A SERVICE TO AUTOMATE THE TASK ASSIGNMENT PROCESS IN YAWL

by

KRISHNA NAGARJUN REDDY SAMANTHULA

B.Tech, JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, INDIA, 2008

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College Of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2010

Approved by:

Major Professor
Dr. Gurdip Singh

Abstract

Developing an optimal working environment and managing the of work load in an efficient manner are the major challenges for most businesses today. So, the importance of the workflow's and workflow management in an organization is unquestionable. Many organizations use sophisticated systems to organize the workflows. One such workflow system based on a concise and powerful modeling language called "Yet Another Workflow Language" is YAWL. YAWL handles complex data, transformations, integration with organizational resources and Web Service integration.

Workflow comprises of three main perspectives: control-flow, data and the resources. In Yawl, the control-flow and the data-flow are tightly coupled within the workflow enactment engine. But the resource perspective is provided by a discrete custom service called Resource Service. Administrative tools are provided using which the administrator has to manually select the resource (referred as participant) which needs to perform a particular task of the workflow. This project aims at developing a service which can automate the assignment of the tasks to the participants by using the Resource service which provides number of interfaces that expose the full functionality of the service.

The application of this project with respect to Healthcare domain is presented. Healthcare domain is the one of the most demanding and yet critical business process. Hospitals face increasing pressure to both improve the quality of the services delivered to patients and to reduce costs .Hence there is significant demand on hospitals in regard to how the organization, execution, and monitoring of work processes is performed. Workflow Management Systems like YAWL offers a potential solution as they support processes by managing the flow of work.

Table of Contents

List of Figures

Acknowledgements

I would like to thank my Major Professor Dr. Gurdip Singh for his constant help, encouragement and guidance throughout the project.

I would also like to thank Dr. Torben Amtoft and Dr. Mitchell Neilsen for serving in my committee and for their valuable cooperation during the project.

Finally, I wish to thank my family and friends for all their support and encouragement.

CHAPTER 1 - Introduction

1.1 Workflows

Workflow is a depiction of series of tasks within an organization to produce a final outcome. At each stage in the workflow, one individual or group is responsible for a specific task. Once the task is complete, the workflow software ensures that the individuals responsible for the next task are notified and receive the data they need to execute their stage of the process[1].

1.2 Workflow Management Systems

'Workflow Management system' is a system that defines, creates and controls a workflow by running it through a workflow engine. It consists of a process design, a system configuration and a process enactment[4]. In hospital scenario, large amount of clinical procedures relating to patient management are repetitive and Workflow Management Systems (WFMS) can automate these repeated activities. The introduction of WFMS would enable healthcare institutions to face this challenge of transforming large amounts of medical data into contextually relevant clinical information.

1.3 Workflows in Hospitals

Hospitals face increasing pressure to both improve the quality of the services delivered to patients and to reduce costs. Hence, hospitals have to focus on ways of streamlining their processes in order to deliver high quality care while at the same time reducing costs. Workflow Management Systems (WfMSs) offer a potential solution as they support processes by managing the flow of work, such that individual work items are done at the right time by the proper person. The advantage of using workflows and workflow management systems is that the processes can be executed faster and more efficiently [2] [3].

Hospitals processes and procedures are typically *diverse*, require *flexibility*, and often involve *multiple medical departments* in diagnostic and treatment processes. Workflow management systems that offer execution flexibility must be needed in order to support the processes in hospitals[4].

1.4 Perspectives of workflows

The workflow comprises of three perspectives. They are control-flow perspective, and the data perspective, and the resource perspective[5]. The control flow perspective deals with determining which task(s) are enabled at certain times during the life-cycle of a process, based on arcs, conditions, splits and joins. The data perspective deals with mapping data values to and from tasks and their parent nets and performing transformations. The resource perspective is primarily responsible for modeling an organizational structure, and the people who populate it, in a computational form, so that a person may be coupled with tasks and data emanating from the control-flow and data perspectives.

1.5 YAWL overview

YAWL is a workflow management system that is used in variety of academic and industrial settings. YAWL distribution has a process editor, YAWL engine and other supporting services. Process editor is used to design workflow nets. YAWL net has more explicit representation of the join and split behavior of the various tasks of the workflow. The YAWL model captures the order in which the tasks need to be presented at runtime.

At runtime, the Engine uses the YAWL model to determine when certain tasks are to be offered to the resoruces. YAWL presents the data input form for the corresponding task to the resoruce. The presentation of these fields is governed by the type of the data involved and whether the fields contain editable values or are for presentation purposes only. Upon completion of the task, this information is sent back to the Engine, which, in general, may pass it on to other tasks or use it to determine which tasks should be performed next.

Models are constructed in the build time component, the Editor and subsequently deployed in the runtime environment. The runtime environment itself can consist of a number of components, but at its core are the Engine and the Resource Service. The Engine deals with the control-flow logic and workflow data, while the Resource Service is concerned with the routing of work items to appropriate resources [7].

1.6 Automating Admin functionality in YAWL

1.6.1 Motivation

The resource perspective is particularly important because, for the most part, workflow tasks are designed to be performed by people, and so a workflow environment should support efficient and flexible ways to associate work with the people who have the required skills and authorizations to carry it out. YAWL supports the resource perspective of the workflow by discrete service called resource service. There is an interface provided for the administrator using which he can choose the available work items (tasks) and assign it to the human resource (participant). But this assignment part is done manually by the administrator and he also need to remember the authorizations of the people to whom he is assigning the task to. Hence, there is a need for the automating this task assignment to a resource without manual interference. In this report, we describe about a service that uses yawl's resource service and will automate the task assignment to the resource without the human interference.

1.6.2 Overview of the System

This system uses the web application called Resource Service which comes along with the distribution of the YAWL. YAWL distribution comes with two things, YAWL Editor and YAWL engine. Engine comes as the deployed web application on tomcat server. There are other supporting web applications which are distributed along with the engine. Resource Service is one such application that is part of YAWL distribution. An interface built using JSF (Java Server Faces) comes along with the resource service application. Using this interface the administrator and the participants can login into their accounts to view their work queues.

For the administrator there are two work queues available. They are Unoffered work queue, which shows the tasks that are not assigned to any resource and the Work listed queue, which shows the tasks which are assigned to the resources along with other details. For a resource (participant) there are four different work queues available. They are Offered, Allocated Started and Suspended queues. The service developed as part of this project removes the work item from unoffered work queue of the administrator and the work item is placed in the Started queue of the resource. The resource to which this work item needs to be offered is decided based on the task type and the availability of the resource. This task type is decided by accessing some parameter of the work items. These parameters are set during the workflow design time.

The participants can access the work queues from the interface, since this is implemented as a web application. In our case, the doctors and nurses will be carrying smart phones from which they will access their work queues by using web browsers and logging into their accounts. This makes the work flow distributed as the tasks can be handled by the resource by being anywhere using their smart phones. Since, the interface interacts with the engine there is also synchronization is maintained. Once the task is completed by the resource, the engine notifies the resource service about the next task enablement and the workflow progresses.

CHAPTER 2 - YAWL

2.1 History and Emergence of YAWL

Yawl language and its supporting system were first developed by researchers at Queensland Institute of Technology. Subsequently several other organizations have contributed for the initiative. The YAWL language was designed to support the different workflow patterns that would have formal semantics. The starting point for development of YAWL was Petri Nets. Petri Nets were used as the starting point for this because Petri nets were close to supporting most of the workflow patterns in a straight forward manner. But some patterns are not easily captured. To support these patterns, developers of YAWL extended the Petri nets. They introduced dedicated constructs to deal with patterns that Petri nets have difficulty in expressing, in particular patterns dealing with cancelation, synchronization of active branches only and multiple concurrently executing instances of same task. YAWL also added some syntactic elements to the Petri nets to capture workflow patterns such as *simple choice* (xor-split), *simple merge* (xor-join), and *multiple choice* (or-split) [6].

YAWL was given a formal semantics based on a state transition system and therefore YAWL nets cannot be simply mapped to Petri nets. Hence, YAWL cannot be seen as notional abbreviations over Petri nets even though it is inspired by Petri nets. Verification approaches have been developed to handle the new constructs offered by YAWL. *Reset nets* were one such method that was useful for reasoning about YAWL. These are Petri nets extended with the concept of a reset arc. The concept of cancellation can be directly expressed using reset arcs. In workflows, cancelation means that the execution of a certain task should lead to other tasks being terminated or not being available for further execution.

After YAWL was defined, support environment for YAWL is created. This demonstrated that it was possible to provide comprehensive support for the (original) workflow control-flow patterns. As such, the YAWL environment can be seen as a reference implementation that supports the Workflow Patterns. The term YAWL became synonymous with both the language and the support environment. Over time as the environment evolved, and YAWL was developed into a complete work flow management system[7].

2.2 Control flow patterns

The control-flow patterns describe language features for managing the flow of control between the tasks of the business process. These patterns are divided into eight distinct groups depending on their area of focus:

- **Branching** patterns are used for scenarios where the thread of control in a process divides into two or more independent execution threads.
- **Synchronization** patterns are used for scenarios where there are several independent threads of control and they need to be synchronized into one branch.
- **Repetition** patterns describe different ways in which a repetitive task or the repetitive sub processes can be specified in a process.
- **Multiple instance (MI)** patterns describe situations where multiple instances of a task or sub process that are concurrent execute simultaneously and may need to be synchronized upon completion.
- **Concurrency** patterns aims at handling the scenarios where there is some restriction on the extent of concurrent execution within a process.
- **Trigger** patterns identify constructs that allow the execution in a process past a specified point to be made contingent on the receipt of a signal from the operating environment.
- **Cancelation** and *completion* patterns describe the various cancelation situations that may arise in business process.
- **Termination** patterns address the issue related to, when the execution of a process can be considered to be finished.

Each of this group contains a number of patterns that describe desirable behaviors in a business process[8].

2.3 Constructs in YAWL

YAWL provides large range of constructs for describing a business process in terms of the control-flow aspects. Figure 2.1 shows the different constructs of language elements that make up the control-flow perspective of YAWL.

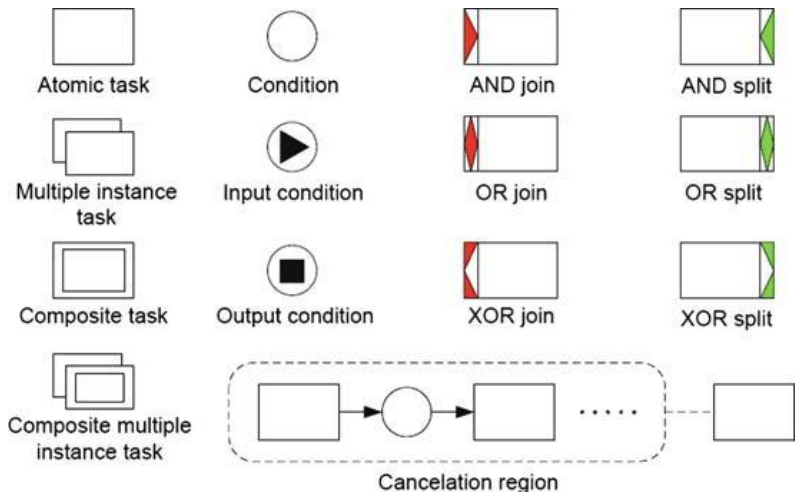


Figure 2.1: YAWL constructs symbols

Image source [21]

Yawl specification typically is made up of set of YAWL nets in rooted graph structure. The rest of this section describes all the available constructs. *Atomic tasks* represent the single task that can be handled by the participant. It has an underlying implementation corresponding to them. *Composite tasks* refer to another YAWL net at a lower level in the hierarchy. The lower level in hierarchy describes the way in which the composite task is implemented. Each YAWL net has one *unique input* and *output condition*. The top level nets input and output conditions signify the starting and ending of the process instance.

A new instance of the process (workflow), called *case* is initiated when a token is places in input condition of the YAWL net. A *task instance* or *work item* is the unit of work that needs to be completed. A work item is an enabled instance of a task. Having multiple instances of both atomic and composite tasks is also possible. (as indicated in Figure 2.1).

Tasks in a YAWL net can have specific join and split behaviors associated with them. The supported join and split constructs are the AND-join, OR-join, XOR-join, AND-split, OR-split, and XOR-split. The operation of each of the joins and splits in YAWL are as follows:

- ***AND-join*** – A task with an AND-Join will wait for all of its incoming flows for the completed work before starting. It is typically used to synchronize pre-requisite activities that must be completed before some new piece of work may begin.
- ***OR-join*** – The OR-Join ensures that a task waits until all incoming flows have either finished, or will never finish.
- ***XOR-join*** – The task with XOR-join will be enabled when one of the incoming branches to the XOR-join in a given case has been enabled.
- ***AND-split*** – The AND-Split is used to start a number of task instances simultaneously. It can be viewed as a specialization of the OR-Split, where work will be triggered to start on all outgoing flows.
- ***OR-split* –** The OR-Split is used to start some, but not necessarily all outgoing flows to other tasks. It is used when we don't know which outgoing flow will result in completion of task until runtime.
- ***XOR-split*** – The XOR-Split is used to trigger only one outgoing flow. It is used for automatically choosing between number of possible exclusive alternatives once a task completes.

YAWL supports the notion of a *cancelation region*, which includes a group of conditions and tasks in a YAWL net. It is linked to a specific task in the same YAWL net. At runtime, when an instance of the task to which the cancelation region is connected completes executing, all of the tasks in the associated cancelation region that are currently executing for the same case are withdrawn[9].

2.4 Support for flexible and dynamic workflows

Business processes are usually rigidly structured and Workflow management systems are generally designed to support the modeling such business processes. But, to remain effective and competitive, organizations must continually adapt their business processes to manage the rapid changes demanded by the dynamic nature of the marketplace or service environment. The term *flexibility* is used to denote to what extent a workflow system is able to support or handle expected or unexpected deviations in the execution of process instances [10].

The support for flexibility in YAWL language is through a number of constructs at design time. YAWL supports parallel branching, choice, and iteration natively, which allow for certain paths to be chosen, executed, and repeated based on conditionals and data values of the instance like many other workflow modeling languages. In addition, YAWL also supports advanced constructs such as multiple-atomic and multiple-composite tasks. With these additional constructs several instances of a task or subnet can be executed concurrently (and dynamically created), and cancelation sets, which allow for arbitrary tasks (or sets of tasks), to canceled or removed from a process instance.

To support dynamic workflow, an integral service called *Worklet Service* is distributed as part of the YAWL environment. It provides dynamic flexibility and exception-handling support for YAWL. To allow, such a system would provide each task of a process instance with the ability to be linked to an extensible set of actions. One of the actions from the set could be contextually and dynamically chosen at runtime to carry out the task. In the worklet service these set of actions are called as worklets. In effect, a worklet is a small, self-contained, complete workflow process, which handles one specific task (action) in a larger, composite process[11].

2.5 YAWL Architecture

Figure 2.1 presents a three-tier view of the YAWL System. It shows how the functionality is provided for user applications at *presentation layers* with the help of *business logic layer* which encapsulate the resources in *data layer* [12].

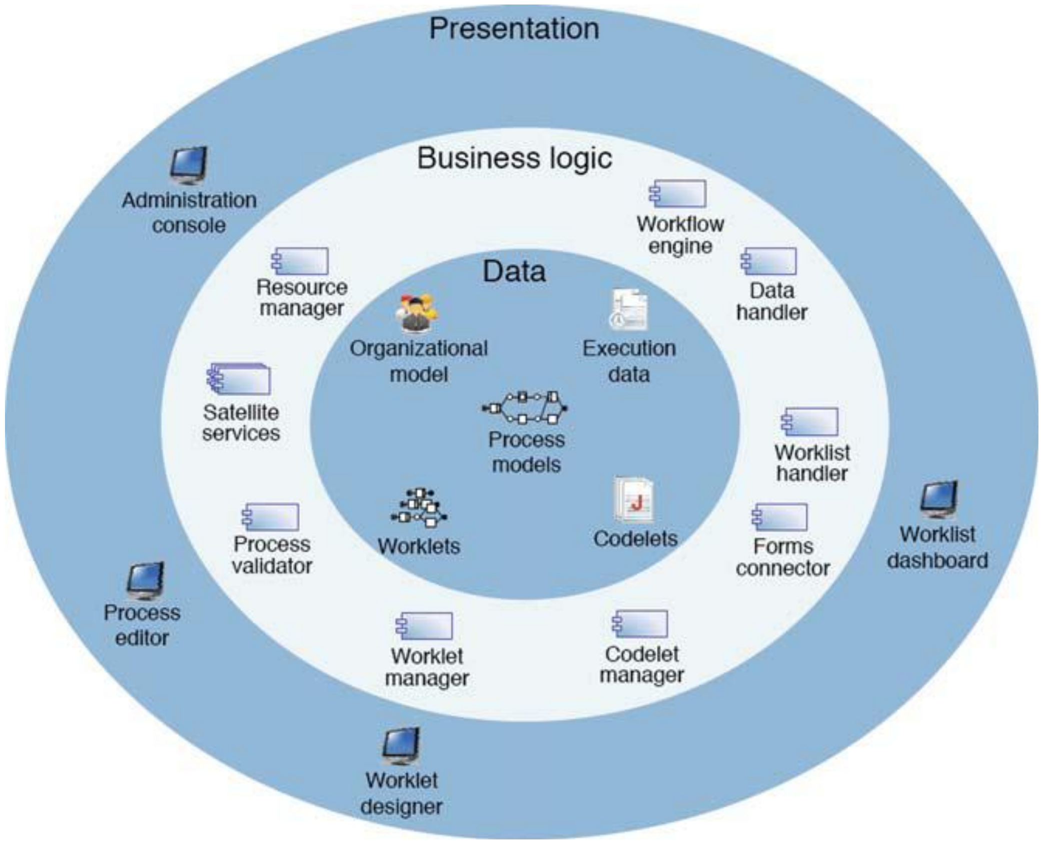


Figure 2.2: The YAWL system architecture

Image Source [20]

2.5.1 Data layer in YAWL

The inner layer in the figure is the data layer, which stores workflow specifications. A workflow specification describes three aspects of workflow: the control flow logic, the data definitions and the resources that are required for execution of the various tasks. The *organization model* represents the resources. It specifies information about the participants, such as roles, capabilities, and privileges. E*xecution data* is also part of the data layer, which includes *case data* and the *execution logs*. “case data” refers to data associated with individual cases, specifically values of variables defined at the level of the root net of a workflow specification and values of variables defined at the level of tasks or subnets thereof. The execution logs contain the entries representing an event such as the creation or completion of a case or the start or end of a task. Log entries include values such as start/end times, input/output data, and end status[12].

2.5.2 Business logic layer in YAWL

workflow engine (*Engine*) is the core service of the YAWL system. The engine is responsible for creating, routing, and synchronizing the execution of work items according to a workflow specification. In order for the other services to work, every other service must first establish a valid session with the Engine before further communication can proceed. The Engine, on receiving a connection request from a service checks to see if the service is authorized and session handle token is passed back. This session handles will be passed as a parameter to engine for all further requests.

For management of individual tasks, a number of services communicate with engine to extend environment. The task related services are responsible for handling the work items. A work item is a runtime instantiation of a task defined in a workflow specification together with its associated data. It is instantiated during the execution of process instance when the control flow reaches the task. When a work item is instantiated, it is said to be *enabled*. Task related service that was associated with the task at design time will process the enabled work item. If no such association of service is defined, it is handled default by the resourcing process.

Work items can be defined *manual* or *automated* for resourcing purposes. Manual work items will be handled directly by a human worker, while automated work items are routed to a software application for automated processing. Manual work items are routed to the *resource manager*, while automated work items are routed to the *codelet manager*. The distribution of manual work items to organizational participants is governed by a resource-based access control mechanism handled via the *resource manager*. The service was built with this resource manger and only handles the manual tasks, which need to be assigned to a human worker.

Before the service was developed, the resource manager used to handle the manual work items in the following way. Resource manager identifies the set of participants to whom the work item should be offered for execution and transfers this information is transferred to the *worklist* service, which handles interactions with the end user. *Execution logs* are used to store all the operations performed by the human resources. Furthermore, the resource manager offers access to system administrators to create and modify the organizational model via the Administration console.

The *worklist handler* is responsible for offering and allocating manual work items to users and transferring the associated business data through a Web form. This service provides a dashboard through which each user can query the set of work items being offered to them, can allocate a specific work item to themselves, start a work item, or complete it. A web form allowing the users to view and enter data related to work item is generated when the work item is started. When a work item is completed, the worklist handler submits to the Engine any data gathered during the execution of the work item. The service uses this worklist handler to populate the dashboard of the users to whom the task is being allocated or offered[12].

The worklist handler and the *forms connector* can be combined to associate manual work items to custom made Web forms. The path of a manual work item through the involved services is illustrated in Figure 2.3.

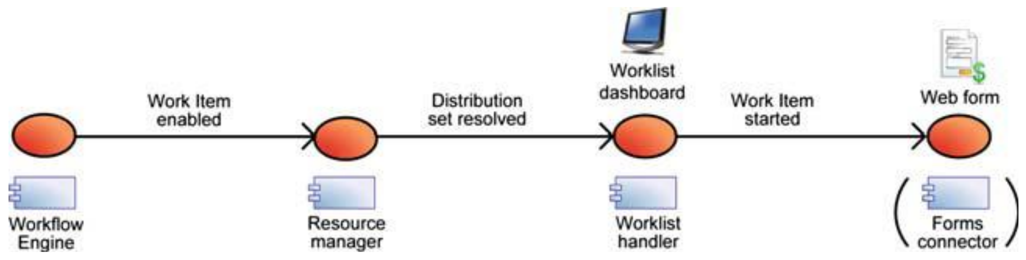


Figure 2.3 : path of a manual work item through the YAWL services

Image Source [19]

2.5.3 Presentation Layer in YAWL

The presentation layer contains all the items that don't directly with the engine. YAWL Editor allows users to create and edit workflow specifications. The Editor uses *process validator* to handle the validation of the workflow specifications, both syntactically and semantically. Developers can introduce the custom services which interact with engine or with other task related service described above.

CHAPTER 3 - YAWL Services and Interfaces

Yawl offers variety of services and interfaces using which services can interact with each other. Applications can also be built to interact with the services using these interfaces. Fig 3.1 shows different services and interfaces in YAWL [13].

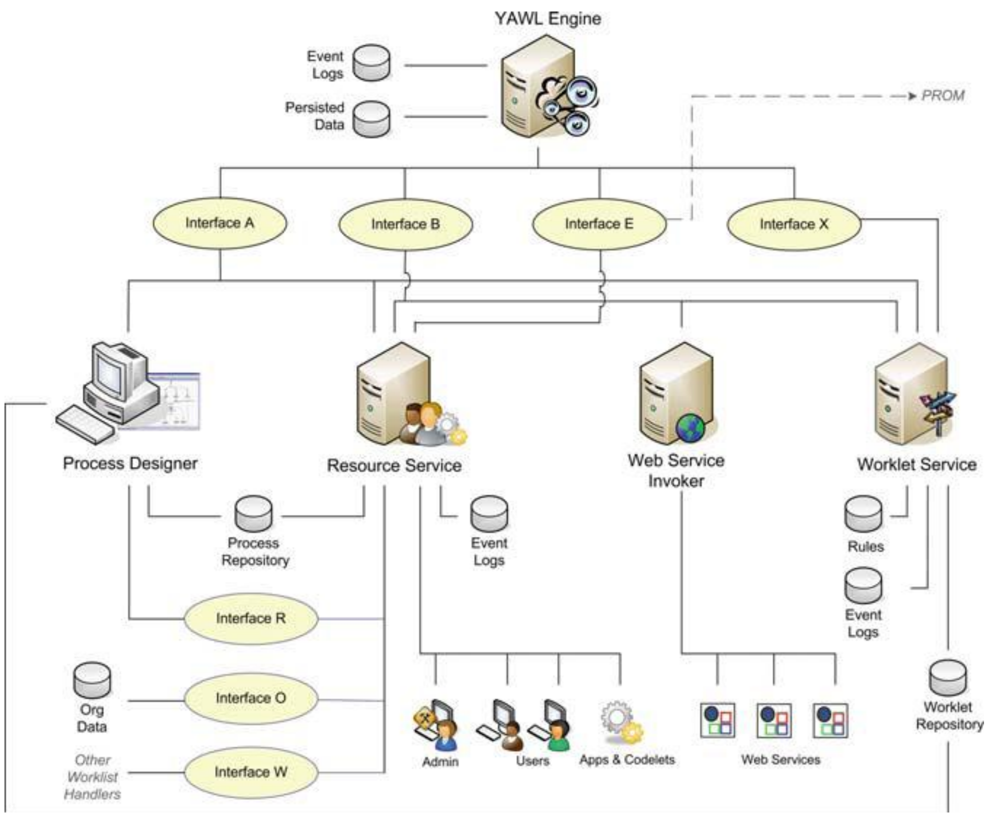


Figure 3.1: YAWL System’s core services and their interfaces

Image Source [18]

3.1 Engine

Engine is the core service of the YAWL system. The workflow specifications are loaded into the engine and the repository is used to store these specifications. This repository is referred by the engine to instantiate and produce cases. Engine is responsible and manages execution of each case according to its data mappings between the tasks, control flow description and the current state of the task. At each state, the engine decides which are the work items that need to be offered for processing and what events should be announced to the YAWL environment. Every task should be associated with one of the YAWL service during the design time. If not specified then Resource service will handled that particular task. The YAWL System's design is in such a way that it should be completely unaware of services interacting with it so that they could be served in generic way. From an engine perspective, each service is a "black-box" that avails itself to process data served by the engine through its interfaces. Thus, the Engine needs no knowledge of its external environment. This approach makes the Engine more lightweight, while providing a flexible and extensible framework for plugging in additional custom services into the YAWL System [13].

The Engine interacts with other services in the YAWL System through four interfaces. They can be accessed by "org.yawlfoundation.yawl.engine.interfce" package from YAWL distribution. The Engine interfaces are the following:

Interface A, which provides endpoints for uploading and unloading the workflow specifications. It can be used to register and remove the association of tasks to external services. It can also be used for user connections and disconnections. It can be accessed using the "org.yawlfoundation.yawl.engine.interfce.interfaceA" package in YAWL distribution.

Interface B, which can be used by services to establish a session with the Engine, launch process instances, check work items in and out of the Engine, and retrieve process data and state information. Interface B delegate's management of the interaction of those component types to a YAWL Service. Thus all communication between the Engine and external components are handled through a single, generic interface.Interface B also generates a number of events in the lifecycle of each case. They are: EnabledWorkItem, CancelledWorkItem, CompletedCase, CancelledCase, TimerExpiry, EngineInitialised, WorkItemStatusChange.

Interface E, which can be used for the retrieval and analysis of process logs.

Interface X (for exception), which can be used for the detection and handling of runtime process-level exceptions.

3.2 The Editor

YAWL workflow specifications are designed and verified in a design environment called the YAWL Editor. The Editor is distributed as a Java application, but other YAWL system services are distributed as web services. It uses Interface A to communicate with the Engine and obtains the list of YAWL services that are registered with the YAWL engine so that the tasks can be associated with the YAWL service. It also communicates with a running Resource Service through Interface R (described in Sect. 3.3) to obtain lists of the various organizational resources and codelets currently available. A tool palette is provided by the editor for modeling tasks conditions. There is a canvas provided where we can choose and place the items to design the process specifications. Routing constructs may be added to tasks and arcs added to join tasks and conditions to form a complete workflow graph representing a particular business process. There is also provision for verifying and analyzing the completeness and soundness of the specifications by using various algorithms.

When design is complete, the process definition may be saved to a disk file. The disk file will contain an XML representation of the specification conforming to the YAWL specification schema. By passing the reference to this file or give its actual content via interface A, this specification file can be loaded into the engine[14].

3.3 Resource Service

The role of the Resource Service is to allocate enabled work items to resources so that they can be processed. There are four sub-services which come under Resources Service:

- A ***Resource Manager***, which manages the allocation of resources to work items.
- A ***Worklist Handler***, a web-form based user interface that provides users with the ability to interact with and process work items.
- A ***Forms Connector***; This sub service allows the designer to implement specialized forms for particular work items. These kind of forms are manages by this service.
- A ***Codelet Service*** that maintains and executes codelets selected for automated tasks[13].

Resources may be people or may be an application, service, or codelet of some kind. At design time, each task in a specification may be marked as *manual* or *automated*. In the Resource Service, a human resource is referred to as a *Participant*. Each participant may perform one or more *Roles*, hold one or more *Positions,* and possess a number of *Capabilities* which will be mentioned later in this chapter.

The resource service communicates with the engine through interfaces A and B and at the same time exposes functionality through three interfaces of its own:

- ***Interface O*** provides an interface to organizational data sources. The organizational data model can also be populated using this interface. Using this interface any pre existing organizational data sources can be directly plugged in to the resource service. The interface is designed to be generic so that it can allows various data sources like LDAP servers, web services, spread sheets or even plain text files by only small implementation.
- ***Interface R*** provides access to the organizational data by authorized external clients or services like editor and custom service. This interface provides sets of both human resource and codelet descriptors.
- ***Interface W*** exposes the entire worklist routing functionality to allow external, specialized worklists to be developed and implemented. Such worklists may be used in conjunction with, or completely replace, the default worklist handle.

In order to access the Resource Service using Interfaces R and W, first a session must be established with it similar to the way in which sessions are established with engine. Resource manager had its own session manager to manage external clients.

3.3.1 Architecture of resource service

Figure 3.2 shows the service's architecture of resource service. It has six separate interfaces using which it communicates with engine and provides support for external environment. The service communicates with the engine via three interfaces:

- **Interface A** is used to check the connections with engine and upload and unload the process specifications and add/ remove custom services.

- **Interface B** is primarily used to handle work item and case events, to launch and cancel cases, and maintain a running cases list to checkout and check-in work items, to suspend, unsuspended, and skip work items, to create new work item instances (for dynamic multiple instance tasks), and to retrieve data schemas and other metadata for tasks and specifications.
- **Interface E** is used for querying the Engine's process logs when formulating an allocation strategy based on the historical information.

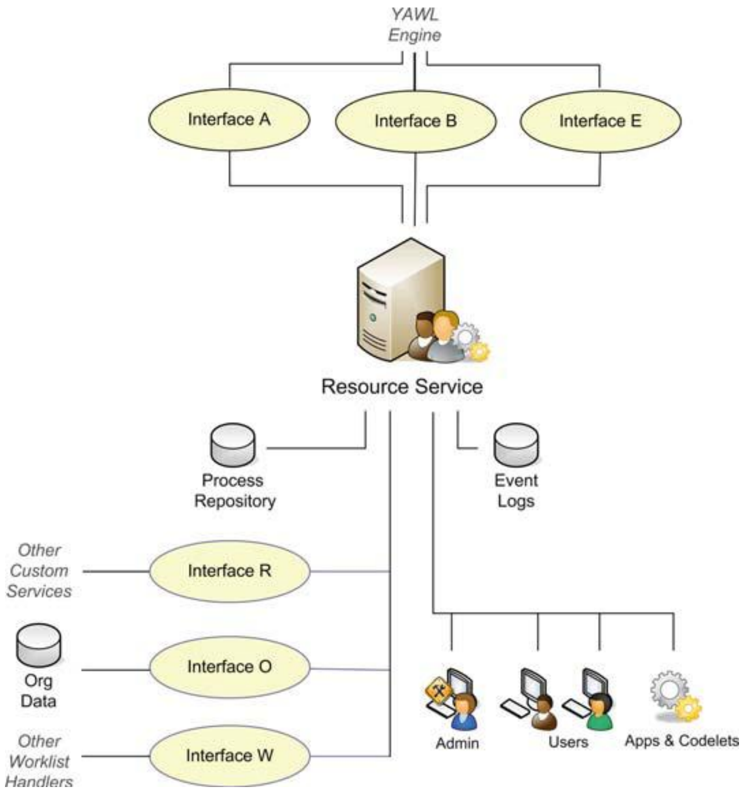


Figure 3.2 : Resource Service Architecture

Image Source [16]

In addition to the engine interfaces, the Resource Service provides three interfaces *Interface O, Interface R* and *Interface W* which were described in previous section.

The *Process Repository* is the place where the process specifications are stored. These specifications are created by using the YAWL editor and loaded into the engine using the resource service's administration tools. The *Event Logs* is a data store where a process log is written containing all resourcing decisions made for each work item handled by the service [15].

3.3.2 Work list for participants

Worklist is a graphical representation of participants work queues via a series of web forms. Each participant has access to their own worklist. Participant's queues are held internally by the service, and the default worklist is one representation of them, allowing a participant to interact with the YAWL environment. Each worklist of a participant consists of four work queues: *Offered*, *Allocated*, *Started*, and *Suspended*. Some queues are concerned with processing the work item, while others provide for changes to the work item's resourcing. The administrator work list consists of two queues *Unoffered* and *Worklisted*. The graphical interface also shows the possible actions that could be taken for a currently selected work item.

- **Offered queue**: The Offered queue lists the work items that have been offered to a participant. Each work item in an offered queue may have potentially been offered to a number of participants. As a work item on an offered queue may have been offered to a number of participants, there is no implied obligation to accept the offer. A participant may take the actions on a work item in an offered queue. Those actions are *Accept Offer*, *Accept &Start* and *chain*.
- **Allocated queue**: The Allocated queue lists the work items that have been allocated to a participant. A work item on an allocated queue means that it has been allocated to that participant alone, and it is implicit that the participant will at some time start the work item and perform its work. A participant may take one the *Start, Delegate, Deallocate, Skip, or pile* actions.
- **Started queue**: The Started queue lists the work items that have been started by or for a participant. Each work item on a started queue has begun execution in a system sense, but may or may not have had any actual work begun for it by the participant – such work is performed by the participant viewing, editing, and finally completing the work item. The participant may take any one of the *view/edit, suspend, complete* actions.
- **Suspended queue:** The Suspended queue lists executing work items that have been suspended [15].

3.3.3 Organizational data

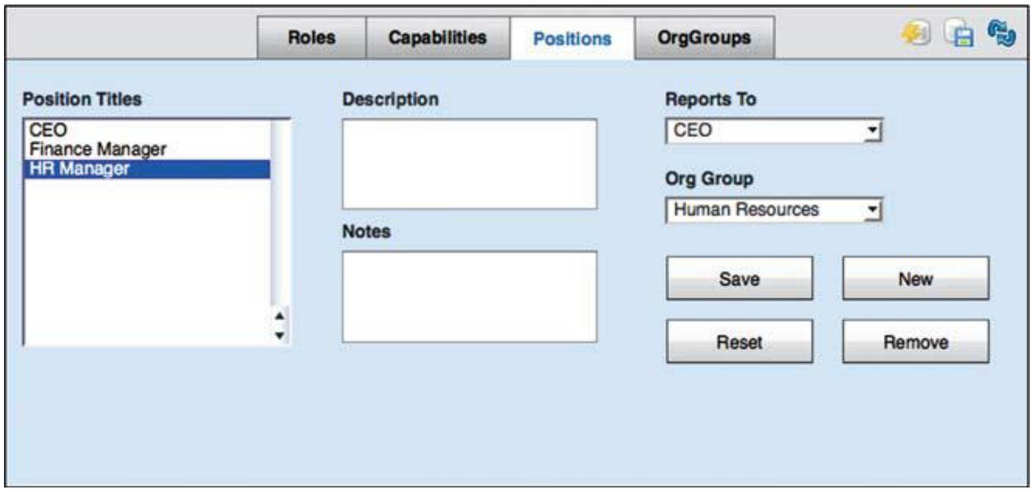
The organizational data is the data source that describes the resources. Resource Service needs access to data that details the attributes of a set of human resources and the relationships between them. In the Resource Service, a human resource is referred to as a participant. A participant is someone who willingly participates in the performance of tasks within a workflow instance. A participant is a member of some kind of organization, and therefore is defined in various ways within an *organizational model*. An organizational model describes the relationships between the participants of an organization, and their jobs, roles, duties, managerial hierarchies, and so on. Resource service has a default organization model database and tools to administrate it.

The participants in the organizational model have the following entities:

- **Role:** A role is a duty or set of duties that are performed by one or more participants. There may be several participants performing the same role (e.g., a bank may have a number of tellers), and so a role in an organizational model may contain a number of participants and a participants may perform multiple roles. Therefore, a role may be considered as a grouping of participants who share the same duties within an organization.
- **Capability:** A capability is some desired skill or ability that a participant may possess. There may be several participants within an organization possessing the same capability, and a certain participant may possess a number of capabilities. In the YAWL model, a participant may possess zero or more capabilities.
- **Position:** A position typically refers to a unique job within an organization for the purposes of defining lines-of-reporting within the organizational model. Although generally a participant will hold exactly one position and each position in the model will contain exactly one participant, to maximize flexibility these restrictions are not enforced in the YAWL model. Within YAWL, a participant may hold zero or more positions.
- **Org Group**: An organizational group (org group) is a functional grouping of positions. In the YAWL model, each position may belong to zero or one org groups [15].

3.3.4 Org Model Maintenance

There are set of database tables which store the various entities and relationships of the default organizational model. These tables may be populated and maintained via the *User Management* and *Organizational Data Management* forms, part of the toolset available to Resource Service administrators. Figure 3.3 shows an example of the *Organizational Data Management* form, with the *Positions* tab selected. Entities can be added, removed, and modified, and relationships between entities can be established or removed using this form.



Position Titles: CEO, Finance Manager, HR Manager

Description:

Notes:

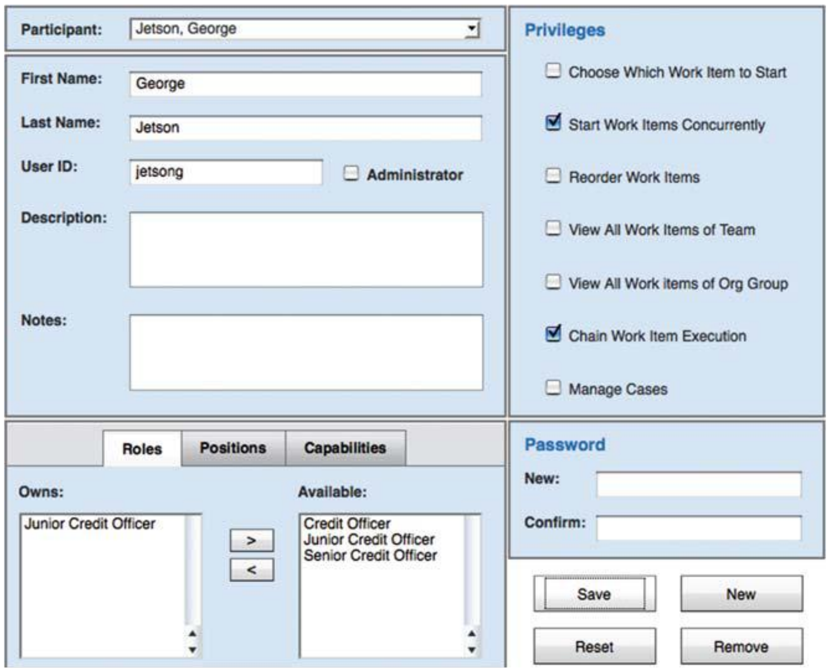
Reports To: CEO

Org Group: Human Resources

Figure 3.3 : The Organizational Data Management form (Position tab selected)

Image Source [17]

An example of the *User Management* form is shown in Fig. 3.4. Participants may be created, modified, or removed using this form, and may be assigned or removed from roles, positions, and capabilities.



Participant:	Jetson, George
First Name:	George
Last Name:	Jetson
User ID:	jetsong
☐ Administrator	
Description:	
Notes:	

Privileges

- ☐ Choose Which Work Item to Start
- ✓ Start Work Items Concurrently
- ☐ Reorder Work Items
- ☐ View All Work Items of Team
- ☐ View All Work items of Org Group
- ✓ Chain Work Item Execution
- ☐ Manage Cases

Owns:	Available:
Junior Credit Officer	Credit Officer
	Junior Credit Officer
	Senior Credit Officer

Password

New:	
Confirm:	

Figure 3.4 : The User Management form

Image Source [17]

3.4 Worklet Service

Worklet service is a combined term for two separate services. They are a Selection service, which enables dynamic flexibility for process instances, and an Exception Service, which provides facilities to handle both expected and unexpected process exceptions at runtime. A *worklet* is a small YAWL workflow specification that has been designed to execute as a substitute for an enabled work item, so that it contextually handles one specific task in a larger process instance. At design time, any task may be associated with the Worklet Selection Service to enable dynamic flexibility for that task.

At runtime, on receiving notification from the Engine of a work item-enabled event, the Worklet Selection Service selects an appropriate worklet from a repertoire of worklets defined for that task, by applying a set of extensible rules to the context of the work item and its parent case. The work item is checked out of the Engine via Interface B, the corresponding data inputs of the work item are mapped to the selected worklet, which is then loaded via Interface A and launched via Interface B, in the Engine as a separate case. When the worklet completes, the service receives a *CaseCompleted* notification, and then maps the worklet's output data to the output data of the original work item, which is then checked back into the engine, allowing the original process instance to continue[13].

The Worklet Exception Service extends from the Selection Service and uses the same repertoire and dynamic rules approach to provide detection and handling support for process exceptions that may occur during the life-cycle of a case. Instead of worklets, the Exception Service maintains a repertoire of exception handling processes (called *exlets*) for each specification, which are selected and invoked in the event of an exception occurring, to perform any required actions and compensations defined. While the Selection Service is invoked for certain nominated tasks in a process, the Exception Service, when enabled, is invoked for *every* case and task executed by the Engine, and will detect and handle up to ten different kinds of process exception. As part of an exlet definition, a process modeler may choose from various actions (such as canceling, suspending, completing, failing, and restarting) and apply them at a work item, case, and/or specification level, using a graphical tool [13].

CHAPTER 4 - Design and Implementation

The service built uses the Resource manager sub-service of the resource service of YAWL. This service is deployed as a web service, along with the other services that YAWL supports. Then service was built in conjunction to the resource manager service to help the administrator in allocating the work items to the participants.

4.1 Architectural Design

The following figure shows the architectural design of the entire execution scenario of the project.

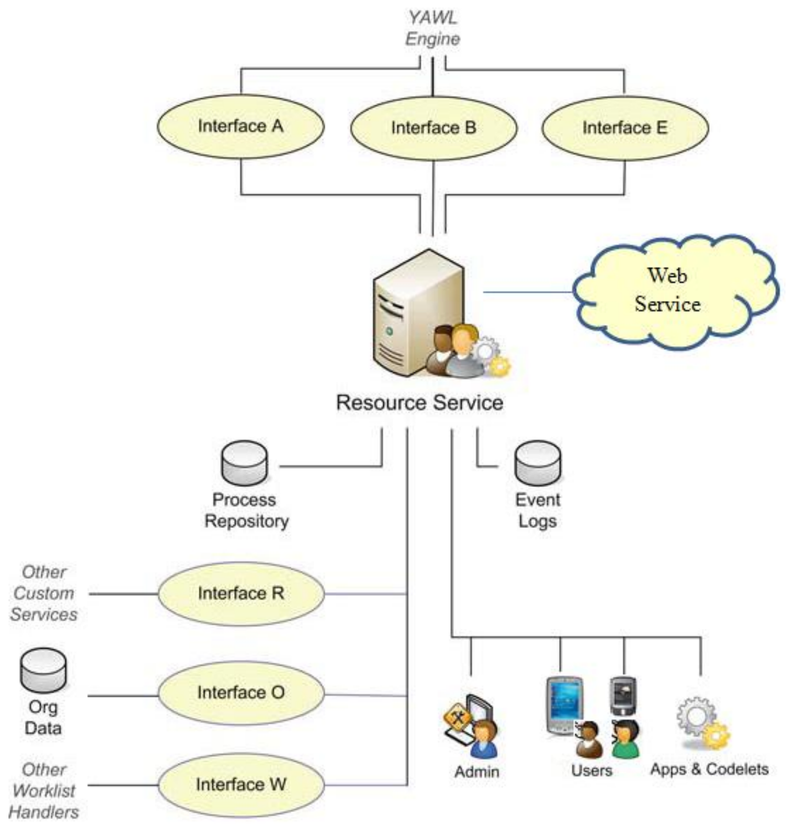


Figure 4.1 : Architectural implementation

Image Source [16]

For the system to function, the web service should be running along with YAWL engine on the same server as the engine, as well as the resource service is on. The figure shows the position of the web service in the architecture of the resource service. Users interact with the resource service to get an access to their worklist by using a smart phone or a PDA. The web service works on the worklist of the Admin by using help of resource manager. Resource manager communicates with the other sub services in the resource service and with the engine to populate the respective participant's worklists. Resource manager users interface B to get these details and it uses interface W to populate the worklist.

4.2 Initial Implementation

Initially, to achieve the goal, the resource service was not completely utilized. Instead of using the Resource manager sub service, the external service called *YAWLWSInvoker* was used. This is an external service to the YAWL environment using automated tasks are handled. The idea was to make the task as automated and use this service to inform the doctors/ nurses about the task that they need to attend to. The following figure shows the initial implementation.

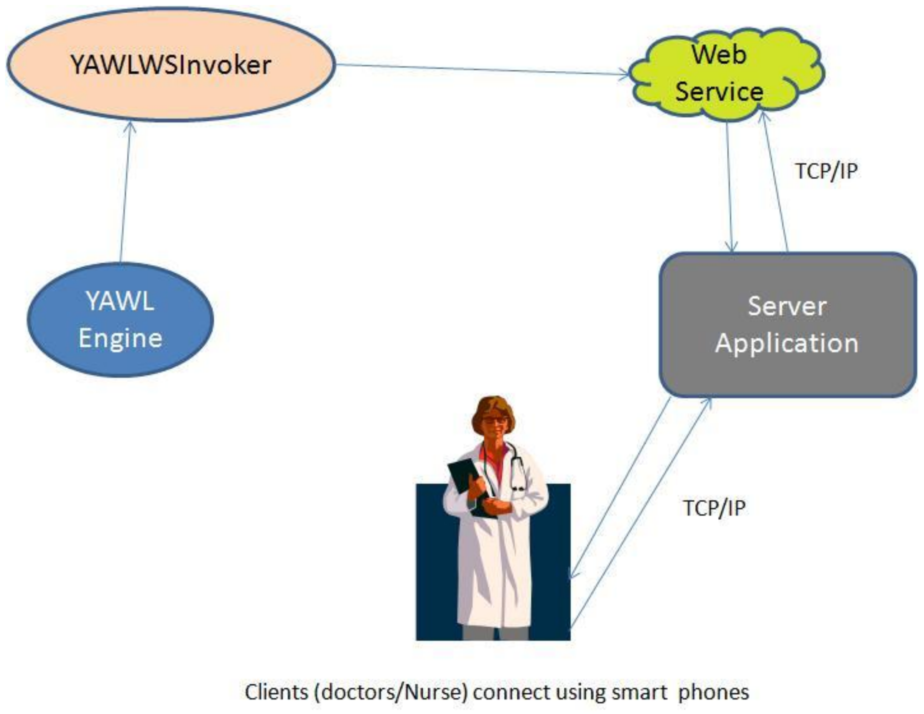


Figure 4.2 : Initial implementation architecture

In the above architecture the Web service, Server application and the client application on smart phones were developed. The purpose of assigning the task to the doctor/ Nurse was achieved by using this architecture. But this architecture has its own drawbacks. In this architecture the authorization for the doctor cannot be provided. Anyone who has the smart phone and the developed application running can register themselves to the server application claiming that they are doctors/ Nurses. And also there is no proper implementation of the organization model that needed in case of hospitals. In hospitals participants like doctors/Nurses are defined with set of roles and duties. But that kind of distinction cannot be achieved by using this architecture.

4.3 Implementation

Figure 4.1 shows the architectural implementation of the project. Now, the functionality of the resource service was fully utilized. In the architecture the Web service was implemented which needs to be deployed along with the YAWL engine on Tomcat server. Web service is a collection of Java classes which are built using the Resource manager sub service of the Resource service. This is because; resource service provides authentication mechanism for the participants who are in the organizational model. The hospital organizational model can be declared using the resource service with the help of interfaces provided. The problem with the initial architecture was authorization was solved by using the resource service.

In the initial architecture, the tasks in YAWL needed to be automated so that it can use the YAWLWSInvoker to invoke the webs service. But with the current implementation, those tasks need not be automated. In fact the tasks needed to be manual tasks. The web service, with the help of resource manager accesses the worklist of administrator and assigns the task to the appropriate participant.

4.3.1 Algorithm

1. The first step is to access the organizational model and constructs the participant lists based on their roles. There will a list corresponding to each role of the organization. The following is the pseudo code to achieve this.

```
getParticipants( )
{
        For each Participant in organization{
                If( list for the participant role already exists)
                        Add the participant to the list
                Else
                Create a new list and add participant to the list.
        }
}
```

2. Second step is to get the unoffered work items and figure out which role needs to handle that work item. Then choose the available participant from that role randomly and assign the task to him. The following pseudo code handles / achieves this.

```
assignUnofferedItems( )
{
        getQueuedItems( admin’s unoffered list)
        get the default value in “toWhom” variable from the task design time.
        Get the value of timer if the task is timer enabled.
        Do
        {
                Get random participant from the role based list constructed in the previous
                Step. the list is decided based on value of “toWhom” variable

                Get the list of logged in participants.
                If( the randomly selected participant is in  the logged in participants)
                {
                        Assign the workitem to that participant
                        If( timer != 0)
                                Start a timer Thread.
                }
                Else
                        Continue the loop
        } until task is assigned to an available participant

}
```

3. The third step is the timer thread step. In this step if the timer of the task expires the work item needs to be reallocated to different participant who is available. The following pseudo code achieves this.

```
timerTask( )
{
        Wait for the timer to expire.
        If( work item is still alive)
                Do
                {
                        Get random participant who is available with particular role
                        If( previous participant ! = new participant)
                                Reallocate the item to new participant
                        Else
                        Continue the loop
                } until the task is reallocated.
}
```

4.3.2 Implementation details

YAWL implementation has “ResoruceManager” implemented as a singleton class. To implement the first step in algorithm, this is to create the list of participants based on their roles. To achieve this, ResoruceManager class provides a method called *getParticipantMap()*. This method returns a HashMap containing the “Participant” objects as the values. Once the participants are obtained they are stored in ArrayList. To determine the list each participant should belong to each participant in the ArrayList is examined. “Participant” class provides a method called getRoles*()*, which will give all the roles of a particular participant. If a participant has more than one role he will be there in both the lists corresponding to each role.

To get the work items that needs to be handled the Unoffered worklist of the should be accesses. “ResoruceManager” provides method called “*getAdminQueues()*” which will return the “QueueSet” of the administrators. QueueSet class provides a method “*getQueuedWorkItems()*”

by taking the appropriate queue as the parameter. Each queue has an associated constant which should be passed to this method. For "Unoffered" Queue the constant . It returns a "WorkItemRecord" object which describes the work item completely. Now to find out the which participants are active Resource manager's *getActiveParticipantsAsXML()* method is used. This method returns the active participants in XML format. This is cleaned to get the id's of the active participants.

Once the active participants are obtained and the work item to be allocated is obtained, ResoruceManager's *assignUnofferedItem()* method is used. This method removes the item for the unoffered queue and is moved to Work listed queue.

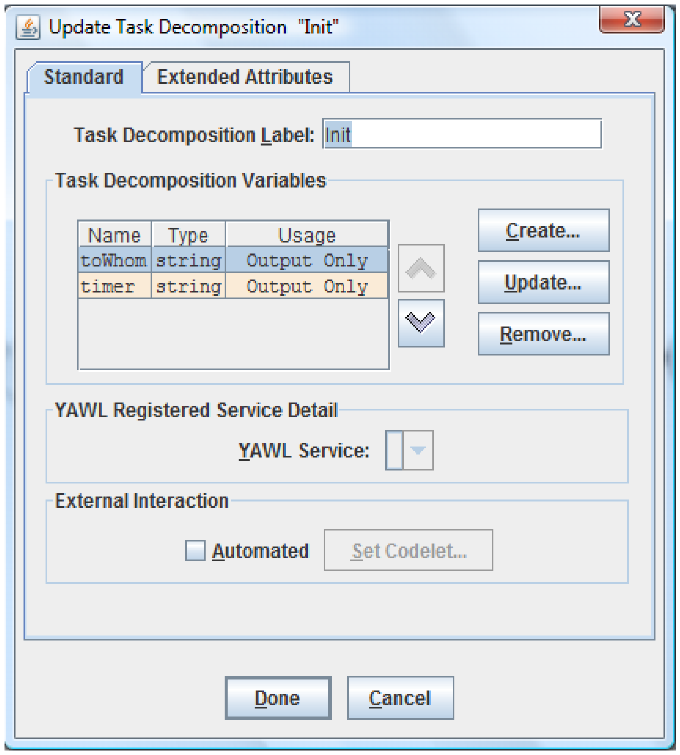
4.3.3 Features implemented

The following are the features that were implemented as part of the project. These are the features that needed to be manually addresses previously.

1. *Selection of the participant based on the role:* In the existing system, once the administrator selects the work item that needs to be assigned to the participant, a page showing all available participants is shown. It was the duty of the administrator to decide who to assign the task to. This required the admin to have the knowledge of each participant's role beforehand. With this service, this part of admin choosing from the available participants based on role is automated by maintaining the appropriate lists as described in algorithm.
2. *Finding the logged in participants:* In the existing system, the admin has no option to figure out whether the participant is logged in or not. He only sees the list of participants of the organization. So, he assigns the task to some participant there is every chance that the participant is not available. But the service built ensures that work items will be assigned to the logged in users.

3. *Categorizing the task with respect to roles:* In the existing system, there is no way to specify whether a particular task needs to be handled by a doctor / nurse. By intelligently designing the work flows using the task attributes the service at run time determines

which role needs to handle which task. This is done using the task decomposition option provided by the YAWL process editor. The following screenshot shows the decomposition option in YAWL process editor.


Update Task Decomposition "Init"

Standard | Extended Attributes

Task Decomposition Label: Init

Task Decomposition Variables

Name	Type	Usage
toWhom	string	Output Only
timer	string	Output Only

Figure 4.3: Task decomposition detail in YAWL editor

Task decomposition is the area where one can define all the data that is needed for the execution of the task. I have used a dedicated variable called "toWhom" for implementing this feature. The default value of this particular variable determines which participant needs to handle this particular task. The default value of this variable should match with the role of the participants who is handling this task.

4. *Task Timer:* The semantics of the YAWL timer is that, when the time out occurs whatever might be the task status the task will lead to completion and work flow progresses forward. This doesn't ensure that the task is completed properly. In critical scenarios like hospital there is a need that every task completes properly. To ensure this task timer was introduced. When the time out occurs the task will be reallocated to

another available participant with the same role. This is achieved by using the Thread control features provided by the Java programming language.

Each timed task is associated with a thread which is responsible for reallocation of the task upon the timeout. The amount of time it should wait before reallocation is determined in a similar way to categorizing the tasks. There is a dedicated variable called “timer” that is used. The default value of the timer variable tells the amount of time after which reallocation can occur in minutes.

On the smart phones the resource service is accessible via a browser. So, a participant can check his worklist to see the task assigned to him and complete the tasks accordingly.

CHAPTER 5 - Limitations

5.1 No support for offering the task

YAWL has a feature to offer the task to participant and it is up to the participant to decide whether the task should be handled by him or not. One can offer the task to more than one participant. But by using the current service, the option of offering the task is entirely removed. With the service, once the task is given to the participant, he has no choice but to handle the task.

5.2 Handling the cancel case

YAWL supports cancelation patterns where in a case which is running can be cancelled by the administrator resulting in the end of the progress of workflow. But when using the service there can be some inconsistencies handling the cancel case event.

CHAPTER 6 - Conclusion & Future Scope

6.1 Conclusion

Using the capabilities provided by Resource service in YAWL, a service was built which will automate the task assignment process in YAWL. The service adds the functionalities like selecting the participant based on roles, checking whether the participants are logged in, and categorizing the tasks based on task design. Additionally new semantics for the Timer is added to YAWL which helps in reallocating the resource to the other participants in case of timer expiry.

6.2 Future Scope

In the current system, even with the automated service the work flow has to be initiated manually. There application could be extended in terms of providing efficient way to initiate the work flow. Also, the User interface can be tweaked to work with the smart phones or PDA’s.

References Or Bibliography

Workflow definition

[1] http://www.webopedia.com/TERM/w/workflow.html

YAWL in healthcare

[2] Ronny Mans, Wil van der Aalst, Nick Russell, Arnold Moleman, Piet Bakker, and Monique Jaspers, YAWL4Healthcare, Chapter 21, pg 543 : Modern business process automation

[3] http://www.yawlfoundation.org/casestudies/health

[4] L Nantika Prinyapol, Joshua Fan, and Sim Kim Lau (2009), A Hospital Based Dynamic Platform Workflow Management. Introduction

[5] Michael Adams, Chapter 10, pg 261, Modern business process automation.

[6] http://en.wikipedia.org/wiki/YAWL

[7] Wil van der Aalst, Michael Adams, Arthurter Hofstede, and Nick Russell, Chapter 1, Section -1.4 :Modern business process automation.

[8] http://www.workflowpatterns.com/patterns/control/

[9] Nick Russell and Arthurter Hofstede, Chapter 2, section 2.4: Modern Business process automation.

[10] Michael Adams, Chapter 4: Section 4.2 – Section 4.3 : Modern business process automation.

[11] Michael Adams, Chapter 4: Section 4.4: Modern business process automation.

[12] Michael Adams, Marlon Dumas, and Marcello La Rosa Chapter 7: Section 7.2: Modern business process automation.

[13] Michael Adams, Marlon Dumas, and Marcello La Rosa Chapter 7: Section 7.3: Modern business process automation.

YAWL user Manual

[14] http://www.yawlfoundation.org/yawldocs/YAWLUserManual2.0.pdf

[15] Michael Adams, Chapter 10: Section 10.3, 10.4, 10.7, Modern business process automation.

[16] Michael Adams, Chapter 10: Section 10.4, figure 10.4 , Modern business process automation.

[17] Michael Adams, Chapter 10: Section 10.3, figure 10.2 , 10.3, Modern business process automation.

[18] Michael Adams, Marlon Dumas, and Marcello La Rosa Chapter 7: Section 7.3.1, figure:7.3 Modern business process automation.

[19] Michael Adams, Marlon Dumas, and Marcello La Rosa Chapter 7: Section 7.2, figure:7.2 Modern business process automation.

[20]Michael Adams, Marlon Dumas, and Marcello La Rosa Chapter 7: Section 7.1, figure:7.1 Modern business process automation.

[21] Nick Russell and Arthurter Hofstede, Chapter 2, section 2.4, figure 2.20: Modern Business process automation.