

Automating Preference and Change in Workflows

R. Baird, M. Hepner, N. Jorgenson, R. Gamble
Department of Mathematical and Computer Sciences
University of Tulsa
600 S. College Ave.
Tulsa, OK 74104
gamble@utulsa.edu

Abstract

Workflow languages can compose and sequence Web service invocations to achieve meaningful task results. Updating workflows when services and task goals change is difficult, and automating the change at runtime requires workflow goals and service preferences to be stated, organized, and controlled. Changing the preferences should be immediately reflected in executing workflows. While there are many applications that could benefit from this dynamic reconfiguration, managing such preferences can be daunting. We introduce NeWT, a Next-generation Workflow Toolkit that combines workflow editing with utilities for introducing and managing change. NeWT allows the novel input of goal preferences to use workflows that accomplish the same task but in different ways to be interchangeable according to the current goal. It accepts that multiple, competing Web services will be available and should be interchanged as dictated by need. We detail the implementation of these concepts using a crisis management example.

1. Introduction

Many businesses today have adopted Web services (WS) as a development choice for composite systems. WS are dynamic, highly-interoperable software entities capable of allowing businesses to uniformly expose component features [1]. Difficulties can arise as WS exhibit their dynamic nature by frequently changing their availability, requiring system reconfiguration. Businesses can utilize technologies such as the Universal Description, Discovery and Integration (UDDI) protocol, as a central repository to maintain status information about deployed services [2]. UDDI assists in managing individual WS, but does little to help businesses reconfigure systems.

The Business Process Execution Language (BPEL) can define workflows of WS [3]. Often workflows

combine procedural rules and business logic with specific service choices to facilitate the transmission of tasks and information between participants [4]. The dynamic nature of WS prescribes that workflow definitions to incorporate rules regarding service availability. Unaltered, the BPEL specification is static in nature and therefore cannot benefit from WS dynamism. It requires defining a single, branching workflow for a task in which the availability of competing services is considered. If an improperly defined workflow is executed, it may fail due to necessary services being either unavailable or slow to respond. Any forced change in a workflow requires the completion of all running instances, the deletion of the current workflow definitions, and the deployment of a new workflow in its place.

Communities of Interest (COI) encompass multiple businesses who access workflows to perform known tasks [5]. Over time, a COI's interests 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 workflow with respect to completing its task in a timely manner and with the best service usage.

Throughout the remainder of this paper, we describe the *Airport Attack Center*, a COI that deals with the operation of a commercial airport during a crisis event. The *Airport Attack Center* utilizes WS related to airport hangers, fueling stations, loading stations, runways, and air traffic controllers. The COI designates certain services according to the immediate needs of the airport (such as a day-usage only runway) and the priority of one service over a competing service due to availability (such as a fueling station

with cheaper operating costs). The COI uses different services if the workflow executes under normal or emergency Airport conditions. Workflow technologies exist to allow a COI to define equivalent services for use when a workflow fails, but these options fail to allow the immediate reflection of preference changes.

Non-standard workflow engines allow businesses some flexibility for WS reconfiguration, which is not normally offered with standard BPEL workflow engines. Many BPEL-inspired workflow systems alter the BPEL specification and allow predefined branches to be strategically placed within a workflow definition so that decisions about which services to invoke can be evaluated at runtime [6-8]. These methods are limited in the types of preferences they support and they do not allow all the types of reconfiguration a COI may require.

Research in dynamic workflow reconfiguration has produced the Change Planning and Coordination Loop (CPCL – "capsule" [9, 10]. CPCL performs automated management of workflow definitions and instances in a WS environment incorporating predefined preferences for service exchange. However, CPCL lacks facilities to change preferences directly. Furthermore, while the current implementation of CPCL allows it to exist decoupled from any one particular workflow engine, an editor is needed for the creation, deployment, and invocation of a workflow. Existing workflow editors lack the support for a COI to express service information and preferences in a way that can be utilized by workflow engines for dynamic reconfiguration.

Workflows deployed on commercial workflow engines expose a WS interface for invocation. Thus, a workflow can have multiple sub-workflows that appear as service invocations. Managing the reconfiguration of these composed workflows is especially difficult because the change must be propagated within the sub-workflows and also to all workflows composed of the affected sub-workflow. For example, the *Airport Attack Center* utilizes many tasks in a workflow defined for the take-off of an airplane. Each of these tasks may be its own workflow. Reconfiguring any such task complicates change management in all workflows referencing airplane take-off.

In this paper, we introduce the Next-generation Workflow Toolkit (NeWT) to tie WS workflow editing and preference entry to enabling automated reconfiguration of dynamic workflows within a commercial workflow management system, such as the Oracle BPEL Process Manager [11].

2. Background

This work leverages the benefits found in many of the currently available workflow technologies for dynamic reconfiguration. Reconfiguration systems are used in a COI to enact business rules that define how systems should be changed in response to pre-defined events. Business rule definitions can range in complexity depending on the size of the COI they are created for.

Previous research on WS workflow reconfiguration focuses on two areas: the dynamic exchange of one service for another and the dynamic composition of services using semantics. WebComposer performs dynamic discovery and composition of WS [6], modeling services with semantic descriptions and making inferences for discovery, selection, and composition of the appropriate services. This allows a business to define semantically equivalent services for automated exchange, but these definitions are static and cannot be changed after the initial definition. Similarly, the BPEL language can be extended for per process instance changes to workflows via a 'find and bind' mechanism [12]. Statically encoded into the definition, this mechanism directs workflows to alternate WS when a WS fails or to adapt to a WS location change. Additional research enables the reconfiguration of workflows to select alternative WS for businesses to dictate preferences for exchangeable services before workflows have begun execution, but lacks the necessary toolsets for preference changes after execution begins [13, 14].

As a workflow language, BPEL offers built-in fault handling capabilities, called compensation actions, that are applied when workflow faults are detected. Common faults experienced in workflows include improper workflow design and service unavailability. The error handling offered by BPEL lacks the foresight typically associated with reconfiguration systems by only offering alternative actions to take place once part of a workflow has failed. Limiting reconfiguration for BPEL workflows can prevent workflow authors from encoding all the reconfiguration actions a COI may need [15]. Furthermore, the fault handling definitions within a workflow are statically defined such that changes made to them require redeployment of the workflow.

NeWT takes advantage of the CPCL loop framework (shown in the right two boxes in Figure 1) to monitor WS for changes requiring reconfiguration, as changes can potentially affect running instances of workflows. The CPCL loop segregates change

management from the execution of workflows. CPCL contains several integral components, each serving a unique purpose. First, the Change Management pieces are:

- *Analysis and Planning* – used for COI event requests, such as invoking or deploying workflows, entering new goal preferences, or changing existing workflows.
- *Inspector* – used for automated monitoring and investigation of COI WS.
- *Coordinator* – used to correlate definition changes and manage the affected instances.
- *Bridge* – used as an integration enabler to assist in communicating with different types of workflow engines. This allows CPCL to support any BPEL compliant workflow engine, assuming the Control Interface is available.

Typical commercial workflow management systems entities are assumed available to CPCL. These are:

- *Control Interface* – allows access to the underlying control infrastructure for normal workflow tasks.
- *Control Infrastructure* – allows workflows to be deployed and invoked
- *Monitoring Infrastructure* – allows metadata to be gathered about instances and definitions.

The available workflow editors that a COI can use for workflow development frequently offer some, but not all, of the features required to fully utilize dynamic WS. Open source workflow editors provide hooks to attach reconfiguration functionality [16-18]. This enables a COI to add the reconfiguration support, but it is not provided natively.

3. Next Generation Workflow Toolkit

The framework we propose for NeWT encompasses two entities that work in tandem with a commercial workflow management system. The first entity, CPCL, 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. This adds a the left-most loop to the NeWT framework as seen in Figure 1.

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 [17]. The Eclipse plug-in sends the generated BPEL workflows to CPCL for deployment on a commercial workflow engine and managed for reconfiguration. By utilizing the open framework provided by Eclipse, TCIL enables a COI to enhance the support offered within NeWT by extending Eclipse with other plug-ins tailored to COI-specific features that may be useful during workflow development, such as service statistics and testing technologies.

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 reconfiguration. Using Eclipse, NeWT 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.

3.1. Controller Interface

A Controller Interface is present within TCIL to assist COI members in determining the necessary functions within the Workflow Toolkit to complete assigned tasks. Common workflow responsibilities include workflow development, reconfiguration, and execution. Workflow *developers* are COI members responsible for creating valid workflows which provide the necessary WS invocations to perform an overarching task. Workflow developers may also be responsible for the reconfiguration of workflows as services change and the COI needs adapt over time. Workflow *users* are COI members who execute workflows to create running instances.

The Control Interface present in TCIL enforces 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.

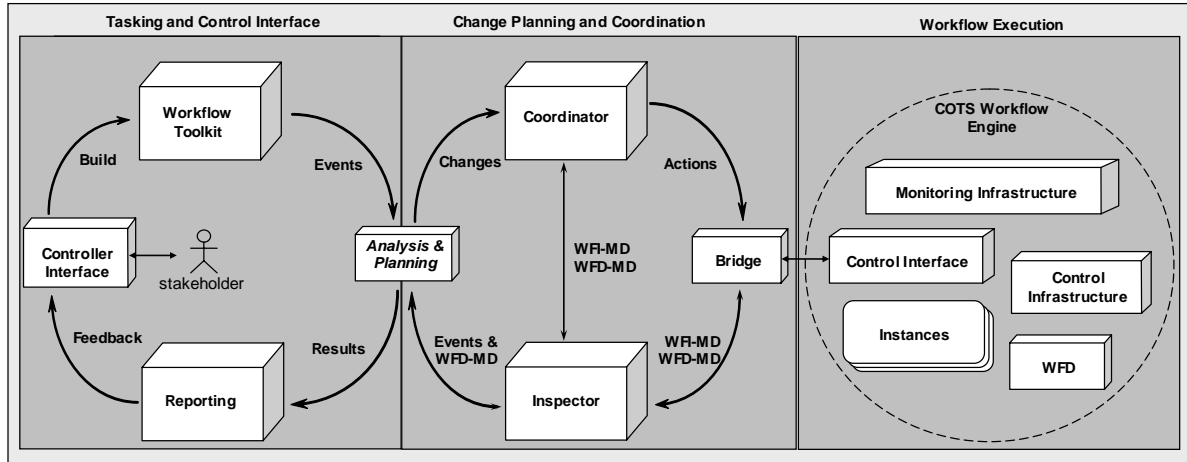


Figure 1. NeWT architecture

3.2. Workflow Toolkit

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. TCIL facilitates workflow creation using a graphical editor containing common workflow tasks which can be dragged and placed into an editor window. A complete workflow developed for the Airport Attack Center using TCIL is shown in Figure 3.

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.

3.3. 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. The built-in EMF viewer within Eclipse lets COI users view the reporting information as it changes within the COI.

TCIL generates reports for each COI. The reporting viewer, shown in Figure 2, 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.

The EMF model used by TCIL supports the ability for a COI developer 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 sending messages to workflow developers via alert dialog boxes or other COI-specific messaging platforms. Examples 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.

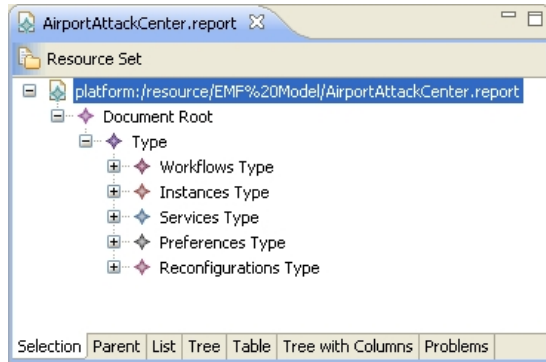


Figure 2. TCIL report viewer

4. Airport Attack Center Example

The benefits of using NeWT to manage workflow operations for a COI can be highlighted by expanding the example airport COI introduced earlier. We examine a workflow used by the *Airport Attack Center* to track airplane information for airplanes undergoing takeoff procedures at a commercial airport. The "AirportTakeoff" workflow shown in Figure 3 contains invocations to WS that implement common tasks performed within airports.

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 WS responsible for removing airplanes from a storage location at a hanger. Services 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:

- *HangerWS* stores and taxis airplanes within the airport.
- *TowerWS* represents the airport tower used to route airplanes on the ground.
- *LoadingStationWS* loads passengers and other cargo onto the airplane.
- *FuelingStationWS* fuels the airplane prior to takeoff.
- *Runway1WS* schedules airplanes to leave via the primary runway.

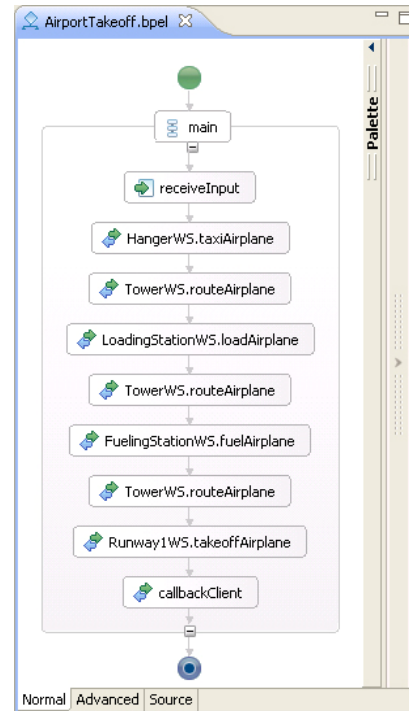


Figure 3. AirportTakeoff workflow definition

The benefits of using the NeWT architecture can be illustrated by examining the reconfiguration opportunities found within the "AirportTakeoff" workflow. Imagine that similar workflows exist within the COI, many utilizing some of the same services as defined within the example workflow. For instance, workflows to land airplanes at the airport utilize the same *Runway1WS*, *TowerWS*, and eventually *HangerWS*. Other workflows may exist which are used to repair airplanes and manage security.

5. NeWT and Dynamism

In our implementation, Analysis & Planning provides an API for propagating CPCL functionality and messages to TCIL within the NeWT architecture. This maintains a loose coupling between the TCIL and CPCL loops, similar to the loose coupling that exists between CPCL and the workflow management engine.

5.1. 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 that are related to these dynamic goal controls.

At the time of workflow creation, a workflow developer may be faced with choosing between two service implementations for a workflow task. Services performing the same task are generally thought to be equivalent, but a COI may view the services distinctly due to differences in implementation or the backend systems used. Workflow *goals* define extra requirements that a COI can place on the execution of a workflow, specifically on the service implementations used to execute workflow tasks. COI *preferences* are defined for workflow tasks referencing specific WS implementations, attaching different workflow goals to each. The goal the COI wishes the workflow to execute under determines the services used when the workflow is executed. Without an architecture that supports the definition of workflow goals, a COI must create each workflow separately, even though the same general task is performed. TCIL allows a single workflow to contain multiple goals, which determine the services selected at execution time.

When multiple goals for workflows exist they may contrast with each other. The goals of "performance" and "accuracy" define a distinction in service implementation where execution may take longer in order to provide more accurate data. In the case of emergency management, a COI may wish to execute workflows according to a goal of "performance" initially, and at a later time switch workflow execution to execute more "accurately". TCIL facilitates the immediate switch of the workflow definitions to the goal being used to generate their specification.

Within the example COI, a major responsibility of the *Airport Attack Center* is responding to emergencies at the airport including possible terrorist attacks or other crisis events. The COI has a set of redundant services available to use in such cases so that airport management can continue as expected in the event of a crisis. Separate workflow *goals* exist for "Normal Operations" and the "Emergency Operations" of the airport. During "Normal Operations," the *Airport Attack Center* COI prefers all primary WS to be used in the workflow. During "Emergency Operations," the COI desires the backup services to be used. The time-sensitive nature of responding to emergency and crisis events places an emphasis on the response time required to switch workflow goals.

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 providing a mechanism for the workflow

developer to define new goal and preferences for the workflow. The wizard, shown in Figure 4, allows the COI developer 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.

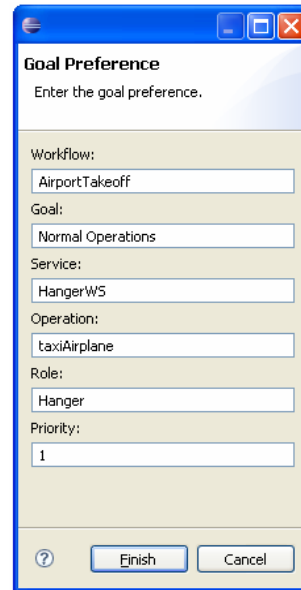


Figure 4. Goal Preference Entry Dialog

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 5. The view lists of all current workflow goals and the service preferences that have been defined. Figure 5 shows the "Normal Operations" and "Emergency Operations" goals defined for the *Airport Attack Center*. All services used within the "AirportTakeoff" workflow shown in Figure 3 have a corresponding backup service defined utilizing a redundant WS implementation.

WFD-ID	Goal	Service	Operation	Role	Priority
AirportTakeoff	Normal Operations	HangerW5	taxiAirplane	Hanger	1
AirportTakeoff	Emergency Operations	Hanger2W5	taxiAirplane	Hanger	1
AirportTakeoff	Normal Operations	TowerW5	routeAirplane	Tower	1
AirportTakeoff	Emergency Operations	Tower2W5	routeAirplane	Tower	1
AirportTakeoff	Normal Operations	LoadingStationW5	loadAirplane	LoadingStation	1
AirportTakeoff	Emergency Operations	LoadingStation2W5	loadAirplane	LoadingStation	1
AirportTakeoff	Normal Operations	FuelingStationW5	FuelAirplane	FuelingStation	1
AirportTakeoff	Emergency Operations	FuelingStation2W5	FuelAirplane	FuelingStation	1
AirportTakeoff	Normal Operations	Runway1W5	takeoffAirplane	Runway	1
AirportTakeoff	Emergency Operations	Runway2W5	takeoffAirplane	Runway	1

Figure 5. AirportTakeoff Goal Preferences

The ability to add goals and preferences to workflow definitions alone does not empower a COI with enough functionality to dynamically change workflow goals. It must be couple with the NeWT toolkit as it continues to execute the default workflow definition created by the workflow developer until the goal for that workflow is changed. The "GoalSelectionWizard" of TCIL enables workflow developers to quickly and effortlessly switch the goal a workflow is executing under. The wizard, shown in Figure 6, gives the COI easy access to change the goal for workflows without worrying about managing the reconfiguration that takes place inside of CPCL.

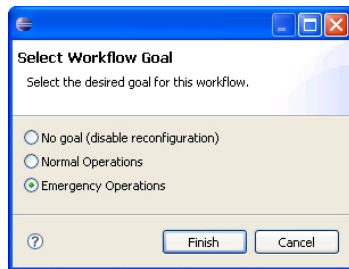


Figure 6. Workflow Goal Selection Dialog Box

Reconfiguration of the workflow is then 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. 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. Figure 7 shows the trace of a failed workflow instance that did not take advantage of the COI goal change to "Emergency Operations".

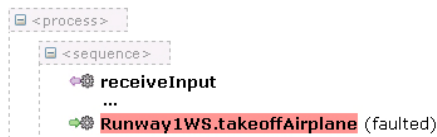


Figure 7. Runway1WS Failure

CPCL manages the reconfiguration of affected workflow instances dynamically. When goal change is received, the workflow definition is altered. CPCL acquires a list of affected instances. The default reconfiguration directive is to configure workflow instances to execute the new definition, allowing them to take advantage of the changes immediately. Since

workflow instances cannot be modified on many commercial workflow engines, CPCL creates a new temporary workflow definition containing the parts of that have not been executed by the instance of concern. Figure 8 shows a partial workflow definition created in response to the goal change to "Emergency Operations" from the *Airport Crisis Center*.



Figure 8. Reconfigured Instance

The location of an executing instance when the workflow goal is changed determines the temporary workflow generated. Workflow instances that are reconfigured successfully prevent the COI from manually inspecting faulted instances. NeWT's ability to quickly reconfigure workflow definitions and instances proves useful for COIs that need to react to dynamic events. In a COI where WS are in high-demand and suffer from performance issues, the COI can switch workflow definitions, potentially before services stop responding to requests entirely.

Results of the reconfiguration are updated in the Reporting component of TCIL, as shown in Figure 9. Reporting allows the COI to review which preferences were used to generate the new workflow definition and the goal that triggered the change.

5.2. Dynamic Priority Controls

Workflows defined by a COI are often vital for continuous execution inside a business. Due to the dynamic nature of WS regarding availability, workflow definitions may fail if instances attempt to invoke an offline WS. For example, Figure 10 shows the trace of a failed workflow that occurred due to the unavailability of the primary fueling station. A COI wants to invoke the best service available at the time of workflow execution. Since there may be a time delay between workflow definition and execution times, it may be difficult for the workflow developer to predict the available set of services to include in a workflow.

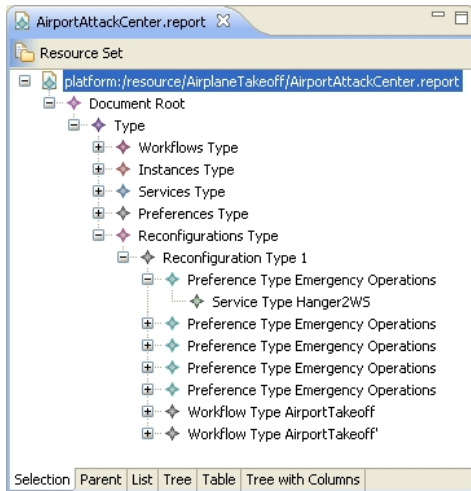


Figure 9. Report for Goal Change Reconfiguration

The Workflow Editor of TCIL provides workflow developers with features to dynamically control WS tasks in workflows via advanced priority controls. Using the same set of goal-preference definitions used to change workflow goals, TCIL allows multiple service preferences to be defined for tasks within a workflow, giving each preference a different priority. CPCL automatically selects the highest priority preference that is available for inclusion in workflow execution. This enriches the task selection process provided by TCIL, yielding extra options for redundancy as well as the ability to take advantage of desirable services with stable availability.

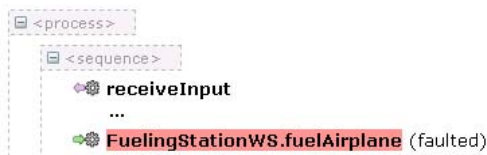


Figure 10. AirportTakeoff Failure Trace

Similarly to how TCIL allows the goal of a workflow to change at runtime, TCIL enables the preferences defined for goals to be dynamically changed by COI users. Investigation of the services used in the *Airport Attack Center* COI may reveal that the primary WS responsible for fueling airplanes is unreliable due to frequently running out of fuel. The TCIL Reporting information would be updated as shown in Figure 11, indicating that the COI needs to create extra preferences for the workflows.

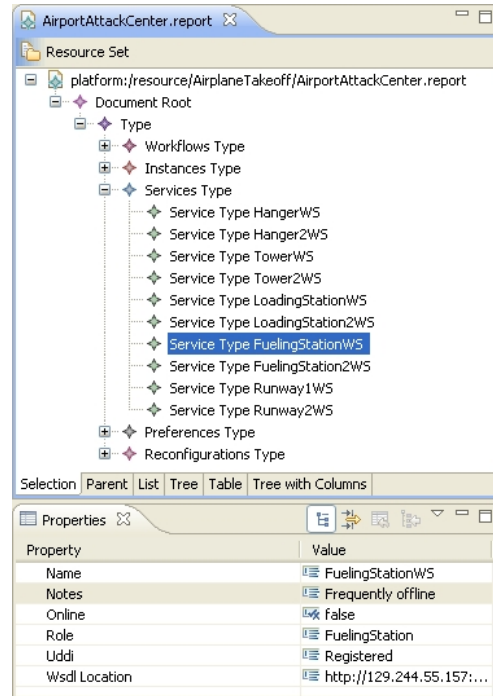


Figure 11. Reporting Properties

Figure 12 introduces a modified copy of the goal preference table found in Figure 5. Additional preferences have been added to the "Normal Operations" goal for the fuel station WS used in the middle of the workflow. CPCL monitors each WS registered in the COI UDDI. When the primary WS responsible for fueling the airplanes becomes unavailable, a workflow reconfiguration is attempted. Analysis & Planning within CPCL checks if changes to workflow preferences in the goal preference table should initiate reconfiguration.

The newly added preferences have a priority value of 2 so the WS they refer to are secondary. The automated reconfiguration of CPCL alleviates the trouble a COI could encounter from workflow executions that failed due to preventable problems. As the primary fueling station WS used in the *Airport Attack Center* goes offline, the workflow and running instances are reconfigured to take advantage of the redundant service provided by the COI.

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	Normal Operations	FuelingStation2WS	fuelAirplane	FuelingStation	2
AirportTakeoff	Emergency Operations	FuelingStation2WS	fuelAirplane	FuelingStation	1
AirportTakeoff	Normal Operations	Runway1WS	takeoffAirplane	Runway	1
AirportTakeoff	Emergency Operations	Runway2WS	takeoffAirplane	Runway	1

Figure 12. Updated goal preferences

5.3. 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 service changing a deployment location). 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. CPCL provides four *reconfiguration directives* for workflow instances:

- *Flushed* instances continue to execute the previously defined workflow definition
- *Aborted* instances cease execution in favor of COI intervention
- *Restarted* instances are safe to begin executing from the start of the workflow
- *Migrated* instances are manipulated so that a partial workflow is created containing the portion of the new workflow needed to execute for the instance to successfully complete using the new workflow definition. This is the truly dynamic reconfiguration option, because the instance is converted during runtime.

Each of these functions appears within the Workflow Toolkit for mandated reconfiguration via CPCL. Manual workflow changes are driven by the information available in Reporting and Feedback, such as a change in the deployment location of a critical WS. In the *Airport Attack Center* example, workflow instances would require manual inspection if Reporting indicates (Figure 13) the *LoadingStationWS* is upgraded. Manual inspection is required in order to ensure all airplanes at the airport have been loaded with the correct passengers. In contrast, changes to the *FuelingStationWS* may have been foreseen or are of less concern, so the COI preferences can be dictated

through the Goal Preference table to automatically reconfigure the workflow using this WS within CPCL without the COI's direct intervention.

Property	Value
Name	LoadingStationWS
Notes	New deployment location, see WsdL Location for more details.
Online	true
Role	LoadingStation
Uddi	Registered
WsdL Location	http://129.244.55.157:8080/loading/FuseletWSService?WSDL

Figure 13. Reporting and Feedback details

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 14 shows the wizard that is displayed when directed changes are made, such as a workflow developer modifying the "AirportTakeoff" workflow to utilize the new service location obtained from Reporting for the *LoadingStationWS* (Figure 13). 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.

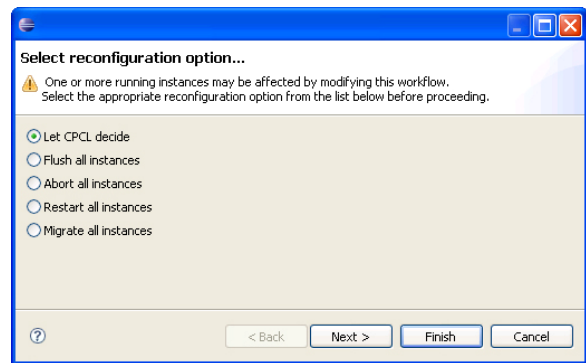


Figure 14. Workflow reconfiguration options

When the choice of reconfiguration directive is made within TCIL, CPCL manages the deployment of the new workflow and the necessary instance migrations. For any reconfiguration, CPCL calculates a *transition point* and *change block* between the new workflow definition and the existing workflow definition that determine the reconfiguration directive to take on designated instances [10]. Rules for reconfiguration dictate which final directive is actually used, given how the directive was put forth (automatic

or manual) and whether the instance has reached the transition point. Manually input directives have the highest priority for satisfaction.

After selecting the reconfiguration directive, reporting within TCIL is updated to contain the reconfiguration information and instances affected by the change, as shown in Figure 15. 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.

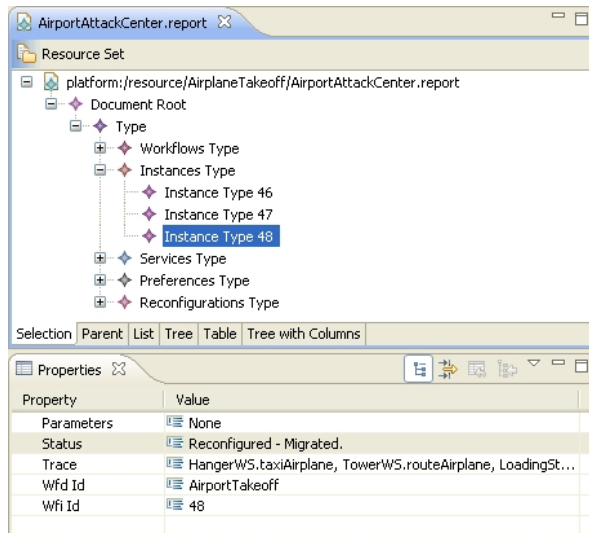


Figure 15. Workflow reconfiguration results

6. Conclusion

Principles of software composition, such as loose coupling of components, plug and play, and dynamic reconfiguration, permeate NeWT. First, NeWT is partitioned into three distinct loops. CPCL is the heart of the architecture [10]. Given that Analysis & Planning and the Bridge can be adjusted at their interface to an outside environment, including any WFMS, the current architecture allows CPCL to be coupled to any interface that feeds it workflow and reconfiguration directives. Second is the composition of the modular components that comprise each loop with asynchronous message-based communication. The combining 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.

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] W3C, "Web Services Description Language (WSDL) 1.1," 2001, www.w3.org.
- [2] Oasis, "UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, Dated 20041019," 2004, <http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf>.
- [3] BEA, Microsoft, SAP AG, and Seibel Systems, "BPEL4WS Specification Version 1.1".
- [4] Workflow Management Coalition, *The Workflow Handbook*. Lighthouse Point, FL, USA: Future Strategies, Inc., 2002.
- [5] S. Renner, "A Community of Interest Approach to Data Interoperability," in *Federal Database Colloquium*, 2001.
- [6] L. A. G. d. Costa, P. F. Pires, and M. Mattosol, "WebComposer: a Tool for the Composition and Execution of Web Service-based Workflows," in *WebMedia & LA-Web Joint Conference*, 2004.
- [7] A. Erradi, Padmanabhuni, S., Varadharajan, N., "Differential QoS support in Web Services Management," in *IEEE Int'l Conference on Web Services (CWS '06)*, 2006.
- [8] T. A. S. C. Vieira, M. A. Casanova, and L. G. Ferrao, "An Ontology-Driven Architecture for Flexible Workflow Execution," in *WebMedia & LA-Web Joint Conference*, 2004.
- [9] M. Hepner, "Dynamic Changes to Workflow Instances of Web Services," Dept. Mathematical & Computer Sciences. Ph.D. Diss., The University of Tulsa, 2007.
- [10] R. Baird, Hepner, M., Gamble, R., Gamble, T., "Reconfiguring Workflows of Web Services," in *IEEE Int'l Conf. on COTS-Based Software Systems (ICCBSS)*, 2007.
- [11] Oracle, "Oracle BPEL Process Manager," 2007, www.oracle.com/technology/bpel.
- [12] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann, "Extending BPEL for run time adaptability," in *9th IEEE Int'l EDOC Enterprise Computing Conference*, 2005, pp. 15-26.
- [13] DAML, "Dynamic QoS based Supply Chain," www.daml.org/services/use-cases/architecture/QosUseCase/DynamicQoSWebProc-SWSA-UseCase-v1.htm.
- [14] J. Salas, Perez-Sorrosal, F., Patino-Martinez, M., Jimenez-Peris, R., "WS-Replication: A Framework for Highly Available Web Services," in *5th Int'l Conference on World Wide Web (WWW '06)*, 2006.
- [15] M. Stefano, M. Enrico, and P. Barbara, "SH-BPEL: a self-healing plug-in for Ws-BPEL engines," in *Proc. of the 1st workshop on Middleware for Service Oriented Computing*, 2006.
- [16] Bull, "Orchestra," www.bullopensource.org, 2007
- [17] Eclipse, "BPEL Project," www.eclipse.org/bpel/, 2007
- [18] Intalio, "Intalio Company and Platform Overview," 2007, bpms.intalio.com.