Enterprise-Wide Workflow Management

What kind of infrastructure enables the management and execution of enterprise-wide workflows? Despite advances in workflow-management-system (WFMS) technology, no comprehensive answers exist.

Complex, end-to-end business processes, such as building a Boeing 747, are certainly not the first process a workflow-management infrastructure implements within an enterprise. As an example of an ultimate workflow-implementation goal, however, the process for building a Boeing 747 serves as an excellent source of requirements. WFMS architects can subsequently debate whether a requirement’s solution needs to be implemented by WFMSs or the next architecture level (for example, a workflow-management infrastructure that encompasses many WFMSs, integrating them as components).

The following case study and discussion highlight the most significant problems an enterprise faces when executing workflows across a heterogeneous computing infrastructure encompassing many different types of WFMSs.

Case study

This case study introduces different end users’ viewpoints. Lead design engineers participate in parts design; orchestrate fellow engineers’ design efforts; and plan, monitor, and replan their work. To deal with this variety of different tasks, they require user interfaces such as

- an end-user access tool for accessing WFMSs (for example, a worklist for participating in design work by accepting workflow task assignments or starting new workflows),
- a workflow-modeling tool (for defining the work of the engineers they supervise),
- a workflow-instance-management tool (for tracking and adjusting fellow engineers’ ongoing workflows), and
- a project-planning tool (for allocating resources and tracking work in progress).

They also need access to design tools (for doing actual design work), but such tools fall outside this article’s scope.

In addition to lead design engineers, standards engineers establish and consistently advance standard parts. This is a complicated workflow because its components are part of many different subassemblies that, in turn, are parts of assemblies, and so forth. A change in a standard part has a potentially significant impact on other parts or assemblies, so change management needs to be extremely reliable.

Design engineers are asked to use standard parts where possible because they are not only proven technology, but also their delivery is guaranteed in a specific timeframe due to established supplier relation-
Impediments to homogeneous enterprise-wide workflow management

Many technical and nontechnical issues hinder enterprise-wide workflow management. The most significant technical issue is the inability to deal with the heterogeneity among users, workflow types, and WFMSs. Not all users demand the same workflow functionality, so user interfaces of different levels of sophistication are required. Because workflow types cannot always be fully predefined, they often need to be adjusted or extended during execution. Unlike relational database management systems, however, each WFMS often has differing workflow metamodels. This leads to incompatibility between WFMSs, making integration into an environment comprising many heterogeneous WFMSs a troublesome and sometimes impossible task.

An important nontechnical issue is the assumption that enterprises have a homogeneous workflow-management infrastructure—or that workflow heterogeneity exists only between enterprises. This assumption includes the belief that one enterprise uses only one WFMS product (maybe installed several times) to manage its workflows, which is generally not true. Reports indicate workflow heterogeneity within enterprises will increase through domain-specific developments (vertical markets) rather than decrease by normalizing architectural components (that is, using one WFMS for all domain-specific application software). WFMS vendors have started seriously focusing on vertical-market-oriented development.1,2 If an enterprise deploys domain-specific solutions based on different WFMSs, a heterogeneous set of WFMSs is slipped into the computing infrastructure without notice.

Another nontechnical issue is that many nonworkflow vendors decide to embed workflow functionality into their products rather than bundling (reusing) existing standalone WFMSs. Embedded workflow functionality cannot be isolated easily, if at all. This adds further to workflow heterogeneity, making WFMS integration an even harder task.1

References

Workflow management

Several WFMSs are involved in executing the discussed workflows. Some might be very specialized in the type of data they can handle or they might handle a single database. However, these WFMSs are not run in isolation; they must exchange information and share workflow execution (such as when the human resources workflows require all employees throughout the company to be contacted).

Several kinds of workflow types can be observed. Very predefined workflow types don’t change too often. This workflow executes over and over, needing adjustment in its definition only if corporate policies change. Workflow types are built according to specific needs and can be reused over time for similar work with no, or few, adjustments. Some of these might execute only once. However, this does not mean they are simple in structure: they might encompass many design steps and engineers.

When managing running workflows, a lead engineer might have to adjust a workflow by adding, removing, or reordering tasks. This requires highly adaptable workflow functionality—a workflow metamodel must be expressive and flexible, so all workflows can be defined and managed easily.

Whereas engineers use WFMSs to accomplish work, managers need to be able to observe the status of ongoing work. They do this by requesting status information from WFMSs. A typical question is, “Which workflows are related to part XYZ and to what extent are these workflows already executed?” A related question is, “Given the current state of the workflows related to part XYZ, how long will it take, according to the plan, to complete them?” These questions can be answered easily when applied to the travel-management workflow (where part XYZ is a form) and where the workflow resides within one WFMS. However, if part XYZ refers to a complete airplane, the questions become very difficult to answer because they not only involve the querying of all heterogeneous WFMSs, but also require a homogeneous presentation of the state so that information is meaningful across the whole set of running workflows (see the “Impediments to homogeneous enterprise-wide workflow management” sidebar). An office overseeing an entire company asks the same question, but for all products at the same time.

The representation in Figure 1 shows...
the functional domains, their WFMS type, and the relationships in terms of workflow execution. Each WFMS type is from a particular vendor and is distinguished from one or several installations of it. According to the case study, the relationships are fully developed (that is, every functional domain is in some kind of workflow execution relationship with every other functional domain).

**Heterogeneous workflow types and execution**

For the sake of categorizing the requirements, I discuss only the most important workflow functionality (based on the case study). Workflow distribution introduces another level of requirements. Because distributed workflow execution across heterogeneous WFMSs is currently not possible in a transparent way, we must examine the problem of workflow functionality isolation.

**Workflow functionality**

The case study analysis produces an interesting mix of required workflow functionalities across different functional domains. The following workflow perspectives serve as an excellent means for requirements and for WFMS capability classification. At least five different perspectives must be described when defining a workflow type. Figure 2 shows a definition for an oversimplified human resources workflow for gathering education information.

After a new employee joins the company, the education-history workflow, which is a subworkflow of the new-hire workflow (not shown), collects his education information. The new hire’s unique identifier is stored in a new-hire data item on invocation by the new-hire workflow. According to the workflow definition, the WFMS assigns the first workflow task to the new hire as specified in the new-hire data item. This task is elementary because only one employee (resource) performs it. The new hire invokes the HR application to enter his education history. The next task is assigned to an education specialist, who verifies the information using the HR application program and enters the status data item. The third workflow task is assigned to the human resources specialist, who verifies the education history again and returns the status, possibly overwriting the education specialist’s result. If the education history information cannot be verified, the WFMS starts an unstructured workflow as a subworkflow for contacting internal and external sources, based on the value of the status data item. The subworkflow’s particulars are not further specified because it’s a freestyle workflow, coordinated by the HR specialist, that’s defined as it’s executed.

**Functional decomposition**

Functional decomposition by means of workflow tasks or subworkflows, but also replacing elementary workflow tasks with other composite workflow definitions. This last situation exists if a workflow participant replaces an assigned workflow task with a complex workflow.

**Data and data flow**

Throughout an enterprise, workflow execution involves several types of data to pass on information to users and to make control-flow decisions. The latter requires data-mapping between the (WFMS-external) data-management system and the WFMS-internal data representation, if they are used for internal decisions. This means the WFMS-internal data flow’s data-type system must be able to accommodate data types mapped to it. Usually this requires data transformations between the involved data-type systems to map the data back and forth between the WFMS and the data-management systems. Between workflow tasks, data transformation might be necessary, too, if the applications involved in task execution require different data formats. WFMS-internal data should not be globally available to all workflow tasks. A more directed data flow is necessary to specify which data is produced or consumed by which workflow task. Therefore, data flow is both directed and dedicated. Dynamic changes of data and data flow are necessary when additional data need to be attached to a work-

---

**Figure 1.** Workflow management systems, users, and workflows in a complex enterprise. The circles indicate the type of WFMS used and the arrows show communication relationships.
flow not previously defined by the workflow definition. Sometimes data flow requires redirecting, too, when an initial assumption about data consumption turns out to be wrong.

**Control flow**

Usually, control-flow constructs (such as sequence, parallel execution, loops, and conditional branching) are sufficient to define workflows. However, in many cases these constructs prevent appropriate control-flow definition—for example, three subworkflows can be executed in any order, but only one at a time, using the above control-flow constructs. To accomplish more complex constructs, WFMSs need to allow for their definition. In other cases, it is not possible (and not necessary) to specify control flow at all. The workflow participants define the control flow indirectly, by accepting assigned tasks according to their needs, and the WFMS assigns all subworkflows, according to the workflow definition, as soon as the workflow starts. A combination is possible, too: some workflow tasks are related by control flow, others are not. For example, after designers complete creative design tasks, a design must be signed-off. The latter part is structured whereas the former usually is not. Dynamic changes are also sometimes necessary. For example, the workflow owner or administrator needs to change the execution order due to data availability.

**Organization assignment**

Role-based assignment of workflow tasks is sufficient for many workflows. Roles usually reference a set of employees. As soon as a WFMS assigns a workflow task to a role, that assignment appears in all referenced employees’ worklists. However, role-based assignment is no longer sufficient when workflows become more complex. In the case of the education-history workflow, the HR specialist must be from the new hire’s organization. This is an assignment based on organizational relationships, such as an organization-membership criterion. Dynamic changes are also sometimes necessary, because, for example, managers sometimes need to reassign workflow tasks to balance work, or to accommodate unavailability due to sick leave. Sometimes assignment rules can’t be defined. In such cases, a manual assignment must be possible as well.

**Application integration**

Application integration can be very easy when application programs are built for integration and when the WFMS has a rich functionality for integrating applications. However, application integration can be very difficult when applications are not made for integration. For example, when an application expects input data in files and writes results to files, the WFMS must write and read these files in addition to calling the application. This requires the invocation of several pieces of software, in addition to the application itself. A rich set of APIs makes it possible for the applications and WFMSs to pass data back and forth.

Table 1 highlights the main issues of the different workflow perspectives and adds a short explanation for each entry.

WFMS-provided constructs for defining workflow types must be very expressive to address a large and heterogeneous enterprise’s workflow definitions; flexibility in terms of dynamic changes is key. To avoid being stuck in the WFMS, dynamic changes should not lead to inconsistencies from the workflow-execution viewpoint. However, from the application viewpoint, domain-semantics changes could cause inconsistencies. For example, a signature task’s deletion can cause company-policy violation. Ideally, we could define which changes are correct and don’t cause a violation when applied.

**Workflow distribution and heterogeneity**

Distributed workflow execution in a large and heterogeneous enterprise adds more dimensions to the workflow functionality than are discussed within the above workflow perspectives. Basically, different functional domains have their own autonomously managed WFMSs. This means functional domains develop their own workflow definitions and execute them in isolation. However, as soon as cross-domain workflows have to be put in place, functional domains must integrate workflows.

**Direct and indirect distribution**

If the involved WFMSs know about each other and invoke each other’s functionality, this is categorized as direct distribution. If the involved WFMSs do not know about each other, it’s indirect distribution. In the former case, the WFMSs natively implement some form of distribution to achieve distributed workflow execution. In the latter case, the
WFMSs do not implement distribution natively, and system architects must attach distribution functionality to the involved WFMSs. One way is to establish communication buffers between the WFMSs, such as a database or persistent queues. Because the involved WFMSs do not know about indirect distribution, programs run by applications in workflow tasks must update the buffers. Agents monitor buffers and invoke, for example, the corresponding WFMS to start a workflow (see Figure 3). Workflow definitions must contain tasks to manipulate the buffers; agents that monitor buffers and invoke workflows appropriately must be in place.

Homogeneous and heterogeneous distribution
Workflow distribution is called homogeneous if the involved WFMSs are of the same type, heterogeneous otherwise. Homogeneous distribution assures all involved WFMSs support the same metamodel. In addition to invoking each other through interfaces, the WFMSs can exchange workflow definitions easily because no translation between different workflow metamodel concepts is necessary. If the involved WFMSs are heterogeneous, they need a least-common de-

<table>
<thead>
<tr>
<th>WORKFLOW PERSPECTIVE</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional decomposition</td>
<td>Defined and cannot be changed during workflow execution.</td>
</tr>
<tr>
<td>Static decomposition</td>
<td></td>
</tr>
<tr>
<td>Dynamic changes of decomposition</td>
<td>Allows functional decomposition to change during workflow execution.</td>
</tr>
<tr>
<td>Workflow definition reuse</td>
<td>Enables use of an existing workflow definition without copying it (dynamic binding).</td>
</tr>
<tr>
<td>Version/variant/configuration</td>
<td>Ensures changes are only propagated when necessary.</td>
</tr>
<tr>
<td>Data and data flow</td>
<td></td>
</tr>
<tr>
<td>WFMS-internal data-type system</td>
<td>Supports definition of data types internal to a workflow definition.</td>
</tr>
<tr>
<td>Directed and dedicated data flow</td>
<td>Ensures produced data are delivered to a consuming workflow task.</td>
</tr>
<tr>
<td>Data transformation between WFMS and external data management system</td>
<td>Data types defined externally must be mapped to the workflow-internal data-type system by data transformation rules.</td>
</tr>
<tr>
<td>Data transformation between workflow tasks</td>
<td>If different tasks need different formats, data-type transformation is required.</td>
</tr>
<tr>
<td>Dynamic changes of data and data flow</td>
<td>Workflow end users, owners, or administrators can add or delete data during workflow execution.</td>
</tr>
<tr>
<td>Control flow</td>
<td></td>
</tr>
<tr>
<td>Extensible set of constructs</td>
<td>Workflow modelers can add new control-flow constructs when existing ones aren’t sufficient.</td>
</tr>
<tr>
<td>No definition</td>
<td>If no control flow should be enforced, no control-flow constructs are necessary.</td>
</tr>
<tr>
<td>Partial definition</td>
<td>Control-flow tasks need to connect workflow tasks only if there are dependencies.</td>
</tr>
<tr>
<td>Dynamic change of definition</td>
<td>Workflow end users, owners, or administrators can accommodate exceptional situations by adjusting control flow during workflow execution.</td>
</tr>
<tr>
<td>Organization assignment</td>
<td></td>
</tr>
<tr>
<td>Criteria-based</td>
<td>Role concepts often need enhancement by criteria-based assignment.</td>
</tr>
<tr>
<td>Manual</td>
<td>Necessary when no assignment rules can be defined.</td>
</tr>
<tr>
<td>Application integration</td>
<td></td>
</tr>
<tr>
<td>Invocation of several application programs within one workflow task</td>
<td>The WFMS, or software integrating applications into it, must execute supporting programs before (and sometimes after) applications.</td>
</tr>
<tr>
<td>Data flow between application programs and workflow tasks</td>
<td>Applications and WFMSs must pass data required or produced by those applications back and forth.</td>
</tr>
</tbody>
</table>

Figure 3. Indirect workflow distribution. The WFMSs both invoke an application to trigger remote workflow execution in each other by feeding a buffer that is examined by an agent of the remote WFMS. As soon as the agent detects an entry in the queue to which it is listening, it starts a workflow according to the queue’s entry.
Workflow types can be distributed, too. An application for buffer communication.

Distribution task. The distribution task invokes the better form. Figure 4b shows an example of distribution transparency in a workflow across several installments. If the WFMSs execute a workflow across several installations, each involved object might be subject to distribution. An often-cited situation is workflow-object distribution, where subworkflows are subject to execution on remote WFMSs. Different variants are possible, such as executing a subworkflow synchronously or asynchronously to the invoking workflow. Another variant involves executing some part of a workflow on one WFMS, and continuing on another (see Figure 4a). However, a WFMS can distribute other workflow objects as well. For example, work items can be distributed to other WFMSs, or to the data to which a workflow refers.

In this case, different WFMSs store the objects to which a workflow type refers. For example, workflow definitions for subworkflows can be remote or data-type definitions.

To achieve distribution, a WFMS modeler, a configuration, or the WFMS must place the involved objects. This can be done statically (objects stay where placed) or dynamically (objects change their location) in a variety of ways, such as the migration of objects, their replication across several WFMSs, or the copying of objects.

### Distribution transparency

Workflow definition or execution is distribution-transparent on a technical level if location specification is not visible in a workflow definition. For example, if a workflow definition stating subworkflow Fill should run on node Sammamish is not distribution-transparent because the definition states a particular subworkflow must run on a particular node.

A workflow definition is not distribution-transparent on a workflow-logic level if the workflow metamodel requires location specification. For example, if the workflow covering design and standards engineers states which subworkflow must be executed in which WFMS, distribution transparency no longer exists. Distribution transparency is the best form of distribution because workflow definition and execution are presented homogeneously to end users, workflow modelers, and other roles. Any type of heterogeneity, any issues regarding object distribution or indirect distribution, are hidden from WFMS users. Current technology is far from achieving distribution transparency when more than one WFMS type is involved. Even in cases where only one WFMS type is involved, distribution transparency is not guaranteed automatically. Most WFMSs do not recognize the existence of multiple installations, but run in isolation.

In contrast, the worst form of distribution is indirect distribution across heterogeneous WFMSs. In this case the involved WFMSs do not recognize each other and have different workflow metamodels. Users must add any form of distribution to these WFMSs as external components, because the WFMSs do not provide distribution functionality. This means users must be aware of the situation, especially those who define workflows, because they have to take heterogeneity into consideration while doing so. Workflow modeling tools do not expose modeling constructs for distribution.

### Workflow functionality isolation

It is unrealistic to assume distribution transparency across heterogeneous WFMSs will become the standard functionality in the near future. Different types of WFMSs are heterogeneous because their workflow metamodels and execution semantics cannot be mapped to each other easily, if at all. This causes workflow functionality isolation, which occurs when different functional domains define and execute their local workflows. If a distributed workflow execution needs to take place, system architects must make a special effort to build distribution functionality between the involved WFMSs. Most likely this will result in a subworkflow distribution as shown in Figure 4a. The distribution is not transparent, and workflow users and modelers are aware of it.

Having different WFMSs for different functional domains works fine as long as the functionality supported by the WFMSs fits the functional domains' requirements. If a requirement the local WFMS cannot support comes up, the problem of how to support it arises. For example, should the need to modify run-
ning workflow instances arise in a local WFMS that doesn’t support such a function, limited solutions exist. The functional domain can either ask the vendor to upgrade the WFMSs and add the required functionality, try to build the required functionality by adding a WFMS-external component, or install a WFMS that provides the functionality. All three solutions are costly, far from ideal, and take a lot of time.

With transparent distribution,

- a WFMS that supports dynamic modification could execute the workflow,
- the end user wouldn’t have to deal directly with another environment, and
- the enterprise wouldn’t have to add functionality.

Without transparent workflow distribution, each type of WFMS’s workflow functionality is isolated from other types of WFMSs. With transparent distribution, the isolation would disappear and the individual WFMS’s functionality would be available to every workflow end user.

**Heterogeneous WFMSs**

The case study makes clear that WFMSs need to interoperate in terms of workflow execution. Each functional domain has its own WFMS in place supporting domain-specific workflows, such as human resources workflows and design engineering workflows. A worse case would be several installations of multiple types of WFMSs within one functional domain: workflow heterogeneity. This happens when a functional domain buys vertical products from different vendors, each implementing its own workflow functionality. As a result, heterogeneous WFMSs are deployed throughout the enterprise.

**Basic integration alternatives**

Figure 5 illustrates the following basic WFMS integration alternatives that will serve as a frame of reference for discussing some of the requirements from the case study. I will discuss each alternative briefly in terms of its technical implications and then apply the alternatives to the case study. This will show that workflow metamodels are key to achieving enterprise-wide workflow execution and extreme openness of workflow architectures.

**User-interface collocation**

Different WFMS’s end-user interfaces are collocated at a user’s desktop. Depending on which WFMS users want to access, they invoke the appropriate interface, as shown in Figure 5a. Whether the interfaces are collocated as installed clients or are different Web pages or Java objects in browsers does not matter. The criterion is that one WFMS’s interfaces are unaware of the other’s.

**User-interface integration**

This type of integration uses the same set of user interfaces to access different WFMSs (see Figure 5b). Interfaces accessing two different types of WFMSs don’t necessarily transfer objects between them. The interfaces separate the involved systems’ objects (separated object sets). One example is a worklist displaying assigned tasks from two WFMSs. Accepting a task from one WFMS doesn’t affect the other.

**User-interface integration**

This type of integration uses the same set of user interfaces to access different WFMSs (see Figure 5b). Interfaces accessing two different types of WFMSs don’t necessarily transfer objects between them. The interfaces separate the involved systems’ objects (separated object sets). One example is a worklist displaying assigned tasks from two WFMSs. Accepting a task from one WFMS doesn’t affect the other.

**Execution-logic integration**

This type of integration uses the same set of user interfaces to access different WFMSs (see Figure 5c). Interfaces accessing two different types of WFMSs don’t necessarily transfer objects between them. The interfaces separate the involved systems’ objects (separated object sets). One example is a modeling tool capturing the activities in one WFMS, doing something with the captured objects, and then storing the results in the other WFMS.

**Database integration**

This type of integration uses the same set of user interfaces to access different WFMSs (see Figure 5d). Interfaces accessing two different types of WFMSs don’t necessarily transfer objects between them. The interfaces separate the involved systems’ objects (separated object sets). One example is a modeling tool capturing the activities in one WFMS, doing something with the captured objects, and then storing the results in the other WFMS.
workflow definition. This new workflow definition contains objects from both WFMSs and, consequently, one WFMS knows about the other’s objects.

**Execution-logic integration**
In this type of integration the WFMSs know about each other’s existence. They contact themselves directly to invoke the other systems’ functionality or query status information. Figure 5c shows the scenario.

The execution-logic parts of both WFMS are connected. Note that no mediator or other component exists between the two WFMSs. This indicates they know each other directly as WFMSs. The invocation of workflow instances on a remote WFMS is a possible interaction on this level. Another variant of this architecture is the additional integration of the interfaces as shown in Figure 5b.

**Database integration**
The fourth type of integration involves several WFMSs sharing one database (Figure 5d).

Analogous to the interface integration, the two involved WFMSs’ either recognize each other’s object sets or strictly separate them within the database. WFMSs cooperation is only possible in the latter case, because they’re sharing objects. Variants of the architecture shown in Figure 5d include the WFMSs integrating on an additional execution-logic level, or having integrated end-user interfaces, as discussed previously.

If two WFMSs disagree on their definition of a role, a tight integration is possible only if a way to transform one WFMS’s understanding of role into that of the other’s exists.

If system architects don’t build the involved WFMSs with integration in mind, their only possibility is to build a mediation environment that translates between the involved WFMS. Even if they install one WFMS type (from one vendor) several times, no true integration, in the sense that the two installations recognize each other, is guaranteed. If the installations are independent in the objects they manage and if they cannot reference objects from each other, interoperation isn’t natively possible, and a mediation infrastructure must be built.

Also, the fact that WFMSs can invoke each other by means of a logic integration doesn’t mean workflows crossing WFMS boundaries can be modeled as one workflow within one modeling tool. The interfaces might deal with workflow instances only and not with workflow definition information.

**Architecture selection for heterogeneous enterprises**
The case study indicates various forms of workflow execution distributed among functional domains. Revisiting these in the context of the WFMS-integration alternatives I discussed earlier will indicate the best alternative for satisfying all the requirements for distributed workflow execution.

**Design engineering work**
Lead engineers and their coworkers operate in context of one WFMS type. Any of the previously mentioned integration alternatives is fine if one instal-
lation is enough to handle all their workflows, but not if several installations are required. If workflow instances are executed across WFMS installations because design engineers managed by different installations are involved, execution-logic or database integration is required. Otherwise the WFMS installations cannot share the executing workflow instances. Additionally, integrated user interfaces are necessary, because without them design engineers must log into all the WFMS installations separately to look for their workflow task assignments.

**Design and standards engineering collaboration**

Design and standards engineers work together to find and modify standard parts by means of common workflows. They share data in their local installations and transfer data, such as the specification for a standard parts modification, back and forth between their installations. They require execution-logic integration to execute their workflows across heterogeneous WFMSs. Database integration would work, too, but cannot be expected due to the involved WFMSs’ heterogeneity.

**Human resources work**

These workflows require that end users populate corporate databases. From a workflow viewpoint, this means that after accepting a work item, an end user must invoke an application system that accesses a human resources database. A user-interface integration can achieve this. Execution-logic or database integration would also work.

**Travel management**

The travel-management workflow is very similar to human-resources workflows if end users populate corporate databases. However, if users move documents (such as spreadsheets) around from desktop to desktop, the WFMS must support document moving. A user-interface integration can still accomplish this if the client is capable of dealing with documents.

**Status reporting**

If one WFMS installation generates status or history reports about workflows, any type of integration is sufficient. If a status report spans multiple heterogeneous WFMSs, a user-interface integration with integrated object sets is necessary. Status reporting across heterogeneous WFMSs requires that the involved systems share a common interpretation about workflow status. This might require execution-logic or database integration if integrated user interfaces can’t translate between diverse metamodels.

The scenarios clearly indicate that status reporting requires integrated user interfaces and execution logic. This means that WFMSs must share workflow metamodels to allow distributed workflow execution. Until WFMSs support this functionality natively, an alternative solution must be found.

**Workflow type and instance placement**

The case study indicates every functional domain has its own WFMS installation, but, technically, this doesn’t have to be the case. One WFMS can manage two functional domains’ workflows, as long as it is able to satisfy both domains’ requirements simultaneously. For example, travel management and human resources could share a document- and application-integration-supporting WFMS. In this scenario, the workflow types and instances are within the same WFMS.

Alternatively, all lead engineers and their fellow engineers, including standards engineers, have access to their functional domains’ WFMSs, so instead of having a WFMS for travel management, the engineers can have the travel-management workflow installed in their WFMSs. This would eliminate the travel-management WFMS and would use the existing infrastructure; the workflow types would be replicated across existing WFMSs. However, in the absence of transparent workflow distribution, the workflow-type-management problem of translating the travel-management workflow into all WFMSs arises. This approach is still advantageous over the installation of another WFMS, because of deployment and maintenance costs.

The more workflow metamodels integrate, the better enterprises can manage their workflows in an environment with heterogeneous WFMSs.

**Workflow management infrastructure**

Distributed workflow execution across functional domains is necessary, but distribution transparency is currently impossible because, as I discussed, different types of WFMSs implement different WFMS metamodels. Further-
more, WFMSs generally don’t integrate across heterogeneous WFMSs on a logic or database level, and consequently don’t support distributed workflow execution.

One way to enable distributed workflow execution is to build a workflow-management infrastructure encapsulating different and heterogeneous WFMSs. End users would have access to combined functionality because they access the workflow-management infrastructure, not individual WFMSs. The architecture is general and can accommodate as many WFMSs as necessary.

In addition, a workflow-management infrastructure can extend the existing WFMSs’ functionality by adding more workflow functionality through additional custom code. Furthermore, if WFMSs support their extension natively (for example, directory service access), a workflow-management infrastructure can ensure all WFMSs share common services, such as directory services. If they cannot integrate common services, a workflow-management infrastructure can attempt to feed all WFMSs with the same information (by translating according to the WFMSs’ workflow metamodels).

Basically, a workflow-management infrastructure attempts to present a more homogeneous view to users than what a set of WFMSs can offer natively. Reaching transparent distribution is almost impossible, but reaching some level of homogeneity is realistic.

**BRIDGING HETEROGENEITY**

If different WFMSs don’t recognize each other, one possibility is installing distribution tasks into each one. A distribution task is a workflow task within a workflow definition. However, its purpose is not to be assigned to a user, but to invoke a workflow at a remote WFMS. This distribution task finishes itself once it receives the remote workflow’s results. This form of distribution task is synchronous because it waits for these results.

Another form is the asynchronous distribution task. This task starts a remote workflow, but doesn’t wait for the results to come back. From the time of invocation, the remote workflow runs independently, parallel to the initiating workflow.

If the initiating workflow needs to synchronize later, it must contact the remote WFMS.

Distribution tasks don’t contact remote WFMSs directly, but through a (workflow execution) mediator. This mediator acts as a switchboard and maintains the invocation’s state. Figure 6 shows this in further detail.

The distribution task must be multilingual because every WFMS implements a different set of APIs, and, potentially, any workflow can invoke any other WFMSs’ workflows.

The distribution task invokes workflows on remote WFMSs as subworkflows, but other objects can be distributed as well. One example is when a user’s worklist contacts several WFMSs and displays assigned tasks. In this case, the distribution task distributes the different WFMSs’ work items across several worklists (see Figure 6).

All these different forms of bridging the heterogeneity of WFMSs depend on the availability of the WFMS-implemented APIs. In general, the richer the API set, the easier it is to achieve integration. However, some WFMSs do not expose any API at all; they’re closed in the sense that they cannot be invoked from an external entity. Integration in such a scenario is very difficult, if not impossible.

A higher integration level is possible with a workflow-management infrastructure when the involved WFMSs’ workflow metamodels can be mapped to each other. In this case, the workflow-management infrastructure could either maintain the workflow definitions (feeding them to the WFMSs) or translate workflow definitions from one WFMS to the other. The most appropriate strategy is to support both alternatives; with a workflow-management infrastructure that can feed or translate workflow definitions, the functional domains can manage the individual WFMSs independently, before workflow-definition distribution is envisioned (see Figure 6).

The level of integration and heterogeneity supported by a workflow-management infrastructure depends on the underlying WFMSs. The best integration is possible if they are open—that is, if the WFMSs provide APIs that allow access to all their functionality.

**NORMALIZATION BY COMPONENTIZATION**

WFMSs currently satisfy many requirements, but fall short in areas equally important for building and maintaining reliable enterprise-wide workflow-management infrastructures. From an architectural viewpoint, some of the required functionality might not belong to WFMSs at all, but must be provided by additional components that interoperate with WFMSs. These additional components are as much a part of the workflow-management infrastructure as the WFMSs. Many WFMSs can share these components. In this sense, an architectural normalization takes place: a particular functionality is implemented once and reused by all other components. However, the integration of other components depends on a WFMS’s ability to invoke them. One way would be to define, within a WFMS, which external components it needs to invoke to determine, for example, the set of end users for a task.

Another, very difficult way would be...
to install a workflow task that invokes the external component and then updates the WFMS accordingly. For example, a workflow task invokes the organizational database and derives a set of users. The workflow task then calls the WFMS’s API to set the assignment for the next workflow task, which must be added and executed whenever someone needs to contact an external database to find out a set of users (see Figure 7).

Functional Extension

The workflow-management infrastructure can be termed as an extension architecture in the sense that functionality can be added to the out-of-the-box WFMS functionality. The added, or extended, functionality implements the unfulfilled set of requirements the WFMSs do not address natively.

For example, users can preview assigned workflow tasks. This means they invoke the assigned workflow task without accepting it for work. They can look at the task to determine whether they are the right ones for performing it. If a WFMS does not provide this type of preview, the workflow-management infrastructure can add it (see Figure 6). This still depends on the available API set; if it doesn’t allow such an extension, the workflow-management infrastructure cannot implement the additional functionality.

Workflow standards

Two groups working on workflow-management standardization are the Workflow Management Coalition (WfMC), http://www.wfmc.org, and the Object Management Group, http://www.omg.org. Both efforts focus on interface standards—that is, the definition of interfaces for various purposes—without defining how the interfaces are implemented. In addition, the WfMC has published a workflow-definition language called WPDL (Workflow Process Definition Language). Neither effort attempts to standardize a workflow metamodel that would enable compliant WFMSs to exchange workflow definitions or parts thereof. Consequently, WFMSs can be compliant to the interface standards without sharing a workflow metamodel. However, WFMSs pass data values (for example, a workflow instance’s state) through standardized interfaces. The sending and receiving WFMSs interpret these data values in the context of their own workflow metamodels. A semantically equivalent interpretation can only come about if the involved WFMSs agree on each other’s interpretation (or the workflow metamodel part of which the data value is an instantiation). Once this agreement is in place, they can map the data values to their respective workflow metamodels. With no workflow-metamodel standard, the data-value interpretation is left to the different WFMSs, and there is no guarantee the interpretations will be equivalent across all WFMSs.

One interesting example is the workflow state done. In some WFMSs, this state means the workflow instance still exists but cannot be changed, nor does its state change (analogous to constants in programming languages). In other WFMSs, the workflow instance is deleted from the WFMS as soon as it reaches done (basically, it no longer exists). In other WFMSs, the workflow instance doesn’t change its state by itself once it is in done, but it can be asked to redo portions of its subworkflows. In this case, the subworkflows are executed again and the workflow is set back to a different state, such as running. The interpretation of done depends on which workflow metamodel the WFMS implements.

Publishing a workflow-definition language without defining the underlying workflow metamodel causes the language to be semantically undefined yet syntactically defined. The WPDL document states: “Conformance for process definition import/export is essentially based on conformance to the WPDL grammar and/or the API conformance class with the WAPI (Workflow API) definition.” Figure 8 shows the situation.

If this workflow is fed into two different WFMSs, its interpretation might differ in the following way. Assume data item x’s value is 5 in a particular instance of this workflow type. The first WFMS might execute A, then check B’s precondition. Because it determines that x’s value doesn’t meet the task precondition for x, B doesn’t get instantiated and subsequently C will never be executed. The other WFMS might have a different interpretation. It might execute A, instantiate B, then evaluate the precondition. Because the precondition is not met, it changes the B’s state to done, and subsequently instantiates C.

Because these two interpretations are different from one another, a translator needs to be built that will equalize them to ensure the same behavior in both WFMSs. This translator is simply the establishment of a common workflow metamodel across WFMSs. However, it is unclear whether a translator that can translate workflow definitions from every WFMS type into every other WFMS type—thus ensuring common semantics across all systems—can be built.

It’s clear the current state of workflow standardization doesn’t come close to replacing a workflow-management infrastructure, even though it’s desirable for large and heterogeneous enterprises to have the WFMSs solve their workflow-management integration problems.

Executing Workflows

Across WFMSs is no easy task, due to the latter’s current heterogeneity. Enterprises requiring cross-WFMS execution need to invest in integrating their WFMSs. Only then can they achieve some level of integration, although it still would be far from ideal.

Until workflow standards and WFMSs make integration easier, if not transparent, enterprises must find other ways to deal with the situation. One option is to reduce the variety of WFMSs inside an enterprise, although this generally is very difficult when an enterprise deploys many
vertical-market applications. Another option is to build a workflow-management infrastructure that integrates existing WFMSs across the enterprise, and demands that applications requiring workflow functionality use the infrastructure rather than implementing their own workflow functionality within it.

It’s also possible to try the opposite approach: only buying and integrating vertical solutions. This means that the enterprise doesn’t deploy a workflow infrastructure—all workflow functionality is embedded in vertical applications. The ability to execute workflows enterprise-wide across these applications depends on whether they are designed to integrate with each other (by supporting an extensive set of APIs on a workflow level), enterprise-wide workflow management would involve solving the integration issue between the different vertical applications.

From a software-architecture standpoint, if enterprise-wide workflow management is important, having a common workflow infrastructure as a foundation for vertical applications is preferable.

References

Christoph Bussler is an Advanced Computing Technologist at The Boeing Company’s Phantom Works—Mathematics and Computing Technology research group based in Seattle. As project manager of the Workflow Management project, he is responsible for conducting workflow research, transferring workflow technology into The Boeing Company, applying workflow technology in workflow projects, and representing The Boeing Company in workflow standardization efforts. He earned his workflow management PhD at the University of Erlangen-Nuremberg, Germany. He is a member of the IEEE Computer Society and the ACM. Contact him at The Boeing Company, Phantom Works—Mathematics and Computing Technology, POB 3707, m/s 7L-70, Seattle, WA 98124-2207; christoph.bussler@pss.boeing.com.