

AFRL-RI-RS-TR-2007-251
Final Technical Report
October 2007



FUSELET WORKFLOW INSPECTION AND FEEDBACK

University of Tulsa

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2007-251 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

NORMAN AHMED
Work Unit Manager

/s/

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.					
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) DEC 2007		2. REPORT TYPE Final		3. DATES COVERED (From - To) Feb 06 – Aug 07	
4. TITLE AND SUBTITLE FUSELET WORKFLOW INSPECTION AND FEEDBACK				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER FA8750-06-2-0059	
				5c. PROGRAM ELEMENT NUMBER 62702F	
6. AUTHOR(S) Rose Gamble and Robert Baird				5d. PROJECT NUMBER FUSE	
				5e. TASK NUMBER 06	
				5f. WORK UNIT NUMBER 02	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Tulsa 600 S. College Ave. Tulsa OK 74104-3126				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RISE 525 Brooks Rd Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2007-251	
12. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 07-0456					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The main goal of this effort is the introduction of the concept of a reconfigurable workflow of Fuselet processes into the Joint Battlespace Infosphere (JBI). The marriage of information manipulation and Web services has the potential to increase the net-centric capabilities of the JBI by making it easier to gain access to only needed or pre-processed information designated by a Community of Interest (COI). This technology is especially needed to accommodate the migration of clients accessing the JBI and the JBI itself to Web services. Coordination and orchestration of Fuselets as Web services within the current Fuselet Runtime Environment (FRE) are fundamental to the changes needed for this migration, requiring new construction, deployment, and management technology to be used by the Information Management Staff. The novel outcome of the research is a system design that is poised to support dynamic reconfiguration in the form of policy changes, workflow definition updates, environmental changes, MIO versioning, and changing Fuselet services or locations.					
15. SUBJECT TERMS Fuselets, Community of Interest (COI), Web services, Joint Battlespace Infosphere (JBI)					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UL	18. NUMBER OF PAGES 44	19a. NAME OF RESPONSIBLE PERSON Norman Ahmed
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

Table of Contents

Table of Contents	i
List of Figures	ii
List of Tables	ii
Introduction	1
Initial Migration of Prior Fuselets to Web Services	2
Investigation and Survey of Workflow Engines	2
Change Planning and Control Loop (CPCL)	4
<i>The Reconfiguration Bridge Component</i>	6
<i>The Coordinator Component</i>	9
<i>The Inspector Component</i>	12
<i>Analysis and Planning Component</i>	14
<i>Reconfiguration Planning</i>	15
Plan Rule Formation	17
Meta-data Specification	19
CPCL Component Schema Definitions.....	20
Tasking and Control Interface Loop (TCIL)	22
<i>Controller Interface</i>	23
<i>Reporting and Feedback</i>	24
Workflow Controls	25
<i>Dynamic Goal Controls</i>	25
<i>Dynamic Workflow Change Controls</i>	28
Jython Fuselet Wrapper	30
Example Scenarios	32
<i>Terrorist Chemical Attack Scenario</i>	32
<i>Air Base Battlefield Scenario</i>	36
Conclusion	38
References	38

List of Figures

Figure 1. NeWT architecture.....	2
Figure 2. CPCL Activity Diagram.....	5
Figure 3. <i>Bridge</i> Processing.....	6
Figure 4. Class Diagram for the <i>Reconfiguration Bridge</i> Component.....	7
Figure 5. Use case diagram for the <i>Reconfiguration Bridge</i>	8
Figure 6. <i>Coordinator</i> processing.....	9
Figure 7. Processing paused instances	10
Figure 8. Use Case diagram for the <i>Coordinator</i> Component.....	11
Figure 9. Class diagram for the <i>Coordinator</i> Component	12
Figure 10. Use Case diagram for the <i>Inspector</i> Component.....	13
Figure 11. Class diagram of <i>Inspector</i> Component.....	13
Figure 12. <i>Analysis and planning</i> processing	14
Figure 13. Use Case Diagram for the <i>Analysis & Planning</i> Component	15
Figure 14. Definition Metadata (WFD-MD).....	19
Figure 15. Instance Metadata (WFI-MD).....	19
Figure 16. <i>Analysis & Planning</i> Change Request to <i>Coordinator</i>	20
Figure 17. <i>Inspector</i> Workflow Event to <i>Analysis & Planning</i>	21
Figure 18. <i>Inspector</i> Service Event Sent to <i>Analysis & Planning</i>	21
Figure 19. <i>Coordinator</i> wfd Request for Metadata to the <i>Inspector</i>	22
Figure 20. NeWT Interaction with UDDI Interface Specification.....	22
Figure 21. NeWT UML Diagram	23
Figure 22. TCIL report viewer	25
Figure 23. Goal preference entry dialog	26
Figure 24. Goal preference table for example workflow	26
Figure 25. Workflow goal selection dialog box	27
Figure 26. Report for goal change reconfiguration	28
Figure 27. Workflow reconfiguration options.....	29
Figure 28. Workflow reconfiguration results.....	29
Figure 29. Fuselet Wrapper Use Case Diagram.....	31
Figure 30. Automated Fuselet to Web service Wrapping	32
Figure 31. TCA process flow.....	34
Figure 32. Terrorist Chemical Attack - Simulation Setup	35
Figure 33. Terrorist Chemical Attack - Simulation Map.....	35
Figure 34. Terrorist Chemical Attack - Simulation Web Service Test Interface.....	36
Figure 35. AirportTakeoff Workflow Definition	37

List of Tables

Table 1. COTS workflow engine differences.....	3
Table 2. Dynamic workflow research summary	4
Table 3. Community Goal-Preferences.....	17
Table 4. COI Rules for Change Actions	18

Introduction

The main goal of this effort is the introduction of the concept of a reconfigurable workflow of Fuselet processes into the Joint Battlespace Infosphere (JBI). The marriage of information manipulation and Web services has the potential to increase the net-centric capabilities of the JBI by making it easier to gain access to only needed or pre-processed information designated by a Community of Interest (COI). This technology is especially needed to accommodate the migration of clients accessing the JBI and the JBI itself to Web services. Coordination and orchestration of Fuselets as Web services within the current Fuselet Runtime Environment (FRE) are fundamental to the changes needed for this migration, requiring new construction, deployment, and management technology to be used by the Information Management Staff. The novel outcome of the research is a system design that is poised to support dynamic reconfiguration in the form of policy changes, workflow definition updates, environmental changes, MIO versioning, and changing Fuselet services or locations.

In the following sections, we provide a technical discussion of the *Next Generation Workflow Toolkit* (NeWT) framework. The framework we propose for NeWT encompasses two entities that work in tandem with a commercial workflow management system. The first entity, the *Change Planning and Coordination Loop* (CPCL, pronounced “capsule”), is extended by NeWT to support the reconfiguration directives generated by the COI. The second entity within NeWT is the *Tasking and Control Interface Loop* (TCIL, pronounced “tassel”), which houses the functionality for the COI to designate overarching workflow tasks, goals and service priorities while at the same time allowing users to generate, deploy, control, and reconfigure workflows of WS. TCIL establishes the base from which business rules, workflow goals, and service priorities, are encoded into a workflow definition that can be reconfigured while executing using CPCL. The NeWT framework is shown in Figure 1. The current implementation of NeWT uses the Oracle BPEL Process manager. BPEL is Business Process Engineering Language, a commonly used specification language to represent workflows of WS.

NeWT uses the concept of *Communities of Interest* (COI) to encompass multiple businesses who access workflows to perform known tasks regularly. COI members are related by a geographic area, an administrative domain, or common goals. Over time, COI member interactions can be refined to include preferences for service usage within the confines of a particular workflow. These preferences distinguish among multiple workflows that perform the same task but relate to a different performance goal. Priorities among competing WS may be known based on service quality and availability. However, because a COI’s mission may be critical, redundant services may be in place to take over when a workflow may fail at runtime. Accounting for these issues in workflow definition and change can increase the performance of a COI with respect to completing its task in a timely manner and with the best service usage.

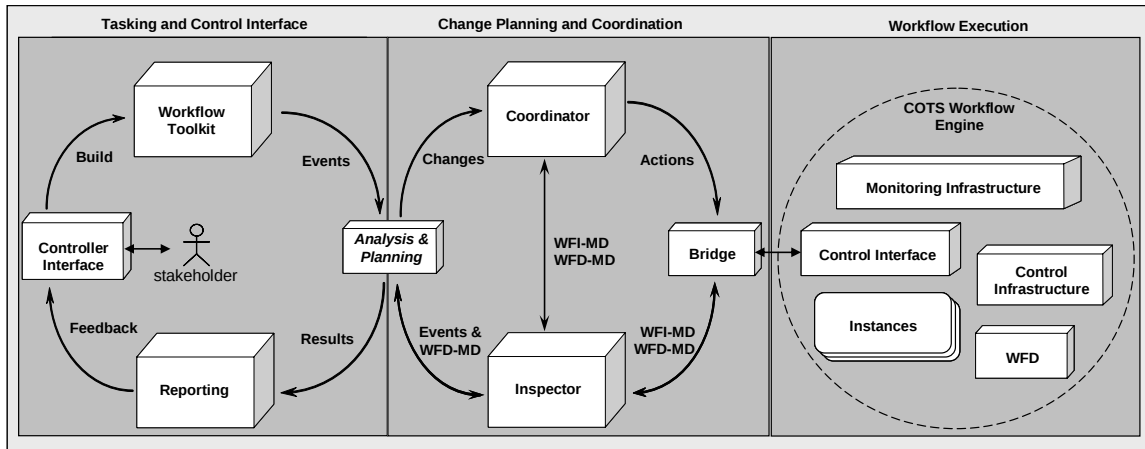


Figure 1. NeWT architecture

Initial Migration of Prior Fuselets to Web Services

One of the early objectives of the research was to push existing fuselets into a Web service representation and assess their deployment on the Joint Battlespace Infosphere (JBI). To address this objective, several pre-existing component-based Fuselets were migrated to Web services. Specifically, Fuselets were taken from the previous University of Tulsa *Enemy Alert* application and manually converted for deployment into a J2EE Application Server, specifically JBoss. This experience readied the project to begin using Web services as the main client base. As Fuselets were deployed, metadata describing their exposed functionality was viewable via their Web Service Definition Language (WSDL). This information was stored and retrieved using the Universal Description, Discovery and Integration (UDDI) protocol.

Investigation and Survey of Workflow Engines

Investigation into the type and granularity of the interfaces provided by some common COTS-based workflow engines is presented 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. From the data accumulated, we concluded that the Oracle BPEL Process Manager is the workflow engine most adaptable for the inspection and feedback of executing workflows within the newly devised second loop.

Table 1. COTS workflow engine differences.

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	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.

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 research-based workflows we evaluated were Autonomic Capabilities [1], Adaptive Workflow Management in WorkSCo [2], Dynamic Evolution using Micro-workflow language [3], Ontology-driven Architecture [4], WebComposer [5], Worklets (YAWL) [6], OPENFlow [7], and Dynamic Schema Change [8].

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. One solution only permits workflow changes before the workflow begins execution. Another solution specifies workflows abstractly but limits dynamism to those abstract points. Finally, support for only dynamic binding of endpoints fosters only certain kinds of changes.

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. Second, allowing dynamism through abstractly defined structures results in flexibility but limits a community to specific types of change. Third, implementing rule driven decisions and policies allows flexible change management specific to the COI and is a concise way to state COI policies.

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. Therefore, we proceeded to use Oracle BPEL Process Manager in our development.

Table 2. Dynamic workflow research summary

Name	Supports									
	WS	Meta-model	Dynamism	DbI-loop	Language Modification	WFD	Versioning	WFI Migration	Inspector	Coordinator
Autonomic Capabilities (JOpera)	No	No	Yes	Yes	No	No	No	No	No	Yes
Adaptive Workflow Management in WorkSCo	No	Yes	Yes	No	Yes	Yes	Yes	No	No	No
Dynamic Evolution using Micro-workflow language	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes
Ontology-driven Architecture	No	No	No	No	Yes	No	No	No	No	Yes
WebComposer (BPEL)	Yes	No	No	No	Yes	No	No	No	No	No
Worklets (YAWL)	No	No	Yes	Yes	No	No	Yes	?	?	Yes
OPENFlow	No	No	Yes	No	New	No	Yes	No	No	No
Dynamic Schema Change	No	No	Yes	No	No	Yes	Yes	No	No	No
Our Approach	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes

Change Planning and Control Loop (CPCL)

The most important expected outcome of the project was defining, specifying, and prototyping an inspection and feedback loop for workflows of Fuselets as Web services on the JBI, and then reconfiguring them dynamically according to predefined rules. This involved the investigation and construction of several distinct components. We describe these components within the

Change Planning and Control Loop (CPCL) that performs the inspection, feedback, and reconfiguration of executing workflows.

Research was conducted to determine the necessary technologies that must be in place in order to realize CPCL, as well as the specification of the meta-information that must be exchanged in order to ensure its successful usage. Allowable changes which can trigger workflow reconfiguration include the addition or deletion of services from the UDDI repository and the changing of workflow definition directly. Furthermore, if the functionality of an implemented web service changes, the workflow may need to be adjusted manually and its instances reconfigured dynamically to incorporate these changes in a timely fashion. Information about the structure of the workflow, such as how data is passed between the Web services, is important in mitigating these changes. Therefore, we have researched how to extract state information from within an executing workflow instance.

A brief overview of the processing done by CPCL is shown in Figure 2 with a UML activity diagram illustrating the common COI changes and the triggers that can be handled by NeWT. The labeled steps are detailed as follows:

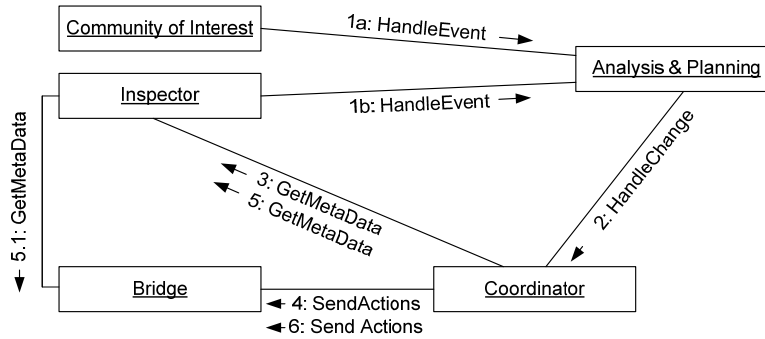


Figure 2. CPCL Activity Diagram

1. To begin the process, either the COI or the *Inspector* informs *Analysis & Planning* that an event has occurred within the COI.
2. *Analysis & Planning* receives the event and formulates an overriding change that needs to be deployed within the WFMS. The change is directed to the *Coordinator* for implementation.
3. The *Coordinator* initially gathers meta-data from the *Inspector* to assist in determining the change that will be deployed as a new WFD to the WFMS.
4. The *Coordinator* sends actions to the *Bridge* that result in the deployment of a new WFD. The *Bridge* recognizes the need to migrate the corresponding WFD instances.
5. The *Coordinator* gathers meta-data from the *Inspector* to assist in the new changes that will be associated with instance migration.
 - 5.1 The *Inspector* communicates with the *Bridge* to gather the relevant information required to assist the *Coordinator*.
6. The *Coordinator* sends actions to the *Bridge* that result in the attempted migration of instances affected by the initial event received from the COI.

In the subsequent section, we detail each component of CPCL.

The Reconfiguration Bridge Component

In the right-most panel of Figure 1 is a typical workflow management system (WFMS), such as the commercial Oracle BPEL Process manager. The WFMS interacts with CPCL (middle panel of Figure 1) via our *Bridge* component, or simply *Bridge*. The *Bridge* is used to hide engine specific details from the double loop, making normal workflow actions uniformly available in accordance with a specification that the double loop understands. The *Bridge* abstracts the interaction with a commercial workflow engine, exposing the interfaces necessary to allow for deployment, inspection, feedback, and the implementation of the workflow changes. It performs its tasks in a manner that is independent of WFMS implementation. The basic functionality of the *Bridge* component is illustrated below in Figure 3, where WFD stands for workflow definition and WFI stands for workflow instance. The workflow engine is embedded within the commercial WFMS.

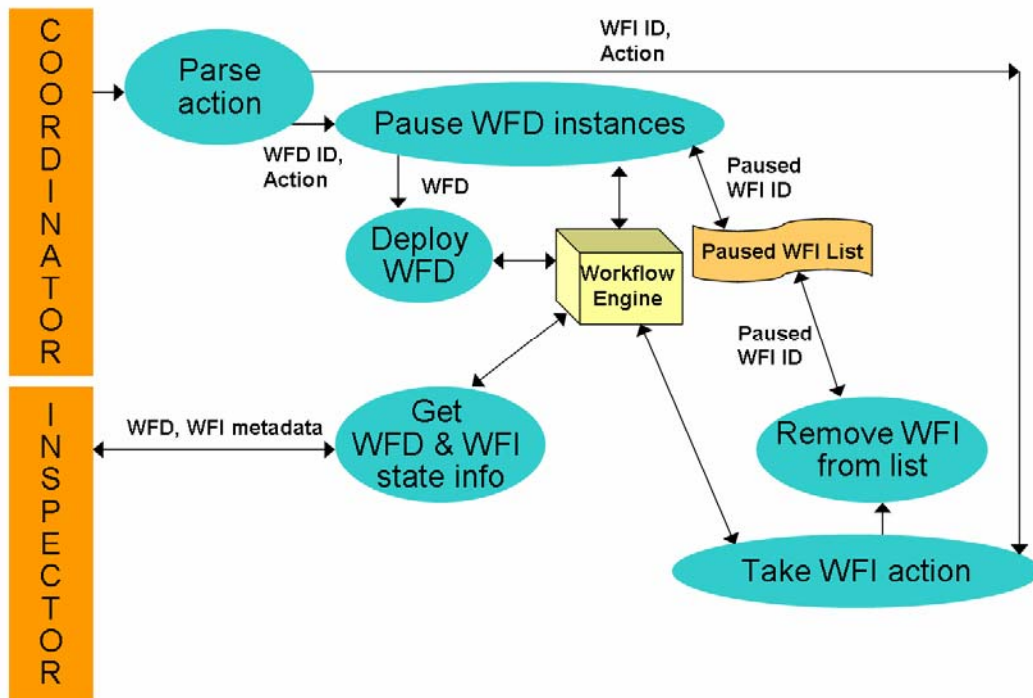


Figure 3. Bridge Processing

The initial set of functions performed by the *Bridge* include the ability to deploy new workflows, undeploy existing workflows, and instance management via actions such as abort, flush, migrate, or restart. Actions such as re-deployment of a workflow definition (to update it with a newer version) can be accomplished by deploying a workflow definition with the same name.

Many widely used WFMS support the use of the BPEL specification language to define workflows. However, the format that these workflow engines accept the BPEL document varies from implementation to implementation. The *Bridge* exposes the functionality of the WFMS

engine, so that if the BPEL document needs to be compiled or packaged in a specific format it can be done by the *Bridge* and not affect the remaining components of CPCL.

There are some drawbacks to the use of the Oracle BPEL Process manager. Primarily, these are in the area of instance management. The Oracle API does not support a native method to flush or migrate instances in an automated manner. Because the *Bridge* ensures the availability of these actions, its implementation utilizes a “pause list” data structure that assists in instance management. If a workflow may later need to be restarted or flushed, it is maintained within the pause list.

The *Bridge* interacts with the CPCL components via publish and subscribe, currently utilizing JBI message passing functions. The *Bridge* subscribes to a set of messages that refer to definition or instance actions. These messages have a schema definition that exposes the functionality of the *Bridge* in a simple manner. Deploying a new workflow can be accomplished by crafting a message that is sent to the *Bridge* containing the new workflow. Internally, the *Bridge* enacts the requested operation. When appropriate the *Bridge* publishes sets of meta-data back onto the JBI. It is the responsibility of the *Bridge* to publish this information in a format that is understood within CPCL.

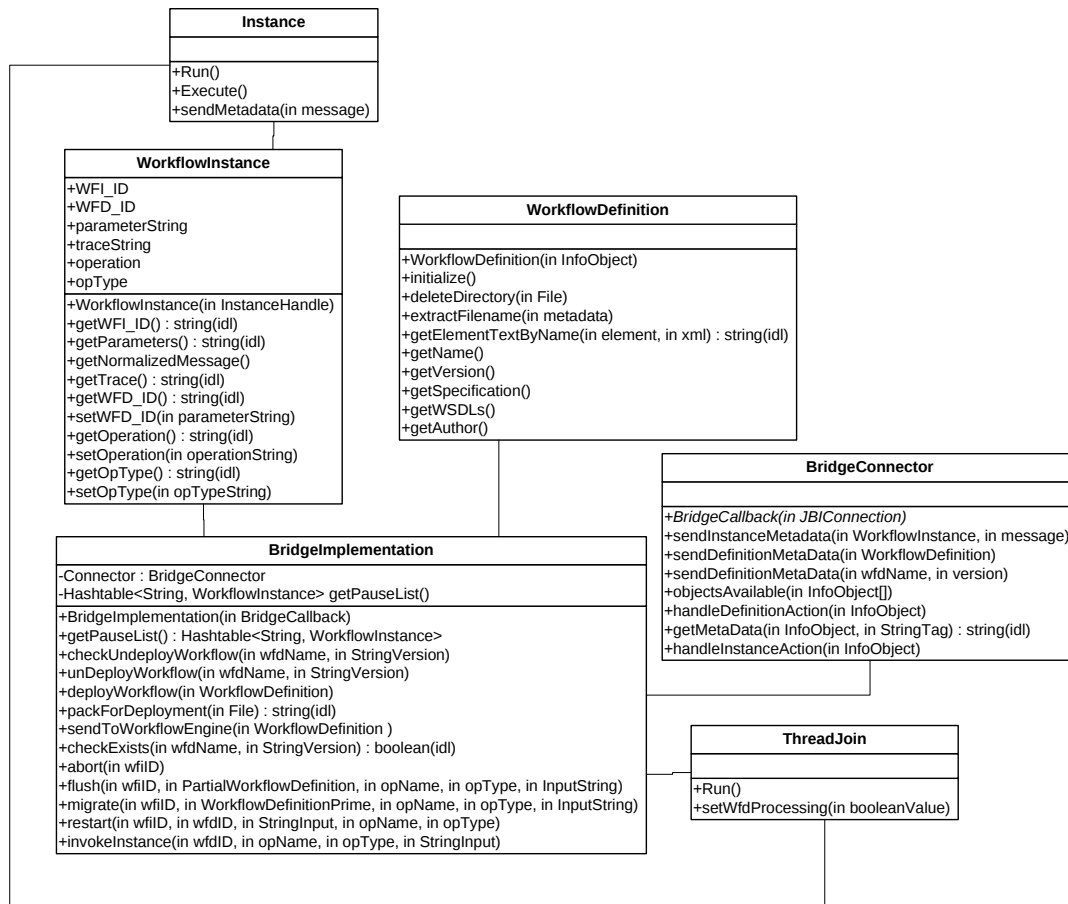


Figure 4. Class Diagram for the *Reconfiguration Bridge* Component

Figure 4 is a class diagram of the *Bridge*. The *Bridge* is composed of a BridgeConnector class which subscribes to definition and instance action messages published by the *Coordinator*. The WorkflowDefinition and WorkflowInstance classes manage metadata about the deployed workflows and executing instances on the WFMS engine. These classes are tightly coupled to the CPCL process but lack WFMS engine-specific implementation details.

The *Bridge* makes use of the *Oracle BPEL Process Manager API Specification*, Java implementation, to make specific calls to the workflow engine. The API is made available from Oracle and available via several Java jar files packaged with the workflow engine. The *Bridge* primarily uses the API classes IDeliveryService, IBPELProcessHandle, and IInstanceHandle for communication with the workflow engine. The current *Bridge* implementation uses the API specification v10.1.2.0.2, upgrades to later version of the workflow engine should be simple due to the API usage.

Therefore, the *Bridge* the following classes are coded. The BridgeImplementation class performs the processing required to perform actions such as definition and instance management. The ThreadJoin and Instance classes manage important execution thread processing to enable the *Bridge* to continue processing while waiting for a reply from the commercial WFMS.

Testing has shown the *Bridge* capable of workflow deployment, undeployment, and invocation actions as well as workflow instance actions to abort, restart, migrate, and flush stale instances.

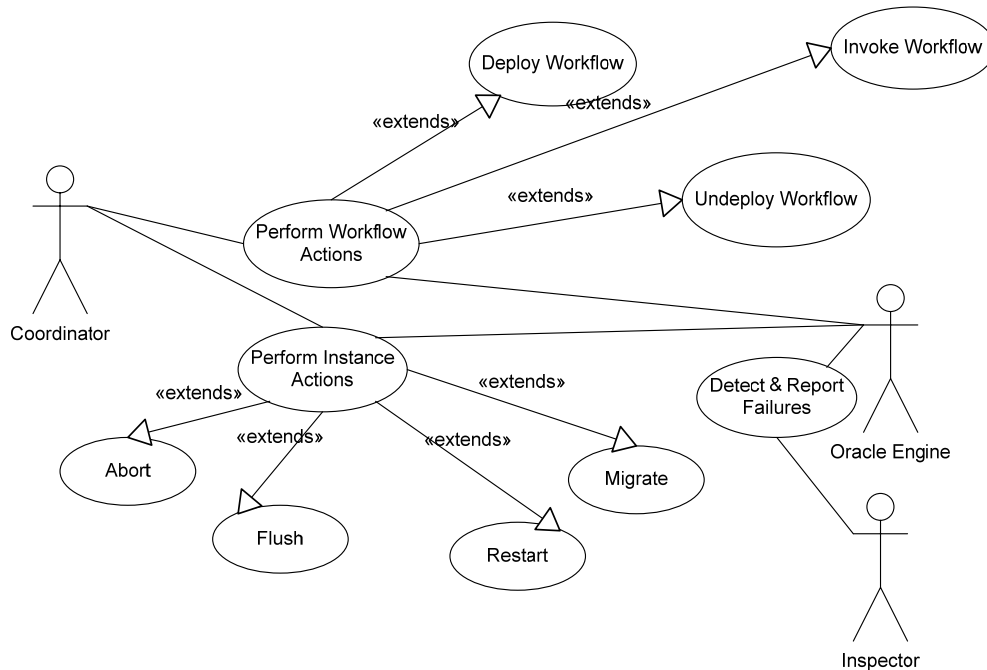


Figure 5. Use case diagram for the *Reconfiguration Bridge*

Figure 5 shows the use-case diagram for the *Bridge*. In addition, an instance's state is communicated by the *Bridge* publishing the instance's execution trace, which is published to the

JB1 via workflow instance meta-data. Change to an existing workflow definition is accomplished by deploying a changed workflow definition with the same original name.

The Coordinator Component

As depicted below, the *Coordinator* accepts a workflow definition (or a modified workflow as a re-definition), analyzes the definition metadata, and deploys new or changed workflows onto the engine via *Bridge* commands. As a result of workflow deployments, workflow instances are suspended. The *Coordinator* evaluates the workflow definition and instance metadata obtained from the *Inspector* (described in more detail in the next section) for each workflow instance affected and determines the appropriate action. These instances are managed through instance actions sent to the *Bridge*.

Implementation and testing of the *Coordinator* prototype revealed that performance of the process path that manages suspended workflow instances is crucial. Poor performance of this path will result in a degradation of performance for all COI workflows. It was imperative that the *Coordinator* not become the bottleneck. Thus, more processing related determining the appropriate reconfiguration directives became the responsibility of the *Analysis & Planning* component (described later in more detail). Figure 6 displays an overview of the processing performed by the *Coordinator* component.

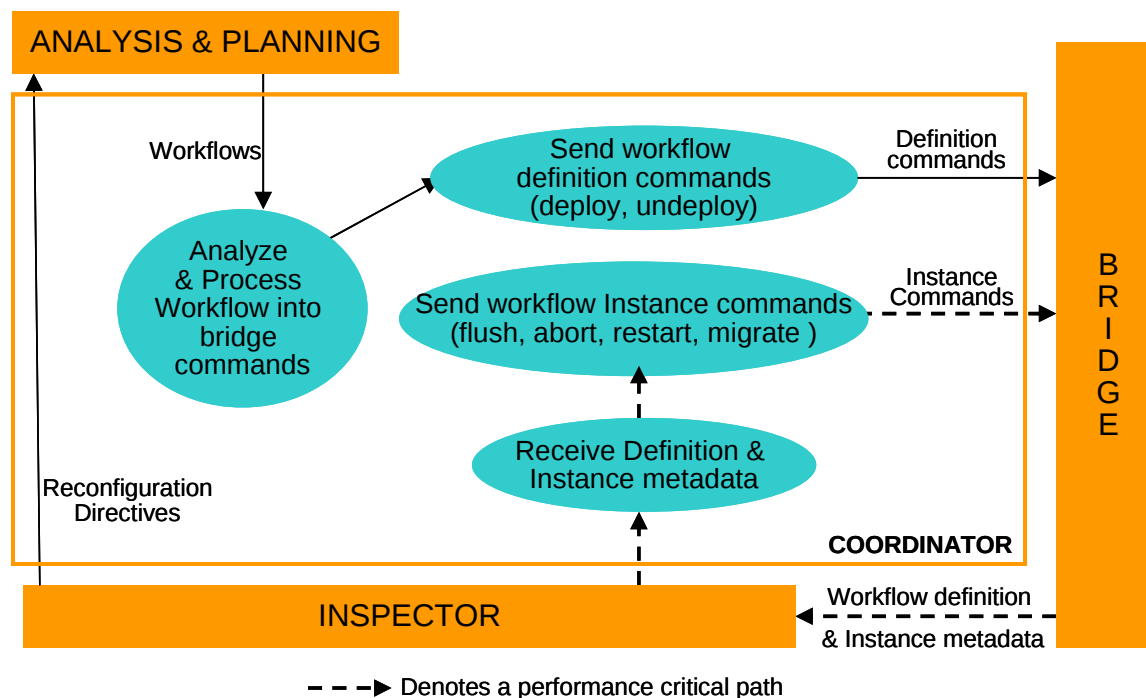


Figure 6. Coordinator processing

The *Coordinator* publishes commands to the *Bridge* as definition actions or instance actions. The definition actions are

- deploy a workflow,

- undeploy a workflow or remove it from the engine, and
- invoke a workflow to create an executing instance.

Instance actions are those actions taken on a workflow instance that has paused due to a new deployment of its corresponding workflow definition. When this changed workflow definition is deployed, the *Coordinator* analyzes the old and new workflow definition to determine the *transition point*. The transition point is the point that change in the workflow processing begins. Trace information for all paused instances is received from the *Bridge*. The *Coordinator* compares trace information obtained by inspecting the workflow instance up to the transition point to determine the appropriate action to take for each suspended workflow instance. For example, in Figure 7, Instance 1 can migrate to the new workflow definition (Workflow A') since its execution has not yet reached the transition point. Instance 2 has executed the transition point and so must either flush, abort, or restart.

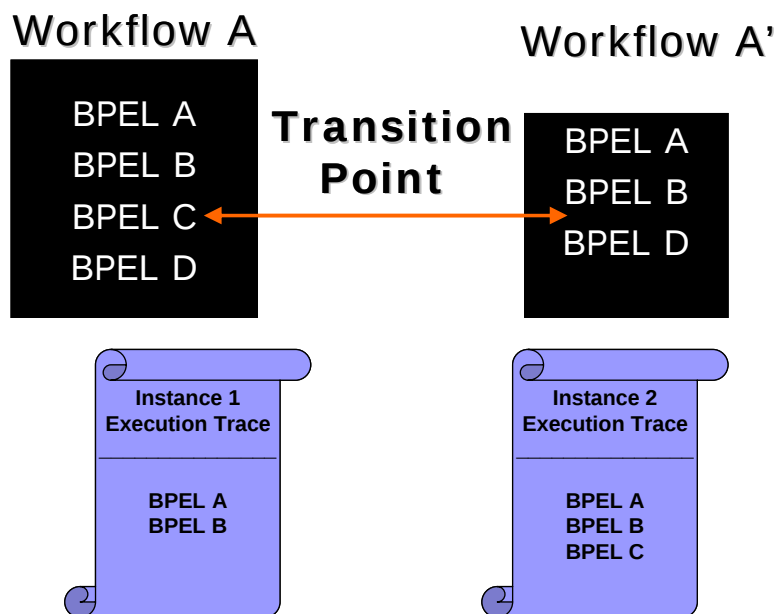


Figure 7. Processing paused instances

The actions that can be taken are

- *flush* the instance which means allow it to resume and complete processing using the previous workflow definition,
- *abort* the instance which forces the requestor to re-initiate processing,
- *restart* the instance which involves an abort followed by an invoke, or
- *migrate* which utilizes instance metadata to adapt the executing instance to the new workflow definition.

These actions are illustrated in the *Coordinator* Use Case diagram in Figure 8.

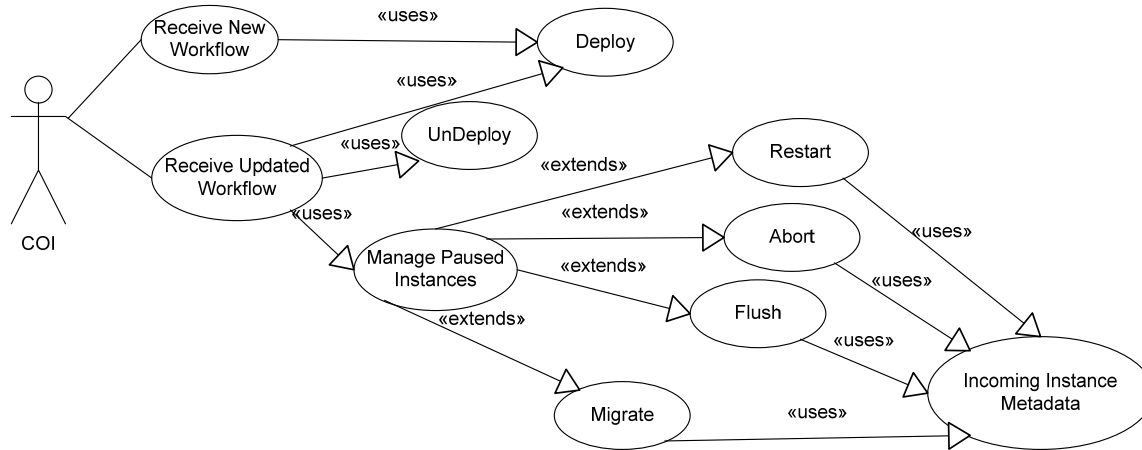


Figure 8. Use Case diagram for the *Coordinator* Component

The *Coordinator* also publishes information to the *Inspector* about deployed BPEL specifications, their supporting services, and WSDL files. This schema is illustrated in the section covering the *Inspector*. The payload of this message is a zip file containing the workflow BPEL file, the workflow WSDL document, a BPEL.XML document which binds the BPEL's partner links to specific WSDL documents, and the WSDL documents for the services that are invoked from the workflow. Together these documents comprise the CPCL workflow artifacts.

The UML class diagram for the *Coordinator* is shown in Figure 9. The *Coordinator* class subscribes and publishes to the JBI and processes incoming messages from *Analysis & Planning* and the *Inspector*. The *TraceParser* class supports the analysis and processing of the workflow instance metadata received from the *Inspector*. This data reports the current execution trace of an instance. The *Coordinator* class must manage all the files that support processing a BPEL on the engine as well as effecting change within a BPEL. Thus, the *Storage* class provides a uniform directory structure for storing the supporting documents for each deployed BPEL including the relevant WSDL documents for invoked services.

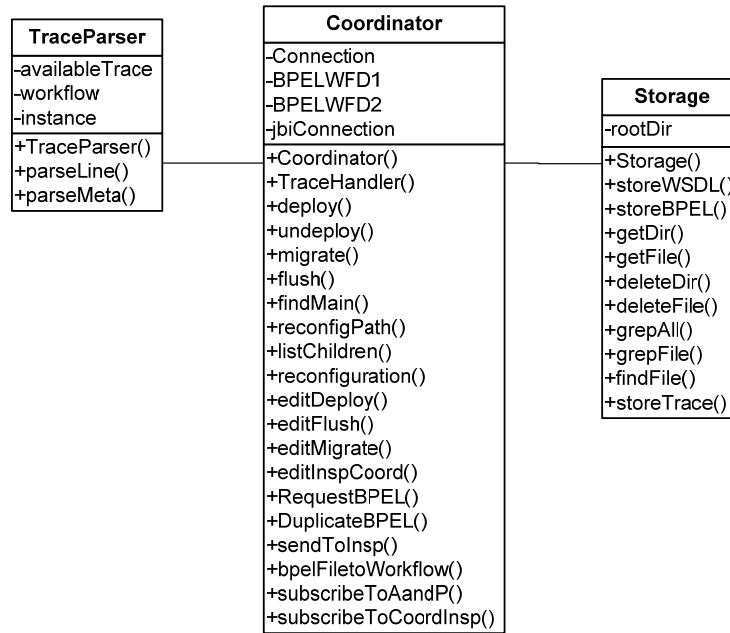


Figure 9. Class diagram for the *Coordinator* Component

The Inspector Component

The *Inspector* gathers metadata from UDDI registries and the *Bridge* (for newly deployed workflows only). From the UDDI, the *Inspector* retrieves the list of services that are available to the COI along with their WSDL documents. From the *Bridge*, the *Inspector* receives the workflow status information.

The metadata is used by the *Inspector* to identify the changes that the COI has dictated that it wants. For example, if a service becomes unavailable, then instead of a resultant failure of those workflows that rely on that service, dynamic update can be performed on the executing instances to use a predefined alternate and available service. As services become available via the UDDI registry, the decision to utilize them is more difficult. If immediate action is taken to include a newly available service, then instability (i.e. a service that frequently transitions between the available and unavailable states) has the potential to create a thrashing behavior in CPCL, through the repeated addition and removal of the service from workflows. It is apparent from detailed analysis that the ideal solution should consider multiple Quality of Service (QoS) parameters such as availability, length of time available, response time, and customer preferences before modifying multiple workflows to include a newly available service. Determining the best combination of factors can be COI specific, has garnered a great deal of research on its own, and is beyond the scope of the current project.

After evaluating the metadata, COI preferences, and UDDI contents for change requirements, the *Inspector* communicates change information to *Analysis & Planning* that may trigger a Reconfiguration Directive. This allows the *Inspector* prototype to focus only on accumulating

COI metadata and identifying changes. The *Inspector* functionality has been refined as shown in the use case diagram in Figure 10.

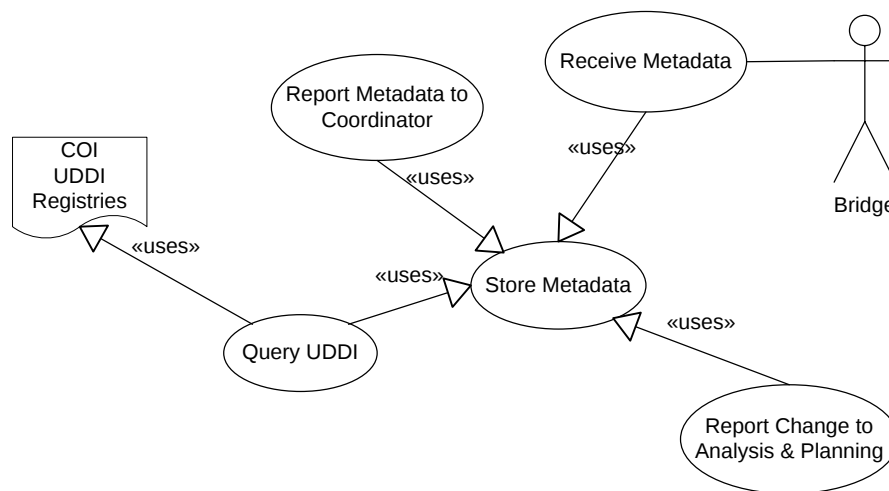


Figure 10. Use Case diagram for the *Inspector* Component

The *Inspector* and *Coordinator* communicate to each other through the JBI. The *Coordinator* requests information regarding the workflow instances. The *Inspector* receives instance trace messages from the *Bridge* and stores it. When requested by the *Coordinator*, the *Inspector* reports this trace information, when requested, to the *Coordinator*. The *Inspector* publishes change triggers and metadata through the JBI to the *Analysis & Planning* component when appropriate. Conditions, such as services that become unavailable or workflows with excessive failures, will prompt the *Inspector* to alert *Analysis & Planning*.

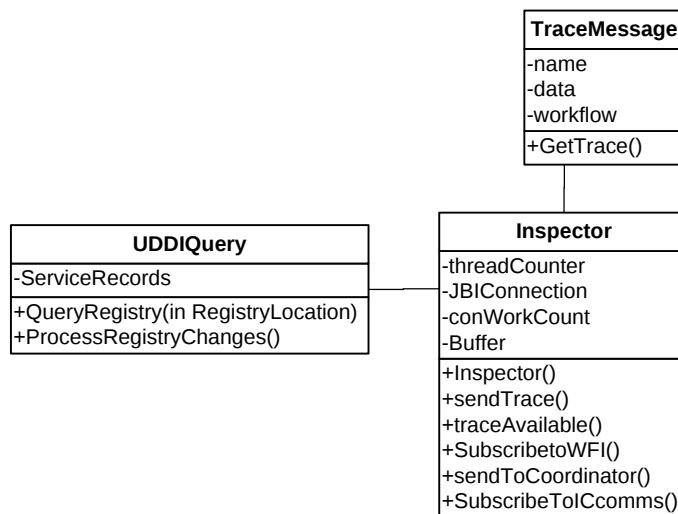


Figure 11. Class diagram of *Inspector* Component

The current class structure of the *Inspector* is shown in Figure 11. The figure indicates the functionality by which *Inspector* component queries the UDDI registry and determines changes in service availability, receives metadata from the *Bridge* and sends messages to the *Coordinator* via the JBI.

Analysis and Planning Component

Prioritizing the set of evoked changes occurs within the *Analysis & Planning* component. The *Analysis & Planning* component receives alerts from the *Inspector* when COI preferred services become available or unavailable. Thus, this component will adapt deployed workflows to minimize timeout errors from unavailable services and to maximize processing by utilizing preferred services over others when preferred services are available. In addition, COI users may request changes to a workflow through the NeWT component. NeWT then publishes change messages to *Analysis & Planning*.

An important function of the *Analysis & Planning* component is analysis of change requests. Since the *Inspector* automatically generates change requests based on service availability, it is possible that a user change request may conflict with *Inspector* generated change requests. User change requests must be given priority over auto-generated change requests in order to give users the maximum control of their environment. However, once conflicting change requests have been removed, the auto-generated requests designed to eliminate errors must take implementation precedence over other requests. This is so that error elimination is given priority over new functionality. If not, then users may request failed workflows to re-execute, creating a backlog of repeat execution while new functionality is deployed to the engine. Control of this processing must be data and policy driven to promote flexibility. Figure 12 illustrates the processing performed by the *Analysis & Planning* component.

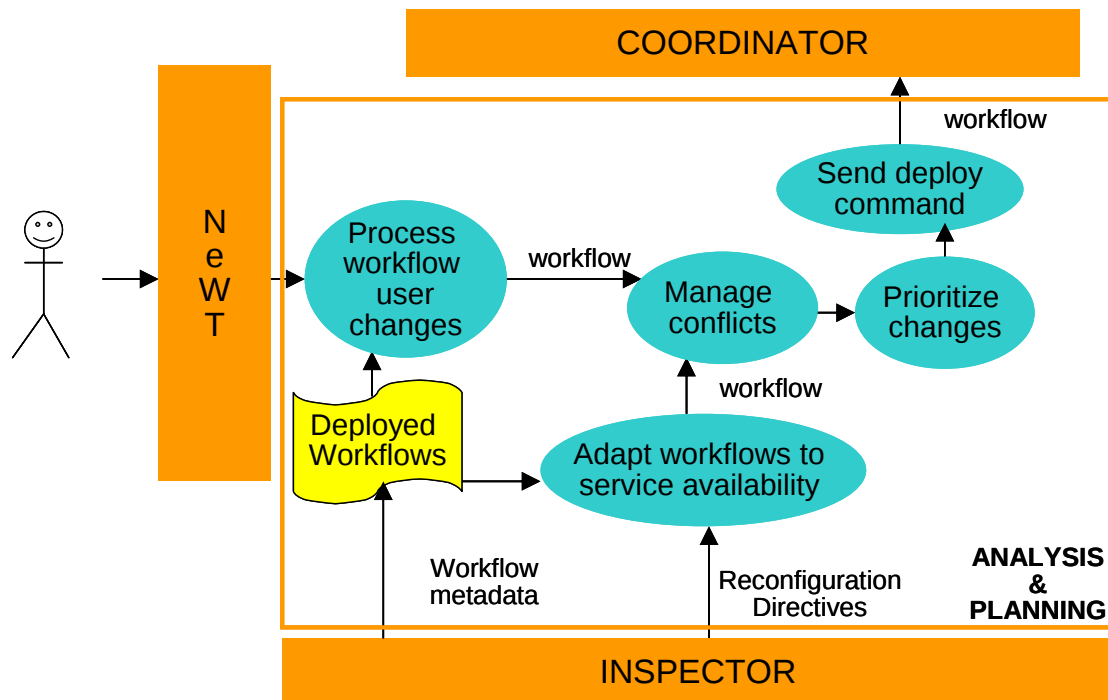


Figure 12. Analysis and planning processing

The *Analysis & Planning* component has been successfully developed to link with the implementations of the *Inspector* and *Coordinator* components in that *Analysis & Planning*

receives all the information needed to properly process the reconfiguration plan and create change requests. Thus, basic change requests are sent from *Analysis & Planning* to the *Coordinator* resulting in appropriate reconfiguration actions. The management of automated changes using the goal preference table and dependency tables is discussed in the TCIL loop information.

Figure 13 displays the use case diagram for the *Analysis & Planning* Component.

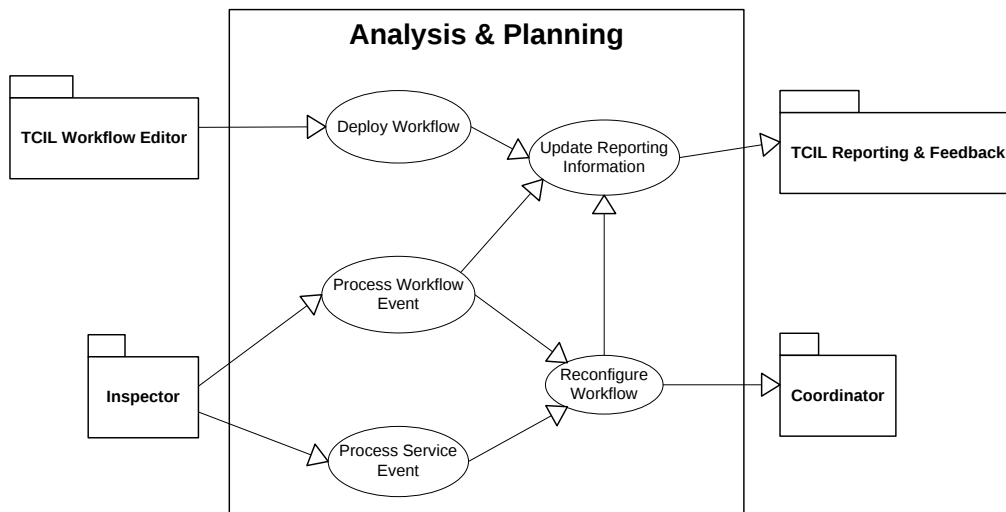


Figure 13. Use Case Diagram for the *Analysis & Planning* Component

Reconfiguration Planning

COI policies are the foundation for workflow change decision making. There are three main areas of metadata that help to define the COI and workflow policies. There are three main areas of metadata that help to define the COI and workflow policies: 1) Dependencies among services and workflows, 2) COI service preferences, and 3) functionally similar services that enable workflow adaptation.

Establishing the dependencies among workflows and services is instrumental in identifying conflicting change requests that, if enacted simultaneously, would threaten the stability of a community workflow. Processing a BPEL specification and its supporting documents provides the data necessary for identifying dependents. Thus, dependencies can be gleaned from the deployed workflow definitions. Service preferences govern workflow behavior in the environment. The workflow environment changes when services become available or unavailable, services take errors, or workflows fail. In addition, some conditions may require a workflow to process information quickly and generate results in the shortest time frame. Other conditions may require the most accurate workflow results no matter the required processing time. COI service preferences handle such variations in environmental conditions allowing the workflow to be automatically adapted to a variety of conditions in addition to accepting manual, user directed workflow changes. These preferences also establish an order of importance among

services. The order is based on community specific goals and groups services providing similar functions but varying by performance, precision, or service provider.

The COI preference data form the foundation for decision making. This data is formed with the understanding that changes to a workflow will be prompted by an overarching community goal. For example, if the user requires an answer immediately then processing time should be minimized. In an alternate situation, accuracy or precision may be the driving user goal. In a precision situation, processing time is of little concern while maximizing analysis and evaluation of all input conditions will take precedence. Table 3 shows a sample COI Preference designation for one of the example scenarios used to validate NeWT. The scenario will be discussed in more detail later in the report. A preference of Speed places priority on querying only the chemical sensors closest to the building on fire (given a value of 1). QueryChemicalSensors, the functionally similar service that takes more processing time to query all the chemical sensors on base and is given a lower preference (value of 2). The next goal identified is Precision. When this goal is designated, then all the base chemical sensors should be queried. In this way, the COI maximizes the variable input into the workflow and enables the most accurate results for chemical plume prediction. Similarly, two services for performing the plume analysis are also prioritized based on the community goals of speed and precision.

By using this data driven method, a COI may easily establish goals that will drive the automatic adaptation of their workflows. The adaptations may be defined to occur based on days of the week, volume of workflow traffic, or preferred business partnerships. For CPCL, the *Analysis & Planning* component supports user designation of the currently applicable goal. The identified design is extendable to incorporate automatec goal setting. In addition, the COI may override the automated environment modification at any time by manually deploying a change to a community workflow, thus maximizing community control over the workflow environment.

Table 3. Community Goal-Preferences

Workflow	Goal	BPEL Statement	Role	Value	Service
TCA	Speed	Invoke... QueryClosestChemSensors	SensorReading	1	ChemicalService
TCA	Speed	Invoke... QueryChemicalSensors	SensorReading	2	ChemicalService
TCA	Precision	Invoke... QueryChemicalSensors	SensorReading	1	ChemicalService
TCA	Precision	Invoke... QueryClosestChemSensors	SensorReading	2	ChemicalService
TCA	Speed	Invoke... PlumeAnalysis	PlumeAnalysis	1	PlumeServiceA
TCA	Speed	Invoke... PlumeAnalysisAdvanced	PlumeAnalysis	2	PlumeServiceB
TCA	Precision	Invoke... PlumeAnalysisAdvanced	PlumeAnalysis	1	PlumeServiceB
TCA	Precision	Invoke... PlumeAnalysis	PlumeAnalysis	2	PlumeServiceA

As workflows are deployed, a goal may be defined for the workflow or the default goal of *priority* may be used. The services and other workflows invoked inside the deployed WFD are identified as dependents. COI UDDI registries are then monitored by the *Inspector* to identify services that become available or unavailable. If a service's availability changes and it is found in the goal-preferences table, then the referenced workflow is evaluated to see if an adaptive change is defined for the combination of workflow, workflow goal, and available services.

Plan Rule Formation

A change can be triggered by different events in the environment. The environment taken into consideration is a COI and the set of workflows within that community, which we refer to as a Community of Workflows (CWF). A Reconfiguration Directive specifies a variety of conditions and alterations necessary for the CWF. The Reconfiguration Directive is a 3-tuple consisting of (Type, Request, ActionOverride), where Type designates whether the reconfiguration applies *directly* to a specific workflow, *globally* to the CWF, or is the result of a change in COI defined workflow goals (such as speed or precision). Multiple services may offer the same or competing operation. Thus, the Request tuple element is used to clearly indicate the operation and service being referenced by a particular Reconfiguration Directive in order to ensure that a newly inserted operation references the desired service. The Request identifies the correct service via the service's WSDL document, which provides information on any new operation specified in the Reconfiguration Directive.

The Request tuple element has the following embedded structure:

<cmd> <{BPEL-stmt1}> { in | from } <Workflow> {at | with} <BPEL-stmt2><WSDL>

- The cmd is the command insert, delete, or replace.
- BPEL-stmt1 is a valid atomic statement in a BPEL orchestration of a workflow.
- Workflow is a reference to a specific, deployed workflow with one or more executing instances.
- BPEL-stmt2 is also a valid atomic statement in a BPEL orchestration.
- WSDL provides a WSDL file name. For insert and replace commands, this optional argument provides the necessary information to correctly identify the service providing an operation specified in BPEL-stmt1 (for insert commands) or in BPEL-stmt2 (for replace commands).

When the COI modifies the goal for a workflow or services change state with regards to online availability, a *ProcessWorkflowGoal* operation processes the goal-preferences table to determine what changes are generated by the change. The *Analysis & Planning* operation *ProcessChangeRequests* executes changes by grouping them by workflows to efficiently implement the changes and resolve conflicting changes. Change request processing prevents inconsistent changes due to conflicting change requests.

Management of the instances running on reconfigured workflow definitions is handled by the *Coordinator* in accordance with the COI rule table shown below in Table 4. The COI rules are executed in an order such that the first rule in the list that is applicable is used to determine a change action for managing the instance. Predicate statements used to analyze the change request and the current state of the workflow instance to determine the best reconfiguration action for the situation.

Table 4. COI Rules for Change Actions

	Predicate 1	Predicate 2	Predicate 3	Result
#	If (ChangeEvent.reason = COI-driven) & (isset(actionOverride))	If last(snapshot(i)) follows tp _{wf} d-id	If ChangeBlock contains "invoke WS _k " with transactional(WS _k) = true	Then WFIactionType =
1	TRUE	TRUE	TRUE	actionOverride
2	TRUE	TRUE	FALSE	actionOverride
3	TRUE	FALSE	TRUE	actionOverride
4	TRUE	FALSE	FALSE	actionOverride
5	FALSE	TRUE	TRUE	flush
6	FALSE	TRUE	FALSE	restart
7	FALSE	FALSE	TRUE	migrate
8	FALSE	FALSE	FALSE	migrate

The resulting WFIactionType is determined according to the following reasoning rules. Emphasis is placed on migrating instances where possible, but not in such a way that an invalid WFD would be automatically created.

- Rules 1-4: These rules contain instances where Predicate 1 evaluates to true indicating that the ChangeEvent is user-driven and that the COI has set an actionOverride policy to be used. The resulting WFIactionType should take the value of *actionOverride*, which can be any of the 4 definition actions (flush, abort, restart, or migrate).

- Rule 5: Instances where predicate 2 and 3 evaluate to true indicate that the instance is executing past the transitional point and that the change block contains transactional services. The action the *Coordinator* uses for reconfiguration is to *flush* the instance, allowing it to finish any transactional statements which have already begun execution.
- Rule 6: When only predicate 2 evaluates to true the instance is executing beyond the transitional point, but it does not involve transactional services. The instance can *restart* to use the new WFD with no negative consequences from partially executed transactional statements.
- Rule 7: When predicate 3 evaluates true with predicate 2 evaluating to false the change block involves transactional service but the instance has not executed within the change block. The instance can *migrate* successfully assuming the transactional point occurs before all of the web service calls associated with the transactional event.
- Rule 8: If all predicates evaluate to false indicating the instance has not executed the change block and the change block does not involve transactional services. The instance can *migrate* successfully.

Meta-data Specification

Definition and instance meta-data specification has been completed in combination with additional JBI Information Object schema definitions for communication used among CPCL components. Meta-data definitions for WFD and associated instances are shown in Figure 14 and Figure 15. Minor changes, including the addition of a <version> element to the WFI-MD have been included in accordance with the need made apparent from development of the *Inspector* and *Coordinator*.

<wfd-md>	::= <wfd-id> " , " <author> " , " <specification> " , " <wsdls>
<wfd-id>	::= <characters>
<version>	::= <integers> " , " <integers>
<author>	::= <username> <wfi-id>
<username>	::= // string with a valid username denoting who authored the WFD
<specification>	::= // valid XML string representation of a BPEL process
<wsdls>	::= <BPELwsdl> <optionalWSDLs>
<BPELwsdl>	::= // XML string representation of the WSDL file for this BPEL process
<optionalWSDLs>	::= <wsdl> <optionalWSDLs> ""
<wsdl>	::= // XML string of a service WSDL used in this BPEL process

Figure 14. Definition Metadata (WFD-MD)

<wfi-md>	::= <wfi-id> " , " <wfd-id> " , " <trace> " , " <status>
<wfi-id>	::= // instance ID obtained from workflow engine
<wfd-id>	::= <characters>
<version>	::= <integers> " , " <integers>
<trace>	::= <BPELstatements> ""
<BPELstatements>	::= <BPELstatement> <BPELstatements> ""
<BPELstatement>	::= // valid BPEL element, a member of the main process sequence
<status>	::= "FAULTED" " STALE" "CANCELLED" "COMPLETED"

Figure 15. Instance Metadata (WFI-MD)

CPCL Component Schema Definitions

This section contains InfoObject metadata schemas as defined within the JBI for the communication between the components of the double-loop workflow reconfiguration processing. The *Analysis & Planning* component sends a change request to the *Coordinator* to cause a new workflow to be deployed such as when a user manually modifies a workflow specification and places it to be deployed onto the engine. Shown in Figure 16, the **wfd-id** field provides a unique reference to the exact workflow definition to be modified. The **action** field provides the action to take on the workflow definition. **Stmt1** is the statement to find inside the workflow to replace, the statement to insert before **stmt2** or the statement to be deleted. If the action is deploy or undeploy, **stmt1** will be set to null. **Stmt2** is null for delete, deploy, and undeploy actions. If the user has specified an action to take on affected instances, that action is specified in the **actionOverride** field. A payload containing workflow artifacts is attached if the action is deploy.

```
<xs:element name="changeRequest">
  <xs:complexType>
    <xs:sequence>
      <!-- can be either the name of the wfd, or the word "all" -->
      <xs:element name="wfd-id"/>
      <!-- either "replace", "insert", "delete", or "deploy" or "undeploy". -->
      <xs:element name="action"/>
      <!-- the bpel statment to either "replace", "insert", or "delete". -->
      <!-- if action is "deploy" this is set to the string "none" -->
      <!-- if action is "replace", this is the statement to replace stmt1 with -->
      <!-- if action is "insert", stmt1 will be inserted immed. after every stmt2 found -->
      <xs:element name="stmt1"/>
      <!-- if action is "delete" or "delete", this is set to the string "none".-->
      <xs:element name="stmt2"/>
      <!-- used to override the default reconfiguration action, -->
      <!-- can be set to the string "none" if not specified by the COI -->
      <!-- or if an inspector-driven reconfiguration -->
      <xs:element name="actionOverride"/>
      <!-- if action is "deploy", the payload of this message follows the syntax used within -->
      <!--Coordinator changeRequest. The payload carries a COI-driven reconfiguration -->
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 16. *Analysis & Planning* Change Request to *Coordinator*

A workflow event (Figure 17) is sent from the *Inspector* to *Analysis & Planning* to report when a workflow has been deployed or undeployed. The **wfd-id** identifies the workflow involved, the **author** identifies the COI user, the **specification** is included along with the **wsdl** for the workflow as well as all invoked services. The **event** identifies if the workflow has been deployed or undeployed.

```

<xs:element name="workflowEvent">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="wfd-id"/>
      <xs:element name="author"/>
      <xs:element name="specification"/>
      <xs:element name="wsdls">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="wsdl"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    <!-- event type can be: deployed, undeployed -->
    <xs:element name="event"/>
  </xs:sequence>
</xs:complexType>
</xs:element>

```

Figure 17. Inspector Workflow Event to Analysis & Planning

When the *Inspector* component recognizes an event among the services utilized by the COI, an *Inspector* service event (Figure 18) is reported to the *Analysis & Planning* component. The **service** element specifies the exact web service which has changed status, the **status** element specifies if the service has either come online or gone offline, **wsdlLocation** and **wsdl** elements hold the respective wsdl information for the service which will be used by *Analysis & Planning* later to manage reconfiguration.,

```

<xs:element name="serviceEvent">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="service"/>
      <xs:element name="status"/> <!-- either "online" or "offline" -->
      <xs:element name="wsdlLocation"/>
      <xs:element name="wsdl"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figure 18. Inspector Service Event Sent to Analysis & Planning

After *Analysis & Planning* has determined the appropriate change request to send to the *Coordinator*, the *Coordinator* component of CPCL begins to enact the reconfiguration. When suspended instances are managed by the *Coordinator*, instance actions are sent from the *Coordinator* to the *Bridge* that enacts the appropriate action. To assist in reconfiguration, the *Coordinator* requests metadata from the *Inspector* by sending a wfd-request message shown below.

```

<xs:element name="wfd-request">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="wfd-id"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figure 19. Coordinator wfd Request for Metadata to the Inspector

The *Bridge* reports meta-data for each deployed workflow definition (WFD) and each suspended, stopped, completed, or aborted instance. The format follows that as defined in the previous section and is collected by the *Inspector* so that it can be sent back to the *Coordinator* when it receives the wfd-request message.

Additionally preliminary specification of the links between the NeWT and CPCL components has begun during the previous reporting period. For example, a link between the COI UDDI repository and CPCL components has been defined allowing services to be registered or removed from the community UDDI. As shown in Figure 20, the **serviceName** and **wsdlLocation** are published to the JBI along with an **action** of "register" or "unregister".

```

<xs:element name="uddiAction">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="serviceName"/>
      <xs:element name="wsdlLocation"/>
      <xs:element name="action"/> <!-- either "register" or "unregister" -->
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figure 20. NeWT Interaction with UDDI Interface Specification

Tasking and Control Interface Loop (TCIL)

TCIL is implemented as an Eclipse plug-in, with new views associated with workflow management. It extends an existing Eclipse BPEL project which provides a graphical user interface for workflow development with an underlying BPEL model using the Eclipse Modeling Framework (www.eclipse.org/bpel/). The Eclipse plug-in sends generated BPEL workflows to CPCL for deployment on a commercial workflow engine and managed for reconfiguration. TCIL extends the basic BPEL editor provided by the Eclipse platform by adding several user interface extensions and the ability to communicate with an extended version of CPCL capable of handling advanced reconfiguration (Figure 21). Using Eclipse, TCIL provides an extended BPEL editor, several new wizards associated with automatic reconfiguration, new views capable of showing reconfiguration options and reporting information, and a modified version of CPCL supporting advanced dynamism controls. TCIL is divided between three main functional pieces providing a controller interface, a workflow toolkit, and reporting and feedback mechanisms. Each is discussed in further detail in the section below.

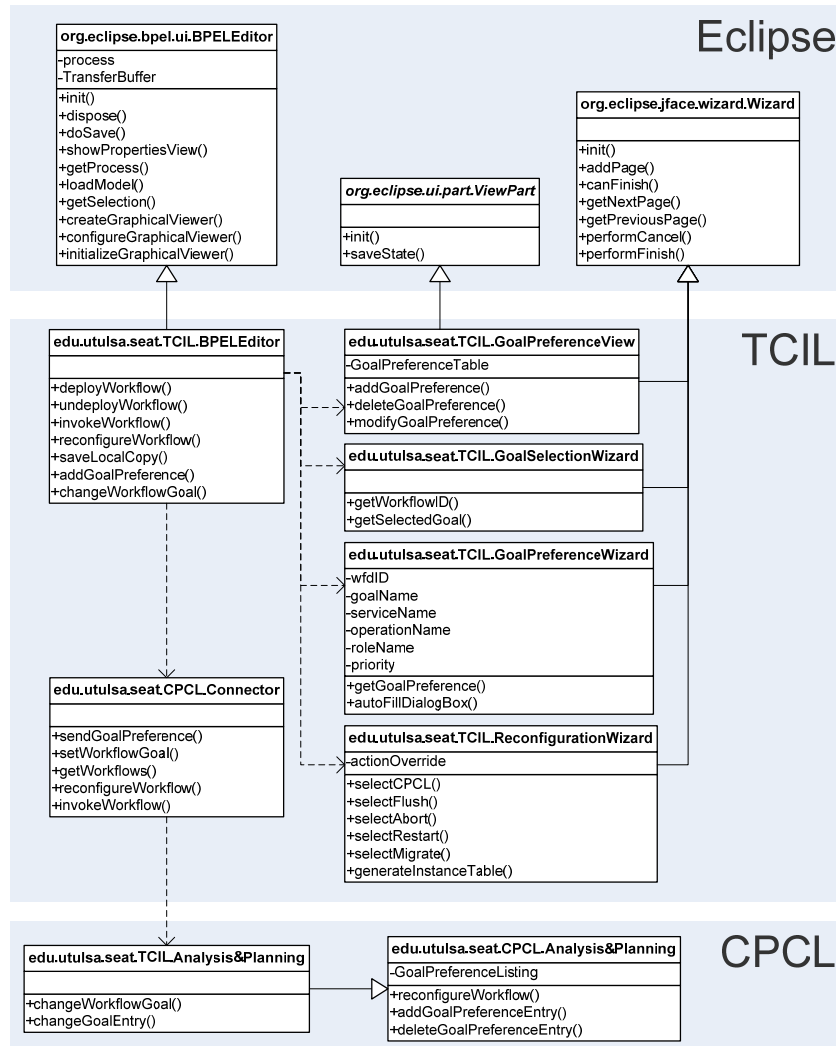


Figure 21. NeWT UML Diagram

Controller Interface

A Controller Interface is present within TCIL to assist COI members in determining the necessary functions within the Workflow Toolkit necessary to complete assigned tasks. Common workflow responsibilities include workflow development, reconfiguration, and execution. The Control Interface present in TCIL allows isolation between workflow responsibilities to be inserted if needed by the COI. A COI may require that certain users have access to execute workflows but not make modifications, requiring an access control mechanism to be inserted before allowing access to certain functionality within the Workflow Toolkit. Throughout the remainder of this paper we focus on the functionality TCIL provides to COI users tasked with workflow development and reconfiguration.

The Workflow Toolkit contains the implementation behind the Controller Interface and forms the majority of the functionality of TCIL. Pictorially, it is separated in the loop simply to signify that there are stakeholder designations that may restrict use of the toolkit. The Workflow Toolkit

sends workflow definitions to *Analysis & Planning* for deployment as well as reconfiguration information necessary for CPCL to execute properly.

Since TCIL extends an existing workflow editor, workflow developers can use the Workflow Toolkit like any other workflow editor to sequence the invocations of specific services. The BPEL editor provided by TCIL extends the basic features found in the `org.eclipse.bpel.ui.BPELEditor` package, providing the functionality to deploy, reconfigure, and invoke the workflow on CPCL, as shown in Figure 21. TCIL facilitates workflow creation using a graphical editor containing common workflow tasks which can be dragged and placed into an editor window.

Dynamic change controls are used within TCIL to provide each COI with the ability to influence how CPCL will handle reconfiguration of the workflow management system. The Workflow Toolkit provides access to dynamic change controls used to attach a priority to redundant services within specific workflows and the ability to group those priorities into goals for the workflow. Dynamic goal change controls exist to assist workflow developers in quickly switching the goal that a specific workflow is executing under. The NeWT architecture aids workflow developers in handling these workflow changes by allowing them to decide what specific change to make to those workflows managed by TCIL. With TCIL in place, CPCL can determine the appropriate reconfiguration action to take for affected instances.

Reporting and Feedback

Reporting within TCIL is accomplished using an Eclipse Modeling Framework (EMF) model of the data made available from the combined source of the workflow management system, CPCL, the COI, and the WS that the COI has deployed on an UDDI registry. The information is stored in an XML file format adhering to the model, and the built-in EMF viewer within Eclipse allows COI users to view the reporting information as it changes within the COI.

TCIL generates reports for each COI. The reporting viewer, shown in Figure 22, allows COI users to traverse information including: metadata about deployed workflow definitions, status information of running instances, information about the WS used by the COI, preferences the COI has related to reconfiguration, and information related to the results of previous reconfigurations. Details about specific reporting items are shown via a properties viewer provided by Eclipse.

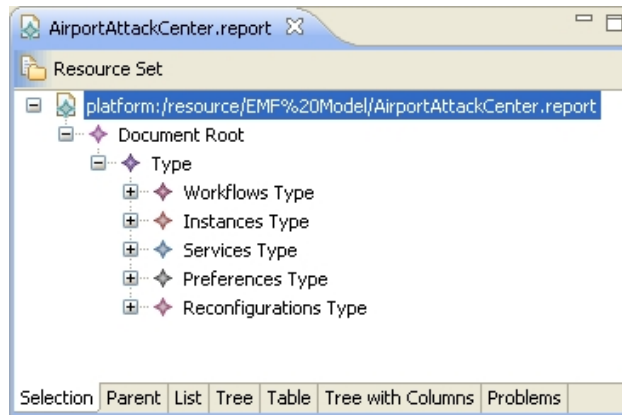


Figure 22. TCIL report viewer

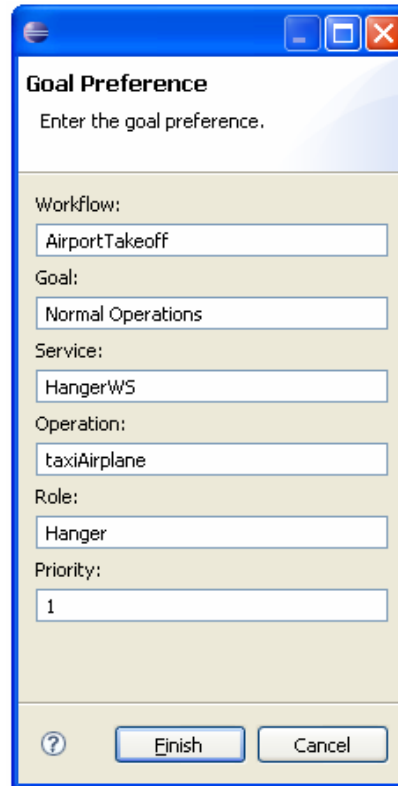
The EMF model used by TCIL supports the ability for a COI to create and register event adapters that wait for changes to the reporting model. When changes occur COI code can execute to determine if the change is significant enough to require COI involvement, possibly sending messages to workflow developers via alert dialog boxes or other COI-specific messaging platforms. Example of significant reporting events can include WS failures, the registration of a new WS with the COI UDDI, or the creation of new workflow definitions. The feedback mechanism allows TCIL to report information to the COI as soon as it is made available for analysis.

Workflow Controls

Dynamic Goal Controls

Workflow reconfiguration lets a COI adapt to the needs of stakeholders quickly and efficiently. TCIL builds upon a *goal-preference* structure found within CPCL, giving workflow developers the ability to embed reconfiguration directives in workflows without modifying their original BPEL syntax. The remainder of this section describes the specific features of TCIL provides that are related to dynamic goal controls.

COI users can add specific goals to workflows by selecting individual services in the workflow editor and opening the goal preference wizard. The wizard implements features of the *GoalPreferenceWizard* class (Figure 21), providing a mechanism for the workflow developer to define new goal and preferences for the workflow. The wizard, shown in Figure 23, allows the COI to define the name of a goal and the alternate service to use. Other information such as the workflow id, operation, and role are provided by TCIL based on the selected workflow task.



Goal Preference
Enter the goal preference.

Workflow:
AirportTakeoff

Goal:
Normal Operations

Service:
HangerWS

Operation:
taxiAirplane

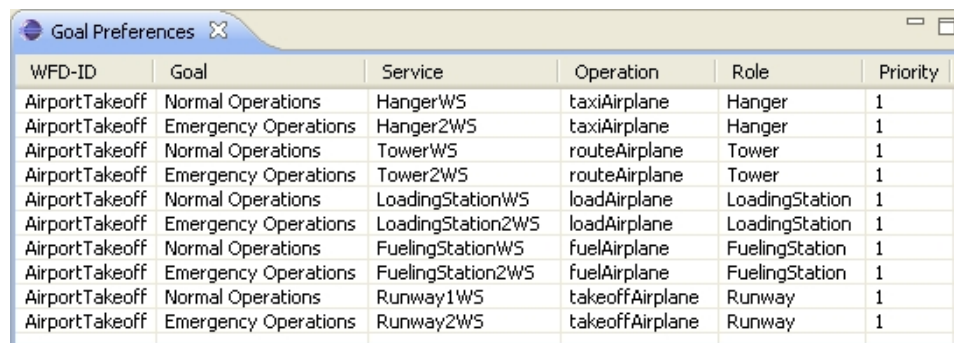
Role:
Hanger

Priority:
1

? Finish Cancel

Figure 23. Goal preference entry dialog

The goal preferences created by workflow developers are sent to CPCL to be stored and used in processing goal changes. A summary view of all current workflow goal preferences is made available inside the Workflow Editor as shown in Figure 24. The view lists of all current workflow goals and the service preferences that have been defined.



WFD-ID	Goal	Service	Operation	Role	Priority
AirportTakeoff	Normal Operations	HangerWS	taxiAirplane	Hanger	1
AirportTakeoff	Emergency Operations	Hanger2WS	taxiAirplane	Hanger	1
AirportTakeoff	Normal Operations	TowerWS	routeAirplane	Tower	1
AirportTakeoff	Emergency Operations	Tower2WS	routeAirplane	Tower	1
AirportTakeoff	Normal Operations	LoadingStationWS	loadAirplane	LoadingStation	1
AirportTakeoff	Emergency Operations	LoadingStation2WS	loadAirplane	LoadingStation	1
AirportTakeoff	Normal Operations	FuelingStationWS	fuelAirplane	FuelingStation	1
AirportTakeoff	Emergency Operations	FuelingStation2WS	fuelAirplane	FuelingStation	1
AirportTakeoff	Normal Operations	Runway1WS	takeoffAirplane	Runway	1
AirportTakeoff	Emergency Operations	Runway2WS	takeoffAirplane	Runway	1

Figure 24. Goal preference table for example workflow

The ability to add goals and preferences to workflow definitions does not provide a COI with enough functionality to dynamically change workflow goals. The NeWT toolkit will continue to execute the default workflow definition created by the workflow developer until the goal for that workflow is changed. The "GoalSelectionWizard" of TCIL (Figure 21) enables workflow developers to quickly and effortlessly switch the goal a workflow is executing under. The

wizard, shown in Figure 25, gives the COI easy access to change the goal for workflows without worrying about managing the reconfiguration that takes place inside of CPCL.

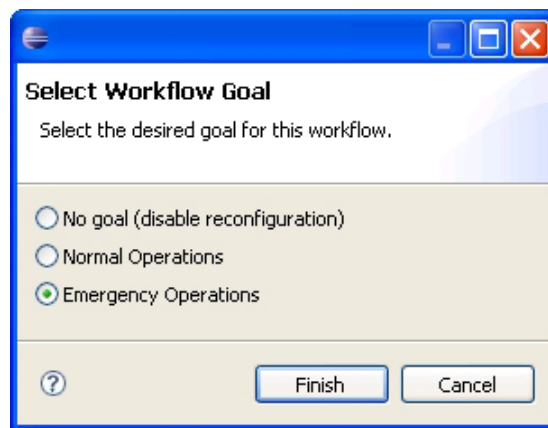


Figure 25. Workflow goal selection dialog box

Reconfiguration of the workflow is performed via the interface shared between TCIL and CPCL. The *Analysis & Planning* component of CPCL is extended to monitor for goal change requests from TCIL as shown in Figure 21. The core functionality contained within CPCL is used to manage changes in service availability, but not goal changes. The extensions made by NeWT examine all the services used by the workflow, matching the goal preferences defined and replacing service invocations. After generating the new workflow definition, a number of instances may require reconfiguration to take advantage of the change. CPCL manages the reconfiguration of affected workflow instances dynamically.

Results of the reconfiguration are updated in the Reporting component of TCIL, as shown in Figure 26. Reporting allows the COI to review which preferences were used to generate the new workflow definition and the goal that triggered the change.

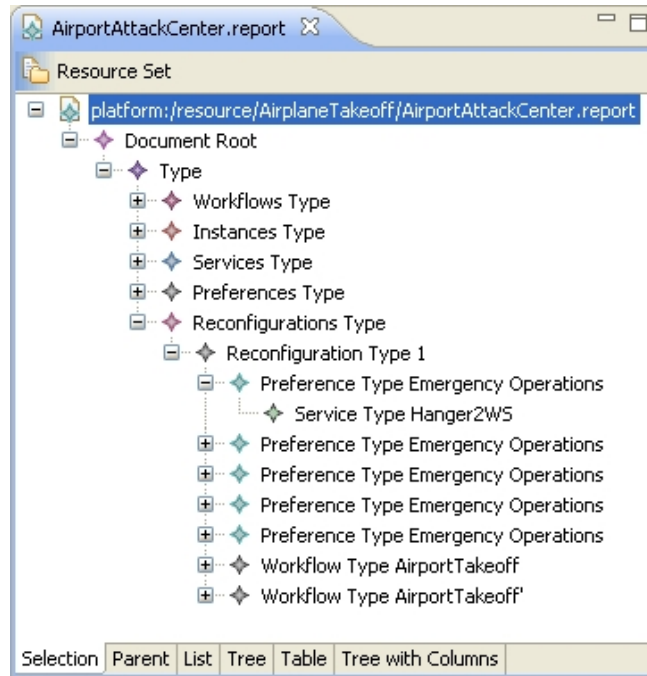


Figure 26. Report for goal change reconfiguration

Dynamic Workflow Change Controls

TCIL provides automatic reconfiguration, as described in the previous two sections, as well as the ability to perform manual reconfiguration. Manual reconfiguration can be triggered by a variety of COI events. For example, a COI may decide to modify an existing workflow due to changing business practices (such as using an upgraded service) or due to WS-specific reasons (such as a COI choosing to no longer utilize a particular service). TCIL accesses functionality contained within CPCL to assist workflow developers in managing manual reconfiguration. Specifically, manual reconfiguration requires exposing certain CPCL functions to dictate the reconfiguration directly via the Workflow Toolkit.

After a workflow developer manually changes a workflow, TCIL asks the user to choose one of the reconfiguration directives used for running instances of the workflow that are affected by reconfiguration. Figure 27 shows the wizard that is displayed when directed changes are made. CPCL's reconfiguration strategy places emphasis on migrating instances, which may not always be the best choice, requiring COI intervention at this level. Therefore, TCIL presents the option to override the migration default reconfiguration directive using the wizard where an *actionOverride* message is coupled with the chosen reconfiguration directive and delivered to *Analysis & Planning*.

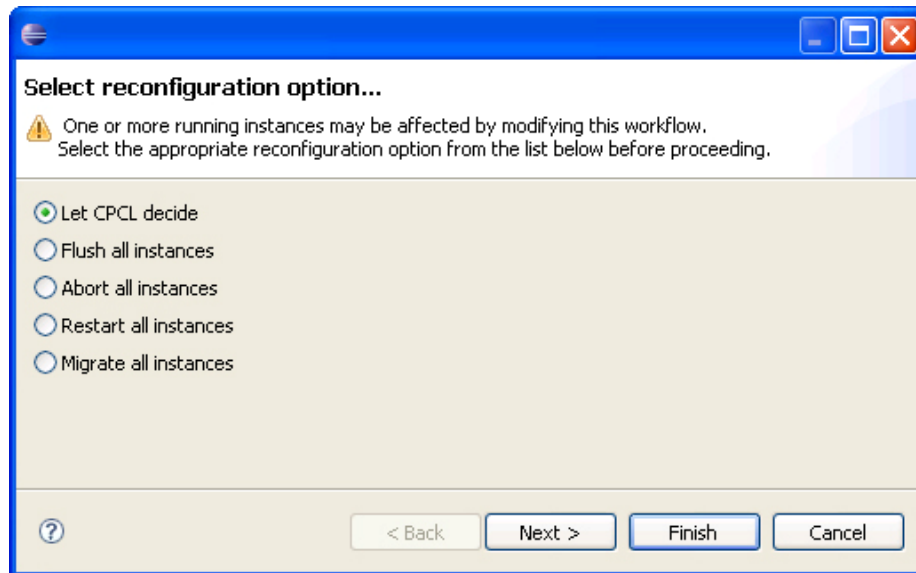


Figure 27. Workflow reconfiguration options

When the choice of reconfiguration directive is made within TCIL, CPCL manages the deployment of the new workflow and the necessary instance migrations. After enacting the reconfiguration directive, reporting within TCIL is updated to contain the reconfiguration information and instances affected by the change, as shown in Figure 28. Trace information is provided as the workflows continue to execute, which allows the COI to inspect instances that warrant investigation at a later point in time.

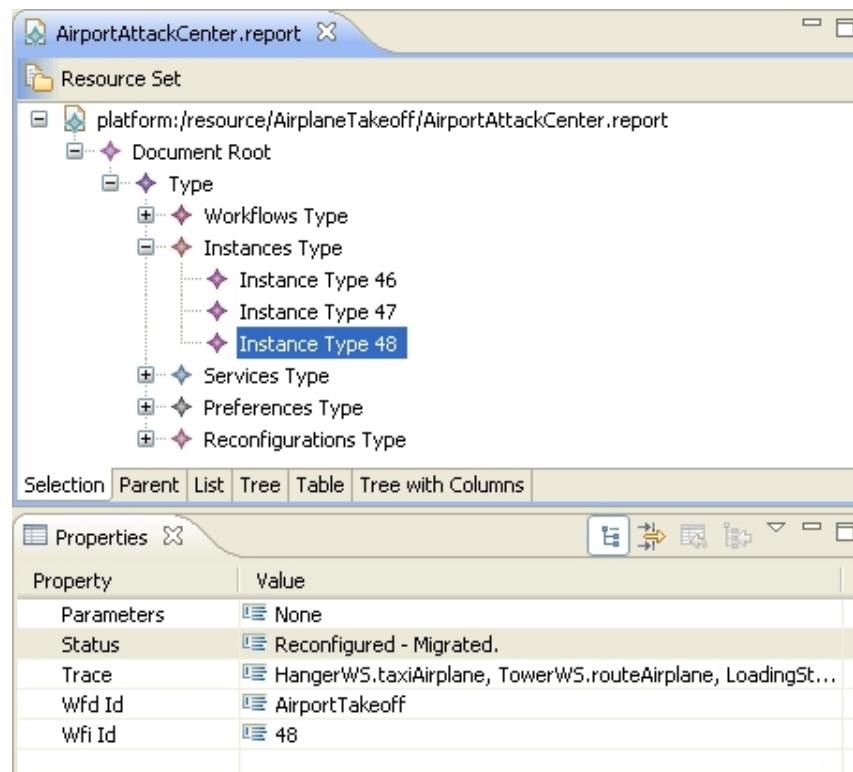


Figure 28. Workflow reconfiguration results

Jython Fuselet Wrapper

A Web Service wrapper that encapsulates Jython-coded Fuselets has been developed to assist in the development of test cases for use in NeWT. Previous research has shown that the functionality within Jython Fuselets can be completely re-written to run within a web service architecture. JyWAR assists in the migration of existing Jython Fuselets that are developed, so that they may be included easily within workflows. A set of requirements for the toolkit included the following functionality.

- Toolkit should be an Eclipse-based plug-in capable of interfacing with an existing Jython FDE and reading existing Fuselet specifications.
- The toolkit should be able to automatically create and deploy a packaged file that contains the necessary files to call a wrapped Jython Fuselet in the JBI.
- Fuselet meta-data should be made available via the web service interface or WSDL document, so that this information can be queried.
- The Wrapper should be able to handle calls to both stateful and stateless Fuselets. Operations to invoke stateful Fuselets should block, operations to invoke stateless Fuselets should be non-blocking.
- Wrapped Fuselets should still be able to communicate with a running JBI.

This JyWAR plug-in reads an existing Jython Fuselet specification, automatically creates an appropriate wrapper to the Fuselet, deploys the packaged file to an application server, and registers the service in an UDDI registry. The wrapper recognizes JBI subscription requests and modifies them to JBI queries for the invocation of Web Services. The wrapper was developed to support the Fuselet specification language provided to us by the Air Force.

A use case diagram is shown in Figure 29 and illustrates the interaction among the Web Service requestor, the wrapper, the Fuselet, and the JBI. A user or workflow will make a request to the wrapping Web Service, the application server receives the request and invokes the appropriate Web Service. The Fuselet Web Service will parse the Fuselet metadata and runs the Jython Fuselet. A JBI connection is utilized to publish to and query from the JBI.

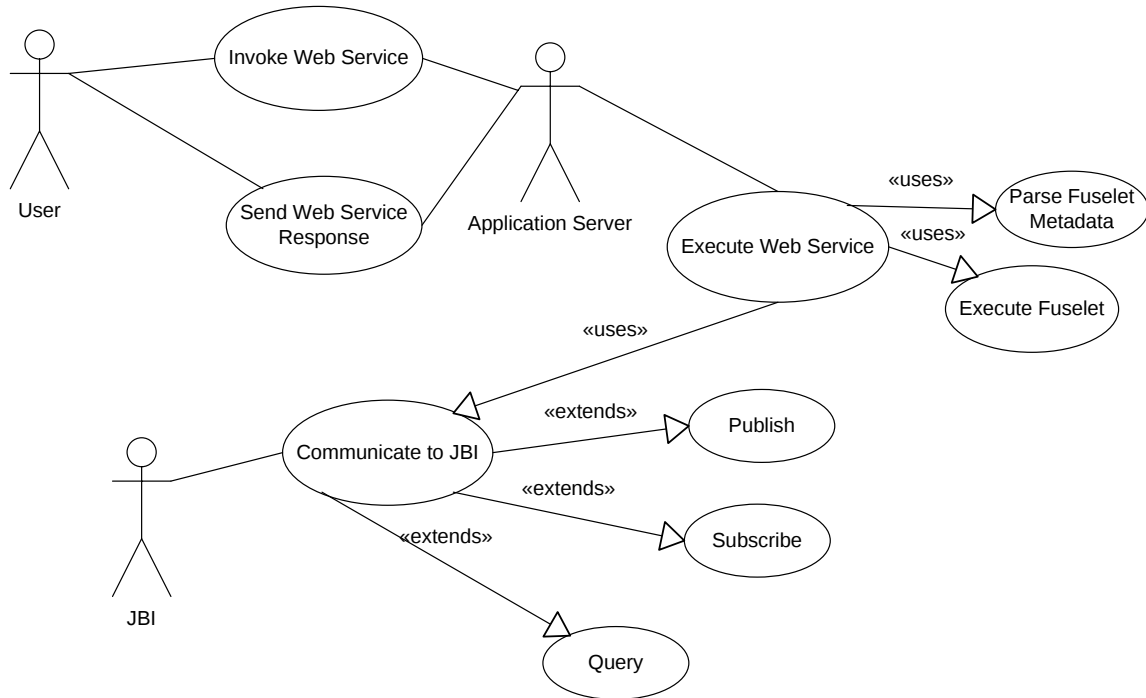


Figure 29. Fuselet Wrapper Use Case Diagram

When executed, the plug-in provides an interactive step by step process, as shown in Figure 30, for creating a Web services. When launched from within Eclipse, the Fuselet developer can select an existing Fuselet in the workspace allowing JyWAR to fill in necessary deployment parameters automatically. Preferences are saved so that multiple Fuselets can be created quickly.

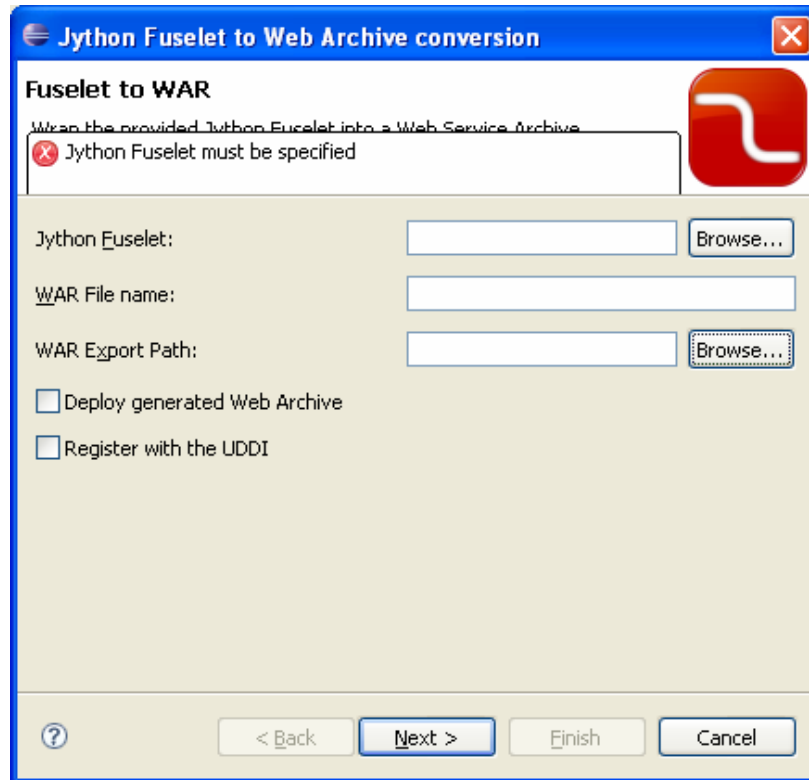


Figure 30. Automated Fuselet to Web service Wrapping

Example Jython Fuselets have been successfully wrapped using the plug-in deployed on a Java Sun Application Server (Glassfish J2EE). These include example Jython Fuselets provided by the Air Force as well as in-house Fuselets developed to test the plug-in and NeWT (see “Air Base Battlefield Scenario” section later for more details about these Jython Fuselets).

Example Scenarios

Terrorist Chemical Attack Scenario

An example web service scenario simulating a Terrorist Chemical Attack (TCA) and the release of chemicals into the air due to the attack has been created for testing in conjunction with the development of CPCL. In the example, chemical sensor readings are simulated and a chemical plume model is created based on the readings and live weather conditions. The architecture for this scenario is composed of Web Services, a BPEL workflow, a web portal, and the JBI. The Web Services are invoked from a BPEL workflow and each publishes and queries the JBI for information. This simulation utilizes weather data downloaded from the National Oceanic & Atmospheric Administration (NOAA). TCA enables analysis of different workflow goal types.

When a fire occurs, the immediate goal of the emergency response team will be to get feedback as quickly as possible. Plume contents along with the general plume location and direction is sufficient to allow emergency response to begin. Shortly thereafter, more detailed information will be required especially as evacuations are evaluated. Further, as emergency response

personnel enter the field, their chemical sensors may be included in the plume modeling data to enable the most precise prediction that is possible. Redundant services can be defined for querying chemical sensors directly surrounding the fire, querying sensors in a wider area, and querying sensors on emergency personnel. Querying fewer sensors means a faster response time. Different plume modeling services may be offered from different vendors. These models may be chosen based on priority, availability, or weather conditions.

The workflow can utilize dynamism by querying a small set of sensors to generate an initial plume estimate. Once an emergency response has begun, a dynamic change to the workflow can be performed to query all area chemical sensors and generate a more detailed prediction of the chemical plume. Occasionally, weather data for the desired weather station is unavailable via NOAA. When this occurs, the user can enact a workflow change to utilize a different weather station. Figure 31 illustrates the processing flow for the TCA scenario. Dotted lines designate redundant services that have been defined for replacing services in the normal flow as needed.

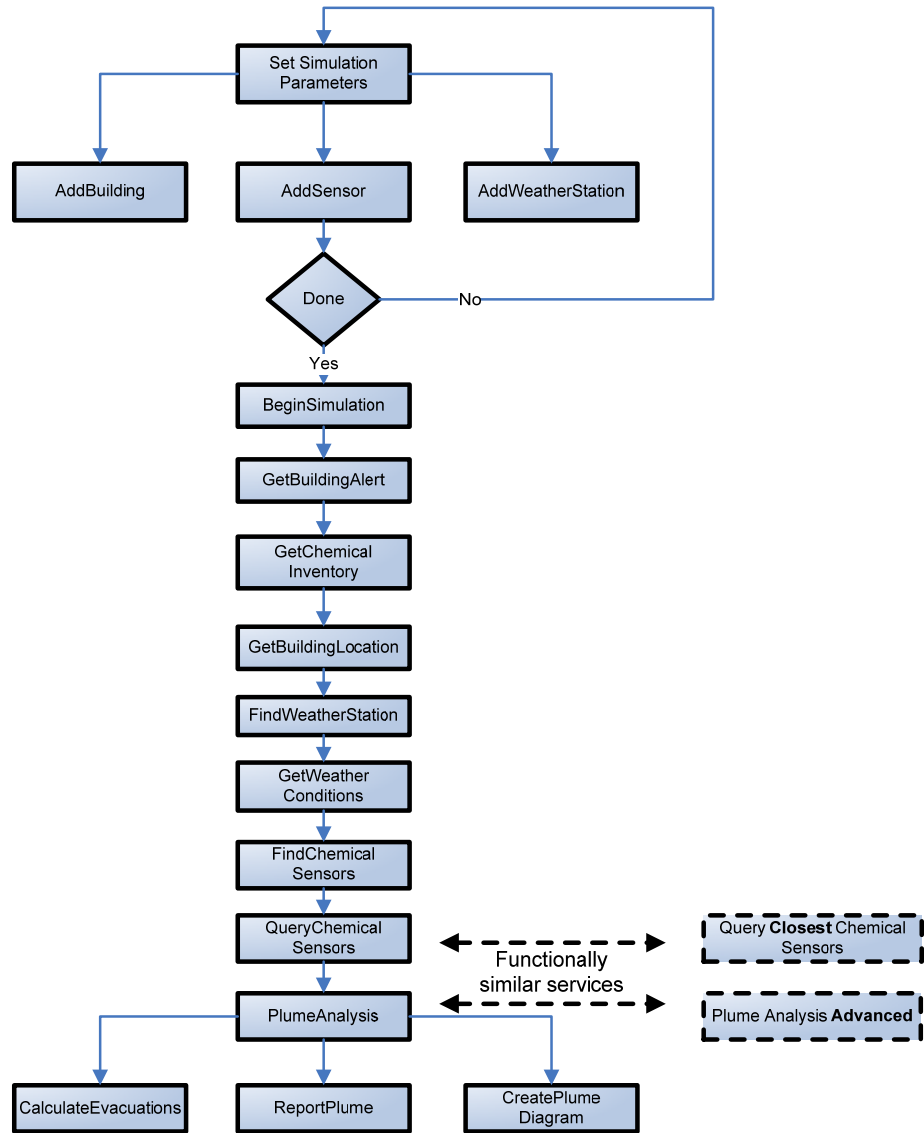


Figure 31. TCA process flow

The simulation utilizes a web portal to establish simulation parameters. The portal provides an interface for defining the simulation parameters such as the simulation area radius, chemicals, buildings, chemical stores in the buildings, and area chemical sensors. In Figure 32, three chemicals are defined for an example simulation. The portal includes a map of the defined chemical stores and buildings defined during setup as shown in greater detail in Figure 33.

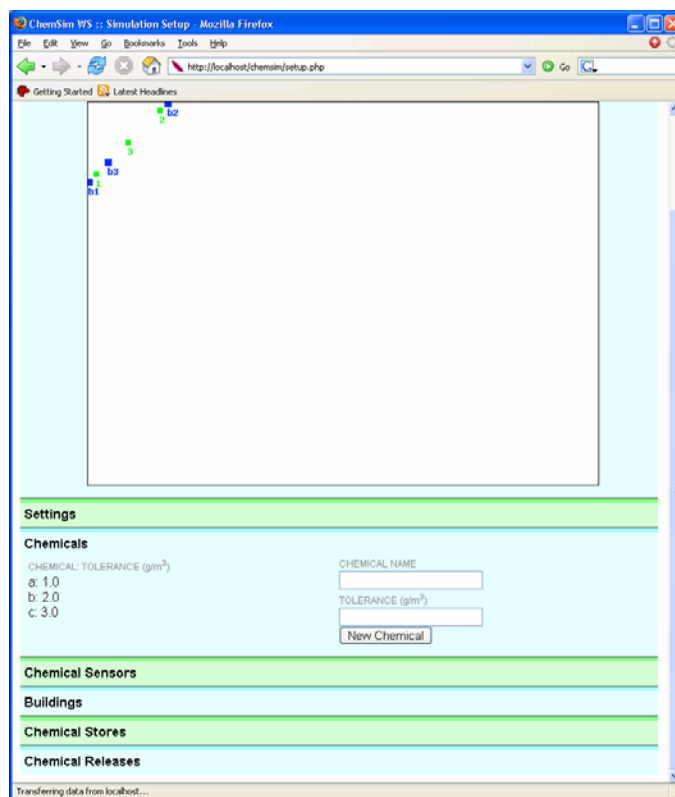


Figure 32. Terrorist Chemical Attack - Simulation Setup

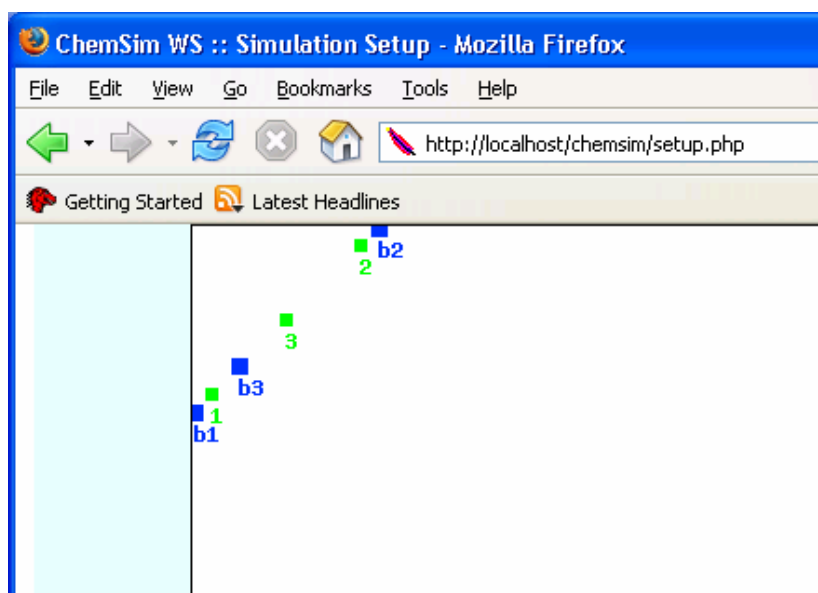


Figure 33. Terrorist Chemical Attack - Simulation Map

Currently documentation of the exact installation and setup for the ChemSimWS simulation is being developed for easy transition to the Air Force at the end of the research effort. The simulation's Web services are deployed on a J2EE server. The portal calls the services to define the initial parameters and a workflow invokes the services to execute the actual simulation. Redundant service operations are available to provide different weather station data.

Figure 34 illustrates the two operations available to retrieve weather information as shown in the J2EE server's testing interface. A rapid simulation is available for making a quick estimate of plume direction. More detailed analysis can be performed by a redundant plume analysis service. These operations are being used in the dynamism testing with the CPCL system.

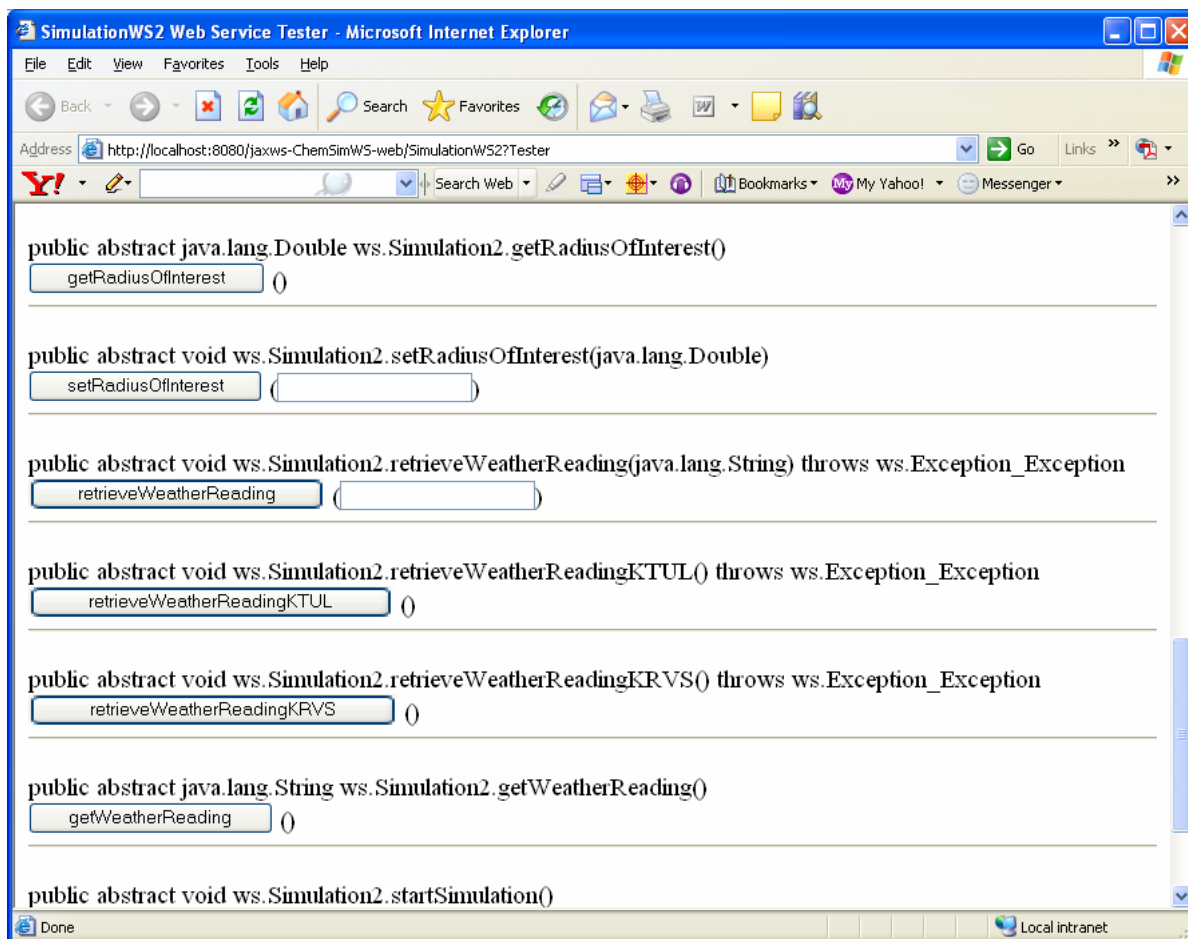


Figure 34. Terrorist Chemical Attack - Simulation Web Service Test Interface

Air Base Battlefield Scenario

The second scenario investigated, an Air Base Battlefield (ABB) scenario, examines services involved with the working of an Air Force base. Fuselets have been generated in Jython which have been incorporated into a workflow that manages the exchange between an aircraft tower, ground support crew, and aircraft crew at an airport. The "AirportTakeoff" workflow shown in Figure 35 contains invocations to WS that implement common tasks performed within airports.

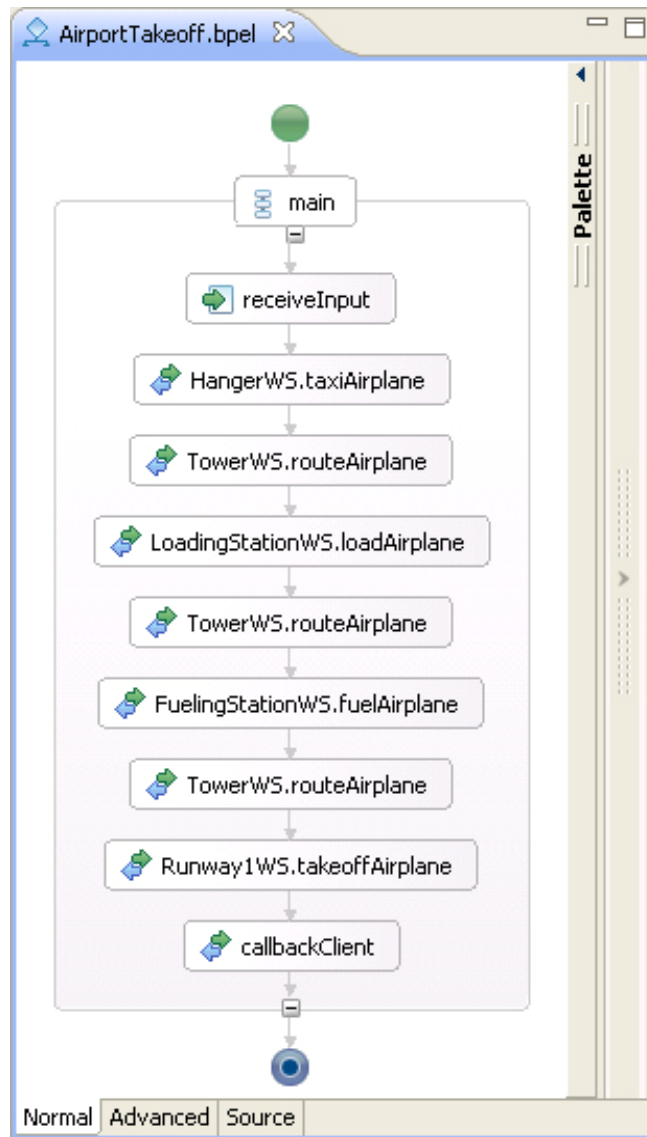


Figure 35. AirportTakeoff Workflow Definition

The services used by the Airport Attack Center COI route airplanes on the ground through the necessary takeoff procedures before an airplane leaves the runway of an airport. The services are sequenced by the COI starting with a Fuselet responsible for removing airplanes from a storage location at a hanger. Fuselet services are used to route planes through loading and fueling stations at the airport, and then send the airplane to the appropriate runway for takeoff. The specific services are:

- *Hanger*: stores and taxis airplanes within the airport.
- *Tower*: represents the airport tower used to route airplanes on the ground.
- *LoadingStation*: loads passengers and other cargo onto the airplane.
- *FuelingStation*: fuels the airplane prior to takeoff.
- *Runway1*: schedules airplanes to leave via the primary runway.

Normal operations of the *AirportTakeoff* workflow involve invoking the appropriate Fuselet allowing necessary messages to be transferred to the JBI. Redundant services have been created for each of the Fuselets used in the workflow. Under attack, the air base may experience a variety of changes. One or more runways may become unavailable, all operations on a runway may be suspended until the runway is cleared which may include delaying and/or re-routing landings, and when operations resume, aircraft landings may be re-ordered based on priority rather than pre-attack schedule.

Redundant Fuselets for each of the services used within the airport have been created and tested showing how CPCL can automatically migrate from a non-functioning workflow to a backup workflow utilizing the appropriate services. A text-based toolkit has also been developed which subscribes to the JBI and outputs console messages showing the progression of airplanes as they move throughout the airport and updating the current status of airport services such as available fuel.

Conclusion

Given the task of workflow inspection and feedback NeWT has been designed and tested to provide a variety of dynamic reconfiguration responses to events surrounding COI WS. NeWT is partitioned into three distinct loops providing workflow creation, automated reconfiguration, and execution management. Partitioning NeWT into TCIL, CPCL, and Oracle provides NeWT with principles of software composition, such as loose coupling of components, plug and play, and dynamic reconfiguration. The combining of WS discovery, task allocation, and automated task reconfiguration into a single, malleable architecture yields a core notion of composition that can be extended to branching and embedded workflows for WS. The example scenarios developed have provided a test bed to examine NeWT's effectiveness in responding to dynamic workflow events and have proven successful in managing the dynamic services used throughout TCA and the ABB example situations.

References

- [1] T. Heinis, C. Pautasso, and G. Alonso, "Design and Evaluation of an Autonomic Workflow Engine," presented at International Conference on Autonomic Computing, 2005.
- [2] P. Vieira and A. Rito-Silva, "Adaptive Workflow Management in WorkSco," presented at International Workshop on Database and Expert Systems Applications, 2005.
- [3] P. Dias, P. Vieira, and A. Rito-Silva, "Dynamic Evolution in Workflow Management Systems," presented at 14th International Workshop on Database and Expert Systems Applications (DEXA '03), 2003.
- [4] T. A. S. C. Vieira, M. A. Casanova, and L. G. Ferrao, "An Ontology-Driven Architecture for Flexible Workflow Execution," presented at WebMedia & LA-Web Joint Conference, 2004.

- [5] L. A. G. d. Costa, P. F. Pires, and M. Mattosol, "WebComposer: a Tool for the Composition and Execution of Web Service-based Workflows," presented at WebMedia & LA-Web Joint Conference, 2004.
- [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 International Enterprise Distributed Object Computing Conference, 2001.
- [8] S. W. Sadiq, "Handling Dynamic Schema Change in Process Models," presented at Australasian Database Conference, 2000.