

Reconfiguring Workflows of Web Services

R. Baird M. Hepner R. Gamble T. Gamble
Department of Mathematical & Computer Science
University of Tulsa
Tulsa, OK 74104
gamble@utulsa.edu

Abstract

Workflow reconfiguration traditionally involves modifying workflow specifications to adapt to changing architectural conditions. Causes include the introduction of new services or the alteration of goals. Dynamic reconfiguration is currently achieved in workflow specifications employing Web services using techniques that modify endpoint bindings and control structures. Abstract specification of service endpoints delay the point at which the endpoint for a Web service is bound, and modifications to control structures can allow for a variety of complex workflows to be specified. These approaches work with limited changes. This paper defines an improved process of dynamically reconfiguring Web service workflows. Our double loop approach utilizes companion meta-data specifications and reconfiguration plans that are associated with workflow specifications. The approach maps external change requests to workflow actions to determine an appropriate reconfiguration plan without changing the workflow language. Dynamic reconfiguration concepts are reused from architecture reconfiguration research to offer a wider range of change.

1. Introduction

Web services (WS) expose business functionality using common, well-established communication methods. Frequently, component interoperability is increased by creating a WS interface for a component, enabling a unified communication method. This unified communication helps system developers compose component interactions that deliver complex business processing. A workflow automates a business process

during which documents, information, or tasks are communicated among participants according to a set of procedural rules [1]. Workflows can be defined using a variety of commonly available languages such as the Business Process Execution Language (BPEL) which allows for the definitions of workflows specific to WS. WS are more dynamic in nature than typical components since they employ a standard (e.g., UDDI) to remove location dependency and add the ability to search and select a service from a set of competing but functionally equivalent services.

A Community of Interest (COI) is a group in which the majority of communication remains within the group and for which identity, functionality, and interaction can be predetermined [2]. COI members may be related by a geographic area, an administrative domain, or common goals. It is important to note that community specific policies drive routine tasks that accomplish COI goals. Assuming relatively simple construction, execution, and modification, workflows are well-suited to perform these routine tasks.

Dynamic changes may be required due to changing service priorities, a drop in service reliability, or a service's new location. These dynamic changes may be executed on-the-fly or over time. If the policies of a COI change, it may invalidate parts of defined workflows. Since executing workflows may lose valuable processing time if aborted in favor of a new workflow definition, runtime changes may be the best course of action. Unfortunately, commercial workflow engines that execute WS do not allow workflow modification at run-time. Thus, a new approach to workflow change is needed for WS.

A *workflow definition (WFD)* is deployed into the workflow engine environment before it is invoked as an executing *instance*. A workflow environment is typically composed of a single loop of processes such as those on the right side of Figure 1. A workflow

manager instructs a control infrastructure to manage instances which are monitored by the monitoring infrastructure which, in turn reports to the manager. Commercial workflow engines adhere to this single loop approach without support for dynamic changes to instances. To introduce change in a workflow environment, researchers advocate a second loop to monitor and modify the workflow engine's execution threads [3]. Alternatively, a workflow engine must be newly constructed to support dynamism [4]. This results in a huge divide between research-based dynamic workflow solutions and commercial engines. Thus, while every workflow environment allows WFD to be modified, at issue is how changes to WFD are propagated into the workflow environment at runtime.

In this paper, we present a double loop workflow process utilizing a COTS workflow engine. Our process includes reconfiguration rules tied to a COI and meta-data about WFD, instances, and the overall process in order to separate concerns while guiding a change process. We show the language independent mechanisms and their functions to ascertain the need for change, manage a WFD and its instances, and deploy changes to an instance. Our example implementation uses Oracle BPEL Process Manager.

2. Dynamic change in workflows

When changes occur, a workflow process change request is formulated to dictate what change is required and to which workflow definitions. We recognize three types of changes: (1) a user *directed* change in which the new and old WFD are specified, (2) a *priority* change based on WS availability or COI preferences for particular services, and (3) a *global* change to all workflows referencing a certain service. For example, a user may add new requirements into an existing workflow, a business process may discover a new and better service, or the environment may detect the movement of existing services.

We employ four common approaches for handling change [5].

- *Abort*: An instance is stopped without concern for loss of processing.
- *Flush*: An instance completes as is. The change only affects new instances.
- *Restart*: An instance is stopped immediately then restarted using the new WFD.
- *Migrate*: An instance uses the new WFD immediately. Prior state is retained and incorporated into the migrated instance. This is the most complex to implement.

2.1. Using COTS workflow engines

The proliferation of COTS workflow engines in commercial settings has promoted their thorough testing and operational stability. These engines typically adhere to an open, standardized workflow language and provide assistance with deploying, monitoring, and debugging instances. The type and granularity of the interfaces provided by engines can vary as shown in Table 1. Access to information may be limited to certain times, such as before or after an instance executes. In addition, engines can vary in the versioning mechanism used for modified WFD and their relationship to instances. Most engines provide *abort* and *flush* actions defined above, but not *restart* or *migrate* actions to facilitate dynamism. If the COTS workflow engine does not easily provide information regarding the instance state, migration requires complex analysis and reasoning external to the engine. Oracle is the best engine for partnering with a second loop for change.

Table 1. COTS workflow engine differences¹

COTS Workflow Engine	Versioning	Deployment	State Information
IBM WebSphere MQ Workflow	Server configuration options determine how instances are handled when a new WFD is deployed, options include different termination options such as flush or abort.	Separated Build-time and Run-time environments. Build-time contains tools to convert between BPEL and FDL (IBM specific workflow language), Run-time will only execute FDL files.	Workflow dashboard is available via WebSphere Business Integration Monitor, allows for detailed trace information to be accessed while instances are running, but only if utility is installed.
Microsoft BizTalk BPM	Version information can be included with WFD. Instances associated with previous WFD can be unlisted, such that they continue to execute and new instances are associated with new WFD.	Publishing wizard contains import/export wizard to assist in BPEL standardization. Internally WFD are stored as "orchestrations", a proprietary format specific to the BizTalk architecture.	Business Activity Monitoring architecture supports inserting "milestones" into WFD, such that architecture can extract state information at each defined milestone.
Oracle BPEL Process Manager	Deployment process allows for the specification of WFD version number. If previous WFD is overwritten, instances associated with it abort.	Executable WFD is stored as a BPEL document within a packaged jar file for deployment on server.	Detailed trace and audit information is available after instances have been aborted. API also defines methods to extract the last scope an instance has entered.

2.2. The concept of change in workflows

¹Available from <http://www-306.ibm.com/software/integration/wmqwf/>, <http://www.microsoft.com/biztalk/>, and <http://www.oracle.com/technology/bpel/>

Current research in dynamic changes to WS workflows is summarized in Table 2. Column 1 denotes if the workflow approach supports WS and associated standards. Column 2 states the role of a meta-model in specifying the extent to which the system understands change consequences. There is a difference between built-in flexibility and actual dynamism as reflected in column 3. Column 4 designates whether a double loop separates change concerns from workflow management. Modification to standard workflow languages, a deviation that makes any change method difficult to use in commercial settings, is reflected in column 5. Allowable WFD versioning and instance migration appears in columns 6 and 7 respectively. The final columns 8 and 9 respectively indicate if an Inspector, a process who automatically monitors, and a Coordinator, a process who analyzes and manages change, assists the workflow solution.

The proposed solutions in Table 2 have many positive aspects which must be present for dynamic changes of WS in workflow. However, there are also some clear limitations. For some it is the type or timing of allowable changes. For others, the development of an original workflow engine prevents its widespread use [4, 6]. One solution only permits workflow changes before the workflow begins execution [7]. Another solution specifies workflows abstractly but limits dynamism to those abstract points [8, 9]. Finally, support for only dynamic binding of endpoints fosters only certain kinds of changes [4].

Several important points can be gleaned from the research participating in this table. First, the workflow execution environment must be sensitive to the workflow reconfiguration process through monitoring and maintenance actions, such as provided with a double loop approach [3]. Second, allowing dynamism through abstractly defined structures results in flexibility but limits a community to specific types of change [4, 9]. Third, implementing rule driven decisions and policies allows flexible change management specific to the COI and is a concise way to state COI policies [5-8].

In summary, the goals for a workable solution to dynamic workflows include:

- Limit coupling to any specific engine. Though monitoring is necessary, the monitoring mechanism should be adaptable to different engine types.
- The solution should be flexible and adaptable to different COI with varying policies. Some workflow solutions create a standard method of

enacting change which is then applied to all workflow changes, regardless of the environment.

- The architecture and functionality augmenting a COTS engine should not restrict the type, structure, or timing of dynamic changes.

Adherence to a standard workflow language provides interoperability with other workflows and workflow engines to promote scalability and adoption to commercial settings. Deviation from a standard limits solution applicability.

Table 2. Dynamic workflow research summary

Name	Supports									
	WS	Meta-model	Dynamism	Dbt-loop	Language Modification	WFD	Versioning	WFI Migration	Inspector	Coordinator
Autonomic Capabilities (JOpera) [3]	No	No	Yes	Yes	No	No	No	No	No	Yes
Adaptive Workflow Management in WorkSCO [4]	No	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No
Dynamic Evolution using Micro-workflow language [10]	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes
Ontology-driven Architecture [8]	No	No	No	No	Yes	No	No	No	No	Yes
WebComposer (BPEL) [9]	Yes	No	No	No	Yes	No	No	No	No	No
Worklets (YAWL) [6]	No	No	Yes	Yes	No	No	Yes	?	?	Yes
OPENFlow [7]	No	No	Yes	No	New	No	Yes	No	No	No
Dynamic Schema Change [5]	No	No	Yes	No	No	Yes	Yes	Yes	No	No
Our Approach	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes

3. Architecture of CPCL

The double loop process we advocate assumes one loop within the workflow engine and the other external to it. This second loop, called the change planning and coordination loop (CPCL – pronounced “capsule”) is the foundation for the process to monitor and manage change. Thus, the internal workflow engine loop manages the execution of instances and the execution of workflow changes as directed by the CPCL.

We currently assume the following in CPCL that works with the Oracle BPEL engine.

- The engine provides a unique WFD identification that allows identification of all instances bound to one particular WFD.
- The engine provides abort, suspend, resume, and start commands to operate on specific instances.

- Workflows are simplified to sequential processing with no external parameters.

These assumptions allow us to abstract out certain complexity in detecting the exact state of a looping process. The Oracle BPEL engine provides a method to obtain variable values making this limitation easy to remove at a later time.

The architecture described in the following sections is a foundation able to deal with change while still maintaining the ability to utilize a COTS-based workflow engine. Our approach limits the coupling to a specific workflow engine by using an integration enabler which abstracts interaction to the workflow engine. The approach is adaptable to different communities by using COI-specific rules that are evaluated independently of the CPCL platform. Our approach implements dynamism by performing instance migration rather than utilizing dynamic workflow decision points as mentioned earlier in section 2.2 We adhere to BPEL-specific workflow engines that are commercially available, with an emphasis on using the Oracle engine due to reasons stated in section 2.1.

3.1. CPCL architecture

Figure 1 shows how the workflow engine manages a WFD and associated instances by an abstracted loop on the right. This loop interacts with the CPCL on the left via a *Bridge* component. The components listed in the COTS workflow engine are related to those discussed in previous sections. The responsibilities of each of the CPCL components are described in the next sections, followed by a flow of the change process in section 3.2.

The components of the CPCL process utilize a variety of meta-data in order to exchange information that is relevant to the state of the COI and workflows.

For a given WFD, the information that assists the process involves the specific BPEL definition and the WS that are called within it. The BPEL document is used for making changes and comparisons to the definition. Each WSDL document provides the CPCL process with semantic information describing operations provided by the service and location information about where a service is deployed and how the workflow engine can access it.

For a given instance bound to a WFD, another set of meta-data information can be highlighted. This meta-data includes information about the current state of the instance, gathered from a trace of all executed workflow statements and a reference to its WFD. This information assists in determining when the migration of an instance to a new workflow definition is possible.

3.1.1. Inspector. The Inspector performs automated monitoring and investigation of the COI to collect meta-data pertinent to managing change. The triggers for reconfiguration are monitored with the proper frequency and accuracy to correctly gauge the reconfigurations required.

The Inspector gathers WFD and instance meta-data for all WFD deployed on a workflow engine by communicating with a Bridge component that is later discussed in section 3.1.4. Typically, reconfiguration is performed by maintaining a view of the system's architecture in the form of a model or meta-data. At any given point in time the Coordinator component, which is responsible for determining actions associated with reconfiguration, can ask the Inspector for this meta-data.

The Inspector is responsible for monitoring the WS that are deployed within the COI and contains functionality to determine the *priority* of a WS as well as if the WS is of a *transactional* nature, via COI-specific standards. Since COI WS are registered in an

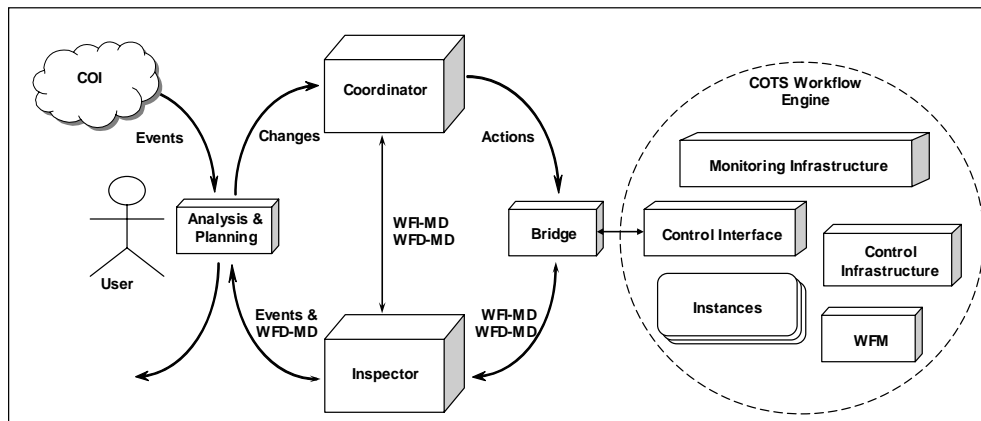


Figure 1. Double-loop using COTS workflow engine.

UDDI registry, the Inspector queries the registry to determine if changes have occurred at regular set periods of time. As WS go on and offline, the Inspector is aware of the change and redirects events to the Analysis & Planning part of the CPCL process.

3.1.2. Analysis & Planning. As shown in Figure 1, requests for reconfiguration enter the double loop either as submitted information from a COI user, by the COI at large, or by the Inspector's automated investigation of the COI. The location that reconfiguration requests are received from determines the type of change request. Events are directed to the Analysis & Planning module and processed according to policy outlined by the COI. Specified reconfiguration actions are determined and sent to the Coordinator for implementation.

Users can submit specific changes to a WFD or a new WFD via an Analysis & Planning interface that verifies the validity of the BPEL statements in the WFD. Our implementation utilizes the toolkit made available from Oracle for manipulating BPEL documents that will be deployed on a workflow engine. This allows the Coordinator to assume changes submitted through the Analysis & Planning process are valid BPEL syntax. Thus, the Coordinator can focus on the dynamic aspect of the change. User submission allows the user to direct the functionality of the Coordinator if required.

3.1.3. Coordinator. The challenge for dynamic reconfiguration is to maintain the system consistency and stability while implementing the reconfiguration changes. For this reason, a configuration management system typically validates and directs the configuration changes of a system architecture [11]. The CPCL process uses a Coordinator component, which can be viewed as a configuration management system.

In CPCL, Analysis & Planning forwards reconfiguration requests to the Coordinator component. The Coordinator is a dynamic process that deduces a reconfiguration plan based on request information, and knowledge of the COI and workflow engine that is obtained from the Inspector. The reconfiguration plan is a set of actions that are forwarded to a workflow engine for enactment. The Coordinator creates changes for each WFD, as well as the running instances associated with the definition. The Coordinator has been implemented to use BPEL as a workflow language, so the Coordinator can create the appropriate WFD changes.

Change within the Coordinator is identified for a single workflow as a *transition point* that defines where

change begins within the WFD, and a definition of the new, valid sequence block, referred to as the *change block*. The *change block* is used to compare a newly submitted change against the running WFI on the workflow engine. The meta-data defined for a given WFI contains a trace of executed statements, which when compared to with *change block* can be used to determine the appropriate action that is necessary to implement the given change. Typically, in configuration management systems, inference engines or rules are used to assist in the evaluation of reconfiguration alternatives [12]. However, if the change is received from a user interacting with Analysis & Planning, the required actions can be overridden by the user's submission.

The following COI-specific rules are provided to outline example community specific concerns with respect to reconfiguration. Additional functions are defined in order to assist the Coordinator with correctly determining the appropriate change action. One such function is *snapshot* which returns the current state of an instance. This allows the Coordinator to determine where the instance is in its execution with respect to the transition point for the change.

We state the COI rules as an ordered list, such that the first rule in the list that is applicable is used to determine the change action.

1. If the change type is user-directed then use the directed change action, if given.
2. If the instance snapshot shows it is past the transition point, and the change involves a priority WS, then *migrate* the instance.
3. If the instance snapshot shows it is past the transition point, and the snapshot shows a transactional WS has been executed, then *flush* the instance.
4. If the instance snapshot shows it is past the transition point, but no transactional or priority WS are involved, then *restart* the instance.
5. If the instance snapshot shows it is before the transition point, then *migrate*.

Using the above rules, the change action (i.e., abort, migrate, flush, restart) is determined that will affect how instances associated with the changing WFD will be treated. The Coordinator queries the Inspector to get the list of instances. Once a set of affected instance are determined, COI Rules are executed for each instance.

3.1.4. Bridge. All communication from the CPCL process is forwarded through an integration enabler known as the Bridge. Previous research on integration

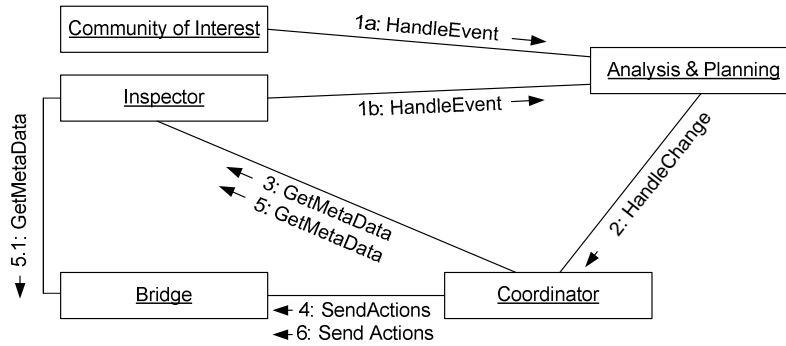


Figure 3. Double-loop collaboration diagram.

enablers had led our approach to a design that is loosely-coupled to the CPCL process, but tightly coupled to a specific workflow engine. Pre-defined information must be present to determine how any dynamism process implements reconfiguration. As shown in section 2.1, interacting with a standard BPEL server can vary between vendors, even when enacting the same task, such as deploying a new WFD. The Bridge is responsible for taking reconfiguration requests and translating the actions to statements that can be recognized by the engine.

To interface the CPCL process with the workflow engine, the Bridge exposes a set of common functionality that the process requires to operate. Available operations include deploying new definitions, gathering information about deployed definitions, and making changes by deploying new versions. These operations have to be custom tailored to the specific workflow engine being used, in order to overcome all of the challenges that exist when using a COTS workflow engine

Figure 2 shows the format of the WFD-MD and WFI-MD that the Bridge is responsible for creating based on the currently deployed WFD and WFI

```

<wfd-md>
  <wfd-id></wfd-id>
  <author></author>
  <specification></specification>
</wfd-md>

<wfi-md>
  <wfi-id></wfi-id>
  <wfd-id></wfd-id>
  <parameters></parameters>
  <trace></trace>
</wfi-md>
  
```

Figure 2. WFD-MD and WFI-MD

contained within the workflow engine. For the WFD-MD, *wfd-id* is a unique identifier assigned to each WFD by the engine. *Author* can refer either to a COI-user that originally created the WFD, or another *wfd-id* which underwent a change related to that necessitated the creation of this WFD. *Specification* is used to contain the actual BPEL specification related to the WFD. For the WFI-MD, *wfi-id* is a unique identifier assigned to a particular instance by the workflow engine. The value held by *wfd-id* is used as a reference to determine which specific WFD the WFI belongs to. *Parameters* hold the initial parameters used to start the instance, if any exist, and *trace* is a BPEL trace which shows the execution history of the instance.

To make this information available, the Bridge must be able to either (a) extract the necessary information via API calls to the workflow engine, or (b) statements must be inserted into deployed workflow definitions so state information can be routinely extracted. Our implementation of the Bridge interfaces with the Oracle BPEL engine and is able to extract state information via embedding of WS calls back to the Bridge that report state. Similar methods can be used with other workflow engines that are BPEL-compliant.

3.2. CPCL flow

In order to understand the sequence in which the CPCL process components are used for standard reconfiguration requests, a UML collaboration diagram is shown in Figure 3. This assists in tracing the route that a change travels within the process.

A short introduction of how reconfiguration occurs with these components is detailed in the following steps corresponding to the sequence numbers in Figure 3.

1. To begin the process, either the COI or the Inspector informs Analysis & Planning that an

event has occurred within the Community of Interest.

2. Analysis & Planning receives the event and formulates an overriding change that needs to be deployed within the COTS workflow engine, and redirects a change request to the Coordinator for implementation.
3. The Coordinator gathers meta-data from the Inspector to assist in the initial change that will be deployed as a new WFD to the workflow engine.
4. The Coordinator sends actions to the Bridge that result in the deployment of new WFD, and the need for instance migration.
5. The Coordinator gathers meta-data from the Inspector to assist in the new changes that will be associated with instance migration.
6. The Inspector communicates with the Bridge to gather the relevant information required to assist the Coordinator.
7. The Coordinator sends actions to the Bridge that result in the attempted migration of instances affected by the initial event received from the COI.

An example of a reconfiguration request and modification to a WFD is presented in the following section.

4. Example process

In the following section, we present a sample COI related to the emergency management sector of the government. The COI has an associated set of workflows that have been previously defined for the normal business operations of the COI. The workflows report valuable information that can be used to direct response actions to crisis events. The following section defines one specific workflow in more detail. Next, dynamism with respect to this COI is illustrated by

traversing the double-loop process with one change which results in a workflow instance migration. Simulation of this example was accomplished by preserving the state of a single workflow instance using embedded statements, changing the workflow, and informing the Bridge to start the instance with the preserved state and newly defined workflow.

4.1. Workflow description

The workflow defined for an emergency management COI deals with gathering data associated with an alert event, processing it, and delivering it in a report to desired officials who need to be updated on the status of the community. The workflow is displayed graphically in Figure 4 as a WFD. Our example process deals with gathering data related to suppliers that are near a crisis event. Using *R* or *P* to denote a regular or priority WS, and *T* or *NT* to signify if a WS is transactional or non-transactional, the *TruckLocation* WFD can be viewed as:

- EmerMgmt.WS_Crisis_Alert – a R and NT service that is triggered, adds information about a new crisis, and stores this information in a central database.
- EmerMgmt.WS_Filter_Crisis – a R and NT service that is called and filters crisis to those that require the delivery of supplies.
- BMart.WS_Get_Trucks – a R and NT service that is called to retrieve current locations of supplier trucks, their capacities, and load information.
- EmerMgmt.WS_Match_Regions – a R and NT service that performs a matching between supplier trucks and crisis locations, then stores this information.
- EmerMgmt.WS_Report_Trucks – a R and NT

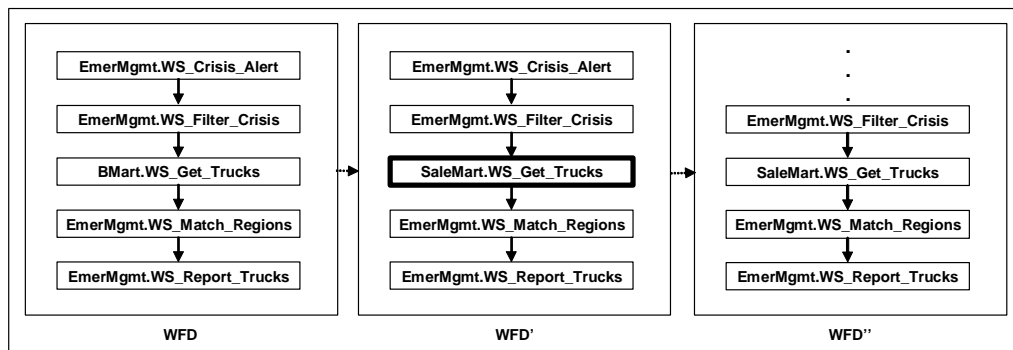


Figure 4. Workflow definition comparisons.

service that retrieves crisis and related truck information, sorts the information by distance, and reports the information to a user.

The processing time for this workflow may vary a great deal depending on the processing load on the supplier, the time of day, and number of trucks in motion. Information computed or retrieved by services is always stored locally since during an emergency, communications may be lost at any time. The workflows are run by different emergency management personnel at distributed locations. Some users are local to the crisis site, some users are at a location securely removed from the crisis but still geographically close, and some users are located securely at emergency management headquarters which may or may not be close to the crisis area. This allows further adaptation by the workflows which can be modified to work off the most recently stored data in the event of a failure of any WS.

4.1.1. Change request. A major retailer, SaleMart, has the ability to provide supplies for emergencies and has more delivery trucks than any other retailer, with a distributed system of warehouses. These conditions make it the highest priority supplier for the emergency management COI. An additional WS is defined as the other services were defined above:

SaleMart.WS_Get_Trucks – a *P* and *NT* service that is called to retrieve current locations of supplier trucks, their capacities, and load information.

SaleMart’s WS was previously offline and the alternate supplier’s WS is being used. Thus, currently the workflow calls BMart.WS_Get_Trucks() for the supplier information.

4.1.2. Change Planning & Coordination Processing. For the emergency management COI, the implementation of the CPCL loop uses the Bridge discussed previously to ensure that the Inspector gathers appropriate information about all of the deployed WS, including their associated WSDL documents and their state indicating if they are online or offline.

Once all information has been registered with the Inspector, the process begins with Step 1 from section 3.2. The Inspector process watches the SaleMart WS’s addition to the COI UDDI registry. When a new WS comes online in the COI, the Inspector recognizes this as an event and reports event details to the Analysis &

Planning process. As outlined in Step 2, Analysis & Planning’s implementation examines the event and creates the corresponding change request for further analysis by the Coordinator. The message that is generated by Analysis & Planning is defined as a Change Request, shown below:

```
SaleMartOnlineCR = (Priority, "Replace
'Invoke SaleMart.WS_Get_Trucks()'
where 'Invoke *.WS_Get_Trucks()'")
```

In this statement, “*.WS_Get_Trucks()” implies all calls to any WS_Get_Trucks() operation. Step 3 requires that the Coordinator receive the change request and query the Inspector for detailed information about all the WFD currently deployed. These WFD are filtered by the Coordinator using all of the WFD-MD to only those including a call to a WS_Get_Trucks() operation. For this example, assume only the *TruckLocation* WFD remains after filtering. Figure 5 shows the associated WFD-MD for this workflow, omitting the BPEL specification to ease readability.

To complete step 4, the Coordinator has been

```
<wfd-md>
  <id>
    TruckLocation
  </id>
  <author>
    COI-userid
  </author>
  <specification>
    BPEL Specification
  </specification>
</wfd-md>

<wfi-md>
  <wfd-id>
    TruckLocation
  </wfd-id>
  <wfi-id>
    TruckLocationWFI
  </wfi-id>
  <parameters>
    None
  </parameters>
  <trace>
    EmerMgmt.WS_Crisis_Alert
  </trace>
</wfi-md>
```

Figure 5. Example WFD-MD and WFI-MD

implemented to modify the *TruckLocation* WFD as shown in Figure 4 such that WFD' is generated. The Coordinator sends the new WFD' to the Bridge. The Bridge, upon receiving this new WFD', first suspends all instances related to the previous WFD then deploys the new definition.

The Coordinator then queries the Inspector for a list of executing instances related to the original *TruckLocation* WFD, as shown in Figure 4 as step 5. In order for the Inspector to fulfill the request of the Coordinator, it queries the Bridge for current meta-data about the suspended instances. Step 6 ensures that the meta-data correctly reflects the current execution location and state of all instances. For this example, assume there is only one instance running on the workflow engine, *TruckLocationWFI*, such that it has just processed the statement *EmerMgmt.WS_Crisis_Alert*. Figure 5 shows the example WFI-MD for this instance.

As part of the Coordinator's reconfiguration processing, a change Block is computed as well as a transition point. *TruckLocationWFI* is evaluated by the Coordinator. Based on the trace information for the instance the rules are evaluated sequentially, the Coordinator determines that the instance has not executed past the transition point, and the appropriate change action of Migrate from Section 2 is chosen. The Coordinator is then modifies WFD' to generate WFD''.

The comparison between WFD, WFD', and WFD'' can be viewed in Figure 4. WFD shows the original definition with a reference to the BMart WS. WFD' has been updated to refer to the priority WS SaleMart. WFD'' is a custom WFD generated for *TruckLocationWFI* to allow instance migration since it can still call the SaleMart WS.

Upon completion of these steps, a message is sent containing WFD'' to the Bridge to complete step 7, which starts a running instance of WFD'' to replace *TruckLocationWFI*. The Bridge has been implemented to enact the required changes to the instances associated with the change from step 4 and the double-loop process has been completed. Alternative change actions that could have been taken by the Coordinator include:

- Abort, which would have terminated the execution of *TruckLocationWFI*. Processing time would be lost due to the fact that a COI user would have to manually update and restart the instance in order for it to run successfully.
- Flush, which would have allowed *TruckLocationWFI* continue to execute the original WFD. No processing time would be lost in

the handling of the instance, but important information available from the priority WS would not be used in the workflow.

- Restart, which would have restarted the execution of *TruckLocationWFI* from the beginning using WFD'. Processing time would have been lost considering the fact that WS *EmerMgmt.WS_Crisis_Alert* would execute twice.

6. Conclusion and future work

Our process enables dynamic WS workflow changes and highlights the obstacles faced when adapting COTS workflow engines to dynamism. We note that COTS engines can enhance their ability to customize and adapt to various COI by including full access to instances during runtime. While every COI may not require like functionality, they can more easily tailor workflow reconfiguration to their unique needs if, for instance, the API details and manipulation were commonly available. Future work includes migrating the existing infrastructure to other BPEL engines and expanding the type of dynamism that can be observed and processed by the Inspector process and the COI specific rules.

Acknowledgement. This material is based on research sponsored by the Air Force Research Laboratory, under agreement number FA8750-06-2-0059. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

7. References

- [1] *Workflow Management Coalition, The Workflow Handbook*. Lighthouse Point, FL, USA: Future Strategies Inc., 2002.
- [2] S. Renner, "A Community of Interest Approach to Data Interoperability," presented at Federal Database Colloquium, San Diego, 2001.
- [3] T. Heinis, C. Pautasso, and G. Alonso, "Design and Evaluation of an Autonomic Workflow Engine," presented at Int'l Conf. on Autonomic Computing, 2005.
- [4] P. Vieira and A. Rito-Silva, "Adaptive Workflow Management in WorkSCo," presented at Int'l Workshop on Database and Expert Systems Applications, 2005.
- [5] S. W. Sadiq, "Handling Dynamic Schema Change in Process Models," presented at Australasian Database Conf., 2000.
- [6] M. Adams, A. H. M. t. Hofstede, D. Edmond, and W. M. P. v. d. Aalst, "Implementing Dynamic Flexibility in Workflows using Worklets," BPM Center Report, BPMCenter.org, BPM-06-06, 2006.

- [7] J. J. Halliday, S. K. Shrivastava, and S. M. Wheeler, "Flexible Workflow Management in the OPENflow System," presented at IEEE Int'l Enterprise Distributed Object Computing Conf., 2001.
- [8] T. A. S. C. Vieira, M. A. Casanova, and L. G. Ferrao, "An Ontology-Driven Architecture for Flexible Workflow Execution," presented at WebMedia and LA-Web, 2004.
- [9] L. A. G. d. Costal, P. F. Pires, and M. Mattosol, "WebComposer: a Tool for the Composition and Execution of Web Service-based Workflows," presented at WebMedia & LA-Web 2004 Joint Conf., 2004.
- [10] P. Dias, P. Vieira, and A. Rito-Silva, "Dynamic evolution in workflow management systems," presented at Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA '03), 2003.
- [11] D. LeMetayer, "Describing Software Architecture Styles Using Graph Grammars," *IEEE Transactions on Software Eng.*, pp. 521-553, 1998.
- [12] M. Wermelinger, "Towards a Chemical Model for Software Architecture Reconfiguration " presented at Int'l Conf. on Configurable Distributed Systems, 1998.