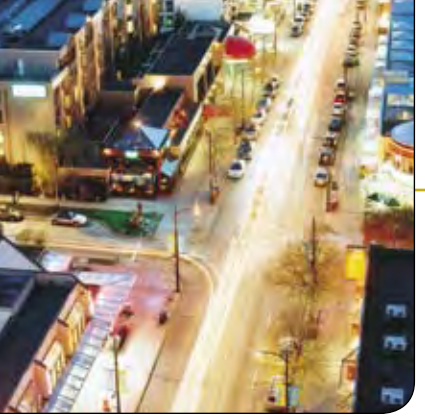
Learn the discipline, pursue the art, and contribute ideas at www.ArchitectureJournal.net
Resources you can build on.

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes
Journal 7

Generation Workflow

Microsoft®



Contents

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes
Journal 7

Foreword

Dear Architect,

I remember the morning well. The day was clear and cold, and I was in the Microsoft office in London sitting with Arvindra Sehmi as he explained his vision and showed me one of the first exciting prototypes of the *Journal*. As you may know, from that morning onward the *Journal* has gone from strength to strength. You are reading the 7th issue of the *Journal*, which reaches over 30,000 subscribers worldwide—both in printed and online format through our new site, ArchitectureJournal.net, that launched last month. As I was chatting with Vin, little did I know that three years later I would have the honor of taking over as editor of the magazine.

Although there are no major changes to the format, as editor I do plan to introduce a couple of new directions for *The Architecture Journal*. First, given the growing subscription base we will be localizing it into eight languages worldwide. If you are reading this issue in a language other than English, I'd like to be first to welcome you to this new format and hope that it sets a continuing trend.

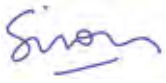
Second, from this issue we are moving toward a themed approach, where the majority of the articles in each issue fit a common theme or pattern. As you may have seen from the cover, the theme for this issue is "Generation Workflow." To kick off the theme we start with an article by Dave Green, an architect for our Windows Workflow Foundation (WF) product. Dave will be sharing the decisions and reasoning his team faced while building a workflow platform, which ultimately shaped the product they are developing.

Next we have Vignesh Swaminathan, a product manager from Cordys R&D. In his article Vignesh will be taking you on an "Amazing Race" and will be looking at the similarities between workflow decision matrices in enterprises and a popular television show. To add more of a human touch to the *Journal*, Jesus Rodriguez and Javier Mariscal will be covering the main components of a human workflow system and describing patterns that can be used to model human-related tasks.

To hook these pieces together, Kevin Francis will be discussing integrating applications with workflow in his article, "Workflow in Application Integration." And to complete the set of workflow articles in this issue, Andrew Needleman will be showing a technique he uses called "dots and lines" to simplify the process of communicating complex workflows with business experts.

Following the articles on workflow we are lucky enough to have William Oellermann's article on the Enterprise Service Orientation Maturity Model (ESOMM), a maturity model that looks at service management in the enterprise. Finally, to wrap this issue, we have decided not to let Arvindra off the hook completely, and he'll be returning with Beat Schwegler to deliver Part 1 of an excellent overview of "Service-Oriented Modeling for Connected Systems." Part 2 will appear in the *Journal*'s 8th issue.

Overall, it's been a great ride putting my first issue together, and I hope you'll find the articles useful and thought provoking. As the new editor I would like to welcome you again, and look forward to hearing from many of you in the near future.



Simon Guest

Build Applications on a Workflow Platform

by David Green

Summary

A workflow can be useful for resolving business issues. Here we'll get acquainted with and investigate the idea of building applications on a workflow platform. A workflow platform supports key workflow concepts and provides a basis for building applications structured using these concepts, including workflow products as hitherto understood. We'll survey a range of applications to explore the necessary characteristics of a workflow platform, which leads to a discussion of the potential benefits of building applications on a workflow platform. We'll also discuss the Windows Workflow Foundation as a means of realizing these benefits in practice.

The idea of workflow has been around for a long time, and it has been consistently attractive as a way to attack business problems. The problems to which a workflow approach has been applied often display three characteristics: The key business value delivered is *coordination*, for instance, in organizing multiple contributions to the preparation of a quote or driving a document review. Each instance of the business process concerned is of *long duration*, measured often in days, weeks, or months rather than minutes. The business process has *human* participants, who usually contribute most of the work product.

However, only a small proportion of the business problems with these characteristics are solved using a workflow approach. Most commonly, the business process is not recorded as machine-readable information at all. Rather, each of the humans participating in the business process interacts with business systems that are not aware of the semantics of the process as a whole, such as a customer information system, and with other human participants through content-neutral communications channels such as e-mail. Each human participant uses a mental model of their part in the overall business process to determine their behavior.

The attractions of creating a machine-readable model of the business process, that is, a workflow, are not difficult to envisage. Three key benefits that a workflow model can bring are insight, monitoring, and optimization. A set of related workflow models can be used to gain *insight* into the flow of work through an organization. For *monitoring*, knowing which individuals are contributing work to which business process is very useful when trying to understand costs and workloads. For *optimization*, having a model of the work being undertaken, and being able to use the model to interpret behavior, together make it possible to reason about how to optimize the business process.

A Workflow Model

Given these compelling benefits, why haven't workflow models been used more widely? The most likely answer is that the cost of using them has been too high. These costs include *product* costs, that is, the direct cost of purchasing a workflow product; *integration* costs, where processes modeled as workflows need to be integrated as part of a larger business system; and *standardization* costs, where it is difficult for a large organization to standardize on a single workflow technology. Variations in workflow products also mean that skills and model portability are issues.

Let's look at the possibility of addressing these blocking issues by building applications on a workflow platform that is low cost, ubiquitous, uniform, and easily integrated in applications. To be clear, the idea is not to replace workflow products. Rather, the hypothesis is that it is useful to factor out support for some core workflow concepts into a platform on which both workflow products and other applications can be built (see Figure 1).

A workflow is a *model*, which means it is a machine-readable description of business behavior that is not code. The meaning and benefits of this concept in the context of the value of a workflow platform will be discussed later.

A workflow model describes an organization of *work units*. For instance, suppose that a document review process specifies that Joe writes the document and then Fred reviews it. Here, the work units are first writing and second reviewing the document, and the organization is that one task must follow the other. This concept is not a radical idea. Code that makes successive calls to two subroutines is a valid example of the concept. The interest lies rather in the forms that this organization takes.

To test the workflow platform hypothesis, we will consider a range of real-world applications and explore the characteristics that a workflow platform should have if it is to prove useful.

A *document review* process takes as an input parameter a set of [reviewer, role] pairs that describe which humans are involved in the workflow in which roles. Possible values for the role are required, optional, final approver, and owner. The review process then proceeds until all reviewers have performed their assigned roles and notifies the owner of the outcome.

Here, the work items are the document reviews organized by the review process. There are three interesting characteristics to call out, namely, multiple points of interaction, human and automated activity, and the need to handle dynamic change.

Workflow Contracts

The workflow has multiple points of interaction, or contracts. First, there is a contract with a reviewer. This contract involves asking a reviewer to review a document, accepting the verdict and any review comments, and also telling a reviewer that his or her input is no longer required (if the review is canceled, or perhaps if enough reviewers have voted yes). The contract might also allow a reviewer to delegate a review. Then there is a second contract with the final approver, which is a specialization of the reviewer contract. Third, there is a contract with the owner of the review that allows the owner to cancel the review and be notified of the outcome of the review. Finally, there is a contract with the initiator of the review process, who instantiates the review and supplies the required parameters.

It is typical of workflows that they connect multiple parties through a variety of contracts (see Figure 2). The document review workflow is essentially a coordinator, initiated through one contract, that is coordinating a variety of participants through one or more additional contracts.

The *document review* workflow drives human activity. However, it might also drive automated activities, such as storing versions of the document in a repository as the review progresses. From the point of view of the workflow, there is no essential difference. A workflow can be thought of as communicating, in general, with services through contracts. One special case of a service is another workflow. Another special case is a human. In many ways, a human is the original asynchronous service: one never knows when or if it is going to respond.

A characteristic of this type of workflow is that the participants will ask for changes to the workflow as it executes. For example, a reviewer might delegate a review task to a colleague or share the work involved in a review task with a subordinate.

There are two ways of addressing this requirement. One is to build an understanding of all the possible changes into the workflow. Then, a delegation request becomes just another function of the contract between the workflow and the reviewer. The other possibility is to see change as something separate from the workflow, where change is implemented as an external function that changes the workflow model. In this approach, the result of delegation is a new workflow model identical to one in which the review task was assigned to the delegate from the beginning.

Requesting an additional approval step would add a new approval task to the workflow model, which might well have contained no approval steps at all in its original form. The workflow no longer has to anticipate all possible modifications; at the most it will be concerned with restricting the areas of the model that are subject to change.

Both approaches are useful. Building understanding into a workflow is simple to model and understand. Generalizing operations is more complex to model, but more powerful and agile.

In an extreme but interesting case of the latter approach, the workflow begins execution with little or no content, and the required behavior is added dynamically by the participants in the workflow. Here, the available operations for modifying the workflow become a vocabulary that a user can use to construct the desired behavior as the workflow progresses.

Problem-Resolution Collaboration

To look at a specific example of a *problem-resolution collaboration* application, consider an inventory shortfall. An assembly line is making a gadget, and the computer indicated that there were enough widgets in stock for the purpose. However, when the stockroom manager went to fetch the widgets for delivery to the assembly line, a shortfall of 10 widgets was discovered.

Collaboration among the stockroom manager, the customer's account manager, the supplies department, and the production manager is required to resolve the problem. Each role in the collaboration may take characteristic actions. The supplies department could order more widgets, perhaps using a different supplier or paying an existing supplier more money for faster delivery. The account manager could go to the customer and request deferred delivery or split the delivery into two parts and bear the extra shipping cost. The production manager could divert assembled gadgets from an order for another customer. The stockroom manager could search his or her physical stock in an attempt to find the missing widgets. Any given action might be performed multiple times.

> "IN MANY WAYS, A HUMAN IS THE ORIGINAL ASYNCHRONOUS SERVICE: ONE NEVER KNOWS WHEN OR IF IT IS GOING TO RESPOND"

One obvious constraint is that the collaboration is not completed until the shortfall is resolved by some combination of the previous actions. There will often also be business constraints. For instance, there might be a rule that says deferral of delivery to gold customers is never permitted. Also, the actions will affect each other. For instance, there might be a policy that states that total added cost from corrective action may not exceed 5 percent of original factory cost. Thus, placing an order for accelerated supplies at a higher price might prevent a shipment from being split.

Figure 1 A monolithic workflow to stack

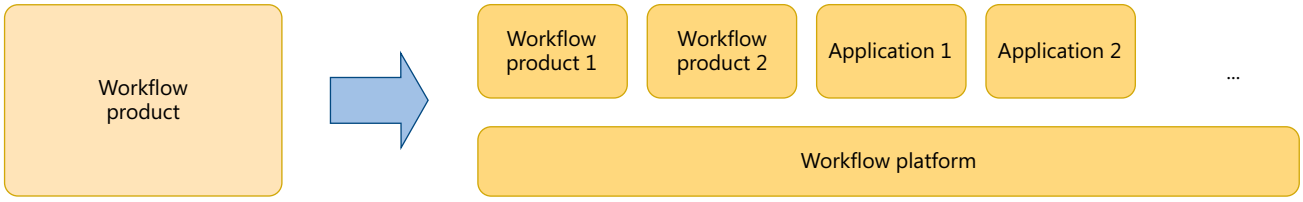
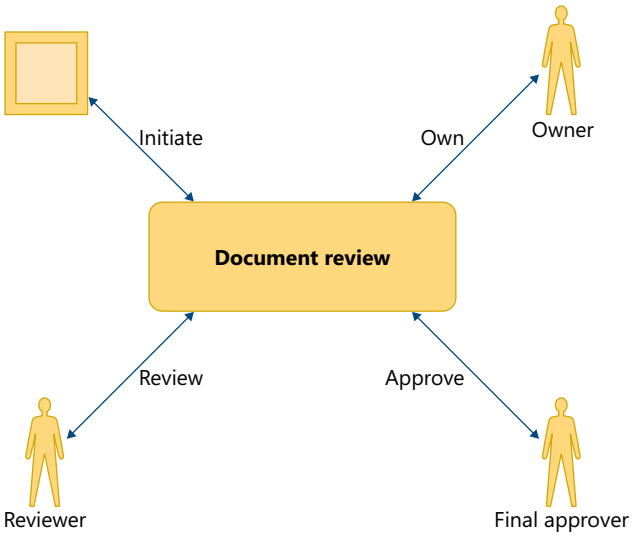


Figure 2 A contract diagram for the document review application



In this case the work items are the actions that the various participants can take as they seek to resolve the inventory shortfall. The organization, however, is not the same as that required in document review. The participants are not dictated to; instead, they choose which actions to perform and when to perform them. However, these choices are constrained by the organization of the workflow, which has two aspects: 1) The actions are focused on achieving a goal; in this case, resolving the inventory shortfall. A bounded collaboration space is created when the problem resolution starts, and is not closed until the goal has been reached. 2) The participants are not free to perform arbitrary actions. Instead, the available actions are determined by the role the participant is performing and the state of the collaboration. The set of actions available is determined by policies related to the goal and global policies such as the restriction on shorting gold customers. The actions available vary as the collaboration progresses.

The experience of the participant is no longer that of performing assigned tasks. Instead, a participant queries for the actions currently available to him or her, performs none or one of these actions, and then repeats the cycle.

The main new requirement here, therefore, is for a form of organization of work items that is essentially data state and goal driven. There is also a requirement to support a query/act-style of contract with a workflow participant.

Scripted Operations

Scripted operations are simply a set of operations that are composed using a script. An example might be a desktop tool that allows a user to define and execute a series of common tasks, such as copying files and annotating them.

It would be unusual to consider using a typical workflow product for this purpose. However, it does fit the workflow platform pattern of a set of work units organized by a model. In this case the model is a sequence, perhaps with support for looping and conditional execution. Therefore, if a workflow platform were sufficiently low cost and ubiquitous, it would be possible to consider applying it to this sort of problem. Would doing so add any value?

One feature of scripted operations that is not addressed by their typical implementations today is the question of data flow. It is common for the data required by one operation to be the output of some previous operation, but this information is not typically modeled in the script. Thus, a user assembling tasks using a desktop tool might not be told when creating a script that the prerequisite data for a task hasn't been supplied, and would only discover the error when running the script. A workflow model that can describe these data dependencies would add clear value for script authors.

One approach is simply to include data flow constructs in the workflow model. It is highly arguable that the basic workflow model needs to include basic structural features such as sequences, conditions, and loops; but it is not clear that data flow is sufficiently universal to be represented by first-class elements of the model.

An alternative approach is to layer support for data flow on top of an extensible, basic workflow. A workflow model that can be enriched with abstractions appropriate to a variety of problem domains fits well with the notion of a workflow platform. This approach avoids both the complexity created by including in the base model a large variety of semantic constructs specialized for different problems and also the limitations imposed by limiting the workflow model to a fixed set of constructs.

Now let's look at a *guided user* application. One example is an interactive voice response (IVR) system, and another is a call center system guiding telephone operators through support or sales scripts.

> "A WORKFLOW MODEL THAT EXPRESSES THE CORE BUSINESS PURPOSE OF THE APPLICATION, STRIPPED OF ANY IRRELEVANT TECHNICAL MATERIAL, IS AN EFFECTIVE WAY TO ACHIEVE COMMUNICATION BETWEEN IT STAFF AND BUSINESS PERSONNEL"

The essence of these applications is to guide users through the series of operations needed to achieve their goal. The organization of these operations is typically used to drive the presentation to the user, whether this is generated speech or a set of enabled and disabled command buttons on a form.

A characteristic of this type of application is that the workflow is the most frequently changed part of the application. Also, the business sponsors of the system are often heavily involved in specifying the changes, making it important to provide a way for IT staff and business personnel to communicate clearly and efficiently about the changes. A workflow model that expresses the core business purpose of the application, stripped of any irrelevant technical material, is an effective way to achieve this communication.

These applications also require flexibility within the workflow structure. In an IVR application the user will typically be heavily constrained, moving through a hierarchically structured set of menus. However, there will also be escape commands—for example, "return to root menu" or "skip out of current subtree."

A call center application will have more flexibility than an IVR application, changing the options offered to the user in response to

the state of an order or in response to the input from a customer, such as skipping sales prompts if the customer starts to react negatively.

This sort of application requires support for a mix of organizations of work items, combining sequences, loops, and conditions with jumps from one state to another, and also the kind of data-driven behavior seen in problem-resolution collaboration.

Rule and Policy

As discussed previously, one way in which the workflow approach can deliver value is by isolating the focus of change in an application. Often, this focus is on the way in which the work items are structured, but in some applications the focus of change is on expressions tied to a relatively slow-changing structure.

An example of this focus is an insurance policy quotation system, where a set of frequently changing calculations is used to drive decision making in the quotation process. The requirement is for the workflow to model these expressions, which has two key benefits: First, the testing and deployment costs are much lower than those that would typically be incurred if the expressions were written as code, since the model provides a strong sandbox restricting the scope of possible changes. Second, the changes can be made by personnel who understand the business significance of the expressions but do not have the skills to understand the technical code in which expressions written as code would inevitably need to be embedded.

The Model-View-Controller (MVC) pattern often is used to wire a UI to an underlying object model (see Figure 3). The *model* represents the behavior of the system, independent of any particular UI representation. The *controller* is a part of the UI layer that is used to map the events generated by the UI into the method invocations required to drive the model. The UI itself is thus not polluted by any assumptions about the underlying model.

The workflows considered so far, viewed from this standpoint, all fall into the category of Models in the MVC sense. However, the controller can also be seen as a workflow. The work items it organizes are the methods provided by Model objects. The controller also interacts with the UI and the model through well-defined contracts. A model of this kind is often termed a *page flow*.

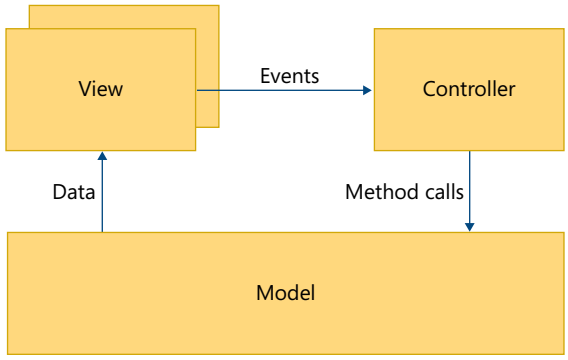
As with scripted operations, page flow would not today be implemented using a typical workflow product. There are two reasons to consider building a page flow using a workflow platform. First, a model readily can be represented visually, helping developers and analysts to express and communicate the required behavior. Second, if the page flow is frequently changing, then the abstraction of the page flow as a model improves agility.

There are two main requirements if this problem is to be addressed using a workflow platform. The workflow runtime must be lightweight, since a page flow may be running within a small application on a desktop, and the contracts supported must include the event-based contract characteristic of UIs, as well as the object-method contracts exposed by the Model.

Now let's look at a *test record/replay* application example. The intent of this final example is to test the limits of the applicability of the workflow platform hypothesis.

The application here is a tool for testing applications built as a set of services. The tool uses an interception mechanism to record all the interaction between services that occur during manual performance

Figure 3 An MVC application



of a test case for the application. This recording can then be replayed. During replay, externally sourced messages are generated without manual intervention, and messages between the set of services that make up the application are checked for sequence and content against the original recording.

The workflow is the test case, organizing the work units that are the participating services. The workflow is both active, in that it simulates the behavior of externally sourced messages, and passive, in that it monitors the interactions between services.

A unique feature of this application is that the workflow is written, not by a developer or a user, but by a program, as part of the act of recording a test case. Workflow model creation must be fully programmable. There are also requirements for extensibility and dynamic update.

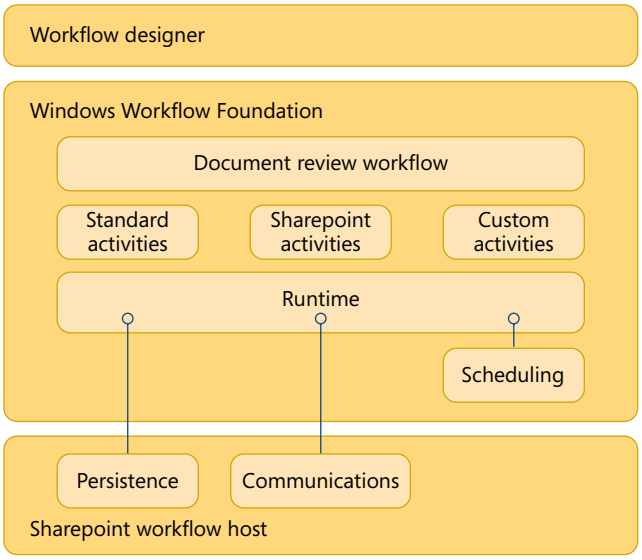
Extensibility is required because the structural semantics are rich. For instance, just because two messages arrived at a service one after the other in the recording, there is no necessary implication that this order needs to be preserved in a replay. If there is no causal dependency between the messages, then a replay that reverses the order of the messages is correct. So the semantics of sequence in the model used to record the test cases need to include a notion of causality, which is not likely to be a feature of the core workflow model of sequence.

Dynamic update is required because human interaction with the model will occur during replay. Discrepancies discovered during replay between recorded and observed behavior are popped up to a tester. If the discrepancy is because a message includes a timestamp that varies from run to run, then the tester would update the model to mark the field "don't care." If the discrepancy occurs in a regression test because the software has changed, then the tester might approve the change and update the test to expect the new behavior in all subsequent runs.

Workflow Platform Value

A workflow platform does not, by definition, have the full set of features offered by today's typical workflow products. Rather, the workflow platform considered here focuses on supporting the concept of a workflow as a model of the organization of work items. We have seen that this idea of an organization of work items is indeed applicable across a broad range of applications, but the fact that a workflow platform can be used doesn't mean that it should be used.

Figure 4 The document review implementation schematic



Two questions must be asked: What additional value is derived from the workflow platform approach? And, is this approach practical? This value of the workflow platform approach must come from the expression of the organization of work as a model, which we'll discuss later. Let's summarize the characteristics that a practical and effective workflow platform must display.

To demonstrate how a model differs from code, this code snippet is a valid workflow by the definition used here, that is, an organization of work units:

```
public void HandleLoanRequest (
  string customerID,
  Application app)
{
  if (CheckCredit(
    customerId, app.Amount))
  {
    MakeOffer (customerId, app);
  }
}
```

And, in a sense, it is a model. It is possible to parse this code and build a CodeDOM tree that represents it.

However, the semantics of the resulting model are so general as to be opaque. It is possible to tell that the code contains function invocations, but it isn't too easy to distinguish a function that represents the invocation of a work item from a function that converts integers to strings. A workflow model explicitly distinguishes these ideas. Typically, a specialized model element is used to represent the invocation of a work item, and conversion functions cannot be expressed directly in the model at all. A workflow model, then, is one in which its graph is built from elements that are meaningful in the workflow domain. The semantic richness of such a model can be exploited in several ways.

Visualization. Visual representation of the model—typically in graphical form—is useful for the developer, during both development and maintenance, and also for workflow users who want to know why they have been assigned a given task or the IT operations worker who wants to understand what a misbehaving application should be doing.

Insight. The workflow model is amenable to programmatic access for a variety of purposes. An examples is static analysis to determine dependencies and flow of work across a set of cooperating workflows or using the model to drive a simulation that predicts the workloads that will be generated by a new version of a process.

Expressiveness. The specialization of the workflow model to the workflow domain means that characteristic problems can be expressed more quickly and compactly. It is a domain-specific language (DSL), specialized to support characteristic problems. Consider a document review process where three positive votes out of five reviews mean that the document is good, and any outstanding reviews can be canceled. This process is quite difficult to code, but a workflow model can supply out-of-the-box constructions that address such problems.

More Semantic Exploitation

As we have seen in the scripted operations application discussion, extending the workflow model to further specialize the out-of-the-box model language is a very powerful technique for delivered additional value. An example is the creation of a language intended for end users, as in the document review conducted using an improvised definition of the review process that was discussed previously.

Execution. The specialization of the model makes it possible to add run-time support for common problems. A good example is long-running state. Of the applications discussed here, management of long-running state is required for the document review process, problem-resolution collaboration, and guided user applications. The workflow platform runtime can solve such difficult problems once, using simple expressive model elements to control a common capability and freeing up the developer to focus on the business problem.

Monitoring. The existence of a model makes it possible to produce an event stream with a meaningful semantic without any additional developer effort. Of the applications described here, this

> "WF IMPLEMENTS THE IDEA OF WORKFLOW AS AN ORGANIZATION OF WORK ITEMS, ABSTRACTED AWAY FROM THE RELATED IDEAS WITH WHICH IT HAS BEEN COUPLED IN TRADITIONAL WORKFLOW PRODUCTS"

event stream is useful in the document review, problem-resolution collaboration, test record/replay, and guided user applications. The event stream can be used to monitor instances of workflows or build aggregate views of the state of a large number of workflow instances. The standardization of the event stream makes it much easier to build such aggregate views across workflows that were developed independently of each other.

Another powerful idea is the presentation of errors using a business semantic. Often, a technical failure such as the nondelivery of a

message leads to escalation to a technical expert because the significance of the failure is unclear without specialist investigation. If the error can be mapped to a workflow model—so that it is clear that the error concerns a noncritical change notification, for instance—then escalation can be restricted to cases where it is necessary.

Composition. If an application is factored into work units, then these work units, with their well-understood interfaces, can be reused by other workflows. Workflows themselves also define work units that can also be used by other workflows.

Customization. Suppose that an ISV ships a workflow, which is customized by a VAR, and then again by a customer. Reapplying these customizations when the ISV ships a new base version is a challenging maintenance problem. The use of a shared, well-understood model for the workflow makes the consequent three-way merges much more tractable. Customization and composition together enable ecosystems where definitions of work and flow become shared or traded artifacts.

Manipulation. As we have seen in the discussions of the document review and test record/replay applications, there are often requirements to invent or modify workflows on the fly. This modification cannot be done securely if changing code is required. Using a model makes possible dynamic manipulation that is both controllable and comprehensible.

These benefits make a compelling list, and it demonstrates clearly that the description of an organization of work items as a model has a lot to offer.

Platform Characteristics

There must be support for basic structural concepts like sequences, conditions, and loops. However, there also needs to be support for data-driven approaches to deal with the less-structured organizations that appear in applications like problem-resolution collaboration and guided user.

It is also important to allow new semantic elements to be added to create rich, specialized languages such as the data flow-aware composition in scripted operations. Adding new semantic elements might go so far as to require the redefinition of such fundamental ideas as sequence—for example, in the test record/replay application.

The workflow must also be able to communicate in a rich variety of ways. Workflows respond to UI events, drive different types of services (human, programmatic, other workflows), and support queries over the current state of their contracts—for instance, when determining the actions available to an actor in a problem-resolution collaboration application.

If the workflow platform is to be used in all the applications where it adds value, such as MVC, then it must be lightweight. Equally, it needs to address the scale and performance requirements implied by applications such as document review.

In addition, the workflow model itself must be fully programmable, which includes model creation—such as in the test record/replay application—and dynamic model update to support unanticipated change, as in both the document review and test record/replay applications.

Now let's look at the realization of these required characteristics in the Windows Workflow Foundation (WF). Thus far we have effectively recapitulated the thinking that drove the development of WF. WF as a realization of these concepts is the means by which this value can be translated into delivered solutions.

WF implements the idea of workflow as an organization of work items, abstracted away from the related ideas with which it has been coupled in traditional workflow products. The abstractions fall under three main categories: design and visualization, hosting, and semantics.

Design and visualization. A workflow in WF is a tree of work items (called activities). This tree can be manipulated directly as an object model. A designer is provided, but its use is not mandated. It is possible to create new designers specialized to particular user communities or to particular organizations of work items. It is also possible to specialize the provided designer, which can be used not only within Visual Studio but from within an arbitrary hosting application.

> "THE FOCUS OF THE WF RUNTIME IS TO DELIVER FACILITIES REQUIRED BY ANY WORKFLOW, AND THEREFORE AVOID THE NEED TO RE-IMPLEMENT THEM TIME AND AGAIN IN DIFFERENT APPLICATIONS, BUT WITHOUT COMPROMISING THE FLEXIBILITY OF THE WORKFLOW ABSTRACTION"

Hosting. The WF runtime is sufficiently lightweight to be hosted in a client context such as a controller in a rich-client application shell. It is also performant enough to scale when embedded in a server host, such as the Sharepoint Server delivered by Office 2007. The WF runtime's expectations of its host are abstracted as provider interfaces for services such as threading, transactions, persistence, and communications. Useful provider implementations are supplied out of the box, but they may be substituted as required.

Semantics. Different problems respond to different model semantics. WF supports three main styles of workflow out of the box: flow, state machine, and data driven. Flow is optimal for applications where the workflow is in control such as the scripted operations example. State machine is best when the workflow is driven by external events, as in the MVC or guided user applications. A data-driven approach is suited to applications where actions depend on state, as in problem-resolution collaboration.

These semantics can be extended by building custom activities to create a domain-specific vocabulary for use in any of these styles. However, since the structure of a workflow is itself expressed as a set of activities, the same approach can be used to define new styles, and entirely novel semantics, if required.

A Common Workflow Runtime

The focus of the WF runtime is to deliver facilities required by any workflow, and therefore avoid the need to re-implement them time and again in different applications, but without compromising the flexibility of the workflow abstraction. These common facilities fall into four main categories: activity scheduling, transactions and long-running state, exceptions and compensation, and communications. Let's look at each in more detail.

Activity scheduling. The WF runtime defines an activity protocol that all work items implement. This protocol defines the basic activity life cycle (initialized, executing, and closed) and the additional states needed to handle exceptions (faulted, canceling, and compensating). This definition enables the WF runtime to provide work scheduling for all workflows.

Transactions and long-running state. The WF runtime supports the execution of ACID transactions. These transactions are particularly useful for maintaining consistency across workflow state and external state such as application and message state. However, ACID transactions are not suitable for managing long-running state because of their resource and locking implications. The WF runtime implements a broader checkpoint-and-recovery mechanism to handle long-running state. From this point of view, ACID transactions become units of execution within a larger framework. The developer needs not do any work to get the benefit of WF's support for long-running state, as it is default behavior. However, if more detailed control is required, a set of simple model elements are supplied for the purpose.

Exceptions and compensation. The familiar idea of throw-try-catch exceptions is supported by the WF runtime and represented in the out-of-the-box workflow model. However, the WF runtime also supports a broader view of fault handling that includes the idea of compensation for successfully completed transactional units.

> "ACID TRANSACTIONS ARE PARTICULARLY USEFUL FOR MAINTAINING CONSISTENCY ACROSS WORKFLOW STATE AND EXTERNAL STATE SUCH AS APPLICATION AND MESSAGE STATE"

Communications. As we have seen, workflows need to communicate in a variety of ways, which is reflected in the WF, that supports communication through .NET method, event interfaces, and Web service interfaces. Support for Windows Communication Framework will also be made available in the future. Thus, WF does indeed realize the workflow-platform approach proposed here.

Figure 4 illustrates the high-level implementation schematic of the document review application and how all of the foregoing comes together. An implementation uses Sharepoint as the workflow host. The WF runtime uses the default scheduling service provided out of the box with WF. However, the default persistence and communications services are replaced with implementations specialized for the Sharepoint host. The persistence service stores long-running workflow state in the Sharepoint database, and the communications service makes the rich-user interaction facilities of Sharepoint available to the workflow. Both of these services are in fact delivered out of the box with Microsoft Office 2007.

Three sorts of activities are used to define the document review workflow itself. First, out-of-the-box WF activities are used to provide structural elements such as If-Else and While. Second, activities provided as part of Office 2007 are used to access the user communication services of Sharepoint. Third, custom activities are used to implement organization-specific semantics for forwarding and delegation in a standard and reusable way. The WF designer is used as a means to define the workflow and also provide diagrammatic representations of the state of a document review workflow instance to the workflow owner.

Attacking the Problems

In summary, the workflow platform supports an abstraction of the ideas that have made workflow products an attractive attack on business problems. It does not replace today's workflow products, however. Rather, it factors them into platform and superstructure.

The workflow platform embodies two key ideas: a workflow is an *organization of work units*, and a workflow is a *model*, that is, a machine-readable description other than code. These ideas are valuable in a broad range of applications, both within and beyond the problem domain addressed by typical workflow products. Such a workflow platform is most useful if it is low cost and ubiquitous.

The principal benefits delivered arise from the expression of an organization of work items as a model, which has several advantages over a representation in code:

- *Transparency*. The business purposes of the system are clear, allowing users and IT staff to communicate effectively about the desired behavior and IT staff coming onto the project to get up to speed quickly.
- *Isolation of change*. The areas of the application most likely to change are expressed as workflow rather than code. By isolating the rapidly moving parts of the application, changes can be made more reliably.
- *Agility*. The bottom line of all these benefits is business agility. If business users can understand the system, developers can get up to speed quickly, and the risks associated with change are minimized. Then the system may be termed agile.

A broadly useful workflow platform must have these characteristics: define a core workflow model as a standard that is extensible and fully programmable at design time and runtime, be able to communicate in a rich variety of ways, be lightweight and embeddable, and be able to scale and perform well in high-volume environments. WF is a product that displays all of these characteristics. As a component of WinFx and a part of the Windows platform, WF is also low cost and ubiquitous.

The concept of a workflow platform, as described here, is an appropriate and customizable model with multiple benefits for a wide range of applications, and it is fully realized in WF. •

About the Author

David Green joined IBM as a developer in the Hursley labs in 1977. Since then, he's moved up and down the software supply chain, working for a newspaper company, American Express, Siemens Nixdorf, and a major UK bank in a variety of technical, presales, and post-sales roles. The theme that runs through all of his experience is building applications for the day-to-day business world, thinking it was far more difficult than it should be to create great business solutions, and trying to create approaches and tools to do something about it. David has spent the last two years at Microsoft working on Windows Workflow Foundation, which he believes is a significant addition to the application builder's armory.

The Amazing Race Metaphor

by Vignesh Swaminathan

Summary

Business Process Management (BPM) is buzzing in everyone's ears today, and many enterprises realize the business potential of automating their business processes. As the buzz settles down and BPM approaches the plateau of productivity, there are some practical offshoots. One of them is a need for better management of high-level business processes that are composed from simple, reusable business processes. The answer, interestingly, is not entirely through graphical modeling but is the management of high-level business processes through externalized, process-definition rules. We'll look at the definition, benefits, and implementation of process definition rules.

CBS's *The Amazing Race* is a reality television show aired in the U.S. in which participants compete to race around the globe by going from city to city. To reach one city from another they take a common means of travel such as a car, train, or plane. The participating teams have to find a clue in each city to figure out the next city to where they have to travel. The clues control the choice of the next route map to follow; neither the participant nor the map itself is aware of or contains these clues. That is, the clues are externalized from the process of getting from one city to another.

Using this analogy, the entire process of moving around the globe (let's call it the *globe process*) is achieved by combining smaller processes of getting from one city to another (let's call them the *city processes*), thus creating the game show. The city processes that make up a particular globe process in turn are determined by the clues. The participants also decide, for instance, to take a flight if the city is not approachable by road or rail. These are simple internal rules that are applied within the city process. The clues on the other hand are entirely external to the city process and control the globe process.

An interesting twist to the game would be if the clues determine the next city process based on the characteristics of the players, the city they were previously in, and other relevant parameters. This process would lead to a richer possibility of cities that the participants can travel to next, every time they have traveled successfully to one city. Though this scenario becomes interesting for the participants, it is not a pretty picture anymore for the game show's creators. They now have a whole array of possible city processes that can be combined into the globe process. The city processes that are combined are dynamic and not predetermined (see Figure 1). Since only the choices are predetermined and not the final city process, the more parameters that go into making the clue, the higher the complexity of choices in their hand to create a globe process.

The Amazing Enterprise BPM Race

Now what does *The Amazing Race* have to do with Business Process Management (BPM) in the enterprise? Enterprises have a much similar approach to business processes in their domain. Each enterprise has a rich repository of business processes that are applicable within a particular subdomain in the enterprise. These subdomain business processes are modeled to achieve a particular task or workflow within the context of that domain. In most of the cases in practice, enterprises start to realize that the business processes applicable in the subdomain are reusable in a higher context. The enterprises then end up creating enterprise-wide business processes (high level) by reusing the subdomain business processes (low level).

The maturity of the approach to creating a high-level business process relates back to the analogy of *The Amazing Race*. Enterprises realize the efficiency of combining their city processes into making one of their globe processes (see Figure 2). Then they start to combine their city processes in more innovative ways by using parameter-based rules. At this point they also face a dilemma similar to the one faced by the game show's creators, which is the dilemma of managing the complexity created by the rich possibilities of using parameter-based rules to control their globe process.

A further dilemma with the game show analogy is that the game show's creators have a whole maze of if-then choices to visualize one single globe process that would make up this season's episode. Using this maze they would be able to maintain the globe process for the current season.

Imagine a situation where because of certain unforeseeable circumstances such as natural calamities, an outbreak of war, terrorism, or something that is just outside of their control, the game's participants are unable to use a particular city process. In this case they would have to go back and alter the globe process to ensure that the problematic city process is removed from the globe process or has an alternative. As the globe process is now controlled by various parameters and rules based on these parameters, the entire globe process model has to be scanned manually to ensure that there is no possibility of reaching the problematic city process. In addition, the show is already on for the season, and the globe process is in execution. This example requires that

even though the show is on, none of the participating teams end up in the midst of a war-torn city!

The dilemma is even more critical in the enterprise. The enterprise globe process (high level) will be influenced by changes in the enterprise city process (low level). The reasons for change can be similarly unforeseen as in the game show. The business might require that the enterprise is able to change dynamically in a high-level process to meet new and unforeseen business demands (see Figure 3). The enterprise globe process would also be in execution, and a change would be mandatory even when in execution so that the enterprise does not end up in the midst of its equivalent of a war-torn city. Unlike the game show, parts of the enterprise cannot be just cancelled. The show must go on!

High-Level Business Processes

As mentioned previously, the globe processes are high-level processes, and as such are managed entirely by nontechnical users. The game show's creators are more skilled in managing a show than they are experts in broadcast technology. This scenario is true also for the enterprise where the high-level business process is managed by executive managers who are familiar with the task of running their business and not experts in creating computer models of their business. Thus, the first and foremost aspect of managing a high-level business process is the need to abstract the user from technicalities.

The second need is to ensure that a globe process in execution can be altered within given boundaries. This flexibility is an important requirement for high-level business processes that are subject to more impact from business change than low-level processes.

The third need is to provide a straightforward, easy, and simple interface for the user to manage the globe process. This management interface should provide for creating and maintaining a high-level business process. The same interface should allow the user to conveniently alter the globe processes in execution, and it should allow easy search of parameter-based rules that control the high-level business process. This search would allow the user to quickly locate the rules that they want to alter. The interface should be lightweight in nature to allow users to quickly update the high-level business process without effort.

The final requirement is to understand that the globe process is determined by combining a rich repository of city processes based on parameter-based rules. The number of parameters is in practice not limited, but it can exceed 30 parameters in many real-time enterprise scenarios. We first encountered this practical problem in one of our customer assignments. The customer, a large European bank based in The Netherlands, faced this issue in their retail banking arm. They had an average of 5 defined parameters to handle each step in a loan-application process. Each parameter had anywhere from 3 to 100 possible values that led to a complex business process, deemed unmanageable through conventional means.

Figure 1 Going global

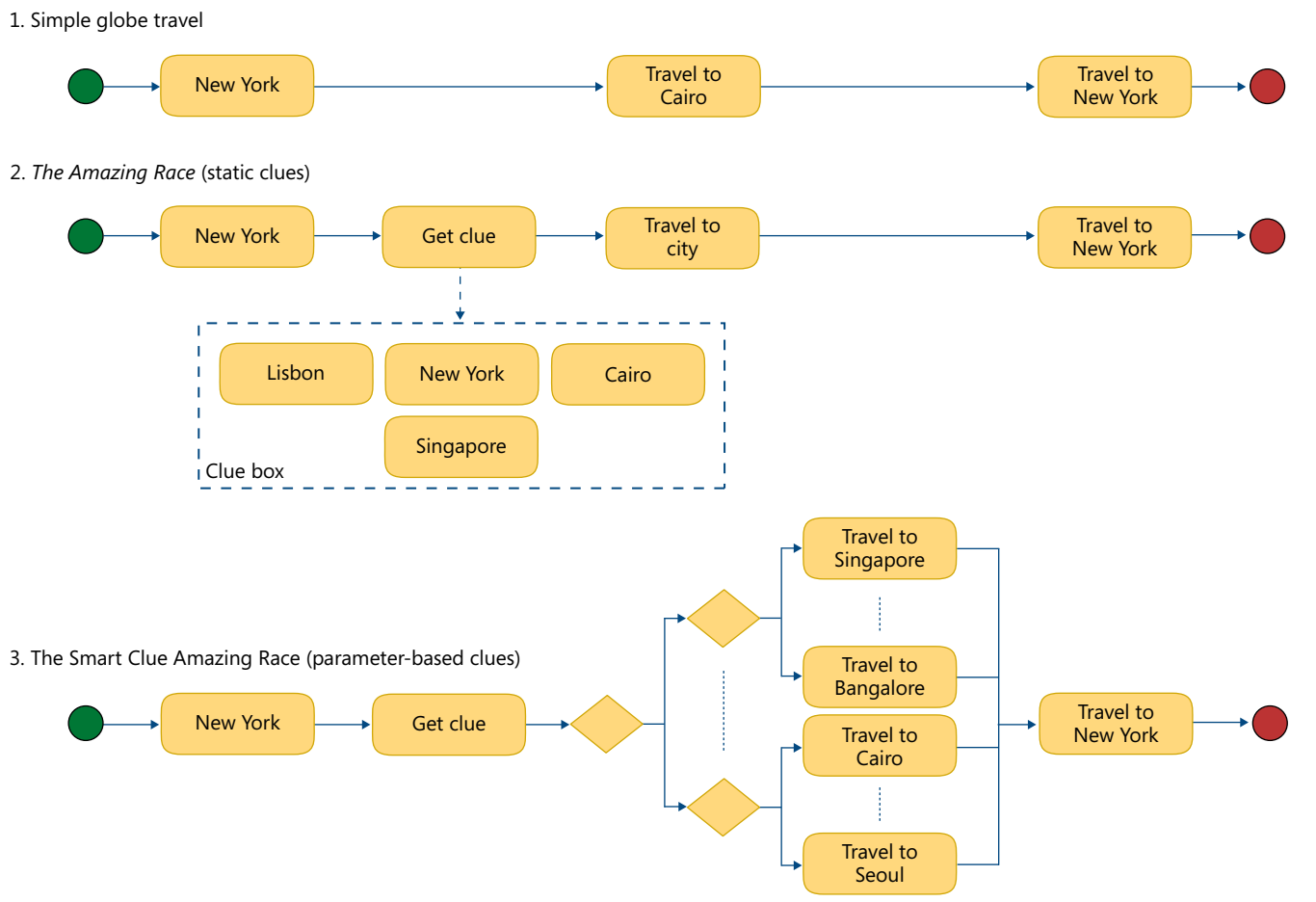
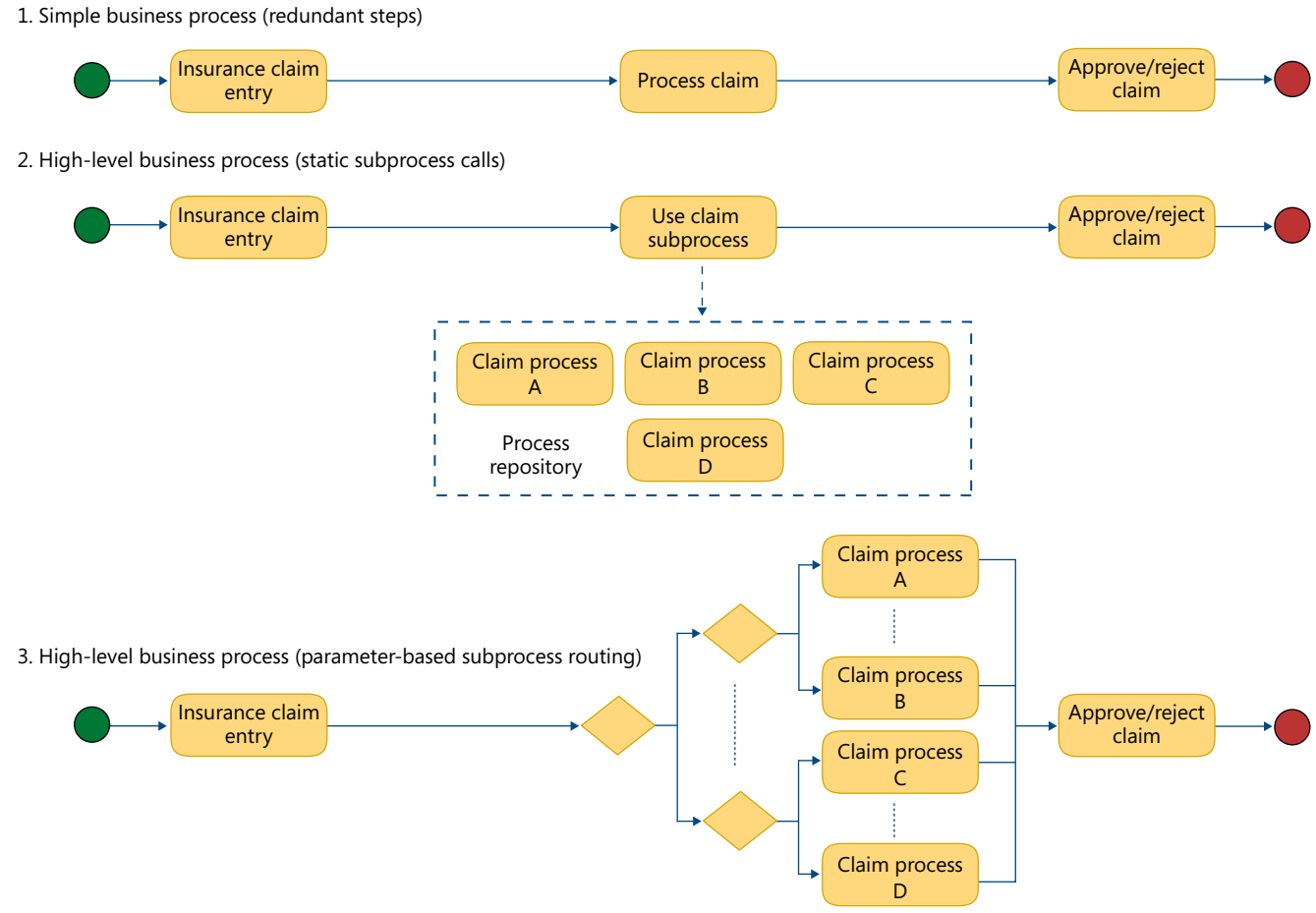


Figure 2 The enterprise



To show this complexity in action, assume that *The Amazing Race* always has five competing teams for each new race (globe process). Each team in any stage of the game would have traveled through a list of cities before they travel to the next city. For this example there are a total of ten cities through which the teams can race, which is ten different possibilities of the city in which the team is currently in. They would also have reached the current city using one of three primary modes of transport such as road, rail, or air.

Using this analogy, the job of the game show's creators would be to make up the globe process using these three parameters of team number, previous city list, and transport mode. For a unique combination of these three parameters a particular city process is to be combined in the globe process. The globe process should ensure that the participants do not use the same mode of transport more than once continuously, and also ensure that the participants end up finally in the city where they started to complete the race.

Applying the Rules

The game show's creators should also use the team number to determine certain special conditions randomly to ensure that the game show has richer content by making sure that the teams travel through different cities in different orders and in different modes of transport. Thus, the next city process to be used in the globe process at any stage is to be determined by the team involved, the current city, and the mode of transport last used.

This combination translates to roughly 150 different flow paths that can determine the next city process to be used in the globe process. The number of flow paths possible is determined this way: 5 teams×10 possible current cities×3 possible modes of transport just used=150 unique choices to determine the next city process (see Figure 4). For example, team number three could have arrived in New York by rail and then given the next process as "to London by air."

> "THE FIRST AND FOREMOST ASPECT OF MANAGING A HIGH-LEVEL BUSINESS PROCESS IS THE NEED TO ABSTRACT THE USER FROM TECHNICALITIES"

This complexity translates into enterprises as well. Both the finance and insurance domains provide several scenarios that necessitate high-level business processes controlled by parameter-based rules. For example, an insurance claim to be investigated by a global insurance firm can necessitate an insurance claim, high-level business process that chooses a country-specific, claim-investigation process based on other additional

parameters. Imagine that the insurance claim is categorized into 5 possible age groups from 10 different countries of operation and belongs to 3 groups defined by claim-amount ranges: 5 possible age groups×10 possible countries of operation×3 possible claim amount ranges=150 unique choices to determine which claim investigation process is to be used.

For example, if age group is B (30–45), country is India, and claim-amount range is greater than $150,000, then use the "High-Value C Group India claim investigation process."

If we go back to the previous financial example involving The Netherlands-based bank, the math works out this way: 5 possible customer interaction channels×3 possible user types×4 possible customer segments×100 possible product offerings×7 possible high-level process steps=42,000 possible implementations in the process repository.

One additional item to consider: In *The Amazing Race* metaphor the number of city processes available reduces at each iteration of the travel-to-city process step. However, in real-time enterprises it is not just one process step that has a wide array of possible process implementations, but each and every step of the high-level business process can have multiple possible implementations. The metaphor has been deliberately simplified, and the increase in complexity for every new step that is dependent on parameter-based rules is shown in Figure 5.

How can we manage these scenarios? Do we still want to use graphical models to manage the high-level business process?

The first and immediate notion that strikes the user when we take a globe process and city processes example is to create the globe process with the city processes as subprocesses that are used inside of the globe process. This notion is not surprising as the use of subprocesses is the most common way to combine simple processes into more complex processes. Using subprocesses is a good choice when we are trying to break down a business process into submodules for better reuse. However, they are not applicable to high-level business processes that are controlled by parameter-based rules because when subprocesses are used the decisions that control the flow of the high-level business processes are embedded and internal to the process.

This characteristic violates the need for flexibility as stated in the requirements of high-level business processes. It also does not solve the problem of managing decisions as the user is left with an inefficient choice of modeling 150 different flow paths into the high-level business process and loading the resulting huge graphical model every time there needs to be a change in the globe process. In addition, this approach is not in line with the service-oriented tenets of loose-coupling as processes would eventually be exposed as services.

Any optimization effort to break down the decision points further into subprocesses would also be futile because the trade-off is visibility of the high-level business process, and the user now has the additional problem of having to drill down to make alterations. Even if the user

Figure 3 The complexity

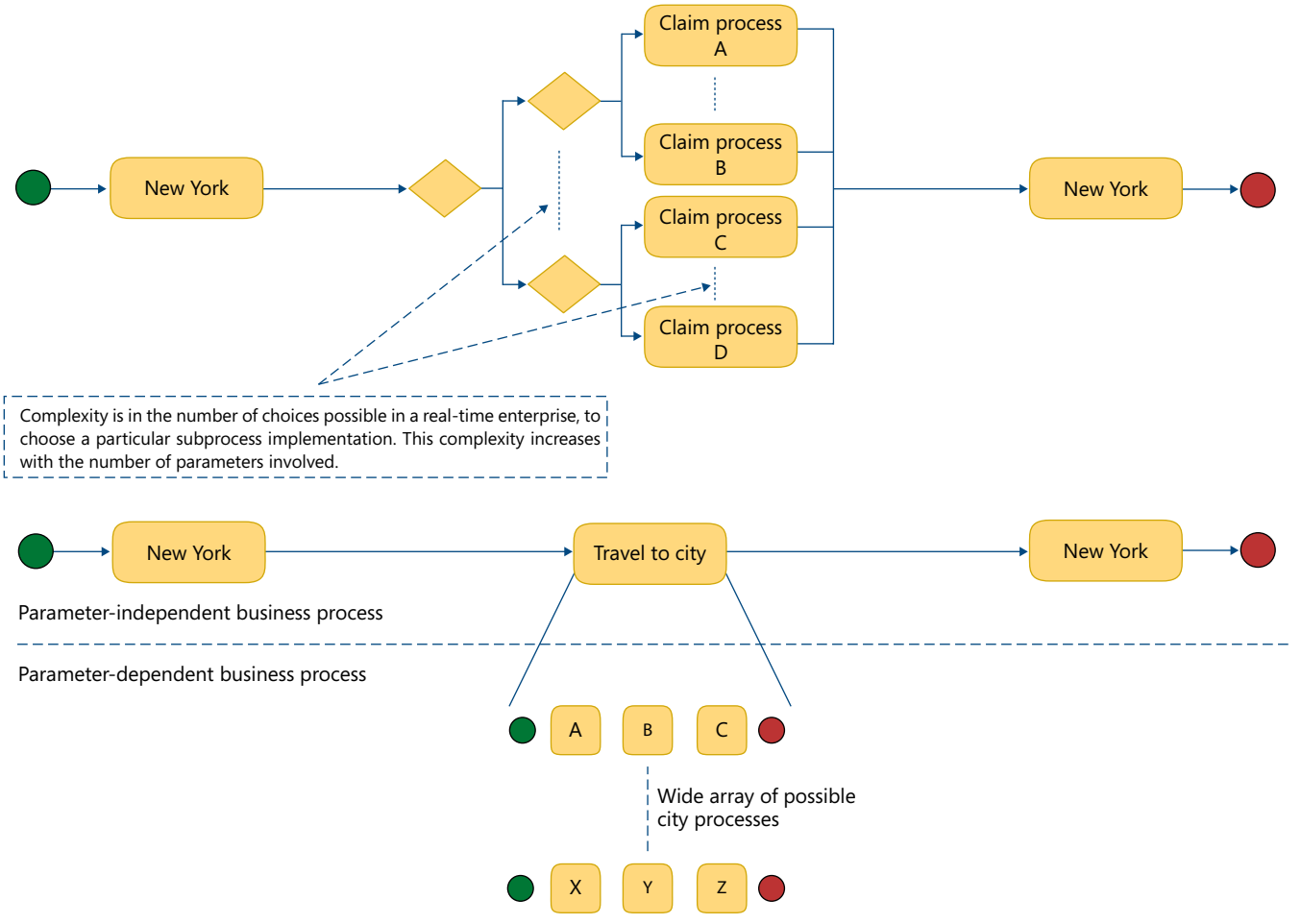
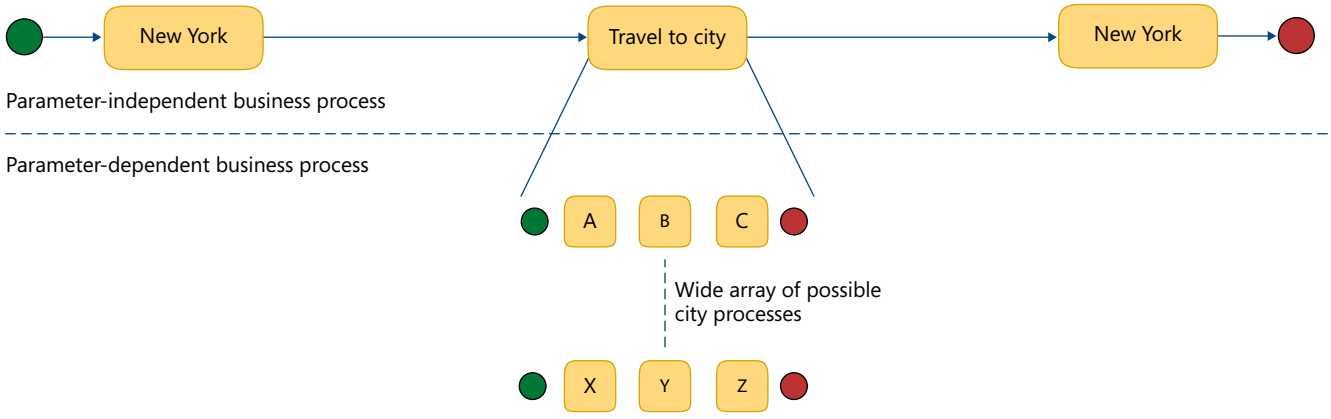


Figure 4 Determining the number of flow paths



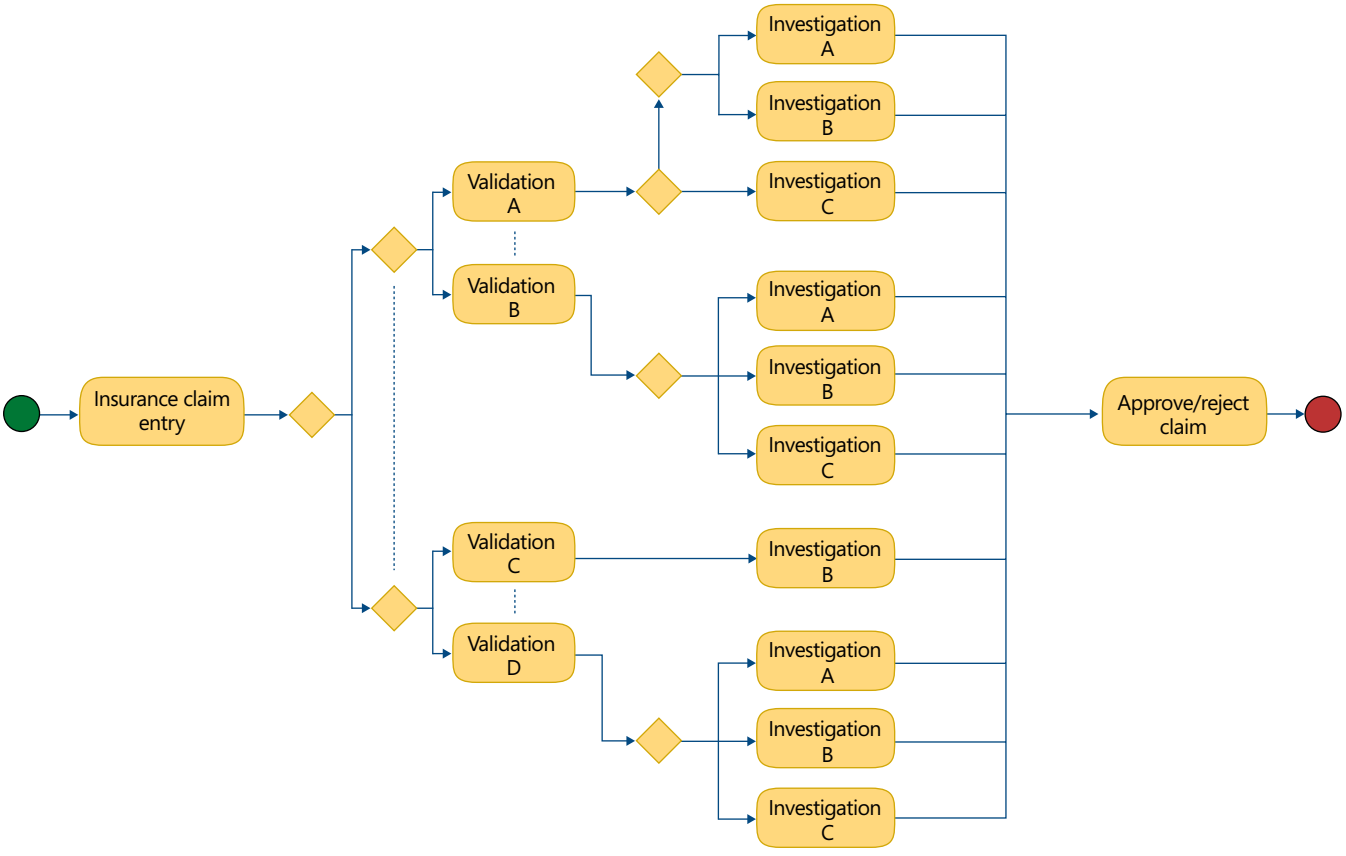
Example: three parameters for a globe race can be team, city, and transport.

Team		City		Transport		Possible unique implementations
5	x	10	x	3	=	150!

Adding one more country of operation and two more claim amount ranges:

5	x	11	x	5	=	275!

Figure 5 One more step



goes through with the alterations, this approach does not guarantee that alterations can be made on a globe process in execution. Hence, using subprocesses in all aspects does not solve the problem of high-level BPM.

Externalized Condition

One of the primary disadvantages of the subprocess solution was that the decisions were embedded and static inside the high-level business process. Some smart BPM tools provide a solution that addresses this specific issue. They provide capabilities to model business processes that take up the decision rule from a lookup repository or refer to an incoming message during the execution for the decision rule.

The approach here is to have a database store of parameter-based business rules that control the flow of the high-level business process (see Figure 6). In the graphical model of the high-level business process a simple call is made to a service over the rule store to evaluate the data in the business process and determine the next logical step. This approach greatly reduces the complexity of the high-level business process in that now the graphical model does not have the flow paths but just the linear stepping of the high-level business process from one stage to another. The solution, however, is still not good enough to solve the primary problem of manageability.

The disadvantages of this solution are that the rule store abstracts the entire logic of the high-level business process, making the model less transparent to the business users. The rule store and the interaction possibilities over the data are nonstandard and do not guarantee that all technical details of conditions would be abstracted from the user. It fails to leverage existing standard tools and methods that are available. Thus, the question of a better solution remains.

Let's take a look at another solution. Process definition rules (PDR) are managed externally from the graphical model, similar to the externalized-condition approach up until the point of keeping the rules outside the high-level business process. The difference is in how these external rules are managed and how the high-level business process is controlled. The PDR matrix solution promotes the rule engine as the control layer and shifts the process engine to an execution layer, making the high-level business process flow more visible in the rules rather than in a graphical model.

> "HIGH-LEVEL BUSINESS PROCESSES ARE CREATED BY COMBINING SEVERAL LOW-LEVEL BUSINESS PROCESSES, THEREBY FORMING TWO TIERS OF BPM"

With this solution there is still a lack of a standard approach to model and maintain the PDR repository. The critical point here is that the technical abstractions provided by a graphical, business-process,

Figure 6 An externalized condition

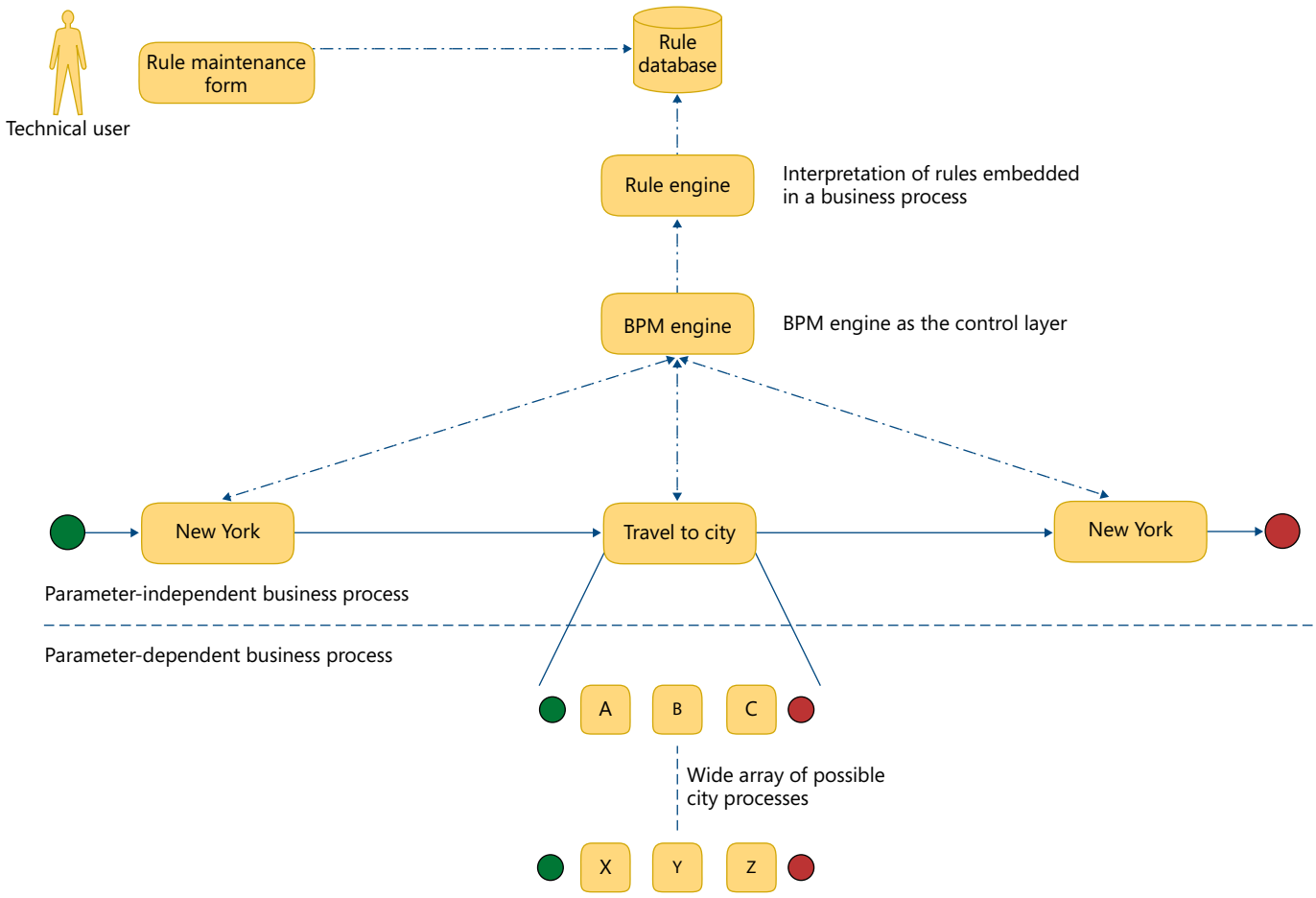
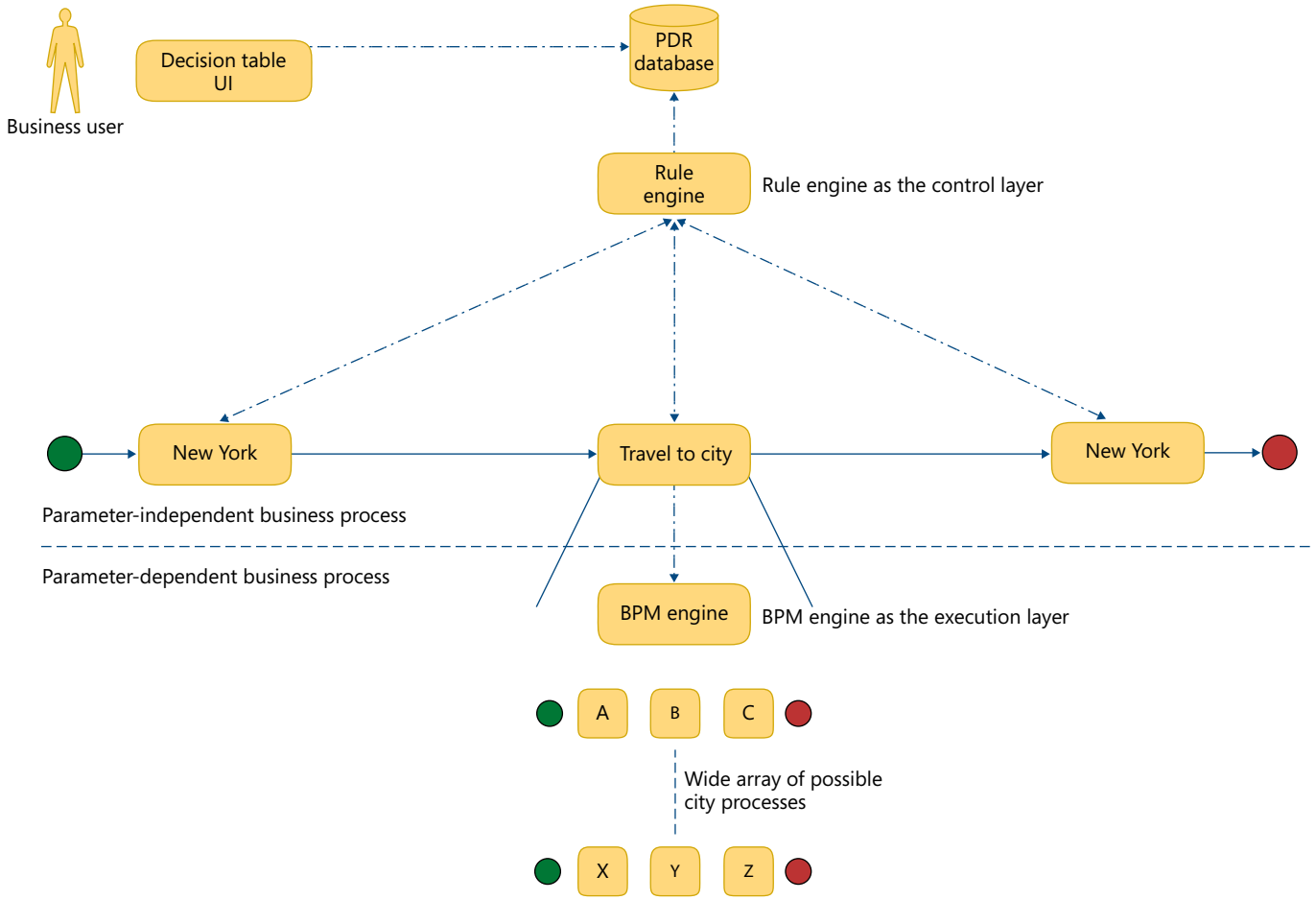


Figure 7 A PDR matrix solution



modeling tool should be intact in the tool to model and maintain the PDR repository. Since the PDR repository is a rule repository, we need to look at the available standard tools in the business rule world. Prominent among them that stand out as a good fit for providing a simple, straightforward, and lightweight tool is the decision table.

The *decision table* is a standard tool to model and manage rule repositories and is an excellent abstraction to business users that is simple to use because it is text based. Text-based decision tables are also good candidates for searching the rule repository and quickly allowing the user to alter relevant process definition rules.

Using this tool we can create a solution that has a graphical, high-level business process that is simplified by externalizing the rules and providing a service layer on top of the PDR rule repository (see Figures 7 and 8). The PDR rule repository is then modeled and managed through decision tables. We call this solution a PDR matrix, and the concept of globe process and city process can be shown as two tiers. The first tier is the high-level, enterprise-wide business process, and the second tier is a reused, low-level business process.

The End Game

We have worked through a very practical problem here with BPM in enterprises: the complexity in defining and managing high-level business processes. High-level business processes are created by combining several low-level business processes, thereby forming two tiers of BPM. The high-level business processes are marked by the need for parameter-based rules to dynamically choose a low-level business process.

These parameter-based rules, when employed to manage a high-level business process, increase the complexity of the graphical model manifold with each new value for each parameter. This increase in complexity makes the graphical model of a high-level business process unmanageable and impractical. The solution to this problem is to use a rule-based approach to define and manage high-level business processes.

You can define and manage such an approach using a decision table-based PDR matrix. Decision tables are lightweight, nongraphical, business user-friendly tools to create and manage rules. The PDR matrix abstracts the complexity of managing a graphical, high-level business process model. Therefore, employ the PDR matrix to manage two-tier BPM and put the business user back in the driving seat in your enterprise. •

Resource

CBS.com
www.cbs.com/primetime/amazing_race5

About the Author

Vignesh Swaminathan is a product manager at Cordys R&D India (www.cordys.com), which provides a state-of-the-art application platform suite going beyond basic EAI and BPM to solve many practical aspects of the enterprise. He has been working with business-process orchestration, business rules, data transformation, and other related technologies for the past five years, and he specializes in data, process, and human integration. Contact Vignesh at vswamina@cordys.com and vigneshs@hotmail.com.

Explore Human Workflow Architectures

by Jesus Rodriguez and Javier Mariscal

Summary

Human workflow systems and some of the most representative patterns of human-to-business processes interactions break down into two major components. The first is human workflow systems and the interactions among them as they are implemented in integration platforms. The second component is human workflow interaction design patterns and how they are implemented using interactions among the human workflow systems. This discussion will take a close look at these processes.

Total automation of business processes is practically unachievable without considering human interaction, a factor that is tied semantically to many aspects of process automation and integration. Human interaction is present in some of the most common business processes like order approvals and human resources management. Those human interactions can vary from a simple task-assignment process to a very complex business process notification and task reassignments.

Frameworks for human workflow are present in a variety of popular integration servers from different vendors. We'll explore conceptually some of the main components that are present in those human workflow platforms and the interactions among them. We'll also cover some of the most common human workflow design patterns and how they can be implemented using those components.

Human workflow systems have to support the communication between people and systems. To accomplish this support every human workflow system has to provide basic functions like task assignment, identity management, notifications, tracking, and interoperation with business process management (BPM) systems (see Figure 1).

Four main components of a human workflow architecture are a task management service, a tracking service, a notification service, and an identity service. Before we explore these services in depth, it's important to understand the role of tasks in human workflow systems.

Tasks are the main communication unit between business processes and people. Typically, a task is assigned to a user who needs to perform some related action. For example, a supervisor might need to approve the request to purchase some items. Normally, a user has to perform a series of tasks that are grouped semantically. That is, our supervisor might group all items to perform each day into a group called "today's approvals." These groups are called task lists.

Task States

During the lifetime of a human workflow system, tasks in task lists are constantly switching from one state to another. For example, the task for our supervisor first enters a state of *pending*; when the supervisor accepts the task, the state changes to *claimed*. Finally, when the supervisor approves the request, the task enters its final state of *completed*. States are used conceptually to describe the task life cycle. Some of the most common states are *pending*, where the task has been created; *claimed*, where a user has claimed the task and has received its input data; *completed*, where a user has finished the task and provided its output data; and *failed*, where a user has finished the task and provided a fault message.

Tasks are normally associated with time frames: expiration, escalation, delegation, and renewal. In our example, the approval task can *expire* if the supervisor does not act on it in a specified period of time. This expired task can be subsequently *escalated* to another course of action or assignment. Also, the supervisor can decide to *delegate* the task to another person (a manager, for example) to act in his or her place. The manager can also decide to *necessitate* another manager to gather additional insight. If this second manager does not act on the task in a given time frame, the task will be *renewed* for another period of time.

In other scenarios, tasks are related semantically to each other. The question, "what's the next task?" will not always have a trivial answer. In some cases, this answer has to be determined at runtime. Tasks can be grouped sequentially within a business process instance so the user knows the next task(s) to perform after completing the current task.

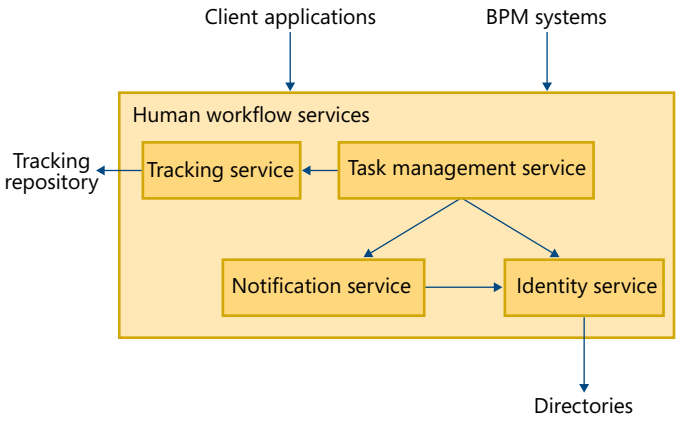
In our example, 20 requests should be approved for the supervisor in sequence to complete the business process. Each time the supervisor completes an approval, the engine should be able to identify the next task. *Task chains* represent a metadata-driven approach to describe the relationship between a set of tasks within the scope of a business process. Task chains can group tasks semantically to help users achieve functionalities like sequence execution and failure management.

Tasks that have a dependency on a specific business process are known as *inline* tasks. Tasks that are totally independent of a particular business process are *stand-alone* tasks. Inline tasks typically have access to data related with a business process and are stored in native, business-process artifacts like variables or messages. By contrast, stand-alone tasks interact with business processes through a well-defined interface without any dependency on the process data itself.

Workflow Services

The architectural component that handles sectional tasks is the task management service. In a typical scenario, the *task management service*

Figure 1 The main components of a human workflow architecture



receives a request to create a task, interacts with the *identity service* to select all people that are eligible for the task, adds the task to the work lists (to do's) associated with the selected users, and assigns the specific timelines and policies. Eventually, one user decides to work with the task by claiming it. The user can then work with the task or request more data.

A key aspect of human workflow systems is the ability to resolve the set of users allowed to execute a task. This user-resolution process can be based on interactions with identity management platforms. In our example, the human workflow services need to identify which users are able to approve the request—in this case the supervisors. To accomplish this identification a human workflow platform has to resolve the concept *supervisor* against a set of users and roles traditionally stored in a user directory.

Multiple types of relationships can be established between people and processes. One of the most common is how people interact with processes (human roles). People in an organization can be grouped in roles that are related semantically to some business activity, such as a process administrator or task owner. Another common relationship is how processes identify which people to interact with (people links and queries). Within a business process, certain groups of users are relevant from the business standpoint. *People links* are used to represent the different groups of people who participate in the execution of the process. A *query* against an organizational directory is used to determine the individuals associated with a people link and is bound to the people link. In our example, the generic human role, finance manager, could be qualified by the people link supervisor, which is bound to the query "select head of department, where department name is finance."

The identity service is in charge of the user-related features like authentication, authorization, or people resolution. User information is frequently stored in organizational directories (for example, Active Directory directory services, an LDAP directory, or a relational database). The identity service can work independently of the organizational directory. Based on the adapter pattern, it is possible to extract the directory access through the provider, which is able to execute the queries to obtain the information from the directory. This approach extracts the human workflow functionalities from the user's store.

In our example, let's assume a manager would like to reconstruct the execution of the request-approval workflow to check for deficiencies. The tracking service keeps track of the state changes related with tasks and task chains. This service should provide the basics of the required functionality to reconstruct the task history changes and perform task analysis.

We also need a service that will notify the supervisor by e-mail when the request-approval task is created. The notification service handles the notification mechanisms to the user related with the task's state changes.

The four run-time services discussed here provide a good overview of some of the most common functionalities required for human workflow systems. Combining these services addresses some of the most common human workflow scenarios. Now let's explore some of the common human workflow patterns.

Task Assignment Patterns

Workflow-oriented processes have been present in the industry for years. The knowledge acquired serves as the base for improvements to build workflow systems. Patterns abstract workflow systems at different levels such as task approval, task creation, and task state management. We don't intend to define a catalog of human workflow patterns here. Instead, we'll cover some common patterns in human workflow systems from the angle of the architecture defined previously.

We'll begin by looking at a *single-user workflow* example. A task can be assigned to one user, and only one user can act on it. For example, an employee, through the employee portal, submits a vacation request. The portal initiates a business process that includes a user task modeled using a simple workflow. The task is assigned to the employee's manager. When the manager approves or rejects the vacation request, the employee is notified by e-mail of the manager's decision.

For this solution combine the human workflow components through four interactions. The task is configured using the client applications that interact with the task management service; the task's life cycle or task states are configured. As part of the business process, the task is assigned to a group of users using the task management service. One of the users claims the task, and the task management service applies the correct policy to prevent other users from acting on the same task. The business process uses the task management service to get the status update of the task.

Now let's consider a *sequential workflow* example. The sequential workflow represents a scenario in which a task must be approved sequentially by a set of users. For example, when a purchase order approval system processes a purchase order using a business process, an employee belonging to the group "Supervisor" initially evaluates the purchase order. After the initial user approves the purchase order, that user's manager approves it. After the manager approves the purchase order, it is forwarded to the billing and shipping departments. This solution has for interactions. Interact with the task management service to configure tasks and set appropriate policies. Define the sequence of users that should act on the task. Start the task interacting with the task management service. The first user will claim the task to start working on it, and upon completion the task management service will route the task to the next user in the group.

A *parallel workflow* pattern represents the scenario in which a task must be approved by different users at the same time. Each approver can add comments and attachments that are independent of the others. For example, a hiring process is used to hire new employees. Each interviewer votes in favor of or against a candidate. If 75 percent of the votes are favorable, the candidate is hired; otherwise, the candidate is rejected. The process is modeled using the parallel workflow, where each interviewer can vote independently from the other interviewers. For implementation this solution has five interactions. Interact with the task management service to configure tasks and set appropriate policies. Define the sequence of users that should act on the task. Start the task interacting with task management service. The task management

service routes the task to all users. The task management service will complete the task only when all users are finished acting on it.

Task Assignment Using Policies

Now let's take a look at using policies for task assignment in which the tasks need to be assigned to the user by following specific rules. For example, the heart-bypass procedure is allocated to the surgeon who has the least number of operations allocated to him or her. This solution's implementation combines the human workflow components through five interactions. Interact with the task management service to configure tasks and set appropriate policies. Configure the assignment policy as part of the task definition—for example, select from the available users the one that has the least number of tasks assigned. As part of the business process, the task is assigned to a group of users using the task management service. Start the task interacting with the task management service. The task management service will execute the task policies against the potential users that can act on the task and, in turn, select a user matching the policy criteria and assign the task to that user.

In a *single-user workflow with escalation* example, a task can be assigned to multiple users, but only one user can act on it. If the task expires, the user's manager has to act on the task. For example, the help desk service request process allows users to file help desk service request tickets. If the person who receives the ticket does not act on it within a specified time period, the ticket is escalated automatically to that person's manager. The ticket is escalated automatically three times if no one has acted on it within a predefined time period, until it gets to the CEO of the company. If the CEO also doesn't act on it, it expires.

For this solution's implementation combine the human workflow components through five interactions. The task is configured using client applications that interact with the task management service; properties such as the task's life cycle, escalation policy, or task states are configured in this step. As part of the business process, the task is assigned to a group of users using the task management service. One user claims the task, and the task management service applies the correct policy to prevent other users from acting on the same task. If the task expires, the task management service applies the escalation policies to escalate the task to the correct user. If the task expires again, the task management service cancels it.

In a *single-user workflow with delegation* example, a user who claimed to act on a task can reassign it to another user. For example, before going on leave, the chief accountant passed all of his or her outstanding tasks on to the assistant accountant. For this solution's implementation combine the human workflow components through four interactions. The task is configured using client applications that interact with the task management service; properties such as the task's life cycle, escalation policy, or task states are configured. As part of the business process, the task is assigned to a group of users using the task management service. One user claims the task and the task management service applies the correct policy to prevent other users from acting on the same task. Using the appropriate client applications the user inspects the list of possible people that are able to act on the task. The user reassigns the task to one of those people and readjusts the task's properties.

For the *task chained pattern* example a task needs to start based on the completion status of another task. For example, immediately commence the next work item(s) in the emergency rescue coordination process when the preceding one has completed. To implement this solution combine the human workflow components through four interactions. The task is configured using client applications that interact with the task management service; properties such as the task's life cycle or states are configured. The task chain metadata is configured, indicating the second task will be triggered by the completion of the first task. The task chain is started through the task management service, causing the start of the first task. When the first task completes, the task management service will use the metadata in the task chain to start the second task.

Business Process Integration

Human workflow solutions can be designed using a combination of services like task management, identity management, tracking, notification, and client applications. We've explored some of the most common components present in human workflow architectures and their implementation in a series of interactions. Some of the most complex human-systems interactions can be modeled using a set of human interaction patterns that are in turn implemented using the human workflow components. These basic components and patterns of human workflow architecture represent key aspects in the business process integration space, and when implemented they create a powerful human workflow solution.

The concepts and scenarios presented here are intended to assist you in better understanding the components found in typical human workflow platforms. For more information on human workflow concepts consult the listed resources. In addition consult the workflow engine architectures from some of the major integration server vendors.

We would like to thank Kirsti Elliot, Ben Elliott, and the architecture strategy team at Microsoft for all their feedback and corrections made to this article. •

Resources

The Workflow Management Coalition
www.wfmc.org

OMG Business Process Initiative
www.bpmi.org

About the Author

Jesus Rodriguez is chief software architect at Two Connect Inc. (www.twoconnect.com), a Microsoft Gold Partner based in Miami, Florida. He is also a Microsoft BizTalk Server MVP. Jesus's extensive experience with business process integration and human workflow has been derived through multiple implementations of loosely coupled systems founded on the principles of SOA. He is an active contributor to the .NET and J2EE communities, focusing on the interoperability aspects between those two platforms. His contributions include several articles for different publications such as MSDN, sessions in Microsoft conferences such as Teched, and Web casts about different Microsoft technologies. He is a prolific blogger on all subjects related to integration and has a true passion for the technology. Contact Jesus at jrodriguez@twoconnect.com or through his blog at http://weblogs.asp.net/gsusx.

Javier Mariscal is president of Two Connect Inc. Javier has dedicated a good part of his 16-year professional career to designing and deploying data and application integration solutions, particularly those involving the merger of mainframe and AS/400 environments with the WinTel platform. He has a true passion for business process automation and workflow solutions that implement the service-oriented development of applications (SODA) approach to globally distributed application development and maintenance. He keeps quite busy working on just such solutions for Fortune 1000 and upper midmarket organizations worldwide. Contact Javier at Javier@twoconnect.com.

Workflow in Application Integration

by Kevin Francis

Summary

One of the greatest challenges facing the architect today is the integration of applications. Let's look at a framework for application integration that moves beyond the common, one-off integration approaches and toward a cohesive structure. The requirements for successful integration are outlined along with the presentation of an architectural approach for meeting these requirements by using tools such as workflow technologies.

Integrating applications has continued to become more common, and the growing availability of tools and standards (such as the WS-I Web services standards) and service-oriented architecture (SOA) appear to hold a promise of easier integration. Many articles exist that promote the simplicity of linking applications using Web services, or even SOA. Two approaches for integrating applications are commonly used today: point-to-point and services bus integration (see Figure 1).

In the point-to-point scenario direct links are created among applications through a direct application program interface (API) link, file transfer protocol (FTP), or batch interfaces. Transformation (translation) of data may take place as data is transferred across the link. Generally, point-to-point interfaces are implemented without the use of an integration product, with translation of data taking place using code at the point of integration at one or both ends of the interface.

Service bus integration makes use of a technology solution to provide a bus, upon which applications are able to place messages to have the bus itself manage the routing of messages among applications. The bus will also generally manage the transformation of message formats among applications.

As organizations strive to bring a greater range of services online, aim to integrate product lines together, and aim to streamline call center experiences, an integration solution requires more back-end

Figure 1 Point-to-point and service bus integration

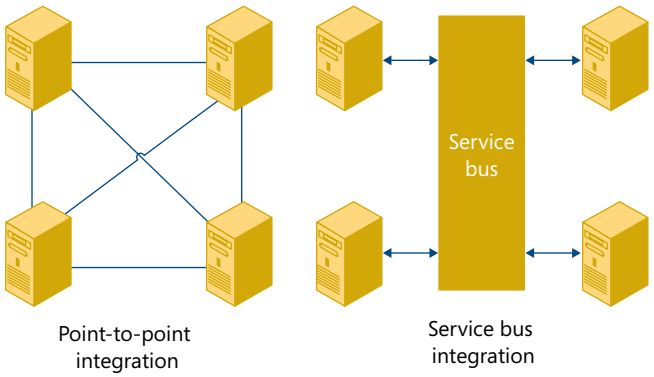
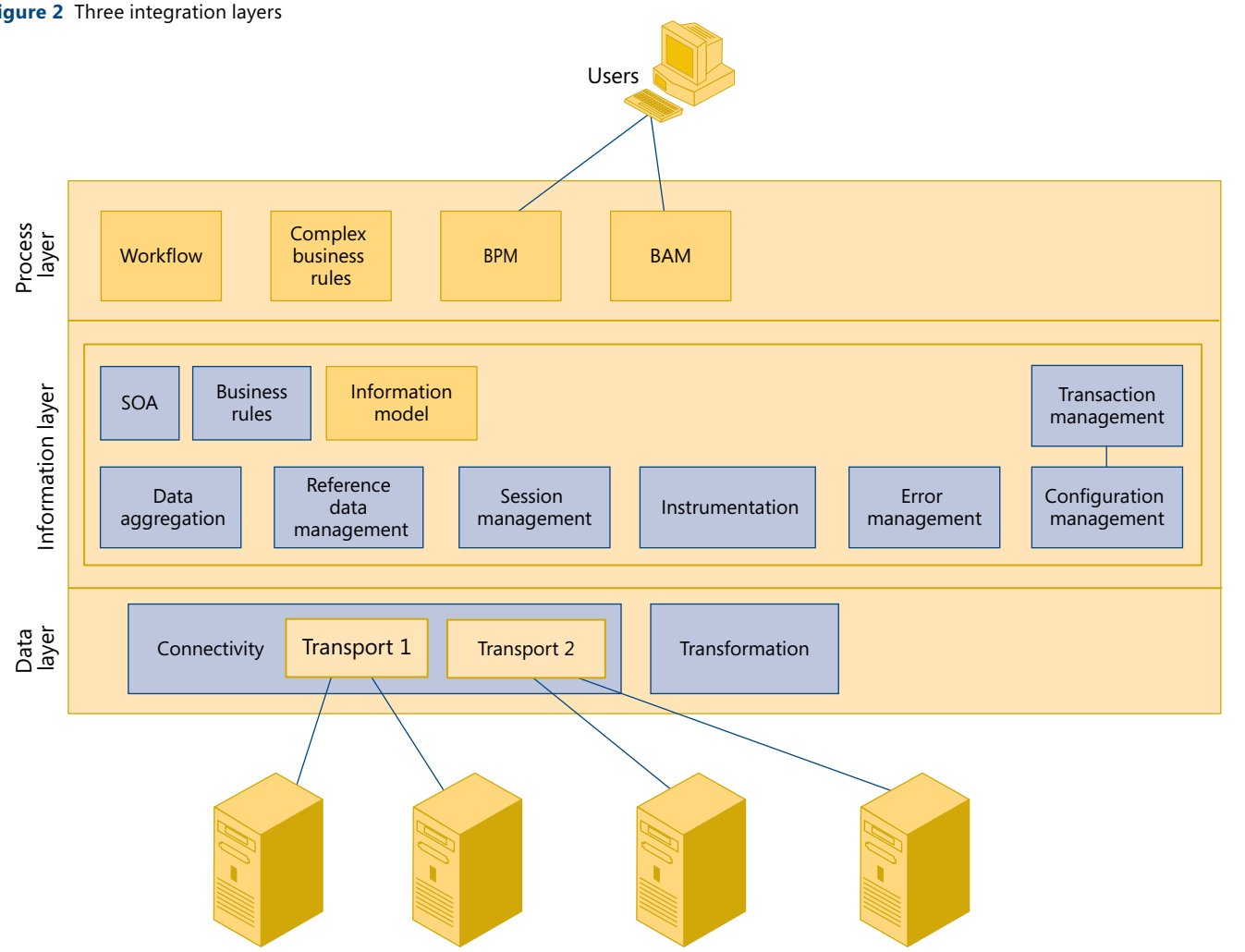


Table 1 The requirements of an integration layer

Layer	Requirement	Description
Data	Connectivity	Basic connectivity in which applications are able to communicate with each other
	Transformation	Translation of data format, and so on among applications
Information	Data aggregation	Aggregated view of data across multiple systems
	Business rules	Development of business rules across multiple systems
	Transaction management	Ability to perform ACID transactions across multiple systems
	Information model	A cohesive data model across all systems where a common understanding of data entities and structure is achieved
	Reference data management	Management of commonly used reference data from multiple systems in a single location
	Session management	Management of session information across interactions and across systems
	Instrumentation	Common point for the logging of operational information
	Error management	Consistent approach to error management from a single rule set
	Configuration management	Ability to configure the run-time operation of the entire system, configure its communication with the various systems that make up the environment, and deploy new versions of the various components
	Workflow	Workflow processes that cross multiple systems
Process	Complex business rules	Shared, reusable business rules that cross multiple systems
	Business process modeling	Modeling of business processes across systems for optimization and integration
	Business activity monitoring	Monitoring the speed and efficiency of end-to-end business processes for optimization and issue tracking

Figure 2 Three integration layers



systems. It is not uncommon, for example, for call center operators to switch among upwards of ten applications in handling customer calls. The replacement of this type of scenario, where the user is effectively integrating the applications by copying and pasting or retyping information into various applications to complete a transaction, is a common driver for the integration of applications.

Enterprise Architecture Backbone

There are a number of methods that can be used to integrate applications behind a Web site or call center application, and the most common solution is to construct a number of point-to-point or service bus interfaces in the first instance.

What happens, however, when a call center solution and a Web application need to access the same information? The ideal scenario is to reuse the interfaces, right? Well, in most cases where development teams (and architects) are disconnected, this is not the common approach, given the complexities of taking an existing interface with its own complex set of code and allowing it to be called by something else. This issue grows exponentially with the number of interfaces that need to exist among systems and is relative to the size of the organization; the experiences of larger organizations are obviously worse than smaller organizations.

Failure to implement a well-architected, end-to-end solution results in duplicated code across the enterprise, inconsistent architectural approaches in each system, and an inability to respond to business needs in a timely manner. These issues are a by-product of a project-centric approach to solution architecture.

> "FAILURE TO IMPLEMENT A WELL-ARCHITECTED, END-TO-END SOLUTION RESULTS IN DUPLICATED CODE ACROSS THE ENTERPRISE"

SOA is commonly heralded today as the solution for the integration issues among applications, but a number of additional capabilities are required for a truly efficient integration solution. These capabilities are listed in Table 1 and grouped into three layers of integration:

- *Data integration* – The most basic layer, data integration generally is achieved in even the most basic integration scenarios. In this layer data is moved among applications with transformation taking place to allow data to be translated among applications.
- *Information integration* – In this second layer, data and calls to applications are aggregated to enable single calls to access multiple appli-

Table 2 Technical requirements for the data layer

Requirement	Description
Connectivity	Connectivity is the ability to transfer information using a variety of protocols or methods, and it includes the provision of a Web services interface. It also includes specific protocols required by specific scenarios, which will change from one organization to another. For example, many organizations that retain mainframes for core business systems would require IBM WebSphere MQ and/or SNA connectivity using Microsoft Host Integration Server (HIS). Other systems may require COM+ interfaces or even HTTP, raw sockets, or FTP. All protocols should be executed through a common interface, with pluggable specific implementations for the specific scenarios. All specific technical requirements of the particular protocols or methods should be handled by the implementation, and external systems should not be required to contain any logic to handle specific cases.
Transformation	Transformation is the rule-based transformation of data from one structure to another. Once again, the transformation engine should be contained within the integration layer.

Table 3 Technical requirements for the information layer

Requirement	Description
Data aggregation	The provision of services that wrap multiple calls brings with it data aggregation. It is important to consider data aggregation as a separate requirement, as it brings a requirement for greater care on the design of the data and the use of careful data design and business rules to allow the data to be aggregated rather than simply accumulated.
Business rules	It is necessary to be careful in the development of business rules in an integration scenario to ensure that only those rules that form part of each application reside within the application, and those rules that are related to the integration of the application are encapsulated within the integration layer. This approach provides the greatest opportunity for reuse, simplifies the maintenance of the applications, and provides a single, centralized point for development and execution of the rules.
Transaction management	Transaction management is both a necessary and complex requirement for application integration. It is necessary because there is a need to commit or roll back transactions across all applications that are being accessed, driven from the calling application, which is critical. It is complex, however, because rollback for many systems may involve providing reversing entries or some other code-based method, which is an ideal example of the need to centralize this complex logic.
Reference data management	Reference data is used commonly by application user interfaces to provide lists of choices or for data validation. It is common for reference data to be common across applications (such as lists of countries, postal codes, products, office locations, and so on). Accessing reference data from a shared location in the integration layer provides better performance, consistent results, and less development effort.
Session management	Session management is used to ensure that all systems in the integration scenario have a current understanding of who is accessing the data, and to ensure that the current actions are synchronized across the systems. A complete use of session management can be used to allow a session to be captured from one access point, such as a customer accessing an Internet portal, and reused in another, such as to allow a consultant to fix and complete the order internally.
Instrumentation	In a similar way to reference data management, the centralization of instrumentation (logging) into an integration layer is logical in that it is the point from which the majority of calls with other systems are made, as well as again allowing the instrumentation code to exist in a single location.
Error management	Error management is similar to reference data management, in that the use of an integration layer for error management allows error processing code to be shared and allows a single set of error responses to be used.
Configuration management	Configuration management of the type of environment outlined here is a complex matter and worthy of its own content. Configuration management should, however, provide a central point where the addresses of systems can be altered, access to systems can be controlled (preferably with functionality gracefully deprecated), and where overall configuration settings can be stored and loaded once by a shared configuration manager.

cations, with the basic business rules in place to allow single calls to bridge applications. The use of these techniques provides service aggregation and meets the minimal requirements to achieve an SOA implementation.

- *Process integration* – The third layer of integration builds on top of data integration by aggregating and integrating the processes and data that are involved with executing a business process that operates across application boundaries.

As you move through the three layers of integration, the focus changes from technology to business. The end result, however, allows for greater ability to quickly add value to the business.

Many views of SOA provide a common connectivity model and assume to provide transformation, but these views, in reality, only provide data integration. The challenges faced by an architect in integrating applications in an efficient and repeatable manner are far more complex, particularly when an integration scenario includes applications that are already in existence.

Why does all of the foregoing matter? When you build the first interface between two applications, simple data integration can be sufficient. As you build more interfaces the design becomes more important, however. Failure to design and manage a complete integration scenario leads to an exponential increase in the cost of each interface in turn. Conversely, as the functionality that is provided in the integration layer grows, the code that is needed in each layer is able to be reduced.

Integration Layer Components

The components of an integration layer can be split into three layers: data, information, and process (see Table 1). Furthermore, they can be separated into components that make-up a technical framework;

Table 4 Business requirements for the information layer

Requirement	Description
Information model	As the number of applications that are linked into the integration layer grows the amount of data that is available grows. This situation provides the ability to begin to build an information model across the systems that are in scope, which can be closer to building an information model for an organization than most are able to achieve. When implemented correctly, the information model that sits within the integration layer can provide a single point of truth for key data entities. An example of the single source of truth is an understanding of a customer that covers all systems with addresses, products, order history, and support history. Two commonplace methods of creating a single point of truth are to utilize one key system and attempt to synchronize across other systems, or to use a database that is updated with changes to multiple systems. The use of a shared information model in the integration layer is particularly powerful as the result is a shared single point of truth that is available through the combining of data from multiple sources without impacting data or code in those sources. Also, given that no synchronization of data is required the data is more likely to be current.

Table 5 Business requirements for the process layer

Requirement	Description
Workflow	Workflow systems can be implemented as user (human) driven or system driven. Both can be utilized in an integration layer; although, it is important to separate the two. System-driven workflow includes the execution of the steps involved in interacting with the various systems in the environment.
Complex business rules	Complex business rules provide an additional value as the integration layer becomes complete and as it becomes a place where business rules begin to reside there in their own right. Given the central role that an integration layer begins to provide, it can become the host for business rules that provide new or extended functionality beyond the integration of applications, which could be new products, combining products together, additional discounts, better credit check, and so on.
Business Process Modeling	Workflow systems can also be used for the execution of human-facing workflows, and this reason is predominant for their existence. The provision of an integration layer provides the ideal host for a business process modeling environment to exist, allowing human workflows to be automated. The use of workflow tools allows processes to be documented and executed in a graphical form. The use of the integration layer behind the workflow tools allows access to multiple applications through the workflow tool more easily and readily, particularly when the capabilities outlined here, such as the information model and transactional support, have been implemented. Thus, business process modeling is allowed to facilitate greater improvements than would otherwise be possible.
Business Activity Monitoring (BAM)	Business Activity Monitoring (BAM) is the tracking of each transaction through the end-to-end system to identify roadblocks, slow performance, issues with particular transactions, and opportunities for overall performance. Based on the instrumentation capabilities provided by the information layer, the capabilities provided by end-to-end BAM are considerable.

the components appear in blue in Figure 2. The *technical framework* provides plumbing and engineering requirements. A *business framework* provides business enablers, which are shown in gold in Figure 2. Let's take a closer look at the components that make up the three layers of an integration layer.

Data layer. The two components of the basic data layer are connectivity and transformation, both of which form the foundation of the technical framework (see Table 2). Single point-to-point interfaces using Web services are frequently used as examples of SOA, but many principles of SOA refer to the provision of services that wrap multiple calls across systems. As mentioned previously, however, Web services exist in the data layer, which does not provide for these principles. In this context, Web services provide for the connectivity, but we need to explore the information layer for service-related integration.

Information layer. The information layer is made up of components that build on the technical framework (see Table 3). The business framework is built on the base of the information model (see Table 4).

Process layer. As the focus moves from technical requirements to business value, the focus of the functionality that should be delivered through the process layer shifts to providing the true value of the integration layer concept—quickly configurable business processes (see Table 5).

In bringing it all together, there are a number of ways that an integration layer can be implemented: custom development; the use of frameworks, toolkits, open source, or operating system-provided components; and the use of a more complete integration package. Custom development could be applied completely to the integration layer but is not recommended given the availability of toolkits and frameworks.

> "SOA IS COMMONLY HERALDED TODAY AS THE SOLUTION FOR THE INTEGRATION ISSUES AMONG APPLICATIONS, BUT A NUMBER OF ADDITIONAL CAPABILITIES ARE REQUIRED FOR A TRULY EFFICIENT INTEGRATION SOLUTION"

Some frameworks that are available provide an opportunity to quickly integrate systems at the user interface layer, such as the Customer Care Framework from Microsoft. This type of integration solution provides an excellent solution to the rapid integration of back-end systems to provide a consolidated user interface, but it is important to understand the relative merits of this type of solution. Frameworks that enable the integration of systems at the user interface layer provide rapid solutions for certain issues encountered by businesses, such as improving call times for call center operators. An

integration layer can provide further benefits, at an additional initial implementation cost.

Extending the Integration Scenario

An integration layer allows key functions to be hosted centrally, shared among applications, allowing more than the integration of the user interfaces. The range of data, information, and process integration capabilities can be used across applications to lower the cost of each development or integration in turn. Experience has shown that the use of an integration layer as described here produces growing cost savings across a range of applications over time, with the range of creating new applications beyond the original integration scenario.

For example, while billing, trouble-ticketing, customer-management, and ERP systems may be integrated behind a CRM user interface in the first instance, the fact that these systems are already linked provides simple solutions for Web self-help, and then extending to online ordering and a B2B channel. Should there be a purchase of a new application, such as a supply-chain application, the integration of

> "FRAMEWORKS THAT ENABLE THE INTEGRATION OF SYSTEMS AT THE USER INTERFACE LAYER PROVIDE RAPID SOLUTIONS FOR CERTAIN ISSUES ENCOUNTERED BY BUSINESSES"

that application into the environment is still not a trivial process. However, the integration is much easier than it would be without an integration layer because data integration can be achieved more easily by hooking up a single set of known interfaces into the integration layer, not into each application. Also, the process of integration can be added by extending the integration workflows in the workflow engine, just as BAM integration, data aggregation, inclusion in the information model, and so on are more easily achieved by simply extending that which already exists.

The workflow component is a key part of any integration solution being used to provide both human-focused workflow processing through a business process management capability and in playing the central role of executing the processes involved in linking systems. These two tasks are quite separate and quite different, but they are both suited perfectly to the use of workflow tools. A system-integration workflow can include such steps as retrieving data from each system and aggregating it together, validating user entries, and updating systems in order with entered or updated data.

Workflow tools provide the ideal method of performing these activities, are vastly superior to using code, and they can therefore provide a significant business benefit from an integration layer, given these points:

- Working across multiple systems is inherently process driven, executing tasks in a step-by-step way with decision points, branching, and other core workflow steps.
- System integration workflows sit centrally among applications and are therefore more often impacted by changes to the entire environment, given that each change to each system will be reflected in changes to the integration layer. The use of workflow tools allows processes to be more easily understood by a wider group of people than the original developers, given the graphical and somewhat self-documenting nature of the workflows; processes to be more easily and quickly changed, and more easily debugged than code; and the use of less code, which results in both greater reliability and easier debugging.
- Given the diagrammatic form that workflow tools provide an easier understanding of the business processes, ownership can be extended easily beyond the development teams to business analysts and even core users in the business community.

A Successful Integration Mindset

Having the tool and using it correctly are two very different things, and this discussion outlines a *mindset* that is needed to achieve truly successful integration. Regardless of the approach that is taken, remember that an enterprise-wide view of integration will lead to far greater business benefits; integration itself is nontrivial and should be considered in an enterprise context; and while considering the enterprise context and designing for maximum business benefits it is still possible—and necessary—to begin to implement the solution with a single project.

The enterprise view of integration should be taken as early as possible. While there is clearly a place for point-to-point integration where further integration is unlikely, a more capable integration environment providing the workflow and other capabilities outlined here will provide advantages as the number of systems and integration grows. Assessment outside the scope of each single project is therefore needed to uncover whatever opportunities may exist.

As with all architectural decisions there are business drivers, costs, capabilities, and so on that are taken into account when making architectural decisions. The approach to integration architecture recommended here is well suited to the capabilities of today's tools and today's architectural concepts. It will hopefully help you in understanding some of the decisions that can be made in the area. •

About the Author

Kevin Francis is an IT professional with 19 years of industry experience in a range of positions—from CIO in a manufacturing company, to security consultant, to global architect—and he also operated his own successful software development organization. Kevin has been architecting and managing leading-edge, e-commerce projects since the birth of e-commerce. As an Infosys principal architect, Kevin is responsible for the success of the company's technical implementation for its customers. He consults to large organizations; works with Infosys's architects, development teams, and customers to ensure that each technical solution adds to the overall quality and strategic direction of the customer; and is a member of The Infosys Australia Technology Council. Kevin was awarded a Most Valuable Professional (MVP) award by Microsoft in July 2005 for his knowledge and experience in the Visual Developer – Solutions Architect area.

Simplify Designing Complex Workflows

by Andrew Needleman

Summary

Several factors go into good workflow design. In simplifying the process it's important to focus on diagramming the necessary steps of a particular key transaction such as an e-commerce order or a medical consultation. In this discussion we'll look at representing workflow in a new type of diagram, called a "dots-and-lines process" diagram. We'll begin by diagramming the possible steps that can be taken at each point of the workflow, and then we'll indicate where that diagram takes us in this workflow process. We'll also ensure that we don't miss any of the potential steps or status points along the way.

Good workflow design requires business process analysis, business process redesign, usability analysis, and software design. This daunting list of necessary skills makes workflow a challenge for the most experienced software architects. How do you break down the problem into its smallest parts to get you started in the right direction? We'll look at how to simplify the design process of complex systems with multiple user types and hundreds of potential statuses. In simplifying the process you can think about the workflow solution rather than the workflow design process.

Each software project has a central process that is the main reason why the system exists or is being built. For example, the main goal of an e-commerce site is to make it easy for people to buy products. This main process is the one that needs the most attention during its design and implementation. We'll show you the *dots-and-lines* approach to modeling that will help you to communicate about complex processes with your business experts to determine available actions and statuses as you are creating the workflow.

Let's begin with an overview of the dots-and-lines approach to creating workflow diagrams. In the dots-and-lines approach each *dot* represents a particular status in the process, and each *line* corresponds to an action that can be taken from that dot. Dots and lines are labeled, and a key is made as the workflow is developed. Dots are labeled with letters, starting with A, and lines are labeled with numbers, starting with 1. That way dots and lines won't get mixed up.

The primary goals behind the dots-and-lines approach are to create a simple visualization tool on paper that can be used with business users, eliminate the perfectionism caused by diagramming on a computer, make the process as lightweight as possible to allow concentration on the tough task of designing the workflow, and separate the changes in description of actions and statuses from changes in the process flow (that is, similar to the idea of a lookup table).

Follow the Rules

In addition to these goals there are five rules that must apply to a diagram using this approach. Let's look at the details of each rule.

The diagram always goes in one direction. One of the main ideas of the dots-and-lines approach is that the diagram always reads forward like a timeline. You may end up at a previous status, but your diagram doesn't have lines going in all four directions like a maze.

Statuses can be discovered as you are building the diagram. In a simple workflow you know all of the potential statuses before you start diagramming. However, in a more complex workflow, you may need to discover the potential statuses by building your diagram. Determine what actions you can take at each point in the process, and see where that takes you. As you discover the statuses, you can decide how to label each status for your use. Then, after the diagram is complete you can decide how to present the statuses to each user type and whether to combine them into more easily understood statuses.

> "EACH SOFTWARE PROJECT HAS A CENTRAL PROCESS THAT IS THE MAIN REASON WHY THE SYSTEM EXISTS OR IS BEING BUILT"

The labels of statuses and actions are kept off the diagram. When you have descriptions of each action and status, it makes the diagram cluttered and hard to understand. By putting the labels for the statuses and actions in a key, you make it easier to see a larger process flow in a smaller space, which is almost like normalizing your diagram. You can change the labels of the points without having to change the diagram. This makes it easier to view and diagram complex processes.

Layout is simple and flexible, allowing everyone to focus on the content. Layout is a large problem in traditional process flows.

A lot of effort goes into connecting and fitting in the different items within the diagram. If you leave out a potential action in a process flow, then the entire diagram might have to be rearranged. In the dots-and-lines approach you just have to fit in a line for the action and a dot for the status of the result. If you need to change an action or status, you can change its number or letter, respectively. If you want to change the name of a status or an action, you can just change the key rather than the diagram. The simplicity of adding, changing, and viewing dots and lines to your diagram makes it easy to stay focused on the process itself.

> "STATUSES AND ACTIONS HAVE TO BE INCLUDED IF THEY ARE REQUIRED TO COMPLETE THE PROCESS THAT IS BEING STUDIED"

Statuses can be "torn off" the main diagram. Since you are always going forward with your diagram, you can *tear off* sections of the diagram without worrying about how they connect to previous statuses in the workflow. For example, A connects to B, and we want to tear off B onto its own diagram because it has too many children to fit on the existing diagram. We don't have to worry if B has an action that goes to C and C has an action that sends its status back to A. By labeling the status with A, they are connected, rather than having to get a line back to the original A status. Our diagram only moves forward, not backward, so we don't have to connect back with the original A dot. This rule allows for great flexibility when you are running out of room on a diagram.

Table 1 A key for the e-commerce order diagram (see Figure 1)

Actions	
1	Find and add a product to the shopping cart.
2	Register.
3	Log on.
4	Remove all products from the shopping cart.
5	Log out.
6	Begin the checkout process.
7	Add/select shipping information.
8	Add/select billing information.
9	Billing processing succeeded.
10	Billing processing failed.
11	Confirm order.
12	Exit the checkout process.
Status Descriptions	
A	No products are in the shopping cart, not logged on.
B	Product is in the shopping cart, not logged on.
C	No products are in the shopping cart, logged on.
D	Product is in the shopping cart, logged on.
E	Begin checkout process, not logged on, no shipping or billing.
F	Begin checkout process, logged on, no shipping or billing.
G	Checkout process, logged on, shipping, no billing.
H	Checkout process, logged on, billing and shipping.
I	Checkout confirmed, logged on, temp status (square).
J	Order processing succeeded, logged on.

Actions and Statuses for Workflows

Now that we've covered some of the advantages of the dots-and-lines approach, let's look at how you can determine if an action or status should be on your diagram. Statuses and actions have to be included if they are required to complete the process that is being studied. For example, even though logging on or registering on an e-commerce system does not change the status of an order object, it is required to complete a purchase. Therefore, it should be included as an action on the system, and the statuses that these actions create should be studied and differentiated if we are examining the ordering process.

Statuses and actions do not have to be included if they are not required to complete the process that is being studied and do not change the steps needed to complete the process that is being modeled. In other words, an action that sets you back a step in the process is something that should be modeled. On the other hand, an action that isn't related to your process does not have to be modeled.

Of course, there is definitely a lot of room for interpretation as to what should be placed on a process diagram; because turning on a computer is required to perform any computer process, it is probably a bit silly to include this action as part of a process diagram. Now we'll use an example to try to clear up what should and what should not be on your dots-and-lines diagram.

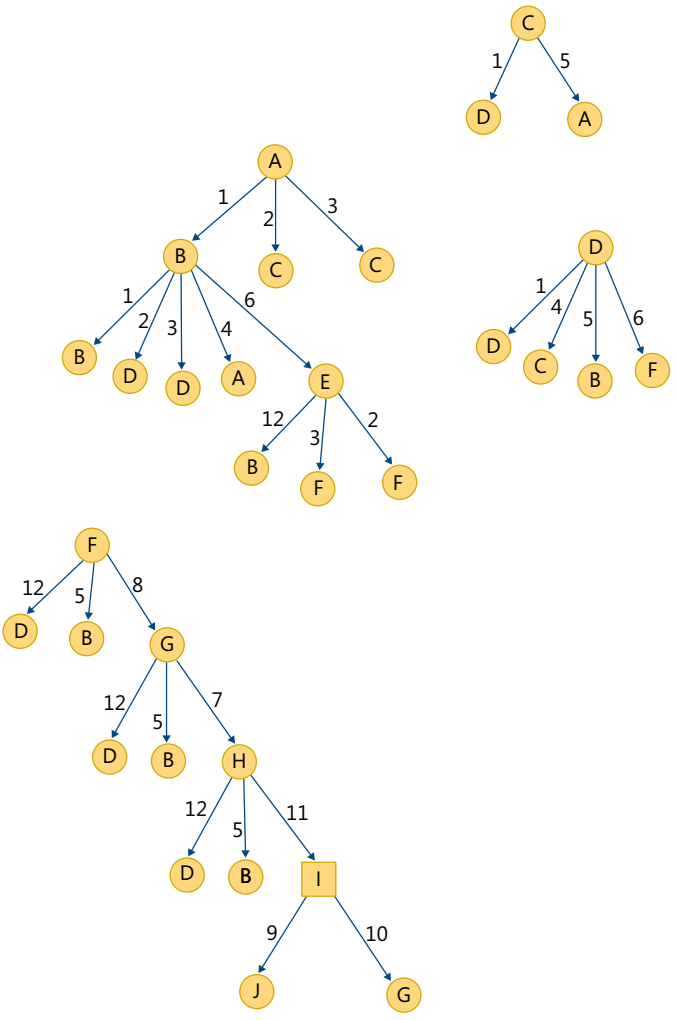
Let's look at a familiar example of a workflow to examine with this process—the purchase of item(s) through an e-commerce Web site. We're going to examine the necessary steps to take for purchasing a product.

When we're beginning to design a purchasing workflow, we'll ignore steps that don't change the status of the purchasing transaction. For example, the process of browsing for the product does not change the status of the order process in any way. It really doesn't matter to us how the user gets around the site unless they do one of three things that change their current status. These three steps that can move the status of purchasing the product forward are adding a product to the shopping cart, registering for the site, and logging on to the site.

For example, pretend you go to the "books" section and click back to the home page. Has this action changed your status in any way in terms of purchasing a product? No. Pretend you search the site for ".NET books," and you receive a list of them. Then you click back to the home page. You still haven't moved forward in the purchasing process.

Even if you pretend that you have an engine that matches product suggestions to your browsing history, it still doesn't move the purchasing process forward until you add a product to your shopping cart. It is very important to keep this step out of your diagram because it will unnecessarily complicate it. We can combine the browsing and then adding a product to the shopping cart into one action in the purchasing-process diagram, called "find and add product." If we wanted to diagram the pro-

Figure 1 A dots-and-lines diagram for an e-commerce, product-order process (see Table 1 for the key)



cess of finding a product, then we would be interested in how the user selected the products to view and add them to his or her shopping cart. For now, we'll focus on diagramming the purchasing process.

Roles in the Purchasing Process

First, we'll start with someone who has just gone to our e-commerce site's home page. We look to see what comes next in our process. The three possibilities in terms of behavior that moves the purchasing process forward are adding a product to the shopping cart, registering a user, and logging on to the site. You can view the dots-and-lines diagram for this entire process in Figure 1 and its key in Table 1.

It is important to understand the impact multiple roles on the site has on our statuses and viewable permissions. For example, assume that our e-commerce site has roles for both shoppers and an administrator. The shoppers are the people who purchase the product, while the administrator makes sure that they get the product. Our dots-and-lines diagram helps us determine what to show the administrator by providing a comprehensive list of statuses that an order is in during the buying process. We can then take these statuses and run down the list to find out which orders should be seen by the administrator and which should not. For example, one potential business rule may be that the administrator can see order transactions that are abandoned at the billing prompt, to allow the administrator to follow up with those customers.

For more complex sites with hundreds of potential statuses for each transaction going through the site, we'll need to roll up multiple statuses into single-status descriptions. For example, statuses E–G can be grouped as an "incomplete" status for the administrator. It is unlikely the administrator will need to know more than that, except for aggregate statistics that would be provided in a report rather than the daily work screen. Plus, if the administrator goes into one of those orders, he or she can provide the status details at that point.

> "WHEN WE'RE BEGINNING TO DESIGN A PURCHASING WORKFLOW, WE'LL IGNORE STEPS THAT DON'T CHANGE THE STATUS OF THE PURCHASING TRANSACTION"

One of the easiest ways to roll up status for each user is to use the "who has the ball?" method. In other words, which user is supposed to do something to advance the workflow? The user(s) that fit this description should see this item grouped with other items that are waiting for their action. Then you can group the other open items in another category and finally the completed/canceled items in a third category.

One final recommendation regarding the dots-and-lines approach is to use a very large piece of paper and a pen to create the diagrams. Another large piece of paper should have the names of the statuses on it, and a final one should have the names of the actions. Using large, separate sheets of paper in this manner allows everyone to get a good bird's-eye view of the entire process rather than having it broken up on multiple screens.

Using paper also focuses everyone on the problem at hand, instead of making it look good. If you are using paper, there is no easy way to put a gradient on each dot or shuffle the items around to fit better. You should find the dots-and-lines approach as helpful as I have in discovering all of the potential actions and statuses with your business experts. •

About the Author

Andrew Needleman is the managing partner of Claricode (www.claricode.com), developers of custom solutions exclusively for the health care industry. He has written articles focusing on software development for many publications in the IT and health care industries. His workflow design experience includes architecting an application with hundreds of distinct states, multiple actions at each state, and nine distinct user types. This application was recognized as an Intel Solution Blueprint for best practices in health care with Microsoft technologies. Contact Andrew at andrew@claricode.com.

Enable the Service-Oriented Enterprise

by William Oellermann

Summary

Building Web services is pretty easy today. Building lots of Web services is a little more difficult, and managing those Web services is really difficult. As the number of services and service consumers grow, the fundamental benefits of service orientation diminish if certain IT capabilities are not provided. Let's look at a model that can facilitate the identification and prioritization of these capabilities for the service-enabled enterprise.

When talking with customers and colleagues about Web services, you'll find that much of the excitement and interest is focused on the specifications that have helped this technology quickly become a de facto standard: SOAP, WSDL, XML, WS-Security, and so forth. While this standardization is critical, it often distracts us from the next level of discussions about how to really take full advantage of this common stack. These protocols give us greater reach and reuse than we have ever previously achieved with software, but their use alone will not fulfill that potential.

All the work invested in standards and consortiums allows us to develop services that can interoperate with other applications and systems, but it doesn't necessarily buy us reuse. Even more importantly, it doesn't buy us manageability. If you happen to work in or with an organization that has attained some success with Web services, you have no doubt encountered the need to address, if not the pain of not addressing, service testing, deployment, versioning, and management. These areas become very problematic when they are not identified prior to building a portfolio of services.

This condition has already made some organizations victims of their own success. Approaches that work with a handful of services often falter as the number of services and their usage increase. If you don't have the principles and practices in place to promote reuse and provide manageability as your use of services grow, you will hit the "scalability ceiling." (See the sidebar, "A Word on Maturity Models.")

Discussing this issue is very difficult, even among the savviest and most experienced of architects, much less other less-technical participants that play a role in the services ecosystem. Some of this difficulty stems from the use of overloaded terms such as *service*, but much of it comes from the sheer complexity involved with a massively distributed architecture that touches many areas of an organization. If something is difficult to discuss, just think of how difficult it can be to define a strategy or plan for it.

These challenges led us to look at the use of a model to help facilitate the discussion of service enablement and the planning of a service-enabled enterprise. With the contributions and feedback of many organizations and thoughtful contributors over the last two years, we have developed the Enterprise Service Orientation Maturity Model (ESOMM).

A Capability-Driven Maturity Model

All 4 layers, 3 perspectives, and 27 capabilities defined in the ESOMM are designed as a road map to support services—not any specific service with any specific use, implementation, or application, but any service, or more specifically, any *set* of services. If you intend to develop only one service, the value this model provides will be limited. The objective of ESOMM is to help you overcome the problem of scalability at a group, department, division, or enterprise level. This objective means it can be applied as part of a grassroots initiative or an enterprise strategy. Your ability to provide certain capabilities might be limited by the extent of your overall organizational alignment, but that factor should not impede your ability to start building and implementing a plan leveraging ESOMM.

Developing a service-enablement strategy is not a trivial undertaking and should not be treated as a short-term or one-time effort. At its apex, service orientation is intended to enable a higher level of agility for an organization, so it can provide an expedient response to the needs of the business and its customers. Successfully achieving this agility means providing a plan that is both extensible and durable. By focusing on the service, and not the message, implementation, or usage, ESOMM can help you to build such a plan that can be applied broadly by any organization. As the preferred service implementations, technology choices, and usage patterns change, the model should provide a durable road map that can endure those changes.

> "ALL THE WORK INVESTED IN STANDARDS AND CONSORTIUMS ALLOWS US TO DEVELOP SERVICES THAT CAN INTEROPERATE WITH OTHER APPLICATIONS AND SYSTEMS, BUT IT DOESN'T NECESSARILY BUY US REUSE"

ESOMM consists of four layers, each containing a set of capabilities that as a whole establish a solid level of accomplishment and value to the organization for that layer (see Figure 1). This value should be fully realized if all the attached capabilities are provided. It is possible, and certainly likely, that an organization may address parts of multiple lay-

ers; the objective is not to satisfy a layer so much as it is to identify the level of value an organization desires.

The first layer of maturity in ESOMM, called *usable*, consists of the appropriate use of standards and protocols to design and develop services that are usable across an organization's platforms and technologies. Building services *correctly* is absolutely critical before you can successfully establish many of the higher-order capabilities up the SOA stack. The true test of how well your services are built is how much commonality and consistency is present across all of your services, regardless of the tools and technologies used.

The *repeatable* layer addresses many of the capabilities that make services efficient to develop, deploy, and maintain across your organization. Without addressing key capabilities in this area, it will be very difficult to successfully scale your use of Web services.

The *supportable* layer encapsulates many of the capabilities necessary to reliably provide services for mission-critical applications that are supportable. These capabilities include the operational support of services, but just as important, the self-service capabilities provided to consumers. Self service is an important tool to enable your services to scale by keeping your developers and architects from spending an inordinate amount of time in correspondence with potential consumers of their services. This layer is very difficult to approach successfully without some success in making services repeatable in their deployment and management.

The *extensible* layer represents the pinnacle of realizing the business agility promised by services and the reuse of services as the building blocks of other services, or service aggregation. This notion usually involves providing capabilities direct to customers and/or partners through services that generate new revenue channels, if not new business models. There are several capabilities required to make this realization possible, and they make up the majority of the focus at this extensible layer.

Figure 1 The four maturity layers of ESOMM

Extensible	An enterprise is capable of aggregating services and extending their use beyond its own borders.
Supportable	An enterprise can effectively manage increasing numbers of services to guaranteed SLAs.
Repeatable	An enterprise implements, consumes, and reuses services efficiently and consistently.
Usable	An enterprise is capable of writing and consuming standards-conformant services with excellent reach.

Road Map Perspectives

Through engagements and interaction with customers over the last couple of years, we recognized a need to address different perspectives through this model. As we discussed the capabilities in each layer of the model, it became apparent that there were varying levels of appreciation or consideration for them, depending on the audience's role and challenges. For instance, one organization had a hard time appreciating the need for contract-driven interface design. They were quite content using tools to dynamically generate service contracts and felt they were very successful with that approach.

When talking with the users of the group's services, however, we got an entirely different perspective by learning that the use of specific data types and element patterns caused a lot of difficulty during implementation. By recognizing that not all capabilities are driven by the service owners or providers, stakeholders are able to have greater appreciation for some of the capabilities and make more informed decisions about approaching and prioritizing them.

The capabilities in each layer of the ESOMM are categorized into one of the three perspectives: implementation, consumption, and administration (see Figure 2). *Implementation* capabilities target the development and deployment of Web services from the provider's perspective. This perspective is the one most organizations appreciate by default and requires the least amount of clarification. *Consumption* capabilities are those that cater to the consumers of your services, making them easier to implement and therefore more widely and successfully adopted. We believe this is the most neglected perspective, yet it can be the most critical in determining your success with service orientation. If it is difficult to leverage your services, your success will be limited, no matter what other areas you have optimized. *Administration* capabilities use the operational and governance aspects of Web services across the organization. This perspective is becoming more recognized and appreciated, but there is still more that needs to be understood about its value and impact.

The additional value of perspectives is the aid they provide in prioritizing the implementation of specific capabilities in the model. While there is an ability to ignore certain capabilities while progressing up the model, it is fairly risky to skip capabilities at a lower layer within the same perspective. Those prerequisites are intended to not only help you grow your system logically, but also to help you learn how services are utilized and supported across your organization because this information will impact the design of certain capabilities at higher layers.

> "THE EXTENSIBLE LAYER REPRESENTS THE PINNACLE OF REALIZING THE BUSINESS AGILITY PROMISED BY SERVICES AND THE REUSE OF SERVICES AS THE BUILDING BLOCKS OF OTHER SERVICES, OR SERVICE AGGREGATION"

It is also important to realize that an organization can choose to focus on one or two of the perspectives, but the overall maturity, and, hence, the overall value of your services, depends on the appropriate level of attention to all three. Neglecting any of the three will have consequences—some more readily apparent than others. Such consequences would be akin to ignoring one of the food groups; you can do it, but it would limit some of the meals you could enjoy and not be ideal for your overall health.

ESOMM Capabilities

There are 27 capabilities outlined in the ESOMM model, and each could easily take an article to explain in full detail (see Figure 3). Here, we'll look at a brief overview of them to help articulate the premise of each capability and its alignment to the associated perspective and layer.

The implementation perspective includes the development processes and design patterns capabilities in the usable layer (1):

Development processes. Like design patterns, development processes speak to a concept with which development teams are very familiar, but is focused specifically on the implementation of services. At a basic level, this concept would generally consist of the use of common standards supported across the most common platforms and tools, including SOAP, WSDL, WSA protocols, and the WS-I Profiles. At a more advanced level, the capability would include the definition of repeatable processes for the design of service interfaces, the deployment of services, and the publication of services. Again, the objective is to make these processes broadly available across platform boundaries.

Design patterns. Whenever you develop software, it is good to take advantage of *proven* design patterns that are appropriate for that implementation. This particular capability speaks to the design patterns that are appropriate and necessary, specifically for use with services. We have good information available in this space, starting with the four tenets of service orientation (see Resources). Other examples of relevant patterns include asynchronous execution, callback support, exception handling, and the use of custom SOAP header data. The key to fulfilling this capability is defining patterns that can be used by developers across all the appropriate tools and platforms.

From the consumption perspective of the usable level there are the explicit contracts and testability capabilities:

Explicit contracts. The most basic information you should provide to consumers of your services is explicit contracts, which includes not only richly detailed schemas (leveraging the power of XSD), but rich policy data that shares appropriate context information. This information allows them to know not only the makeup of the body of the message payload, but the header elements that are supported and required. The difficulty of this capability is that there is very little support for some of the standards emerging around the sharing of policy, so it requires some thought and effort to address sufficiently.

Testability. This capability provides consumers with a mechanism to test the invocation of a service to support implementation and troubleshooting from their side. One of the most challenging aspects of implementing distributed systems is integration testing. With so many dependencies, it can take a lot of coordination to simply complete a call through the entire chain, even for a null operation or an invalid request. Web services are in a unique position to abstract applications from the back-end systems performing the processing; a testing environment, either physical or virtual, can be a strategic tool in improving implementation efficiency and reducing the need for manual support.

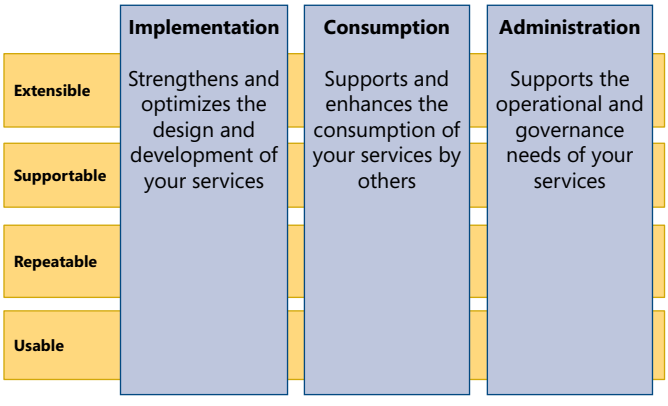
The administration perspective of the usable layer contains the basic monitoring and security model capabilities:

Basic monitoring. The support of Web services starts with support for the basic infrastructure hosting and providing those services. This support includes the basics of operational excellence, which includes the monitoring of key counters and logs that could indicate processing issues and provide alerts on the triggers indicating outages for critical services. This capability does not include service-level visibility or monitoring, as that is not a trivial undertaking and will come further up the model's stack.

> "ONE OF THE MOST CHALLENGING ASPECTS OF IMPLEMENTING DISTRIBUTED SYSTEMS IS INTEGRATION TESTING"

Security model. Once the adoption of services takes hold, organizations will find them used in a multitude of applications—both external and internal applications. It is important to have a standard security model that can support each of the primary scenarios to meet business requirements. This model involves creating one or more credential stores that can be managed and supported as part of your services ecosystem. Examples of these credential stores include certificate authorities, an LDAP directory, Active Directory directory services, and databases. While credentials make up a critical component of your service-security model, fully addressing this capability also means supporting the requirements around access, authentication, authorization, and encryption. These approaches don't necessarily need to be implemented as features, if they are not required at the time, but a complete security model needs to be defined for when the time comes. This implementation will likely be the first step in defining your service policy.

Figure 2 Three perspectives on the maturity layers



Repeatable Layer Capabilities

The implementation perspective includes the service blocks and common schema capabilities in the repeatable layer (2):

Service blocks. A key to reducing the amount of "routine" logic pushed down to each and every service is the use of a concept called service blocks. This capability is an architecture model that provides common routines that should be leveraged broadly such as transformations, point-data lookups, and utility processes. One example of a service block would be a logging service, and in fact our logging application block could be service-enabled to meet this need. For this model to be successful, these service blocks should be available to any and all services across your organization.

Common schema. When providing services that work with common entities, it is very helpful to have a common definition for those entities to minimize the amount of effort necessary to provide and consume those services. Because of the multiple systems typically involved in providing the business logic and data for services, this definition is usually an ongoing effort that will not necessarily ever be considered *complete*. However, even establishing common definitions for a handful of core entities can reap tremendous benefits for service-enablement efforts.

The consumption perspective for the repeatable layer includes the service discoverability and self provisioning capabilities:

Service discoverability. Once you start to provide more than a handful of services, discoverability becomes a key capability to help efficiently guide consumers in the identification of the appropriate service to use. This guidance can be accomplished manually or programmatically with tools (for example, through UDDI). Unfortunately, just providing a list of services on a Web page doesn't prove sufficient at this level, so one of the keys to guiding the discovery process is intelligent categorization. One or more well thought out taxonomies can help potential consumers look intuitively in the correct places for services based on a domain or specific implementation.

Self provisioning. As you develop more services, it will become more challenging for consumers to request access to them. While you, as a provider, might be interested in automating the back end of the user-provisioning process, it is more important early on to allow consumers to quickly identify and leverage the tools available to request access. If this process is too difficult or problematic, your adoption rate will likely suffer, limiting your overall success. This capability can be achieved with a wide range of solutions ranging from very simple to very complex.

The administration perspective of the repeatable level includes the enterprise policies and deployment management capabilities:

Enterprise policies. Service policies are important to define, support, and enforce as you grow your organization's use of services. The starting point for this effort was developing a security model, but at this level it is necessary to broaden the policy to support a set of possible options available depending on the different criteria of a given scenario. For example, a service used within a department will likely have a very different set of policies from the same service used by partners. At this level, the definition of internal categorization(s) becomes important as it can help you to define baseline templates aligned with some of your most common scenarios to promote consistency and efficiency. Success in this area isn't driven as much by the choices made as it is by the establishment, use, and distribution of policies to those impacted.

Deployment management. The best way to promote and enforce a consistent governance model for your organization is to define a common deployment process for services that provides real value. For instance, if your deployment process automatically provides services with operational support, it will quickly become widely adopted. To further strengthen its role, it is also important that this process is flexible and easy to use, so that developers, architects, testers, and operators across the organization will buy into and adhere to the established model.

Supportable Layer Capabilities

The implementation perspective of the supportable level (3) contains the schema bank, executable policy, and versioning capabilities. Let's look at them in more detail:

Schema bank. Creating and supporting a schema bank is the next step in leveraging a common schema. Doing so allows developers to leverage and reuse existing entity definitions to *build* messages that are able to adapt automatically to schema extensions through includes and imports. This leverage is a very challenging area that very few customers are successfully tackling today, but will quickly become an invaluable tool as your services start to evolve.

Executable policy. While the request message initiates the invocation of a service, you will often want to leverage policy information to determine exactly what should be executed under certain conditions. Policies are often thought of as a tool for external use, but policies can also be a powerful tool for providing context internally. If you need to change the execution process or behavior based on who is making the request, what format the data takes, or what time of day it is, executable policy will play an important role in service enablement.

> "TO MAKE YOUR SERVICE CONSUMERS MORE SELF-SUFFICIENT, PROVIDE THEM AMPLE VISIBILITY INTO THEIR USE OF YOUR SERVICES"

Versioning. The versioning of Web services is something that can and should be handled differently from the typical approaches for other software because of its complexity and extensibility. For instance, the versioning of a service needs to be treated separately from the versioning of the messages involved in the service. Fortunately, with the inherent flexibility of XML and XML schemas, there is an ability to support this complexity through both implicit and explicit versioning meth-

A Word on Maturity Models

Like other maturity models, ESOMM was derived from Carnegie Mellon's Capability Maturity Model (CMM, see Resources). However, we borrowed the notion of a capability-driven maturity model and stopped there. Instead of applying service orientation to the CMM, we took those principles and applied them to the service-orientation paradigm, essentially building the road map from scratch. This model does not focus on processes because the focus is on IT capabilities, not organizational readiness or adoption. While you will recognize some conceptual similarities with CMM, you can also see that ESOMM is a decidedly different application of the maturity model concept.

Although there can be value realized through some of the other service-related maturity models out there, be aware of how much of the CMM implementation it subsumes. CMM was designed to address an organization's processes related to software engineering, which is a decidedly different problem from defining an IT plan for service orientation. If you adopt a model that blends these two, it could be overwhelming and counterproductive for the organization. Even worse, if you adopt a model that simply renamed some of the CMM levels and capabilities, you could be going in the wrong direction. That would be like taking a map of England with the name London replaced by Paris and trying to go to the Eiffel Tower—it simply isn't going to work!

Buying into the capability-driven maturity model is one decision. Choosing the appropriate road map is another. Make sure you are picking a map that applies to your objectives. No maturity model applies to everyone in all circumstances; make sure you choose one that takes you to your destination.

Figure 3 Capabilities of the maturity layers

	Implementation	Consumption	Administration
Extensible	Service collaboration Service orchestration	Service SDKs External policy	Business analytics Automated policy management
Supportable	Versioning Executable policy Schema bank	Explicit SLAs Service portal Execution visibility	Auditing Monitoring Provisioning model
Repeatable	Common schema Service blocks	Self provisioning Service discoverability	Deployment management Enterprise policies
Usable	Design patterns Development processes	Testability Explicit contracts	Security model Basic monitoring

ods. However, this ability is a feature that the typical service development tools do not take advantage of out of the box and requires some planning and support to execute consistently.

The consumption perspective of the supportable layer is where we find the execution visibility, service portal, and explicit SLAs capabilities:

Execution visibility. Perhaps the single, biggest factor in your ability to scale your use of Web services is your ability to support increasing service consumption by providing rich self-service capabilities to potential and existing consumers. If every implementation of a service requires a series of phone calls or e-mails, some simple math will show you that your team's ability to develop services will quickly be consumed by support needs. A single portal that provides access to the right information can be a huge factor in reducing the demand on your development and support teams.

Service portal. This capability speaks to a specific resource that could be accessed through the self-service portal. To make your service consumers more self-sufficient, provide them ample visibility into their use of your services. More visibility builds on the ability to test services by providing a level of partitioned access that helps them to understand what is occurring during execution to use troubleshooting and also deduce whether something is failing to meet their expectations. This capability would also share historical activity to help consumers track how often the service is consumed by their application(s).

Explicit SLAs. To effectively set expectations for your services to consumers, you must provide explicit service-level agreements (SLAs). Providing SLAs is a very challenging capability because there is not a standard available to communicate this information for Web services. However, various forms of documentation or communication can be used to effectively share important information in this area including the expected lifespan, the anticipated processing time for requests, and any planned downtime. It is important to note that without a sufficient level of service visibility and management internally, it will be nearly impossible to support SLAs.

The administration perspective of the supportable level contains the provisioning model, monitoring, and auditing capabilities:

Provisioning model. The next step in making services self sufficient and supportable is a streamlined process for provisioning new users. This model would be used to support external and internal users and would include the handling of requests for access, creating new credentials, and providing alerts notifying consumers of changes to services or account status. This capability builds on the self-provisioning capability defined in the repeatable layer (2) to provide an end-to-end provisioning process for service consumers.

Monitoring. Monitoring and visibility into service execution is one of the most challenging aspects of supporting services on an enterprise scale because of the distributed nature of Web services. The only way to address this support effectively is to embrace your services as a system because support on a service-by-service basis is an inefficient and frustrating endeavor. While there are many ways to establish this system, we believe that a common execution architecture is the most effective means.

Auditing. The key to truly trusting the services you provide in your organization is a reliable audit model that can serve as a tracking mechanism for who did what and when. This mechanism not only helps to resolve any disputes that arise, but can also play a strategic rule in an enterprise-compliance program.

Extensible Layer Capabilities

Now let's look at the implementation perspective's capabilities for the extensible layer (4), service orchestration and service collaboration:

Service orchestration. To leverage Web services in well-defined business processes, you must have the ability to orchestrate. This ability involves workflows that may involve manual as well as automated processes, but it allows you to define a fairly consistent and repeatable usage of services that can be changed over time.

> "WHILE IT IS IMPORTANT TO KNOW WHERE YOU ARE, GETTING AN EXACT BEARING IS LESS IMPORTANT THAN IDENTIFYING THE CAPABILITIES YOU NEED TO ADDRESS TO CONTINUE ADVANCING THE VALUE OF SERVICE ENABLEMENT IN YOUR ORGANIZATION"

Service collaboration. Once you've established services as autonomous functions that are maintainable and supportable, you can start to aggregate them to provide more sophisticated and complex services. We refer to this aggregation as service collaboration. Through collaboration, you can define synchronous and asynchronous execution patterns that allow you to support entire processes consisting of existing services. As an example, you could take a service that sends an e-mail and a service that retrieves all schedule changes and expose a new service that sends an e-mail to customers notifying them of schedule changes.

The consumption perspective of the extensible layer includes the external policy and service SDKs capabilities:

External policy. To establish a secure infrastructure for providing services to external consumers and partners, you must establish external policies, which include not only externally supportable security policies, but the policies you have defined for how your services can be bundled with other third-party services. Such policies include branding, payment, and usage terms that are important for protecting your business interests.

Service SDKs. Building a business model or channel through Web services requires taking the level of support for service consumers to the next level. This advance to the next level could include developing and providing prepackaged software development kits (SDKs) that allow developers to quickly provision and implement your services. These SDKs potentially include UI components for direct use in the developer's application(s) and would be tailored to target development environments, as there is no standard for cross-platform development tools today.

Finally, the administration perspective of the extensible level includes the automated policy management and business analytics capabilities:

Automated policy management: Once you start exposing services to a very broad audience and using policies to define your interoperability support, you will start to get requests for changes in policy—for example, changes to SLAs or security requirements. These requests should be processed in an automated fashion. That way they don't create significant overhead in scaling your interaction with a broad customer base.

Business analytics. The pinnacle of visibility across a system of enterprise services is what we refer to as service analytics. This visibility combines the technical data from service metadata, activity logs, and events and allows you to cross reference it with policy information, business data, and user data to provide rich views into every dimension of your services. This information can be used to determine what services are making money or what user tendencies are present between internal and external consumers of your services.

Applying the Model

Now that we've had a chance to briefly review the components of ESOMM, you are hopefully already thinking about how it can be applied to your organization. To be clear, applying this model should not be looked at as a one-time activity or short-term process. Instead, the model is best leveraged as a working plan that can be modified over time as the usage of services and your experience grows.

Unfortunately, the downside of using the term *maturity* with a model is that people will immediately want to know what layer their organization is at to get a sense of their status or identity. As it happens, there is no appropriate way to answer the question, "what layer is my organization?" Instead of giving an overall grade based on one of the layers, we take the approach of giving each layer its own level of maturity, ranging from one to five, based on half-point increments.

A maturity level of one means you haven't had the opportunity to address the capabilities at that layer, while five means you have mastered all of the capabilities defined at that layer. A typical result is provided in Table 1. This rating for each layer is determined by the strength of the corresponding capabilities, ranging from one to five, with a strength rated as a five and a weakness of one. The average of these qualities will give you an overall grading for the layer, rounded to the nearest half point. For example, looking at layer 1, if you have three of the six capabilities gauged as a strength, two gauged as adequate, and one gauged as a weakness, the overall maturity of the level would be 3.5.

Table 1 A sample ESOMM rating

Services layer	Maturity level
4 Extensible	1
3 Supportable	1
2 Repeatable	2
1 Usable	3.5

What shouldn't be lost in this rating discussion is the fact that ESOMM is intended to be leveraged as a road map, more so than as an assessment tool. While it is important to know where you are, getting an exact bearing is less important than identifying the capabilities you need to address to continue advancing the value of service enablement in your organization. As long as you are willing to ask yourself some hard questions in an objective manner across all the relevant groups, you should be able to get a good understanding for your current challenges. If you apply the strategy and objectives of your organization, you should be able to identify which capabilities you will need to address in the near, short, and long term.

It is terribly difficult to have a concise, constructive conversation about the service-enabled enterprise. The goal here is to empower you as a software architect with some tools and some of the information necessary to address the right questions in helping your organization achieve real value through this very powerful enablement technology.

Hopefully this brief overview of ESOMM has provided you with enough information to start applying some of the concepts in your service enablement efforts. Each capability could easily fill an entire article, so your feedback of any specific area is welcome.

Many people have contributed to ESOMM and the concepts presented here, but I would like to thank especially Wolf Gilbert, Scott Beaudreau, Michel Burger, and Byron Howard for their contributions. •

Resources

Carnegie Mellon Software Engineering Institute
Capability Maturity Model for Software (SW-CMM)
www.sei.cmu.edu/cmm/

Microsoft Developer's Network
"Architecting Industry Standards for Service Orientation," Josh Lee (Microsoft Corporation, 2005)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/aisforsoa.asp

About the Author

William Oellermann is a solutions architect with Microsoft Consulting Services, who is focused on driving and supporting strategic service enablement opportunities in the communications sector. William came to Microsoft in February 2004 and has spent over ten years as a software architect, spanning consulting and corporate roles at Avanade, Ericsson, and Micrografx. He has been a thought leader in the Web services space for over six years as a frequent speaker at industry conferences, author of multiple articles in trade magazines, and author of *Architecting Web Services* (APress, 2001). When not keeping up with the latest service enablement trends and developments, William enjoys helping his wife keep up with their two very active daughters.

Service-Oriented Modeling for Connected Systems – Part 1

by Arvindra Sehmi and Beat Schwegler

Summary

As architects, we can adopt a new kind of thinking to essentially *force* explicit consideration of service model artifacts into our design processes, which helps us identify the artifacts correctly and at the right level of abstraction to satisfy and align with the business needs of our organizations. Here we offer a three-part approach for modeling connected, service-oriented systems in a way that promotes close alignment between the IT solution and the needs of the business. We'll start by examining the current perspective of service-orientated thinking and explain how the current thinking and poor conceptualization of service orientation has resulted in many failures and generally poor levels of return on investment. Then we'll examine the benefits of inserting a service model in between the conventional business and technology models that are familiar to most architects, and we'll discuss the Microsoft Motion methodology and capability mapping to identify business capabilities that can be mapped to services. Part 2 of this discussion will appear in Journal 8, and it will show you how to implement these mapped services.

Service orientation is high on the agenda for many organizations. The goal is generally to embrace service orientation to enable better alignment between the requirements of the business and the IT services within the organization and ultimately to enable and support more agile businesses. Today, however, many organizations are failing to align IT services closely enough with the business requirements, and as a result they are failing to see the return on investment that service orientation was expected to deliver.

In addressing service orientation, organizations are attempting to answer these questions: How do we avoid making the same mistakes with service-oriented architectures (SOAs) that previous, hopeful initiatives have resulted in? How do we ensure that the chosen implementation architecture relates to the actual or desired state of the business? How do we ensure a sustainable solution that can react to the dynamically changing nature of the business? In other words, how can we enable and sustain an agile business? How can we migrate to this new model elegantly and at a pace that we can control? How can we make this change with good insight into where we can add the greatest value to the business from the outset?

A key premise we promote here is that to build successful service-oriented systems you need to modify the way you think about service orientation and ultimately about SOA and the way you build your systems. Failing to approach the problem in the right way and getting the thinking wrong up front is arguably the cause of many problems. Without the right thinking, service orientation as an approach is unlikely to deliver the expected benefits.

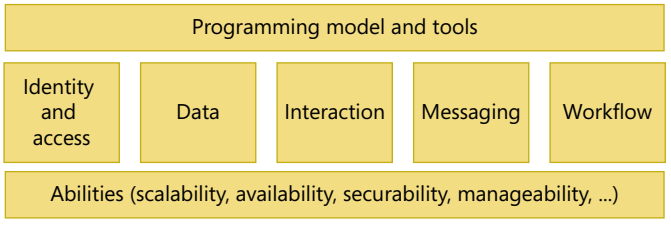
An essential part of the thinking is about being able to conceptualize your business in the right way, and by doing so you can arrive at solid, well-aligned services that map well onto the capabilities required by your business processes. Having identified the right services, you can go on to implement those services by using your choice of technologies such as those provided by the Microsoft application platform. We'll now look at a model that can be applied to connected, service-oriented systems to promote close alignment between IT solutions and the needs of the business.

Five Pillars of Connected Systems

Messaging is important for service orientation, but it is not the only aspect required to model services, and a number of other pillars exist too. While messaging enables you to connect disparate systems and supplies the underlying connection fabric, there are a number of other important issues that need to be addressed, and additional pillars are required to complement messaging. Let's take a closer look at the five core pillars of connected systems (see Figure 1):

- *Identity and access*. This pillar is concerned with the notion of federated identity (Web single sign on) and policy-based authorization. It addresses trust relationship management and how access to the systems now connected should be controlled. Governance and compliance regulations are other important factors that need to be given proper consideration under this pillar.

Figure 1 The five pillars of connected systems



- *Data*. This pillar addresses entity aggregation and is concerned with providing a single, coherent view of a particular business entity such as a customer, even though the customer data might be duplicated many times across multiple systems.
- *Interaction*. This pillar is dedicated to the human consumption of services—for example, through the provision of rich, composite user interfaces with online and offline capabilities. Interaction on the *edge* of networks through the Web, peer-to-peer mechanisms, and mobile devices is also included here.
- *Messaging*. This pillar provides the underlying fabric of connected systems and needs to support secure, reliable, transacted messaging.
- *Workflow*. This pillar addresses business process workflow or automation, external to the services themselves. Workflow is concerned with the orchestration of business processes but also with other aspects such as managing user interaction, ad hoc processes, and exception management.

While the three-part model discussed here is largely concerned with the messaging pillar, consideration is also given to certain aspects of the other pillars. Before examining a new approach to connected-systems modeling, it is important to recognize that many of today's problems stem from conventional thinking that needs to change to realize the true benefits of service orientation.

The Old Perspective of Service Orientation

The traditional approach for architecting and building solutions is to take a set of business requirements and from them derive a technology model that typically involves object orientation, component technology, and perhaps mainframe systems. By failing to work closely enough with the business, there is often a large disconnect between the business and the IT solutions provided. Instead of being inward looking, development needs to become an outward-looking engineering discipline.

The other big issue is that while business people might be very good at what they do, it is not their job to describe what they do to people who are not in their business, which is really what is happening when they talk with their IT people about requirements. Historically, business people have not had great tools to help them to create clear, objective requirements. For example, process maps are subjective views that fail to expose a lot of objective content and metrics—and IT needs objective views.

Another problem often occurs because of the way IT departments are organized. Sometimes they are organized according to specific technologies—for example, mainframe, Internet applications, intranet applications, and so on—and other times they are organized around hard human organizational boundaries and departmental boundaries. With both approaches this organization quickly leads to *silos*, and the way in which information is transferred is largely through prescribed document exchange.

This suboptimal organizational structure also makes it very difficult to create systems that span multiple technologies, and it is very difficult to determine how expensive a solution really is because of the involvement of many people from many different departments.

To help solve this issue, IT departments need to become more outward looking and more closely connected to the business, and a new way of thinking about service orientation and of modeling connected systems is required. Ideally, the IT department should only provide the infrastructural aspects of the solution, and the solution development should be much more strongly aligned with the relevant business groups. Service orientation and the associated software development process will not on their own enable service-driven organizations. They need to be part of an overall strategy that maps the transition toward service-driven IT.

Figure 2 A classic system model

Classic System Modeling

Today's modeling approaches usually focus on business models and technology models (see Figure 2). Ideally, there is a close correspondence and alignment between the business model and technology model, but in practice this relationship is often not the case. Inward-facing IT departments that do not work closely enough and effectively enough with the business are a key reason. While entity domain models are often used in a layer *above* the technology model to help address this issue, this approach tends not to work well mainly because they serve two conflicting purposes: internal implementation and business functional exposure. It is arguable that true alignment between the technology and business models is rarely achieved because the gap between the two perspectives is simply too large. So what can we do to close the alignment? Consider these four essential ingredients:

> "TO BUILD SUCCESSFUL SERVICE-ORIENTED SYSTEMS YOU NEED TO MODIFY THE WAY YOU THINK ABOUT SERVICE ORIENTATION"

Outward-facing IT professionals. IT professionals must become more outward facing and connect more deeply with the business side of their organization. While they do not need to become experts in the business, they need an objective language that allows them to talk with the business about the business. Architects in particular provide the communications channel between the business and the IT departments, and they need to ensure that business requirements and solutions are as tightly interdependent as possible. This interdependence can be achieved by working very closely with the business to ensure that the contracts that the IT department offers the business are much more aligned with the business requirements. In our opinion, taking this perspective is perhaps the best (if not the only) way to arrive at the right granularity for the services that you ultimately build with a service-oriented approach.

In our common vernacular today we talk about "exposing the business architecture." Since the capabilities of a business in a given industry are very similar, or even identical, both IT and business people can use a standard set of questions to elicit relevant information about the business architecture for requirements gathering. By using a standard set of questions, even nonexperts in a busi-

ness domain can facilitate a very useful discussion about business requirements and expose important information on metrics, performance, maturity, interconnectedness, and governance and compliance. Because the businessperson is answering questions from the architect, the architect in fact helps to expose a view that the business may not yet have.

Understanding business change. While we don't suggest that all of IT becomes "business-driven" it can be of great value for IT to understand how the business evolved in the past. This knowledge will influence implementation decisions on the provision of extensibility points, versioning, needed resources, and project management issues like number and length of iterations and release management including the major, minor, and delta releases to expect in a given time frame.

A common operational infrastructure. Common infrastructure is required to support business applications that provide holistic, cross-organizational business practices. Building health models for how to operate and manage the infrastructure and deploy applications onto it is vital to success. Healthy applications allow business decision makers to gain key insight into the information that is being moved around among them, which finally enables individuals to collaborate and make decisions to drive the business more effectively.

Web services technology standards. Web services standards enable applications to be connected. Ultimately, the value of connecting the systems leads to more efficient and effective business practices.

A New Perspective of Service Orientation

As architects, can we ensure that the contracts the IT department offers the business are more closely aligned with the requirements of the business? The introduction of a service model in between the business model and technology model is a key factor that can help achieve this goal. The three-part model of service orientation shown in Figure 3 demonstrates an improvement to the classic system model.

Figure 3 The three-part model of service orientation

The three-part model of service orientation interposes a service model between the business and technology models of the classic system model. The introduction of the service model presents several advantages. The service model is where you can capture the semantics required to express the services that make your solution more loosely coupled and more outward or business facing. The service model provides a logical place to define the contracts that ensure that the business side of your organization is aligned with the IT side from a requirements perspective. By inserting the service model, architects are *forced* to explicitly consider service-model artifacts in the design process.

> "THE SERVICE MODEL IS WHERE YOU CAN CAPTURE THE SEMANTICS REQUIRED TO EXPRESS THE SERVICES THAT MAKE YOUR SOLUTION MORE LOOSELY COUPLED AND MORE OUTWARD OR BUSINESS FACING"

The service model helps architects to discover artifacts at the right level of abstraction to satisfy and align with business needs. It also enables the business analyst to have part ownership of the design process and achieve better business-requirements traceability. Lastly, business people already know the word *service*. For example, in the outsourcing context, a service-level agreement (SLA) is well understood. Internal (not outsourced) functions are similarly understood, though they're (usually) associated with a less-formal, less-contractual service level expectation (SLE). Rather than talking about services and service orientation, it is much better to talk about SLEs because business people will understand their value.

An interesting aspect of the three-part model is that many aspects of the four tenets of service orientation, originally devel-

Figure 4 1) Business function representation, 2) modeling service levels, and 3) implementing the models at different levels of abstraction

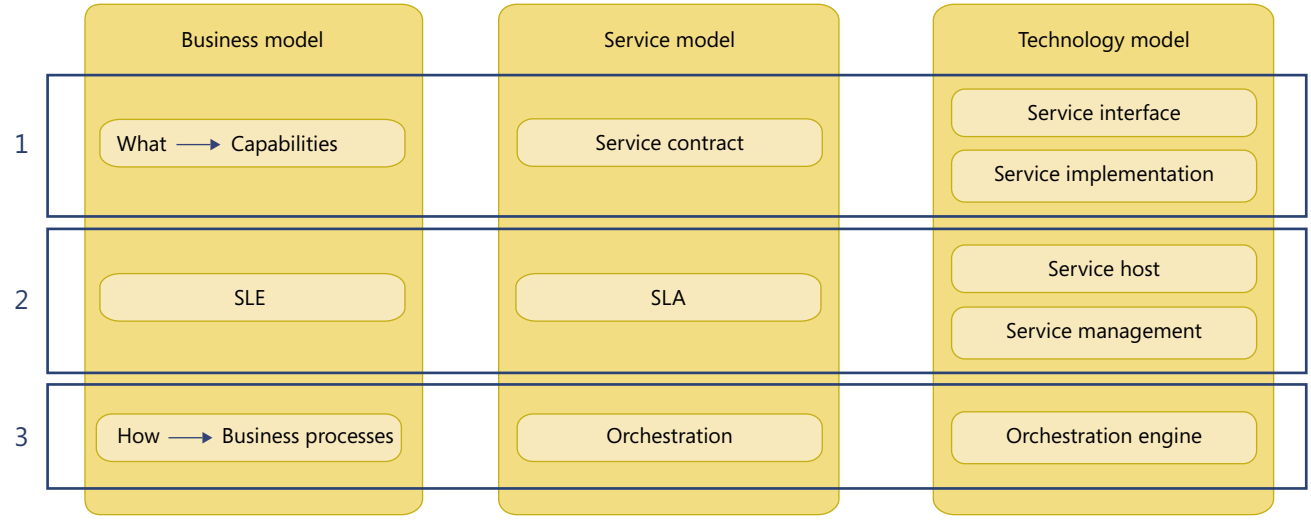


Table 1 The four tenets' requirements and model artifacts

Tenet requirement	Business model	Service model	Technology model
Boundary	Clearly demarcated functional capabilities (Capability map in business architecture)	Externally consumable service interfaces (Service endpoint definition)	Explicit service edge implementation (Concrete hosted service endpoint)
Autonomy	Outsourcing and insourcing capability (Core, operational, and environmental capability definitions)	Interchangeable and loosely coupled services (Spatial, temporal, technical, and operational design considerations)	Implementation independence (Independence of data and behavior implementations)
Contract	Task and process descriptions (Logical units of work and workflow definitions)	Data, interface, and orchestration definitions (message and data XSD, WSDL, and service interaction protocols)	Shareable data, interface, and orchestration implementations (WSDL, XSD, and BPEL)
Policy	Governance, SLE, and SLA definitions (Needs, preferences, expectations, and agreements)	Externally consumable SLA (Compatibility definitions: "I can," "I need," and "I prefer" assertions)	Decoupled operational concerns (Crosscutting concerns security, reliability, transactions, and WSI-Profile)

oped at the technology model level, also apply to the service and business levels.

Service-oriented development is based on four fundamental tenets that were originally derived by the Windows Communication Foundation (WFC and previously code named "Indigo") team at Microsoft. The team originally developed the tenets to describe how to use the WCF programming model to build predominantly fine-grained Web services that are more relevant to the technology model than to the other two models. However, we realized they could lend an important perspective to the three-part model when viewed as *business service tenets* that are applicable at the service- and business-model levels.

Service Orientation Tenets

The four tenets are: boundaries are explicit; services are autonomous; services share schema and contract, not class; and service compatibility is determined based on policy. Let's take a more detailed look at each.

Boundaries are explicit. Service orientation is based on a model of explicit-message passing, and crossing a boundary is considered an explicit act. To use the analogy of crossing into a foreign country, the act of passing from one country to another is an explicit act. From a business perspective, it is equally important that you understand and define organizational boundaries and function boundaries within your own organization to identify clearly demarcated capabilities. From a service-model perspective, this tenet demands externally consumable service interfaces.

> "A KEY BENEFIT IN FOCUSING ON BUSINESS CAPABILITY IS THAT WHILE ORGANIZATIONAL STRUCTURES AND PROCESS FLOWS COME AND GO THE ESSENTIAL CAPABILITIES OF BUSINESSES TEND TO REMAIN CONSTANT OVER TIME"

Services are autonomous. Changes made to one service should in no way impact another service. A service should be able to be rewritten without negatively impacting the consumers of the service. To continue the foreign-country analogy, one country is able to introduce a new taxation system independently from another country because they are autonomous. Similarly, at the business level changes to what happens internally within a particular business process should not impact outside parties who are reliant on that particular process or service. Autonomy at the business level also means that individual services can be hosted within an organization or outsourced with no overall impact on the business process. Autonomy at the service-model level requires interchangeable and loosely coupled services. Autonomy at the technology model level requires implementation independence.

Services share schema and contract, not class. This tenet is largely concerned with services not exposing their internals. For example, countries publish the rules required to complete their visa forms but do not provide the internal resources to complete them. As another example consider J2EE and .NET interoperability. This interoperability is not possible if you try to interchange platform-specific types, but rather an intermediate metamodel or schema is required.

Service compatibility is determined based on policy. At the business model level, this tenet is essentially concerned with governance and SLAs and their definitions. Policy defined at the meta level describes the semantic capability of a service based on a set of explicit statements of its capabilities.

Each of the tenets plays an important role across the three-part model and in different ways influences the business, service, and technology models. Table 1 summarizes the needs and artifacts within each model in relation to the four tenets. Let's see what each model defines and how you should approach their creation.

Models Definition

The three-part model of service orientation must be able to represent *what* a particular business is capable of—its key functional capabilities—rather than *how* the capability is performed. We'll discuss what a business capability is shortly. The way in which the business function within an organization is represented at different levels of abstraction by the three models is shown in Figure 4.

Business function. From the perspective of the business model, what a business is capable of is expressed by identifying business capabilities. Identifying the capabilities of an organization is a critical task while developing the business model. Capabilities and

Figure 5 The base framework for constructing a capability model

approaches for determining capabilities within an organization will be discussed later.

The functional capabilities have a close relationship with the service contract description defined within the service model. Further down the abstraction level this relationship translates to a description of the service interfaces and ultimately the service implementations within the technology model. (Note that while there is a strong correlation between a capability and a service, it is not necessarily one to one.)

Service levels. To ensure compatibility and consistency across the models, the notion of service levels is introduced. At the business-model level, SLEs define the expected levels of service. These are business expectations. When you subsequently build a software system to provide the service, the SLE is translated into an SLA. At the technology-model level the SLAs impact how you host the service and how you manage the service.

For example, if your business expectations or SLAs are very demanding, then it would be inappropriate to host your service on a single processor server offering no redundancy. The expectations and agreements drive the availability, reliability, security, and manageability requirements of your solution. Similarly, if the business views the system as mission critical, this status places additional onus on the management capabilities that you put in place for your solution. The way in which service levels are represented within each of the three models is shown in Figure 4.

At the business model perspective the expectation can be provided with both qualitative and quantitative information. The fidelity from qualitative to quantitative information becomes more specific as you move toward the technology model. You effectively lose information as part of your descriptions. However, the benefit of having a set of connected models, particularly if they can be described at the meta level, means that you still have the capability to understand where requirements originate and what is related to what.

Implementation. You need to be able to describe how you are going to implement the models and express them in detail. Within the business model you need to express the externalized business processes that you have identified, and then you need an orchestration mechanism to describe the business processes in the service model. When you then implement the orchestrations you need some form of technology workflow implementation or orchestration engine. For example, on the Microsoft platform, you could use the orchestration services provided by Microsoft BizTalk Server (see Figure 4). Though we may imply automated workflows by talking about *orchestration*, we do not exclude manual or human workflows as a form of *implementation* in the technology model, which becomes obvious during the business model process definition.

Creating Business Models

To create an effective business model, you need to be able to conceptualize the business and identify its core business functions. Doing so enables you to arrive at well-aligned services that map closely to your business requirements.

Capability mapping. This technique was developed and refined by Microsoft to help conceptualize the way a business works and to help create business models. A business capability models what an individual business function does. It is not concerned with how the business function is achieved, but rather with its externally visible behavior and its expected level of performance (that is, its outcomes). A key benefit in focusing on business capability is that while organizational structures and process flows come and go the essential capabilities of businesses tend to remain constant over time. A business capability abstracts and encapsulates the people, process, procedures, and technology associated with a given business function into a simple building block. The decomposition of the business into capabilities provides the top-level decoupling for the underlying service contracts.

Pay supplier, dispatch product, and generate invoice are examples of business capabilities. At a high level of abstraction a business capability is essentially a black box with certain parallels to services. For example, both have connections that are important and related to, but separate from, their inner workings.

Developing capability models. A capability model is a nested hierarchy of business capabilities that enables you to model the business as a structured network of capabilities, as opposed to a physically integrated network. A business capability model is a diagram that describes the network of capabilities used by the business. The framework used to construct a capability model at varying levels of abstraction is shown in Figure 5.

> "MOTION DESCRIBES HOW YOU EXTRACT AND DOCUMENT STRUCTURAL AND PROCESS-ORIENTED INFORMATION ALONGSIDE INDIVIDUAL CAPABILITIES"

The level of abstraction with which you can view a business capability varies. Level 1 capabilities are the foundation capabilities common to most organizations, regardless of industry sector. Levels 2 and 3 (and beyond) provide additional levels of detail on business capabilities. Before examining each level in more detail, note that it is not necessary to decompose all capabilities to the same level of refinement; rather, it's necessary to focus on those most relevant to the problem or area of business that's under consideration.

Level 1 foundation capabilities. Having studied business across many industries, Microsoft has found that at a high level of abstraction businesses within most industries exhibit five core capabilities together

with a set of operational and environmental capabilities, collectively referred to as *foundation capabilities* (see Figure 6).

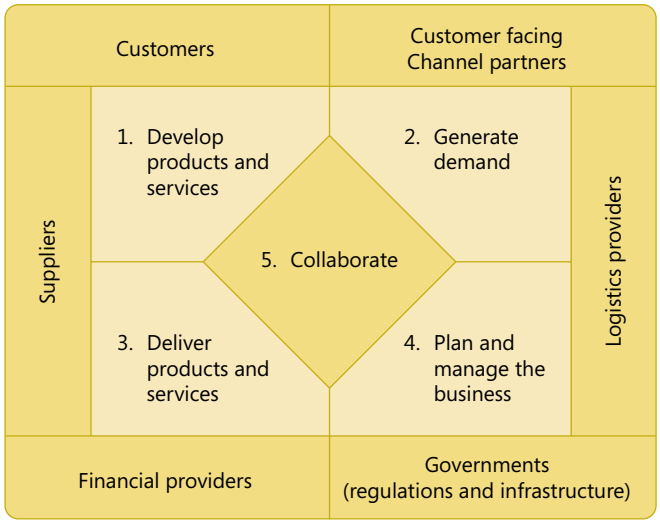
The five foundation capabilities common to most businesses are develop products and services, generate demand, deliver products and services, plan and manage the business, and collaboration. The fifth capability, *collaboration* is the process that orchestrates and coordinates all of the other business capabilities. It is called collaboration because the process can be automated or manual. The process is essentially an orchestration over capabilities.

> "THE FIVE FOUNDATION CAPABILITIES COMMON TO MOST BUSINESSES ARE DEVELOP PRODUCTS AND SERVICES, GENERATE DEMAND, DELIVER PRODUCTS AND SERVICES, PLAN AND MANAGE THE BUSINESS, AND COLLABORATION"

The foundation capabilities also include *operational capabilities*, which are the things inside of the physical business boundary of the organization, and *environmental capabilities*, which are all of the other people and companies who interact with the business that are outside of the physical boundary of the business. These entities might include customers, partners, service providers, and regulatory authorities. They are significant because they all have capabilities that influence the way in which you conduct your business.

Note that the business boundary is not the same as the physical boundary of the corporate entity. For example, if something is outsourced it is still part of your business architecture; although, the work is performed by someone outside of your company. There are some interesting examples of work being moved to the other side of the business boundary such as airport check-in, a function that is now performed frequently by the customer; it is the same capability and has the same SLE. However, there is now different technology and that has allowed different people to perform the work, in this case the customer. This holistic view of the entire ecosystem of capabilities empowers organizations to see a much wider range of options that can inform specific changes they make to their business.

Figure 6 Foundation capabilities common to most businesses



Level 2 capability groups. Capability groups provide the next level of detail in the capability model. Level 2 is where initial analysis begins because it is the level that introduces SLE and impediments and constraints between one capability and another; organizational ownership; and accountability. You also identify the inputs, outputs, supporting functions, and controlling functions.

This example starts with the core capability 3, deliver products and services, and shows a nested hierarchy of capabilities at level 3...n:

3. Deliver products and services
    - 3.1 Provide service
    - 3.2 Advanced planning
    - 3.3 Procurement
        - 3.3.1 Sourcing and supplier contract management
        - 3.3.2 Purchasing
            - 3.3.2.1 Request resources
            - 3.3.2.2 Acquire/purchase resources
            - 3.3.2.3 Manage suppliers
        - 3.3.3 Receiving of indirect/capital goods
    - 3.4 Produce product
    - 3.5 Logistics

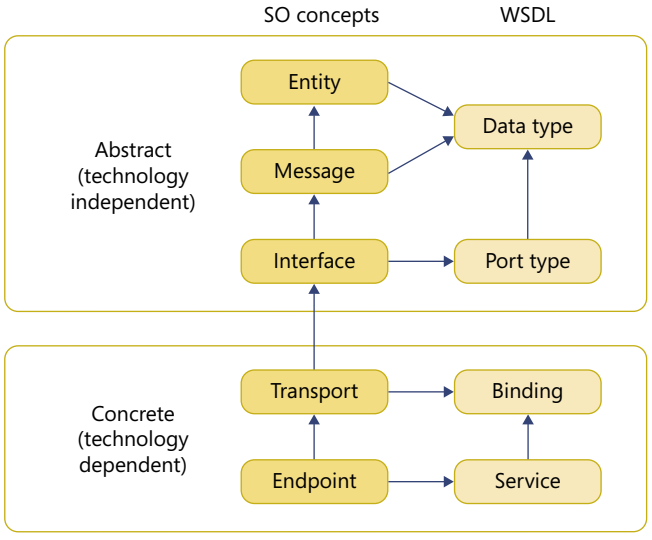
Notice how capabilities are always labeled within their appropriate parent grouping. A capability group is often an important initial level for doing analysis because it is at the capability group level where SLEs, impediments and constraints, organizational ownership, and accountability can first be abstracted and made actionable.

Level 3 business capabilities. Capability groups are subsequently decomposed into *business capabilities*. By mapping individual business capabilities onto a capability model (see Figure 6), you end up developing a nested hierarchy of business capabilities. Capabilities that reside at level 3 and below are the building blocks of the model.

Business capabilities can be decomposed into more granular business capabilities—for example, when more detailed attributes need to be defined. Within the analysis you may decompose some business capabilities to very detailed levels (level 4 and beyond) and aggregate other capabilities at level 3. It is not necessary to decompose all capabilities to the same level. Equally, you do not need to model your entire business. You can pick and choose areas of your business to model based on current business objectives and project requirements.

For each capability that is ultimately identified, you define a set of attributes to describe and document the capability. Some of the key attributes you need to capture for each capability include: Who owns it? Who is its customer? What are the inputs and outputs? What are the required exception-handling and exception-notification mechanisms? What are the performance requirements (past, present, and future)? Are there governance and or compliance implications? Does the performance of the capability impact directly the performance of its parent capability? Its department? The entire organization? What causes the capability to perform the way it does? People? Process? IT? A combination of these causes? And, is

Figure 7 Contract artifacts defined by the service model



the capability part of why customers, partners, or suppliers do business with the organization?

Capturing attributes such as these help you to define SLE at the business level and then SLAs at the service level. This rich description of the capabilities can be passed to development teams who can use the information to help select the appropriate implementation technologies, hosts, and deployment topologies, based on business-driven SLE.

> "BUSINESS CAPABILITIES CAN BE DECOMPOSED INTO MORE GRANULAR BUSINESS CAPABILITIES—FOR EXAMPLE, WHEN MORE DETAILED ATTRIBUTES NEED TO BE DEFINED"

Capabilities to services. One of the key benefits of the capability modeling approach is that it enables you to identify the stable elements of your business to model your architecture around, and it provides a critical layer that closely aligns the definition of services in your connected systems architecture. You can draw boundaries around capabilities—for example, at the group level—and express each capability with contracts. You can then expose these as services and create an orchestration across those capabilities by using the service contracts.

While capability mapping is an effective way for you to analyze the current state of your organization, you can also build a capability map for a desired future state of your organization, and work out how to transform your business to improve its agility. As an example, consider outsourcing. By dividing the capabilities into those that are core for your organization and those that are not, through a process of re-engineering you can decide to outsource non-core capabilities to other organizations. The important thing from an organization's perspective is that the architecture supports business process change and not capability change. The overall business capability remains intact.

Motion Methodology

Microsoft has developed a simple, project-oriented method called *Motion* for exposing your business architecture through building capability maps. Motion is a methodology to organize, measure, and evaluate business capabilities. The four elements of Motion are tools and measures, prescriptive methodology, business capability mapping techniques, and patent-pending model.

The methodology describes *what* the business does in the form of business capability as opposed to *how* the business does it in the form of people, process, and technology. Motion helps expose the busi-

How Does Motion Work?

Microsoft Motion methodology does not encourage you to focus on factoring organizational structure, tools, or processes directly into the capability model. Rather, it provides an effective way to help you arrive at appropriate implementation solutions. The methodology defines these four steps at the top level:

- *Establish the project context*. You start by documenting project objectives, then generate a foundational capability map, and correlate your project objectives with the foundational map capability.
- *Understand the business*. There are two parallel efforts. Capture current business views, which normally involves interaction between the business people and IT people. Capture business architecture detail, which at this stage you start to build your top-level capability map.
- *Complete the "as-is" business architecture*. At this stage you add more detail to create a business architecture.
- *Identify the recommended next steps*. You might be complete at this point, or you might have identified the need to apply process improvement techniques, business process outsourcing analysis, and so on.

The methodology is sometimes referred to as a "phase-gate methodology" because there are distinct criteria that you must be able to satisfy before proceeding to the next phase. It is also important to realize that the methodology encourages iteration at each of these stages. Having completed the stages, you then use the information that you have discovered to construct your subsequent model (see Resources for further information).

ness architecture of an organization and isolate the elements of the business that define the business model and drive performance and differentiation. Business model and business architecture are separable, and Motion helps organizations expose both, letting them better manage change.

For example, consider two companies in the retail industry. They have the same capabilities, but one might make profits from volume

> "THE PRIMARY BENEFIT OF DOCUMENTING CAPABILITIES IN A DETAILED AND CONSISTENT WAY IS IT GIVES YOU A SOLID UNDERSTANDING OF THE SLE THAT THE BUSINESS HAS FOR EACH OF THE BUSINESS CAPABILITIES"

product sales and the other makes all of its money from a membership fee it charges customers. While the industry is the same, both companies have very different business models. Applying retail industry best practices to one will create value in one organization, and destroy it in another. Thus, having a clear and detailed understanding of the business model is very important.

Motion describes how you extract and document structural and process-oriented information alongside individual capabilities. It provides the guidance for performing the modeling activity itself and includes a large number of templates to help you document all of the required information around each capability (see the sidebar, "How Does Motion Work?").

The methodology also provides guidance to show how it complements a number of existing business process improvement frameworks including business process re-engineering, Six Sigma, Lean, and the Zachman framework. Not only does Motion do things these models don't do, it helps with things those other models were never intended to do.

The key to understanding why Motion is different is in understanding capability mapping and how it differs from process mapping. Capability architecture, and not process architecture, is at the core of the Motion methodology. By initially abstracting (even ignoring) processes and analyzing business capability, you can derive an inherently more stable view of your business that is particularly valuable from a versioning standpoint. Capabilities subsequently become the building blocks for processes.

Looking beyond current practices, many of which do not support the rapid rate of change inherent within business today, the Motion methodology was developed as a business architecture methodology that could handle the present and coming challenges of the connected economy.

The Motion methodology defines approximately 80 attributes to help document capabilities. Some of the attributes you should document for each capability were listed previously in the discussion of level 3 business capabilities.

The primary benefit of documenting capabilities in a detailed and consistent way is it gives you a solid understanding of the SLE that the business has for each of the business capabilities. This understanding subsequently allows you to derive SLAs, which in turn helps you to determine the most appropriate implementation technology and deployment topology for the service implementation. Having the full and detailed description of each capability allows your technical teams to implement the solution in the correct way.

Creating Service Models

Given a business model, then, the question now becomes how can you translate that business model into a service model that you can ultimately implement? Before examining an approach that enables you to do this translation, let's consider what you need to define to create a service model (see Table 1). The key contract artifacts that are outputs from service-oriented analysis and design and captured by the service model are shown in Figure 7. Note that only the abstract (technology-independent) artifacts need to be captured for the service model. The underlying transport and endpoint is defined within the technology model.

At an abstract level you need to understand and identify entities, messages, and interfaces. What are the *entities* contained within the data that is passed as messages during the interaction among business capabilities? What are the *messages* that need to flow among the different systems? What are the *interfaces* that business capabilities, and ultimately services, expose?

At the concrete level, the technology model then defines what the underlying transports are and how you should expose the endpoints of a particular capability, which ultimately translates to a service. Within the service model and from a service-definition-language contract perspective, you also need to understand and identify the data types and port types.

To identify and document the preceding items in a service model, you do not need to use radically new analysis techniques. Rather, you can use existing skills such as conventional object-oriented analysis and design (OOAD) skills applied at a different abstraction level to accommodate service orientation rather than object orientation.

With pragmatic, service-oriented analysis and design (SOAD), creating service models does not require completely new methodologies and new approaches. Rather, you can reuse existing skills, and particularly those associated with UML and conventional OOAD. A change of emphasis is required though. You must move away from thinking about RPC-style method call interactions among objects toward message-based interaction among services. By doing so, you can take classic UML-based OOAD and apply it at the business-abstraction level to create your service models. The key difference between SOAD and OOAD is SOAD clearly separates process and entity to decouple services. Decoupled services are more agile and reflect the reality of real business practices.

For example, consider how conventional UML models can be applied to service-oriented analysis and design:

- Use cases including activity diagrams are derived easily from the task and activity descriptions provided by the capability model.
- The collaboration model gives you a good understanding of the process that needs to be enacted between the tasks and activities, which help you to arrive at your orchestration requirements.
- The interaction model or sequence diagram provides information about the underlying message-exchange patterns (synchronous request/response, asynchronous duplex, and so on) that need

to occur among the services. It also helps you to start to derive the externalized canonical domain model, which defines the schema for the data on the wire that goes into the messages. With this information, you can start to define service contracts.

- The component model sketched at an early stage helps focus on operational issues and needs, taking into account the organizational structure and the capability (service) responsibilities and owners. It captures important information for availability, reliability, and scalability from a business perspective and helps to review SLE and SLAs.

By using a pragmatic SOAD approach, you can extract all of the necessary pieces required to build your service model. This extraction includes service contracts, SLAs derived from the SLE defined for each business capability, and the service orchestration requirements. With a detailed service model closely aligned with and derived from the business model, you should now be well placed to map the service model to a technology model that identifies how each service will be implemented, hosted, and deployed.

> "MOTION PROVIDES THE GUIDANCE FOR PERFORMING MODELING ACTIVITY AND INCLUDES A LARGE NUMBER OF TEMPLATES TO HELP DOCUMENT ALL OF THE REQUIRED INFORMATION AROUND EACH CAPABILITY"

The second part of this discussion will be published in the next issue of *The Architecture Journal*, and it will demonstrate how by using the preceding approach and by creating the service model, you can hand over to your IT department data schemas, service contracts, and SLA requirements to inform the definition and ultimately implementation of the technology model.

A New Direction

The old way of thinking about service orientation is not working, and a new way of thinking is required. By adopting this new kind of thinking, as architects we can *force* explicit consideration of service model artifacts in your design process, which helps you to identify the artifacts correctly and at the right level of abstraction to satisfy and align with business needs.

From a modeling perspective, the gap between conventional business and technology models is too large, which is a key contributing factor to the failure of many service-orientation initiatives. We've presented a three-part model with the introduction of a service model in between the business and technology models to promote much closer alignment of your services with the needs of the business. With a detailed service model closely aligned with and derived from the business model, you are well placed to map the service model to a technology model that identifies how each service will be implemented, hosted, and deployed. Capability mapping and the Motion methodology provide an effective way to identify business capabilities and ultimately services. The decomposition of the business into capabilities provides the top-level decoupling for the underlying service contracts, and not the other way around as it usually is today.

Connected systems are instances of the entire three-part model, and they respect the four tenets of service orientation. They can be *more completely* implemented by using the five pillars of Microsoft's platform technologies. Recall that we asked upfront: How do we avoid making the same mistakes with SOAs that previous, hopeful initiatives have resulted in? How do we ensure that the chosen implementation architecture relates to the actual or desired state of the business? How do we ensure a sustainable solution that can react to the dynamically changing nature of the business—in other words, how can we enable and sustain an agile business? How can we migrate to this new model elegantly and at a pace that we can control? And, how can we make this change with good insight into where we can add the greatest value to the business from the outset?

Service orientation with Web services is only the implementation of a particular model. It is the quality and foundation of the model that determines the answers to these questions.

Acknowledgements

The authors would like to thank Ric Merrifield (director, Microsoft Business Architecture Consulting Group), David Ing (independent software architect), Christian Weyer (architect, thinktecture), Andreas Erlacher (architect, Microsoft Austria), and Sam Chenaur (architect, Microsoft Corporation) for providing feedback on early drafts of this article. We would also like to show our appreciation to Alex Mackman (principal technologist, CM Group Ltd.), an excellent researcher and writer who helped us enormously. •

Resources

"Modeling and Messaging for Connected Systems," Arvindra Sehmi and Beat Schwegler
A Web cast of a presentation at Enterprise Architect Summit–Barcelona (FTPOline.com, 2005) www.ftponline.com/channels/arch/reports/easbarc/2005/video/

Obtain a case study on Microsoft Motion methodology by sending a request to motion@microsoft.com.

About the Author

Arvindra Sehmi is head of enterprise architecture in the Microsoft EMEA Developer and Platform Evangelism Group. He focuses on enterprise software engineering best practice adoption throughout the EMEA developer and architect community and leads architecture evangelism in EMEA for the financial services industry. Arvindra is editor emeritus of the *The Architecture Journal*. He holds a Ph.D. in biomedical engineering and a Masters degree in business.

Beat Schwegler is an architect in the Microsoft EMEA Developer and Platform Evangelism Group. He supports and consults to enterprise companies in software architecture and related topics and is a frequent speaker at international events and conferences. He has more than 13 years experience in professional software development and architecture and has been involved in a wide variety of projects, ranging from real-time building control systems and best-selling shrink-wrapped products to large-scale CRM and ERP systems. For the past 4 years, Beat's main focus has been in the area of service orientation and Web services.