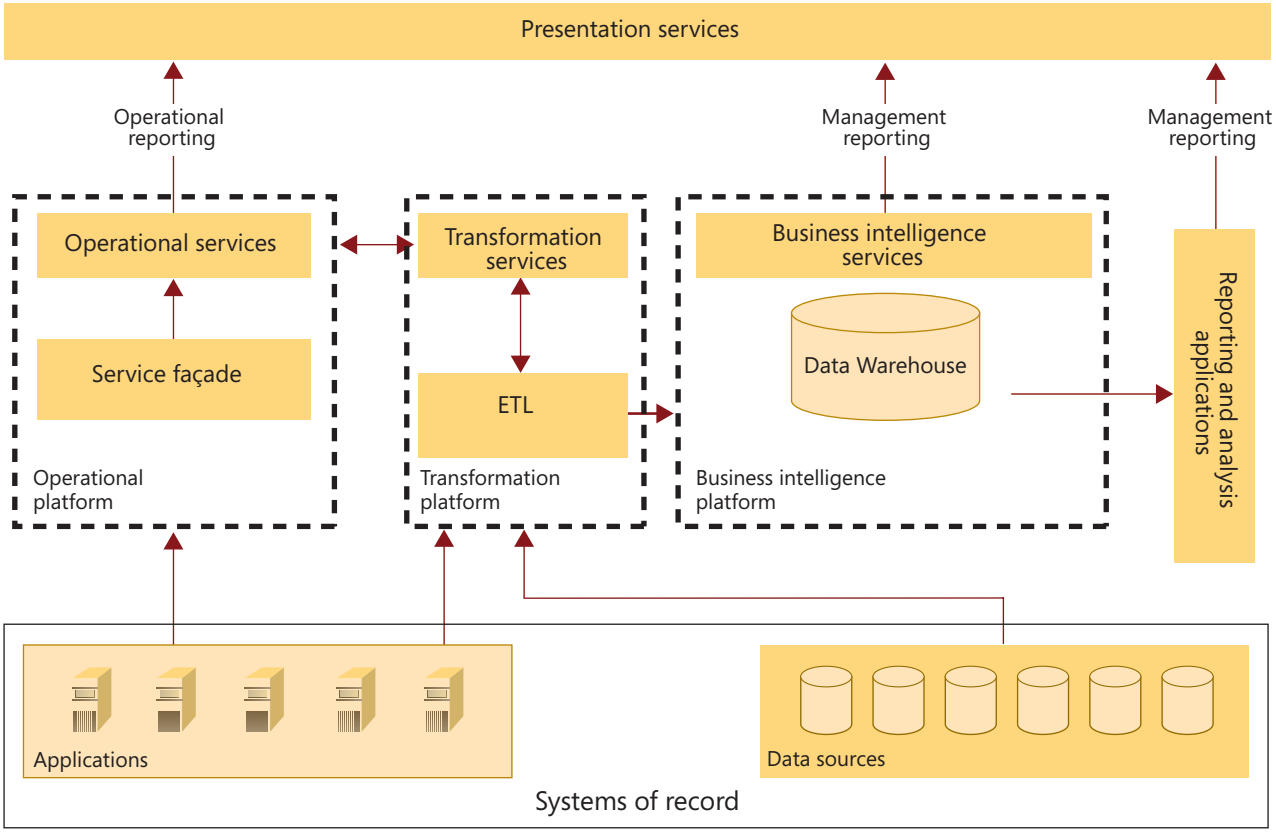
One of the challenges in the development of SoBI was to find an approach that leveraged the core strengths of each architecture and identified the area where the integration caused challenges. Let's look at some of the main challenges inherent in implementing BI in an SO manner. These challenges generally arise because of the specific requirements that each architecture was developed to address.

Figure 6 shows the differences in data granularity that separates the BI and SO approaches. At the extreme ends we have small-grain messages or events, which are naturally in the SO event space. Such small-grain messages are not readily or efficiently consumed in a BI architecture without the use of event agents to collate the events and import them into the data warehouse on a regular basis. Large-grain data, the bulk import or movement of large quantities of data, is most efficiently handled through data warehouse techniques such as ETL. Services in an SO system that expose large amounts of data are inefficient to use and rarely implemented. In the middle ground we have the medium-grained *typical* services, defined to consume sufficient data to fulfill a particular requirement. This middle ground is key to SoBI's value add.

Service-oriented applications exhibit loose coupling between their constituent services. This loose coupling is one of the fundamental principles of SO and supports the development of agile/flexible applications that can adapt to business change. See Table 1 for specific SO and BI benefits.

BI solutions are tightly coupled to the data sources that feed the data warehouse and to the applications that use it. BI has evolved from a batch-centric environment where ETL is used as the means to directly consume and consolidate large amounts of source system data on a known schedule for population of the data warehouse.

Figure 7 The SoBI framework



SO requires that the message interface and the formats of both the incoming message and the eventual response to be tightly defined. When exposing services that describe business capabilities the *questions* that can be asked within an SO application are in effect known in advance. However, there is also an unknown aspect relating to the exposing of services that are aggregated or orchestrated by another application. BI is concerned with the ability to allow applications to ask any question of the data warehouse within the limitations of the data warehouse data model. The question, or the size; content; and format of the result are not known until the question is asked.

SoBI Wins

An SO approach is best suited to the exposing of services that encapsulate business capabilities or services that publish events of interest to other systems. BI provides a closed environment for satisfying the information requirements of the business. BI enables the consumer of data to view it in potentially many different new ways. This flexibility provides the ability to identify trends and relationships that may be overlooked.

Previously we introduced the core strengths of each of the paradigms, each of which can be seen as a fundamental issue if viewed purely from the stance of the other paradigm. Now let's revisit these challenges and see what benefits SoBI can bring.

Surface warehouse ETL functionality to SoBI enables interesting business events caught during the ETL process to be published as events. Let's consider some examples.

Referential integrity. SoBI can provide aggregations of the position "now." For example, data not in the source system may be added in the data warehouse as a placeholder to maintain the integrity of the data in the database. This placeholder may occur when disparate source systems provide data at different times (for example, transactional values supplied before associated reference information has been received). This data will be stored as private *placeholder* data within the BI service. A notification may be sent to the source system to ensure that an error has not occurred, and when the reference data becomes available the data warehouse will be brought back in line with the system of record.

Single-validation service. The functionality required for validation during the ETL stage can be exposed through the SoBI framework for use within other parts of the business.

Timely updates. The key advantage here for BI is that adoption of ETL from an SO perspective can enable real enough time feeds into the data warehouse and therefore potentially real-time data warehousing. In addition, the better support for events and event-driven integration can provide BI with a much better mechanism for invoking ETL than the traditional methods, such as timed scheduling or persistent scan of a known directory for a flag file.

Common business schema. Within any organization there can be multiple systems that hold information about the same entities and that hold this information in different formats. There are clear synergies between these scenarios:

- Building the logical data model for a data warehouse is critical to any BI initiative. The data model is the end product of the efforts to consolidate the data from the disparate source systems. The structure of the data model drives the transformation exercise that occurs as part of the population of the data warehouse and, hence, enables the data warehouse to provide the single version of the truth for management information.

Table 2 SoBI principles

	BI	SO
It is...	...the single version of the truth for BI data.	...the architectural approach for application integration.
It will...	...provide open access to data services, support ad hoc analysis, support *precanned* management reporting, consolidate data from disparate source systems, and support reference data.	...provide application-to-application integration, provide some event feeds to the DW, describe the services provided and the messages passed, fulfill operational requirements, and povide the infrastructure services for all applications.
It will not...	...become a dumping ground for all data, become the data owner, be the default data source to other applications, and support operational reporting.	...be used in every circumstance and replace data import interfaces

- For any entity aggregation service within an SO design it is important to reach agreement on common meanings for the entities that the services will operate on. This agreement is referred to as schema consolidation and is the process of creating master data schemas that contain a superset of the information to describe the entities in the system in sufficient detail that the different services can each locate the data they require.

Another entity service that aligns with this approach is the ability to expose reference data. According to Easwaran G. Nadhan, Principal, EDS, "It is clear from the experiences in the (relatively small) number of organizations that have moved aggressively into SOAs that coordinating reference data is the required first step toward service orientation" (see Resources).

One Truth

One version of the truth. By looking at the functionality offered by the two architectures holistically, SoBI gives us the opportunity to consolidate operational and BI data without the need to physically move all operational data into the BI platform, which is a common approach to providing the "single version of the truth" in a BI project. By adopting the most appropriate architectural choice, the operational data can be left in place but still be available to the SoBI framework as a service, should the need arise to access or view it.

For example, in a BI environment interested in the analysis of health and safety incidents, SoBI would allow factual detail of each incident to be moved to the data warehouse—that is, the fact that an incident happened, where it happened, and the classification of that incident—which would allow all appropriate aggregation and analysis to be performed as expected of the BI platform.

Where SoBI wins is that it provides a means to still access the transactional detail in the system of record (for example, the free-format text that accompanies the incident, which describes the circumstances surrounding the event in detail). This access is commonly referred to as "drill through" or "drill to detail" in BI, and to accomplish it all data relevant to the requirement is traditionally moved into the data warehouse.

In fact, this may not be allowed (such as for data protection reasons) or preferred (it increases the ETL burden and the storage requirements of the data warehouse to hold data, which by definition, is operational in nature), and it has the potential to add additional pressure on the BI platform by becoming the de facto source for all incident data even though it is never up to date.

Business services. The combination of the enablement of real-enough-time BI and the ability to leave operational data in place gives us the opportunity to build a rich service around the SoBI framework that would not be possible if working within only one of the component architectures. Consider a system detecting retail fraud. In BI we have the tools to build an analytical engine capable of mining the potentially huge amounts of transactional data for patterns resulting in a list of suspicious credit cards. The adoption of SO principles allows us to offer a service providing the details of credit cards in use in our store as they are swiped. The SoBI architecture allows this service to be consumed by the BI component of the platform and analyze it against the known list of suspicious cards, and therefore to respond immediately should a potentially suspicious transaction be detected.

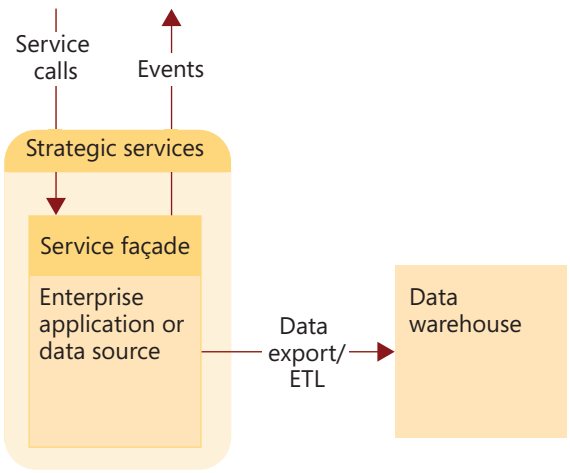
A New Breed of Business Services

Aggregation of transactional and historical data as a service. Given a service exposed through the SoBI framework that can now seamlessly aggregate current transactional data and historical warehouse data, a new breed of business services can be supported. An example would be *slowly changing dimensions*, which is where a value such as a customer name changes over time. Obviously this information is available within the data warehouse, so if there is a requirement to expose an entity service that gives a single view of the customer, this can be achieved more accurately and easily.

Brings interface abstraction patterns to BI. The ability to use the interface abstraction pattern over BI functionality makes the functionality and data more accessible to line-of-business applications and provides the capability to expose complex business rules usually buried in the ETL layer.

Cleansing and consolidation. Data will be changed for purposes of consistency and integrity. Where this change involves a mapping operation, that mapping will be made available to the architecture as a service. Where this change involves a correction to data, details of that correction will be fed back to the system of record through a request

Figure 8 Ideal scalability and flexibility



for change to the owning service, that is, the ETL process cannot change the data as it is not the owner. In turn, this process obviously drives improvement in the quality of data.

Obtain a cross-system, consistent view of the product. During the ETL stage data of the same product (or entity) may require transforming in order that it may be stored in a consistent manner. By service enabling access to the data the organization can expose a single common view of a product.

Mappings available as a service. As previously stated, mapping is a fundamental requirement within the ETL stage. The SoBI framework enables the mapping functionality to be exposed as a service for other uses within the organization. Such uses include EAI and enterprise reference scenarios. This availability of this service can also be used to promote best of breed transformations.

Calculation. The data warehouse is often used to store precalculated values to support the requirements of BI. For instance, sales and forecasting data may be held in different physical systems. The consolidation of the data from these systems into the data warehouse allows us to calculate and store actual versus forecast figures to support more performant analysis and reporting. The business logic used to define such calculations is often interesting to other parts of the business so the calculation to support this *invention* of data in the data warehouse will be made available to the SoBI architecture as a service.

Provides a road map for integration. It is believed that one of the outcomes for the SoBI framework is an ability, at an architectural level, to provide a framework for future integration scenarios.

Compliance/audit. Application of the SoBI framework requires adherence to a formal governance process. Examples include the identification of the system of record or operational data owner, and the definition of the messages that describe the data and functional requirements. Given only the owner of the data can make a change to that data, other systems simply make a request for change, and auditing can be carried out at a single point.

Aggregation. To support fast response times, data in the data warehouse is often preaggregated. For instance, the data warehouse may contain data relating to sales at an individual transaction level, but the majority of management reporting may require seeing the totals at the month level. In this case, it is cost effective to roll the (potentially millions of) individual transactions up to a level more appropriate for the known queries and to store the results in an aggregation, avoiding the need for known queries to perform the aggregation action at query time. Where such aggregations are created, they will be made available to the architecture as a service.

Enterprise Data Quality

Most organizations suffer from the issue of compatibility between their IT systems. This issue is even more apparent where mergers have occurred. No reason exists why systems from completely different enterprises should be consistent, aligned, or satisfy a unified design. A BI initiative has to address the problems of disparate systems, data silos, and data incompatibility through data quality initiatives. If the business users do not ultimately trust the information they are presented, they will not use the system.

Improving data quality is not simply a process of *fixing* problems with individual data elements; it is about designing a business process that improves the management and quality of data as an enterprise resource. In an SO application the data provided by a service is controlled through encapsulation by that service and only published in a specific form that matches the data schema defined for the service.

Figure 9 Collating events for integration

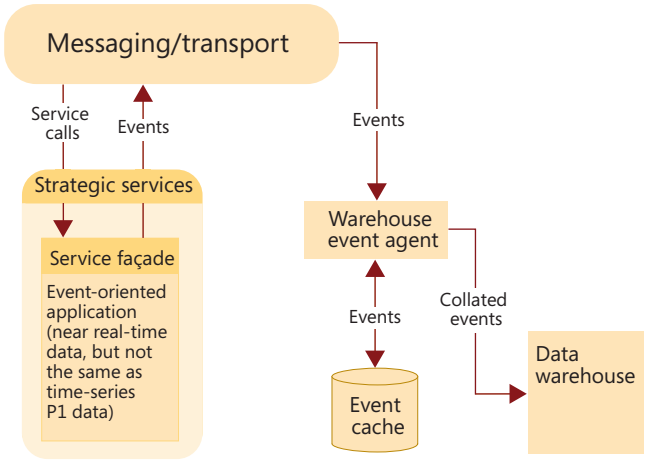
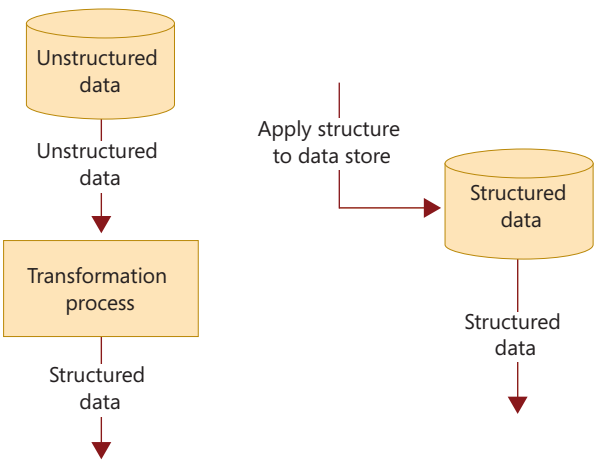


Figure 10 Upgrading nonenterprise data sources

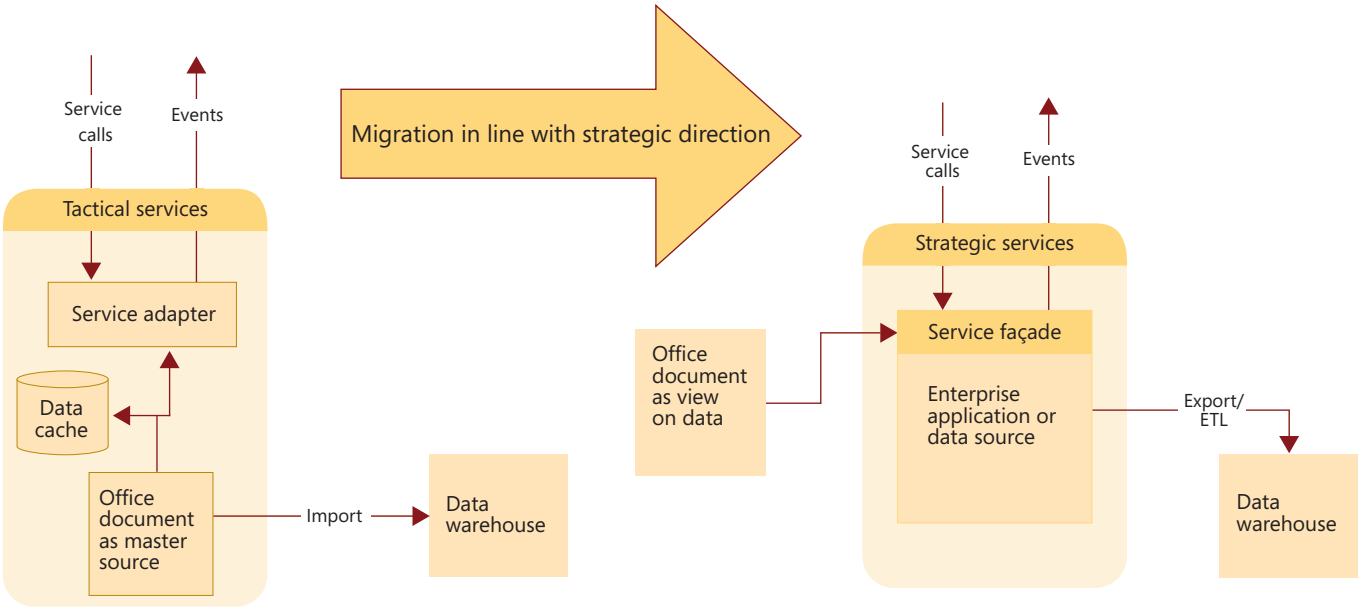


Access to the data is achieved through the messages that the service publishes. The service is solely responsible for maintaining data integrity since it is the only mechanism that directly manipulates the data. The schema that defines entities in the service messages and the definition of these messages themselves strongly enhances the data quality aspect of any solution because of the capability to automatically test messages for compliance to the message schema or contract supported by a service.

Let's take a look at a SoBI framework example. It is important that the SoBI framework clearly defines and distinguishes between different types of data and the associated owners of this data, which ensures integrity as the data can be amended only in one place. It also enables the data to be shaped for its appropriate use. Figure 7 shows the high-level data architecture for the SoBI framework. The purpose of this architecture is to highlight how the different types of data will be handled in the solution.

As you can see, the systems of record are integrated into the data warehouse through traditional ETL methods, with the exception that the transformation services normally contained within the ETL process are exposed for reuse by the operational services. This reuse enables information to be integrated into the data warehouse and exposed

Figure 11 Views on information



through service façades in a common schema as the transformation services are shared by both mechanisms.

It is also recognized that not all information within the data warehouse can be exposed through the service interface and that there may still be a small population of users within the organization who will require direct access to the data warehouse to carry out complex or ad hoc analysis. Let's take a look in greater detail at the factors influencing the successful implementation of the SoBI framework.

At a high level, the architectural guidance for the application of the constituent parts of SoBI can be summarized as shown in Table 2. Success factors for the SoBI project include:

- *Governance.* An SoBI project is unlikely to be successful without an associated organizational change management process in place to support it.
- *Enterprise data and SOA strategy.* SoBI relies on the existence of a strategic plan for SO applications within the organization and on the recognition of the importance of storing system of record data in enterprise-class stores or applications. For example, the organization doesn't store the master of key business information in spreadsheets.
- *Operational versus management reporting.* Clear delineation must be made between the operational and management types of reporting that will be required within the SoBI framework. SoBI BI will be the single version of the truth for management reporting. It will be scoped to meet the specific requirements of the business for BI. The supporting data warehouse will contain only the data necessary to support those requirements.
- *Data ownership.* Although the data warehouse will contain the data gathered from multiple data sources and systems, from a service-oriented perspective the data warehouse should not be considered as the master version or the default store for all data requirements. The owner of this data remains the system of record.

Success Factors

Let's look at these success factors in more detail. Governance is a key success factor in the development of a successful service-oriented solution. It is important that an organization can control and publicize the services, message formats, and structures that are supported to prevent an uncontrolled proliferation of services, messages, and entity definitions in the environment. This is closely tied to the schema definition activities and requires active management by a governance body to ensure that the system adheres to the service-oriented principles.

The key is finding a balance in the level of governance and having enough control to provide a framework for the successful development and deployment of services, but not to the extent of crippling the system's ability to respond in an agile manner to the needs of the business.

In any BI project, a lack of clarity on the role of the data warehouse and the scope of the data to be contained within it can lead to problems. In an ever-lasting design exercise, if the scope of the data warehouse is not managed, the design exercise grows as potential data sources are discovered and the data within those sources has to be consolidated within the data warehouse data model.

The data warehouse model needs to be designed to be extensible to ensure that the data from different assets can be added as a business case can be made for the data. It is not practical to assume that data from every application in the organizational landscape can be included in the design of the first data model. An incremental approach to the design and build of the warehouse is more likely to prove successful.

For enterprise SOA and data strategy the successful application of SoBI relies on the existence of an organizational strategic plan for SO applications and for system of record data to be held in enterprise stores or applications. Where systems and data don't have the capability to support direct integration into the SOA proposed by the SoBI framework, it is assumed that the eventual plan of the enterprise is to migrate these systems and data sources to a platform that would support this level of integration. Examples of the types of systems referred to in this assumption are systems that are heavily loaded and cannot

handle the extra burden of exposing services and systems that do not support online querying and rely on periodic batch export.

Some of the business-critical information is held in stores or applications that are not appropriate for data of this commercial value—for example, Microsoft Excel spreadsheets that hold master versions of data making the spreadsheet the system of record. There must be a strategic direction to move toward the point where such data is held in an enterprise-strength store or application and for the documents to become views to this data rather than the master sources. The goal is to move from documents as systems of record to documents as views on data held in systems of record.

Maintain Integrity

Clear delineation must be made between the operational and management types of reporting that will be required within the SoBI framework. The traditional view of the data warehouse as the single source for all corporate information needs does not hold within the SoBI framework. The objective of the SoBI framework is to uphold the principal of BI as the "single version of the truth" but also to maintain the integrity of the systems of record as owners of the corporate data.

An operational report is likely to meet one of these criteria: requires live access to data; is required for the operational management of the business (for example, information on an individual transaction); doesn't require historical data for comparisons; and doesn't require summarized data (aggregated data except for the basic total). A management report is usually defined as:

- Requiring access to timely but not immediate data and typically requires summarized data to be presented over a predetermined historical time frame (for example, month on month metric comparisons by asset)
- Presenting the user with the capability to explore the presented data at will to quickly research potential performance problems (for example, drilling into the data). The report can highlight areas of failure based on conditional formatting.
- Being defined to only notify the user when a problem occurs (exception reporting).
- Assisting in the forecasting of business performance.
- Assisting in the underlying goals of the individual and company (for example, production efficiency, up selling, and product profit maximization).

Although the data warehouse will contain the data gathered from multiple data sources and systems, from a service-oriented perspective the data warehouse should not be considered as the master version. The owner of this data remains the system of record.

SO clearly discriminates between two different types of data, namely, the data that is held within the service and the data that the service exposes to the applications or other services. Data inside is the data that the service utilizes to perform the operations that it provides. This information is entirely private to the service and is never exposed directly. Data outside is the data that is published by the service and used to exchange information and requests with the clients of the service. This data is defined explicitly in terms of an organizational schema.

The application that is the owner of the data (also referred to as the system of record) is ultimately responsible for maintaining its own data; therefore, to enable other applications to request changes to the data the owning application has to expose the request update functionality through its service interface. (See resources for more information on these two types of data.)

When working with data integration, SoBI will be flexible enough to work with these key integration scenarios.

- Data volumes, which are ad hoc single transactions or large-volume, bulk data loads.
- Integration with third-party packaged applications and proprietary database schemas.
- Database consolidation.
- Integration of legacy systems, relational, nonrelational, structured, and semistructured data sources.
- Support for Web services and integration with messaging middleware.

When working with BI, SoBI will aim to satisfy several requirements. Businesses need to collect and aggregate information from disparate sources within the organization and to be able to share this information in an open manner with a diverse estate of applications in different parts of the organization, without first knowing in detail how the information will be consumed. Businesses also need to isolate users from the underlying data format and structure, instead focusing on the business meaning of the data within the organization. The consumers of the information are concerned primarily with the semantics of the data rather than the syntax of the data. Different parts of the business need to be able to share a common language when describing themselves, and businesses need to publish data more widely within the organization, differentiating between data publication and data for analysis.

The publication of defined pieces of information such as KPIs, or metrics through open mechanisms such as Web services, enables this information to be consumable easily within the organization without recourse to specialized applications. The publication of data for analysis will still be a key part of the services provided by the data warehouse, but this tends to target a more limited audience in the organization with access to the specialized applications necessary for data analysis. One of the goals of SoBI is to enable the open publication of information to a wider audience of users, applications, and other services in the organization.

SoBI Integration Patterns

Another goal for the SoBI framework is to take a pragmatic approach to working with systems that cannot be integrated directly into the architecture—for example, those that cannot support extra load or those that are not scalable enough to support direct integration.

With this in mind, the framework defines a set of scenarios (or patterns) describing categories of typical source systems that the authors have encountered, along with a recommended outline approach for integrating each category of system. It is envisaged that this collection of integration patterns will evolve and grow as more diverse systems are integrated into a SoBI solution. These patterns help address one of the key service-oriented priorities, namely, the ability to be able to support replacement of applications with minimal impact to the enterprise. The SoBI framework describes how

Figure 12 Unsuitable service-interface approach

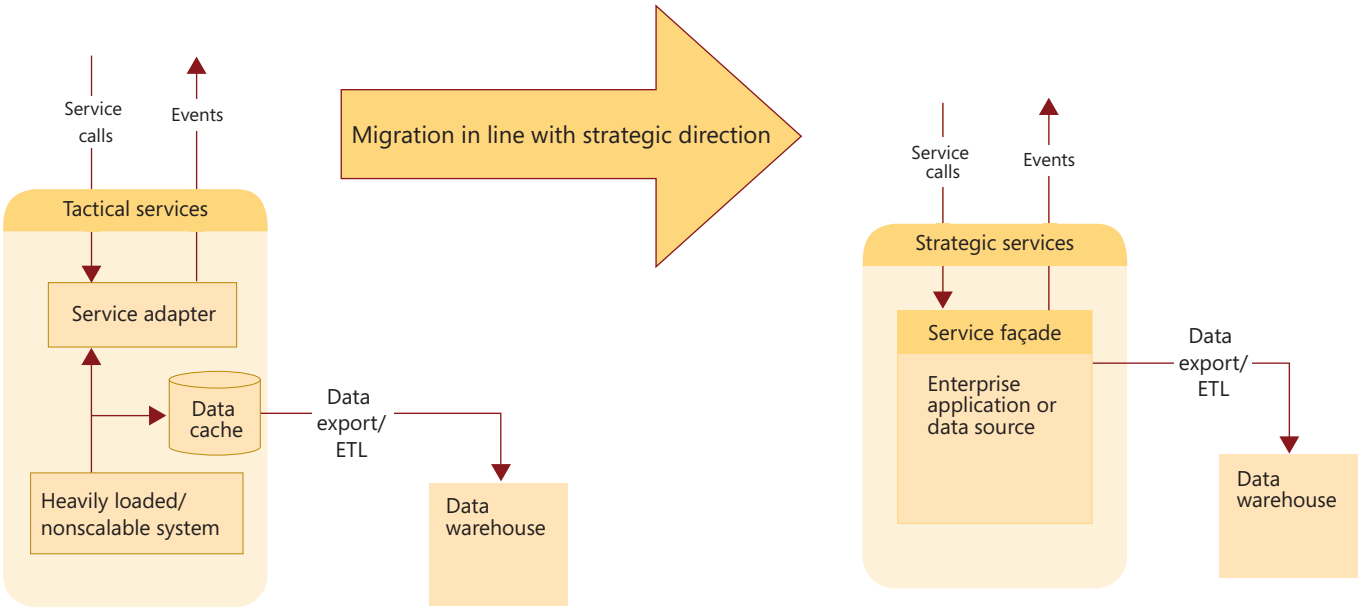
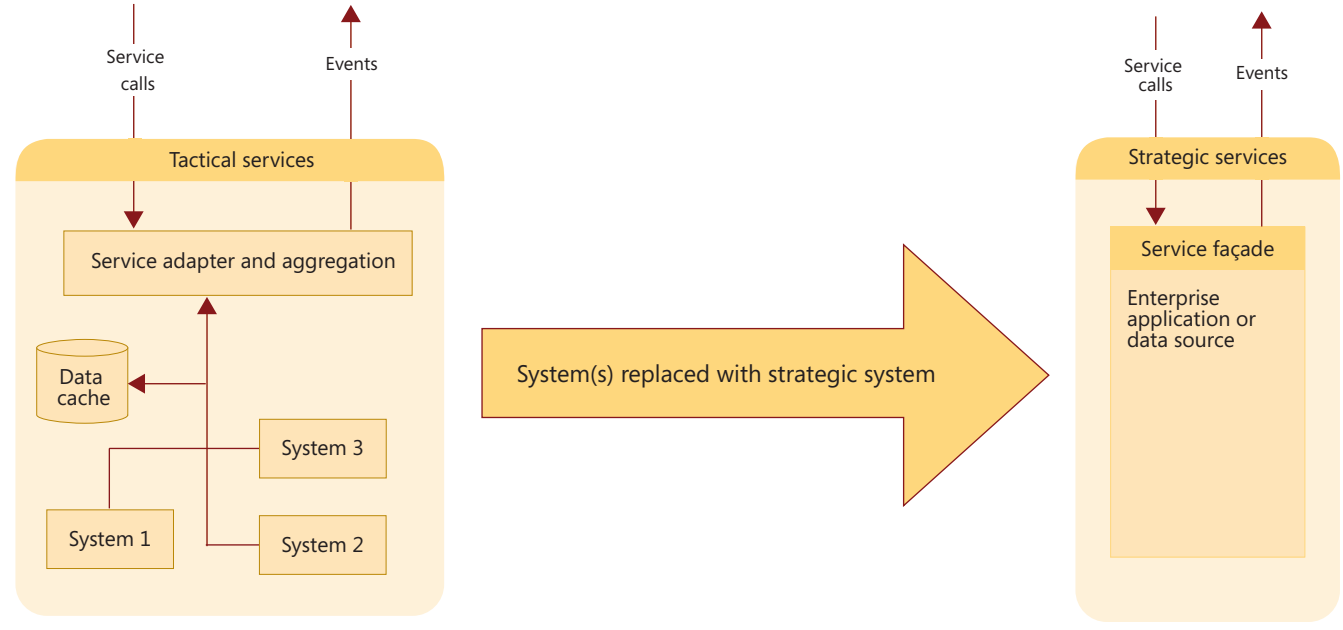


Figure 13 BI implementation replacement



these systems can be abstracted in a way that ensures that there is minimum disruption when the systems are finally replaced.

We've so far framed SoBI in terms of an ideal world scenario in which the various systems of record are able to participate in the SoBI architecture in a service-oriented manner. In a real project environment, it is likely that there will be a number of constraints that impact our ability to implement *pure* SoBI. We have identified several categories of constraint, which will be discussed shortly, and for examples in each case we identified a pattern, which, when used in conjunction with the SoBI principal of enterprise SOA and data strategy, ensures that the pragmatic SoBI implementation stays within the guiding principles of the SoBI framework.

Pure SoBI

An ideal situation is one in which the application is scalable and flexible enough to support the exposing of services and events to the service bus, either directly or through a thin service façade (see Figure 8). This category of application is also capable of supporting the export of data to the data warehouse, as required.

One source system-type constraint is real-time/transactional processing. Some applications will contain information that is interesting from an analysis perspective and from a real-time perspective. Such an application can be service enabled by creating a service façade that exposes the services of the application to the organization, and that fires events when "interesting" changes or updates are made. In this

scenario, the applications that are interested in the events from this application can subscribe to the published events. The subscribers will include a data warehouse agent that collates the events for integration with the data warehouse (see Figure 9).

Non- and semistructured systems are another constraint. In any complex environment there are likely to be a number of data sources that contain information that must be consumed by the solution but that are held in semistructured or unstructured formats, such as spreadsheets and document-management systems. With these kinds of systems it is important to structure the information prior to integration into the data warehouse, which will be achieved in one of several ways. One way is to apply structure to the data store. For example, the provision of structured and change-controlled templates for spreadsheets and documents, such that the information can be accurately and reliably extracted from the document, will involve business process change. Another way is to impose structure on the data read from the store, which is inherently difficult as it relies on making assumptions about the semantics of the existing structure of the data source and relying on this never changing. If this assumption holds, programmatic extraction can be achieved.

We are assuming that nonenterprise data sources will eventually be upgraded to more directly support the services that they provide (see Figure 10). It is important to change documents from information sources into views on information (see Figure 11).

Now let's look at source system limitation constraints. For heavily loaded systems there will be occasions when there are data sources or systems that contain data that cannot be interrogated in real time because of operational or technology constraints. Consider these examples: a heavily utilized system may not be able to support the addition of a service interface that processes a new set of queries on a frequent basis; an information source that does not support concurrent access, such as data held in a spreadsheet; and ad hoc or real-time access to data within a production system is considered to be a risk to the effective day-to-day running of the business-critical system.

In these scenarios the tactical solution is to cache the data in a system that can then provide a defined and published interface to the data or service. This cache allows applications and services to have access to the most up-to-date information possible through an exposed service. This approach provides a means of scaling a data source or application to meet the requirements of the business in a way that gives a clear decoupling between the data or application and the service provided. This decoupling is important for future development when the application or data can be moved to a more scalable solution and provide the service directly, or when the application can be enhanced to support the service directly. One caveat is there will be occasions when the type of data that is required from the system is purely of a BI nature, such as large-scale data export. In these scenarios the service-interface approach will not be suitable (see Figure 12).

We are assuming that the heavily loaded systems will eventually be upgraded to more directly support the services that they provide.

Short Life Expectancy

One of the principles of SoBI is that the organization must put in place a strategic plan for service-oriented applications and for system of record data to be held in enterprise stores or applications. This plan will inevitably mean the replacement of some of the systems of record in the existing landscape, and in some cases these are likely to be replaced after the implementation of the BI project. Therefore, it is vital that an approach is defined that can support these systems in the twilight years of their lives and ease the process of replacement when the successor system goes live (see Figure 13). This approach will need to produce an architecture that gives balance between enabling an easy transition to the new data source while not committing the project team to a large development effort that is ultimately thrown away. •

Resources

"Data on the Outside vs. Data on the Inside," Microsoft Developer Network (MSDN) Whitepaper, Pat Helland, Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dataoutsideinside.asp

"Data Warehousing Lessons Learned: Trends in Data Quality," Lou Agosta, Column published in *DM Review Magazine* (February 2005)

"Information as a Service: Service-Oriented Information Integration," Ronald Schmelzer, Zapthink
www.zapthink.com/report.html?id=WP-0125

"Information Bridge Framework: Bringing SOA to the Desktop in Office Applications," Ricard Roma i Dalfo, *The Architecture Journal 4*, (Microsoft Corporation, 2004)
http://msdn.microsoft.com/architecture/default.aspx?pull=/library/en-us/dnmaj/html/ibf-J4.asp

"Over half of data warehouse projects doomed" Robert Jaques, Gartner, (February 2005)

"Service Orientation and Its Role in Your Connected Systems Strategy," A paper providing an overview of Microsoft's vision for service orientation and service-oriented architecture in enterprise computing. (Microsoft, 2005)
http://msdn.microsoft.com/architecture/soa/default.aspx?pull=/library/en-us/dnbda/html/srorientwp.asp

"Service-Oriented Architecture: Considerations for Agile Systems," Lawrence Wilkes and Richard Veryard, CBDI Forum, *The Architecture Journal 2* (Microsoft, 2004)
http://msdn.microsoft.com/architecture/journal/default.aspx?pull=/library/en-us/dnmaj/html/aj2service.asp

"Service-Oriented Architecture: Implementation Challenges," Easwaran G. Nadhan, Principal, EDS *The Architecture Journal 2*, (Microsoft, 2004)
http://msdn.microsoft.com/architecture/journal/default.aspx?pull=/library/en-us/dnmaj/html/aj2soaimpc.asp

About the Authors

Sean Gordon is an architect in Microsoft's enterprise systems strategy and architecture consulting practice and is based in the company's Scotland office.

Robert Grigg is a managing consultant and enterprise architect at Conchango, UK.

Michael Horne is a managing consultant and business intelligence architect at Conchango, UK.

Simon Thurman is an architect evangelist in the developer group at Microsoft Developer Network, UK.

Planning Technical Architecture

by Waleed S. Nema

Summary

The need for technical architecture planning is well known to managers, auditors, and operational staff in this highly demanding and changing information age. Business drivers for this need include aligning IT with the business and controlling service levels and expenses. This discussion summarizes the first experience of creating an infrastructure Windows Server Architecture (WSA) two-year tactical plan for a computer operations department. Technical architecture is defined in addition to coverage of the deliverables and challenges involved.

The discipline of planning technical architecture has only become more popular recently. Therefore, it is not well understood because of lack of experience, training, and even literature. Technical architecture planning, when done correctly, tries to open all aspects of the existing environment and map them to a business-consistent desired state. It tends to create a wide range of questions and issues some never thought of before: some political, some business, and others. As you try to clarify the vision and scope of architecture planning, you may find resistance because of unclear understanding or hidden agendas. You may even have resistance because of who you are or where you're coming from.

Assuming the vision and scope of architecture planning are agreed upon is no guarantee for keen, open, and full participation on the part of operations staff. The benefits of disclosing information must outweigh the cost of revealing soft spots and vulnerabilities that only operational staff know about. Once the planning cycle is over and it is proven that participation is rewarded by allocating budget and resources, only then can you expect more participation in subsequent cycles. Being at the disadvantage of the first one, you're going to have to make the message very loud and clearly stated by management. To be successful, management must clearly outline the benefits to architecture planning from the start.

As you see, you have your fair share of challenges just getting the architecture planning process started, more challenges maintaining participation from operational staff, and challenges obtaining support from management. And if you follow the model of supervising follow-up action plans, as described later, you may appear as a cop or auditor.

The only way for this effort to be successful is to show and prove good intentions. Management, which usually gives the directive for this activity, along with the architecture team must convince staff that this activity isn't about exposing them but rather about creating a better, more efficient operating environment that better serves the business. In the end, everyone wins but at the expense of accepting change and letting down defenses and barriers.

Giving It Life

Above all, architecture team members are facilitators. They must work with operational staff to understand the existing environment. As a matter of fact, operational staff should be the architects for the existing environment because they know it better than anyone else. The architecture team must work with business and other corporate planners to bring in the business strategy and direction that IT should be aligned with. The team must work with management to understand tactics, constraints, and gain support. They must also bring in the industry best practice perspective covering framework, technology, process, and people. The architecture team needs to bring all of that together in a workable, realistic plan.

Architecture and planning generally give a view of a future state that translates some kind of need or wish. If we're talking about a house, for example, an architect develops a detailed plan that meets

"ASSUMING THE VISION AND SCOPE OF ARCHITECTURE PLANNING ARE AGREED UPON IS NO GUARANTEE FOR KEEN, OPEN, AND FULL PARTICIPATION ON THE PART OF OPERATIONS STAFF"

the guidelines of the owner. If we're talking about a model to be used in a development community, then we're also talking about a standard to be followed to maintain a specific budget and general look. The owner's guidelines and requirements in this sense could also be thought of as a standard for the builder. To bring an architecture blueprint to life, a rather complex construction project plan must be devised and followed. The beauty of the house on paper means nothing until the owner can see it in real life.

Therefore, it makes sense to think of architecture planning as a process that translates a set of guidelines and requirements into stan-

dards that builders can implement. If a building already exists, architecture means projecting new requirements and enhancements to the existing structure.

The technical world has not learned enough from the well-established construction industry. The roles of owner, architect, builder, and inspector are well known in the construction field but not equally so for the technical field, particularly the inspector's role. In the construction industry, the builder is almost always a different entity than the architect. The inspector could also be an independent entity but is frequently the same as the architect, both of whom act on behalf of the owner and must approve the builder's work.

We technical people need to empower and enable architects to take on the role of inspecting and supervising the implementation of architecture plans. In IT, project sponsors frequently are the owners.

> "SINCE ARCHITECTS REPRESENT EXECUTIVE SPONSORS IN THIS MODEL, THEY NEED TO BE AWARE OF THE BUSINESS CASE AND MUST IDENTIFY ACTION ITEM RECOMMENDATIONS AND SET PRIORITIES ACCORDINGLY"

Architects should act on their behalf and assume whatever authority is required. With this new twist, not only is architecture planning responsible for producing blueprints and standards but it also must be responsible for overseeing the implementation of the architecture plans.

Since architects represent executive sponsors in this model, they need to be aware of the business case and must identify action item recommendations and set priorities accordingly. The executive sponsor is ultimately responsible for the architectural process and its successful implementation.

To summarize, architecture planning must address these five aspects: standardization, enhancements, implementation supervision, business-driven priorities, and management follow-up; and we can define it this way: *technical architecture planning* is a process sponsored by executive management that translates existing business needs, challenges, and wishes into a prioritized set of standards, plans, and action items, which recommend and oversee service enhancements leading to better customer experience and operational excellence.

What It Is and Isn't

Technical architecture aims to raise the capability and maturity of service. It is about models, standards, and action (enhancements). It is a high-level solution specification but is not a design specification. Solution specifications are like setting the budget and lot size of a house model in a development community rather than getting into its layout specifics. Once approved, follow-up design and implementation projects are usually born for new services or large-scale enhancements.

Technical architecture is not simply a Visio diagram of file server configurations. Architecture *is*, whether a logical file server name space (such as Distributed File System) is used; whether a centralized or distributed management model is followed; or whether a free-access, shared-hosting model is used for databases or Web sites.

Tactical Plan for the Windows Server Architecture

This outline is a partial table of contents for the Windows Server Architecture (WSA) tactical plan.

- Introduction
- Disciplines
  - Web hosting services
    - Executive summary
    - Introduction
      - Background
      - Business drivers
      - Objectives
      - Scope
    - Solution concept
      - Service architecture
        - MOF optimizing
          - Service model
          - Service offerings
          - Service/operational-level commitments
          - Service requests
          - Quality assurance
          - Team model
          - Support services
      - Technology architecture
        - General
          - Corporate application strategy
          - Definitions
          - Trust zones
          - Hosting profiles (deployment patterns)
          - Architectural guidelines
        - MOF changing
          - Dependencies management
          - Production control
        - MOF operating
          - Monitoring
        - MOF supporting
          - Incident/problem management
          - Problem isolation
          - Enhanced error reporting
        - MOF optimizing
          - Security strategies
          - Business continuity strategies
    - Action plan recommendations
    - Appendix
      - References
      - Diagrams
  - Other services (file and print, directory, data)
- Server road map
- Consolidated deliverables
  - Infrastructure
  - Development
  - Policies and processes

Table 1 Sample of action plan recommendations

Section	Priority/ deadline	Objective	Resources/ skills	Work estimate	Success metric
3.x	Service model				
	Priority A Q4 2005	Publicize the Internet Service Provider (ISP) hosting model	WHG		Publish document in SLC, OLC, and new service request terms and conditions
3.y	Monitoring				
	Priority A Q2 2006	Monitor web sites/ servers for down-time, performance, and optimization	WHG		Assign a monitoring attendant role to a named person who will be accountable for attending to all monitoring console events.
					Explore MOM 2005 new features including IIS Management Pack, and Web sites and Services MP.
					Enable text-messaging and escalation for critical errors such as server-down.
3.z	Production control				
	Priority B Q4 2006	Build a Web site publishing tool	Consulting assignment		Engage in a development contract that will build a Web site Publishing Tool as defined in the existing prototype.

Those types of decisions have far-reaching consequences and thus are architecture-level decisions.

Technical Architecture is most certainly about process. As a matter of fact, it is frequently the process, not the technology that can make or break a service. Processes often involve people issues that imply politics and sometimes are the most dangerous aspect of all! Worse yet is when the system doesn't have checks and balances for working with such issues.

Technical architecture is about measurement. People must feel that measurements are created to help them, not to expose them. Management must make their intentions crystal clear to gain the trust and full buy-in from staff. Only then do we have a chance for *anything* to work. Management must communicate the reasons for measurement, which include realization, or the awareness of how far off we are, improvement to reach targets, and estimation of what's required. Management must also avoid public embarrassment by putting a positive spin on public reporting such as "percentage improvement over last period," while keeping absolute reporting internal

> "ARCHITECTURE IS A CONTINUOUS CYCLE AS ASSURED BY MANY FRAMEWORKS SUCH AS MICROSOFT OPERATIONS FRAMEWORK, DEMING'S CYCLE, AND THE COBIT/ MANAGEMENT CYCLE"

until the numbers are close to the Key Performance Indicator (KPI) targets. Management must prove good intentions either by getting the required resources or by accepting slower progress and less-impressive results.

Technical architecture is about automation. That's the key to efficiency, deterministic processes, and stability. Automation gives operations staff more time to focus on analysis and optimization, which are the basis for service enhancement.

Technical architecture is also about monitoring. The only way to be in control is to constantly look at the road and the gauges. Architecture gives you a road map and makes sure you see the right gauges. It also helps you create an alert and escalation system—possibly an automated corrective reaction.

Architecture is a continuous cycle as assured by many frameworks such as Microsoft Operations Framework, Deming's cycle, and the COBIT/Management cycle. It is basically a planning phase followed by monitored operations, the learning of which feeds optimization and enhancements into a new cycle.

Documenting the Plan

The main body of the Windows Server Architecture (WSA) tactical plan is in the functional area disciplines such as Web hosting services, directory services, and so on (see the sidebar, "Tactical Plan for the Windows Server Architecture"). The server road map applies to specific technology changes in the tactical period. The last section, consolidation deliverables, collects action plan recommendations from all sections and categorizes them in three areas to help follow-up projects: infrastructure, development, and policies and processes.

The architecture's main body of each discipline service—Web hosting services, directory services, and so on—is in the solution concept section, which is divided into service and technology architectures that are in turn based on the Microsoft Operations Framework (MOF). The introduction includes the background, business drivers, objectives, and scope sections, and it sets objectives such as the number of 9s for high availability, among others.

The executive summary is a one-page summary of as-is, to-be benefits, goals, and approach in addition to the most significant action plan recommendations.

Action plan recommendations are probably the most important summary of all the solution specifications and success metrics because operating staff set priorities and deadlines and indicate in the resources/skills column if they can do it themselves or they need it to be contracted out (see Table 1). Establishing architecture planning as a discipline and a role has paid off even before finishing the first tactical cycle. It is a delight to see the completion of the plan and the start of action. This result does not happen without cost or effort and is dependent on teamwork among all players. •

About the Author

Waleed Nema is the team lead for Windows technical architecture planning, computer operations department at Saudi Aramco. For the great success of this project, Waleed would like to thank his management, operational staff, and architecture team.

Behavioral Software Architecture Language

by Behzad Karim

Summary

Software architecture is a hot keyword these days for people in our profession. It seems everyone is out to discover the true potential of the software architecture and what it can bring into play for them. The basic idea of architecture definition is to design software structure and object interaction before the detailed design phase. Although serious architecture definition is being suggested for large projects only, arguably any software construction or implementation work must be preceded by an architectural design and approval phase. Get acquainted with BASL, a language that unifies software architecture definition with software implementation (coding).

One might rightly argue that there are at least a dozen well-defined standards and tools that can be used for defining software architecture. We do have standards and tools to define, communicate, and even generate templates for software design. Some of these tools can even work two ways by translating code to an architecture model and vice versa. Why then is there a need for another language or programming model?

Although there are plenty of ways to define the software architecture in terms of packages, components, and connectors (that is, the structure of software), when it comes to defining dynamic software behavior, we are unable to provide a definition for, communicate, or even clearly design them. However, the *dynamic behavior* of software is really what our software does after it rolls out into the production environment.

Most current, well-known (and established) techniques used for software architecture definition are from an era when software architecture was still a myth. While these techniques (and tools) do a great job defining the structure and components of software systems, they lack the ability to encapsulate and define the software behavior and various interactions with the environment against the dimension of time. Arguably these tools were not meant for the software architect as much as they were meant for the software engineer.

The basic need of the architect to define the essence of a software system in an abstract manner while providing the picture of the *living system* remains unanswered. We are either going too much into the implementation details or merely communicating the outer surface of the system. In both cases we are leaving out the most fundamental ingredient of software character: the dynamic behavioral aspects of the system. In too many projects these aspects of the system are being discovered in the detailed design or coding phase by the development team.

Facing Reality

Another dilemma is the ever-growing gap between the code and the original architecture of the system. We are using different tools and *languages* in the process of software construction. The software architect is using some kind of a modeling language while the developer is using a programming language. As a result, especially after the system rolls into production, documents, diagrams, and code get out of sync. Have you ever had to come back to revise a system you had designed in the past, only to discover that the architecture has no resemblance to the original design?

This analogy holds true for class diagrams, use cases, and test scenarios. The usual problem with these documents is that they all seem to become obsolete after the software rolls into production. These problems arise from the lack of a shared medium or a singular point of reference for the architecture an d physical solution. Ideally, architecture and code are inseparable faces of the very same reality: the software system.

> "THE END RESULT OF OUR LABOR IS NOT MERELY A STATIC PRODUCT OR COMMODITY BUT RATHER *A LIVING, RESPONDING ORGANISM*"

The software-engineering discipline has come a long way since the early days of its existence. Through its journey, ideas from other engineering professions have been utilized. One fundamental factor that differentiates our profession from other engineering fields is the nature of our labor, and more specifically, the outcome of our work. We start with thoughts, ideas, and basic visions and develop them into realities that can change the lives of millions. The end result of our labor is not merely a static product or commodity but rather *a living, responding organism*.

Software engineering as a discipline will continue to work with more complexity in building software systems. As if this were

not enough, the users of these systems will require more intelligent systems, simpler interfaces, and richer functionalities. Software builders will need to handle greater amounts of detail to meet more sophisticated user requirements. Though not statistically proven, experience shows that as the scope, size, and dependencies of software systems rise, complexity rises exponentially. It is evident that complexity is a key concern that we would like software architecture to be able to address.

There are many approaches to software architecture published by respected authors in the discipline. While I respect and enjoy all research documents published on this topic, I would like to keep my list of key points in handling complexity to only two factors:

- *The top-down approach in software design.* This factor emphasizes a top-to-bottom approach in designing the software architecture, starting from the top-most system (product or the big solution) and dividing it into subsystems that can be designed independently, and then approaching each subsystem iteratively in a similar manner to divide it into independent components.
- *Decomposition or breakdown of the system.* This factor means designing components that are loosely coupled and have clear and intuitive interfaces. The precondition of building suitable components is having a clear idea of the dynamic behavior of the system and subsystems. Components should address the functional requirements of the encompassing subsystem while harmoniously fitting together with other components.

These requirements have fuelled the research and the creation of the Behavioral Software Architecture Language (BSAL). The main purpose of BSAL is to:

- Provide an implementation model that can start at the system-architecture level, allowing the architect to define the system, subsystems, states, and dynamic behavior of the system and all the subsystems.
- Facilitate a *unified* platform for software architecture definition (both structure and behavior) and software implementation (coding). BSAL is meant to be the common language of the architect and the developer.
- Enable the architecture design of the system (including behavior) to be enforced in the implementation level. This enforcement can be achieved by the aid of the interactive development environments (IDEs). Separation of roles and authorization mechanisms can be clearly implemented in a BSAL development environment.
- Use a platform that can be changed and enhanced easily by further contributions in the future (BSAL is based on the OOD-OOP model).

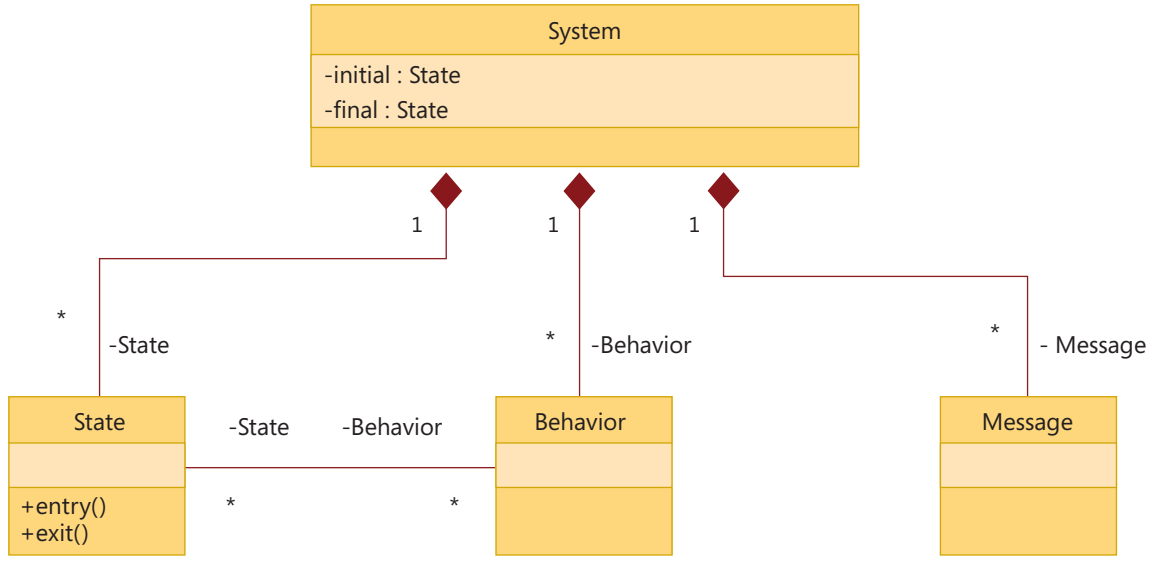
Why "Behavioral"?

To better understand the importance of behavior and why it has been magnified at the architecture design phase of BSAL, let's consider a real-world scenario. We have been given a task to design major enhancements on the membership system of a comprehensible business-to-consumer (B2C) Web site. To make the situation more critical, assume that the system has been built by a couple of previous employees who are not currently readily available to help us. Such a system would normally have features to facilitate creating new members, update member information, manipulate member rights, perform authentication and authorization, and provide member wallets (credit card, address, and invoice information).

> "AN EXPERIENCED ARCHITECT WOULD SEEK TO UNDERSTAND THE DYNAMIC BEHAVIOR OF THE SYSTEM AS IT INTERACTS WITH OTHER SYSTEMS AND RESPONDS TO USER REQUESTS, BEFORE CONSIDERING THE CHANGES OR NEW FEATURES"

Since our job is to be responsible to technically lead a major revision to the core membership system, our first priority would be to try

Figure 1 The basic foundation of systems



to understand this system in its totality. Hence, after examining the working solution we would look for documents and blueprints regarding the architecture of the system. Assuming we are lucky enough to find some class diagrams of the system, we may initially feel relieved. However, upon further investigation and analysis, we would likely find the diagrams themselves to be *incomplete* and *out of date*. As frustration builds, in our search of understanding the overall architecture of the system, we would eventually conclude that nothing but the code itself can reveal the architecture of this system. We would have to roll up our sleeves and start reading the code line by line. The code is always the definite source of information that could reveal (although not easily) to us the architecture of the system.

What piece of information would help us most in visualizing and understanding a software system? Although discovering the interaction rules and interfaces of the system, obtaining the business analysis documents of the system, having up-to-date class diagram blueprints, and getting up-to-date technical design and requirements

> "BSAL MERGES THE ARCHITECTURE (STRUCTURE AND BEHAVIOR) AND THE EXECUTABLE CODE INTO A SINGLE SOURCE"

documents would help, having the original developer of the system explain the *behavior* of the system and explain the event-conditioning and state-change sequence of the system would be the most favorable choice.

This choice is favorable because software behavior and event/state changes are very difficult to be grasped without the direct help of previous designers of the system. Although the lack of up-to-date information is a definite disadvantage in this situation, being confronted with too much information can also lead to a disaster. Both ends of the information-availability spectrum are unfavorable situations.

An experienced architect would seek to understand the dynamic behavior of the system as it interacts and responds to other systems and user requests before even considering the changes or new features. In this particular situation, the dynamic behavior of the system would be deeply embedded in the physical structure of the solution (the source code).

It's important to stress that the *behavioral aspects* of a software system play a key role in revealing the true nature of it, which is why BSAL starts the definition of a software system by outlining the subsystems, states, and behavioral patterns.

Get to Know BSAL

Let's get acquainted with the simple and yet effective language for defining software architecture (and coding). While BSAL is an architecture definition language, it is also a programming language. Although there are other aspects of software architecture that could have been emphasized, the focus in BSAL has been on providing a generally simplified model of architecture definition while facilitating a flexible programming model.

Considering the behavioral aspects of the software systems to be most important, BASL emphasizes the definition of behavior patterns in the architecture definition phase. In fact, without defining the behavioral patterns of the system you cannot proceed to implementation of lower models in the system. The programming language syntax of BSAL could be any modern object-oriented programming language. While the principles of the language can be applied to any modern OOP language (and I certainly hope to see that in the future), C#.NET was used here to convey the BSAL ideas discussed. Before proceeding any further, it may be helpful to provide a role-based usage scenario for BSAL:

- The software architect uses the language to define the major characteristics of the software system. Namely, system, subsystem, state, behavior, and message objects are defined by the architect. These are the high-level component definitions for any software solution.
- The software engineer uses the direct *output* of the architect's work (the high-level BSAL source code) as the *input* to his or her work, provides detailed design, and starts implementing states, messages, components, and classes inside subsystems.
- The test engineer uses the BSAL source code to analyze system behavior and create black-and-white box test scenarios.
- The analyst uses the BSAL source code to analyze high-level system breakdown and component definitions and understand the system behavior before proposing further enhancements to the system.

BSAL is based on the object-oriented paradigm. The important addition that BSAL brings into play is the common, standard usage of a few building blocks in software definition. These building blocks can themselves be created using a modern OOP language. The basic rule of BSAL is that every piece of code must be inside a *system* object. A system object itself can encapsulate smaller systems or *subsystems*. The system object can contain fields (attributes), must have at least two state objects, and can have behavior and message objects. Executable code can only be written inside state objects of the system, meaning that the system object (or subsystem object for that matter), behavior object, and message objects solely define the *behavior and structure* of the system (the architecture). However, almost all of the programmer's executable code will reside inside the state objects, which greatly simplifies the task of architecture design and carries the architecture down into the programmer's code.

Component Breakdown

Low-level components of BSAL could be implemented in any modern object-oriented programming language such as C#.NET, Java, or C++ (the details of these components are beyond the scope of this discussion). Presentation of a full working model of BSAL will be left as the main theme of a future discussion. The basic high-level components of BSAL are system, state, behavior, and message objects. These are the objects that the architects of the system usually define. These objects draw and define the basic boundaries and foundations of the system (see Figure 1).

The system object is the top-most BSAL component. It is the object that encapsulates all other components of the system. It can be spread across multiple files, modules, and/or packages. A system can be composed of one or more subsystems. A system can encapsulate subsystems, states, behaviors, messages, and custom fields and attributes. The system must have at least two states,

namely, *initial state* and *final state*. When the system is started, it immediately goes into the initial state; when the system is signaled to shut down, it goes into the final state. A system can have unlimited custom states.

A system has an initial state and a final state; can receive messages (input to the system); can check behavioral conditions so that it can change and manage states accordingly; can implement behavioral patterns by activating and finalizing states; and can send messages to other systems (output from the system):

```
public class member : System
{
protected State
createMember;
public member(
State istate, State
fstate) : base(
istate, fstate)
{
}
}
```

The state object defines a stage that the system can go through to perform a specific task. State definitions organize work units in a certain logical order. States implement the primary functionality of the system.

A state object has two required methods named entry() and exit(). The entry() method is called when the state is activated; the exit() method is called when the state is signaled for completion. A state object can have unlimited custom methods.

A system can change states internally, or it can change state in response to an outside interaction with other systems or environments. (For example, receiving a message from another system can trigger a change of state in a system.) The common place where a state change is implemented inside a behavior object or another state's exit() method.

A state object has an entry() and an exit() method, can be activated through a behavior object or another state object, can complete itself, must check the behavior objects related to it when completed, can have custom properties and methods, and can run as a separate thread of execution (parallel states):

```
public class MyInitState :
State
{
public MyInitState(
string sn, int sid) :
base(sn, sid)
{
}
public override int entry()
{
// do some stuff
return 0;
}
public override int exit()
{
// do some stuff
return 0;
}
}
```

The behavior object is where the behavioral patterns of the system are defined. Behavior definition is *the heart* of the state management of a software system. Typically, a behavior definition contains

> "AS INFORMATION TECHNOLOGY PROBLEMS CONTINUE TO GET MORE COMPLEX, SOFTWARE ENGINEERS NEED LANGUAGES (AND PROGRAMMING MODELS) THAT CAN HIDE AND ENCAPSULATE GREATER LEVELS OF DETAIL"

one or more conditions-related method calls to complete and/or activate certain states of the system. Behavior definitions should not contain any executable code other than those resulting in change of state, setting a field or attribute, or sending a message to other systems. The sole purpose here is to define the behavior of the system under certain conditions or circumstances. A behavior object basically encapsulates conditions, changes in attributes, sending messages, and completing and activating states.

Good for the Profession

A message object encapsulates the synchronous/asynchronous information flow between different systems. A system object can only receive message objects that have been defined for it. A message definition typically contains a message ID, a message header, and a message body. The underlying implementation of messages could be left to the specific environment and framework in use. The behavioral pattern of the system is checked either when the system receives any new messages or (alternatively) when any state inside the system is being completed.

As information technology problems continue to get more complex, software engineers need languages (and programming models) that can hide and encapsulate greater levels of detail. As we discussed here, the traditional OOP development analogy backed up with software modeling notations can be limiting and at times cumbersome. BSAL merges the architecture (structure and behavior) and the executable code into a single source. Architects and software engineers need to communicate, share, and build ideas. To come up with alternative or improved solutions for live systems, they need to quickly grasp the essence of software systems. Being able to understand quickly and precisely a software system is a great virtue. BSAL is a possible answer to these needs and hopefully a positive step in opening up new horizons in the future of the software-engineering profession. •

About the Author

Behzad Karim (MCSD.NET, MCT) is a software project manager at TEPUM SIGMA Consulting and Development Center (www.sigma.net.tr) in Istanbul, Turkey, where he leads software development teams delivering EAI projects. Behzad has been a software developer and software architect for nearly 18 years. Contact Behzad at **bkarim@sigma.net.tr**.

Microsoft®

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes