Web Curator Tool
Software Architecture Document

Version 1.0 FINAL
31/05/2006

Table of Contents

Revision History

Date	Version	Description	Author
21/03/2006	0.1	Initial High Level draft	Brendon Price
27/03/2006	0.2	Include original non-functional requirements	Brendon Price
04/04/2006	0.3	Initial Review	Nic Waight
31/05/2006	1.0	Updates due to Variation 001 and Variation 002	Brendon Price

Software Architecture Document

1. Introduction

The Software Architecture document outlines all the key design decisions made in creating the web curator system. The web curator system is a joint effort by the National Library of New Zealand and other international Libraries to build a system for the management and harvesting of digital web assets for preservation into the future.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

1.2 Scope

This document only discusses the key use cases that affect the software architecture and how the application will be deployed. Focus has been given to those use cases that are deemed technically challenging. The simpler use cases are not covered in this document.

1.3 Definitions, Acronyms and Abbreviations

WCT = Web Curator Tool
SOAP = Simple Object Access Protocol
RPC = Remote Procedure Call
OMS = National Library of New Zealand's Object Management System

1.4 References

RFP Requirement document - v3A-Web_Curator_Tool__RFP[1].DOC
Use Case Realisation: UCR1 Manager sites and permissions
Use Case Realisation: UCR4 Quality review
Use Case Realisation: UCR5 Submit to archive
Use Case Realisation: UCR8 Monitor and Manage Web Harvests
Use Case Realisation: UCR9 Logon
Use Case Realisation: UCR10 Scheduler
Use Case Realisation: UCR11 Manage users and roles

2. Architectural Goals and Constraints

2.1 Architecturally Significant Design Decisions

The Web Curator Tool specifies several key requirements that influence the way the system is designed. These key requirements can be grouped into the following categories described in the subsequent sections:

- Modularity/Plugability
- Supportability
- Security
- User Interface
- Resource Use

2.1.1 Modularity/Pugability

Modularity and Plugability requirements relate to the flexibility of the system and the ability to interchange components and external tools without additional coding effort. The requirement 6.4.4 has lead to the use of the following patterns and frameworks to deliver these requirements.

Making objects as independent as possible will support modularity. All objects will have clearly defined roles and behaviours, and will collaborate with other objects through clearly defined interfaces. A lightweight dependency injection framework will be used to provide “plugability” making it both easy and non-pervasive to change between different implementations of a module.

Interfaces (fulfills requirement 6.4.4)

An interface is a Java type that defines a “contract” between an object and its consumer. An object that implements an interface declares that it will meet the requirements of that interface. Consumer objects reference their collaborators via interfaces, so any object that implements the interface can be plugged in. As far as the consumer object is concerned, it does not care about the specific implementation as long as it fulfils this interface contract. In some cases, it may be desirable to split an interface between mandatory and optional behaviour.

The Web Curator Tool can query the implementation classes to see which interfaces are supported by that specific implementation. This allows the WCT to be independent of the implementations and still support the fact that different implementations may offer different levels of services.

Adapter Design Pattern (fulfills requirement 6.4.4)

The Adapter Design Pattern is a pattern that is used to simplify the integration with external systems. The Adapter Pattern specifies the creation of an interface to define the behaviour of the external system. For example, the Digital Archive System interface will define a method called submitHarvestResult. Whenever the Web Curator Tool must submit a harvest result, it will do so by making a call to this interface. The Adapter Pattern then calls for an adapter implementation that will convert between the system-defined interface and the real interface of the external system. To support a different external system simply requires that a new adapter be built and plugged into the application. Other external systems, including the Harvester implementation, will also be hidden behind an adapter pattern styled interface. For example, the Harvester will define methods such as (among others) the following:

- Run Harvest Job
- Set Bandwidth restrictions
- Monitor Throughput

Dependency Injection (fulfills requirement 6.4.4)

Objects rarely achieve anything in isolation. Instead, an application ties many objects into a network of collaborators that work together to achieve the goals of the system. The use of interfaces to isolate an object's behaviour from its implementation has already been discussed. However, this leaves the problem of how an object knows about its collaborators.

A solution to this problem is to use a pattern called dependency injection (or Inversion of Control). In this pattern, a management or framework layer is responsible for looking up the collaborating classes and tying them together. These frameworks are generally lightweight and remove all the lookup logic from the domain objects. This allows the domain objects to focus purely on solving the business problem at hand.

The dependency injection framework selected for the WCT implementation is the Spring Framework. Spring is configured through one or more XML files that declaratively describe which implementations should be wired together.

The combination of interfaces and Spring provide a powerful mechanism to build modular applications. For example, to use a different harvester or digital archive system, developers would build a new adapter that implements the defined interface and make a relatively simple change to the Spring configuration files. No changes to the rest of the WCT would be required.

2.1.2 Supportability

Requirements 6.7.4 and 6.7.5 relate to the support for multiple database systems.

Hibernate Persistence Layer (fulfills requirement 6.7.4, 6.7.5)

Another standard feature of an application is the ability to persist information to a database (or other storage facility). The storage and retrieval logic is rarely complex, but it is often time consuming and the error handling can be error prone. Instead of custom-building the persistence layer using JDBC (Java's database connectivity API), Hibernate an Object Relational Mapping (O/RM) tool will be used for this purpose. In combination with XDoclet (a source-code markup tool), Hibernate allows developers to tag properties that need to be stored in the database. The Hibernate framework then builds the SQL statements required for persisting objects to the database. Hibernate has a very wide range of support for different database dialects (variations on the standard SQL language), so helps to meet the database independence requirements of the WCT project.

In order to make Hibernate Persistence layer database agnostic it is important that all query logic be kept to the SQL-92 standard with no proprietary SQL extension being used. In general HQL will be used to define the Query logic, which is Hibernates own Query Language to provide another layer of abstraction.

2.1.3 Security

The identified security requirements 6.5.1 and 6.5.2 allow for a pluggable security framework that integrates with either an Enterprise Directory Service or DBMS repository.

Acegi Security Framework (fulfills requirement 6.5.1, 6.5.2)

The Acegi Security System provides authentication and authorization capabilities for Spring-based projects, with optional integration with popular web containers.

Security involves two distinct operations, authentication and authorization. The former relates to resolving whether or not a caller is who they claim to be. Authorization on the other hand relates to determining whether or not an authenticated caller is permitted to perform a given operation.

The Acegi Security Framework provides a very powerful pluggable interface that meets the requirements for fall-through authentication of different authentication sources. The framework provides interception filters to intercept authentication requests and act upon the security principle and request URL information provided.

The Acegi Security Framework also allows the Implementer to build and deploy a custom group, role and permission structure that can be configured and managed via the XML configuration file and some custom built classes.

2.1.4 User Interface

Requirements 6.2.1 and 6.2.10 enforce that the Web Curator Tool provide a web interface to the application that is compatible with as many browsers versions as possible.

General MVC Web Architecture (fulfills requirement 6.2.1, 6.2.10)

Web applications share requirements for a large number of features including validation, presentation frameworks, security, and controller logic. As these requirements are independent of the application itself, a large number of design frameworks have been built that can be reused to provide this functionality. Many of these frameworks are modelled on the Model-View-Controller (MVC) pattern. This pattern dictates a separation between the business logic (model) and presentation logic (view). The controller is then responsible for bringing the model and view together to provide the application. The view layer will be developed using Java Server Pages for presentation within the browser.

The MVC pattern ensures “separation of concerns” between the different layers. By separating the view component, its development can be assigned to a graphic design team, which requires a different skill set than development of the model or controller. The design team’s work can be easily integrated back into the application with minimum effort. The model is also entirely separated, allowing it to be tested independent of the user interface. Finally, the controller logic is built to bring the model and view together, assembling the full product into a usable web application.

The Spring Framework has been proposed to meet the modularity and extensibility requirements of the Web Curator Tool. The Spring Framework also provides a web MVC framework, which will be used as the framework for building the web component of the WCT. The Spring Framework provides a standard mechanism for implementing features such as the following:

- Per page security, in combination with Acegi Security Framework
- Automated parsing of parameter values
- Validation framework for validating parameter values
- Pluggable controllers, allowing for simple extension of the WCT
- View selection framework.

2.1.5 Resource Use

In order to make the Web Curator Tool flexible requirements 6.8.1 and 6.8.2 specify that the Web harvesting subsystem can be deployed onto physically separate machines to distribute the load.

Requirements 6.4.1 and 6.4.2 relate to the support of multiple platforms allowing for the wide adoption of the code running on varying platforms. Java provides the capability to build once run anywhere.

SOAP RPC (6.8.1, 6.8.2)

The Harvesting subsystem has the requirement that Harvests can be managed and configured from a central location, yet run from distributed physical machines. This drives the requirement for remote procedure calls to communicate with the remote devices.

Both the distributed Harvest Agent and the central Harvest Agent coordinator need to communicate using a standards based communication protocol, that can traverse through Network Infrastructure and Firewalls. SOAP over HTTP allows this type of behaviour. SOAP is also language agnostic, meaning that non-java based Harvest Agents could be built in the future to the same Interface.

For more information on the deployment of the web harvest agent, refer to the Deployment section of the document.

The other component that requires a remote interface is the WCT Digital Asset Store for storing the gathered web harvest materials. The WCT Digital Asset Store will also use SOAP with attachments for the transfer of ARC files from the Harvest.

2.1.6 Other Non-Functional Requirements

While there are additional non-functional requirements specified, that have not been covered in this section, they have been deemed not to affect the architectural components of the Web Curator Tool in any significant way.

2.2 Architecturally Significant Open Source Products/Frameworks utilised by WCT

Requirements 6.6.1 and 6.6.2 specify that the code developed for the Web Curator Tool will be released under an Open Source License and its license be compatible with those other open source products used in its development.

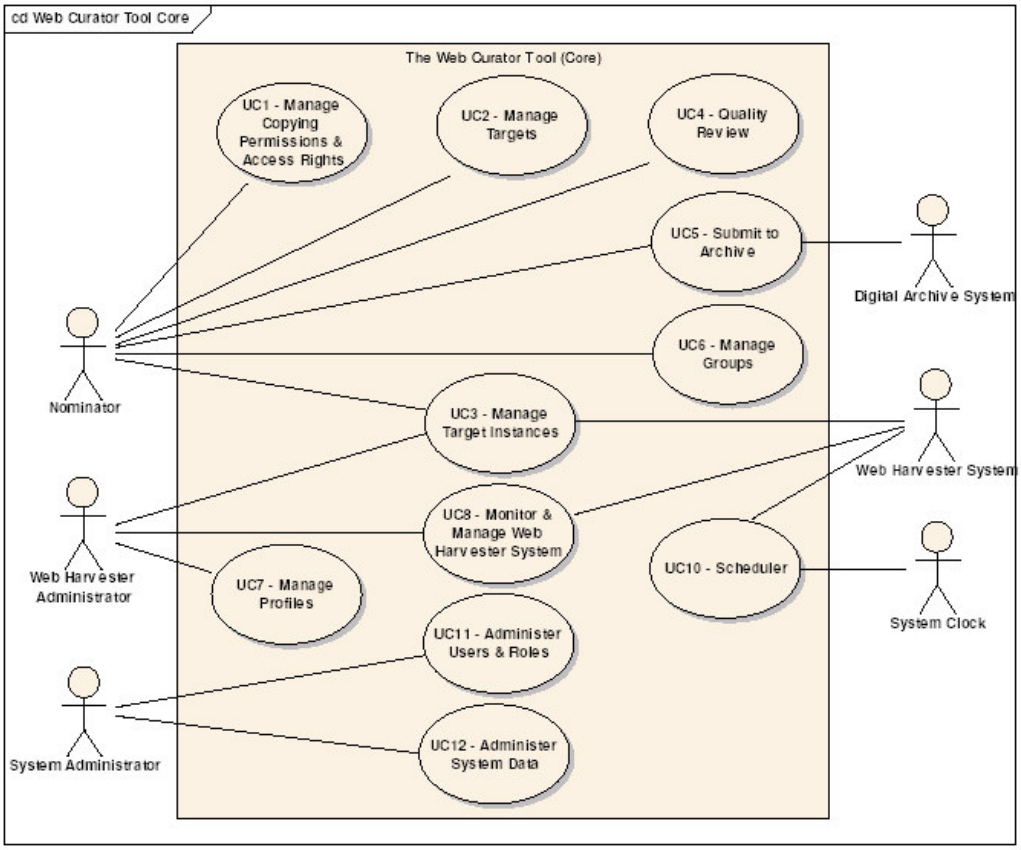
Below are listed the Architecturally significant Open Source components used:

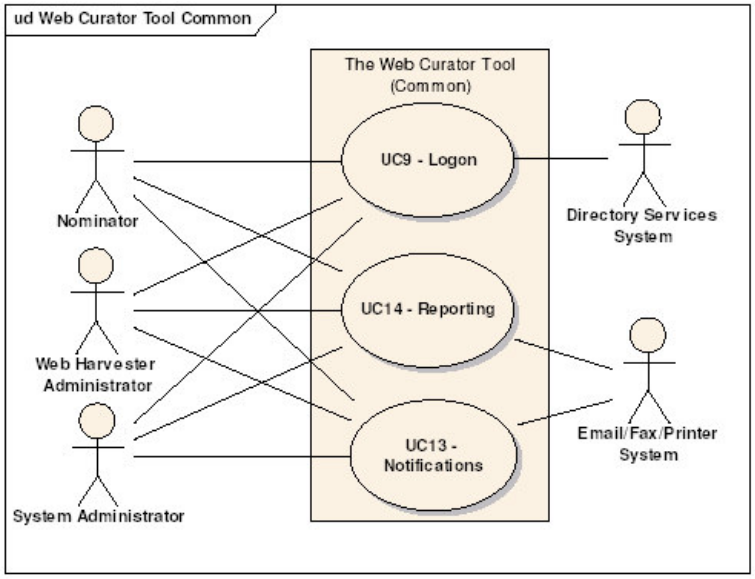
Product	Version	Description
Hibernate	3.0.X	Hibernate is an open source Object Relational Mapping tool that handles persistence of Java objects directly to a database without the direct coding of a JDBC layer. This increases the efficiency and lowers the error rate in developing the data layer for the WCT. Hibernate supports a wide variety of database dialects, including Oracle and MySQL. The JDBC code generated by Hibernate can be changed from one dialect to another by changing a configuration file and requires no application changes.
Spring Application Framework	1.2.X	The Spring Application Framework provides both a Dependency Injection framework and web application framework. Dependency injection promotes a loosely coupled and highly pluggable development architecture. The web model-view-controller (MVC) architecture supports a clearly defined and well understood model for developing web applications. The benefits include ease of extensions and maintainability.
Acegi Security System	1.0.0	The Acegi Security System is a security mechanism that ties into the Spring Framework. It is capable of providing both

		resource-level and method-level security constraints. Furthermore, it is easily extensible to support alternative authentication and authorisation mechanisms, which is critical to provide for the pass-through authentication described in use case 9.
Apache Axis	1.3	Apache Axis is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C. The Apache Axis implementation of SOAP also supports SOAP attachments that provide a way to stream large amounts of data via SOAP.
Apache commons logging	1.0.4	The Logging package is an ultra-thin bridge between different logging implementations. A library that uses the commons-logging API can be used with any logging implementation at runtime. It is intended that Log4j be used to carry out the actual logging.
Quartz	1.5.2	Quartz is a full-featured, open source job scheduling system that can be integrated with, or used along side virtually any J2EE or J2SE application.
Heritrix	1.8	Heritrix is the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project. This is the default Harvester implementation selected for use with the WCT.

3. Use-Case View

The following is taken from the Requirements document for the Web Curator Tool and shows the high level Use Cases and Actor interaction. These use cases form the basis of the Web Curator Tool.





3.1 Actors

3.1.1 Nominator

The Nominator is the primary actor of the WCT. They are responsible for identifying and describing Targets to harvest, recording copying permissions and access rights, reviewing the quality of the Harvest Result, submitting the Harvest Result to the Digital Archive System as well as other related tasks. Note that the role based security access mechanism of the WCT will be used to control the functionality available to individual users. This will allow the WCT to support many different classes of Nominator, some of whom may only require access to a subset of the functionality illustrated by the Nominator actor.

3.1.2 Web Harvester Administrator

The Web Harvester Administrator manages aspects of the Web Harvester System including monitoring it to ensure it is performing adequately. This actor also configures harvest profiles and has overall control of the Web Harvester System schedule.

3.1.3 System Administrator

The system administrator for the WCT. This actor is responsible for configuring the WCT system data and parameters, managing users and security roles as well as any other system administration tasks that may be required.

3.1.4 Web Harvester System

The system responsible for collecting web content. The WCT will support the Internet Archive's open source web crawler Heritrix as its Web Harvester System.

3.1.5 Digital Archive System

The system that will preserve the Harvest Result. For the National Library of New Zealand this will be the National Digital Heritage Archive (NDHA). An interface will be provided for interaction with this system actor, as it will differ from Library to Library.

3.1.6 System Clock

The clock of the operating system hosting the WCT. The system clock triggers scheduled events in the WCT.

3.1.7 Directory Services System

The Directory Services System used to authenticate user credentials.

3.1.8 Email/Fax/Printer System

Peripheral devices used to send and/or print notifications and reports from the WCT.

3.2 Use Cases

The following is a high level description of each use case identified in the Web Curator Tool.

3.2.1 UC1 - Manage Copying Permissions and Access Rights

Nominators can record and/or view details about web sites that require owner permission to

harvest as well as any access right restrictions that apply to the harvested web content. In some cases a Nominator will delegate the responsibility of seeking copying permissions and access rights for the web site to another user of the WCT.

3.2.2 UC2 - Manage Targets

Targets describe the web content a Nominator wishes to harvest. A Nominator can perform various tasks that assist with the workflow and day to day management of Targets.

3.2.3 UC3 - Manage Target Instances

Target Instances represent the harvests that have or will occur for a Target. A Nominator or Web Harvester Administrator can monitor and control Target Instances before, during and after their processing by the Web Harvester System.

3.2.4 UC4 - Quality Review

The Nominator checks the Harvest Result to verify it is of suitable quality and corrects any noticeable errors and/or omissions in the output. If the Harvest Result is not suitable, the Nominator deletes it. The Nominator can then optionally review and update the Target and schedule a new Target Instance for another harvest.

3.2.5 UC5 - Submit to Archive

Once the Nominator is satisfied the Harvest Result is of a suitable standard they use the WCT to submit it to the Digital Archive System. This submission process will involve converting the Target Instance Harvest Result and metadata into a common Submission Information Package format suitable for most Digital Archive Systems to receive.

3.2.6 UC6 - Manage Groups

Groups associate two or more Targets that are related in some way (e.g. Targets that have a common web content theme). A Nominator can use Groups to synchronise the harvest of multiple related Targets or indicate relationships between Targets that may be of benefit to the Digital Archive System.

3.2.7 UC7 - Manage Profiles

Profiles contain predefined settings for harvest control parameters. Nominators with the appropriate privilege can select profiles to use for their Targets. Nominators without the privilege to select profiles are forced to use the default profile of the WCT. A Web Harvester Administrator can create and modify profiles as well as set the default profile of the WCT.

3.2.8 UC8 - Monitor & Manage Web Harvester System

A Web Harvester Administrator can monitor the progress of the Web Harvester System, configure bandwidth restrictions and pause all Running Target Instances on the Web Harvester System.

3.2.9 UC9 - Logon

All users must authenticate with the WCT before they are granted access to any functionality.

The WCT will operate a simple credential management system but will also support integration with an enterprise directory services system for pass-through authentication.

*3.2.10 UC10 - S*cheduler

The WCT includes a Scheduler that initiates harvests for Target Instances by the Web Harvester System(s). It also purges the Harvest Result from the WCT after the associated Target Instance is archived and a set period has elapsed.

3.2.11 UC11 - Administer Users & Roles

A System Administrator can register users, manage user accounts and configure and assign roles to users. Roles are sets of WCT privileges configured by the System Administrator. Roles can be used to restrict the access individual users have to WCT functionality.

3.2.12 UC12 - Administer System Data

A System Administrator can change configuration parameters of the WCT, claim ownership of records and create and modify message templates.

3.2.13 UC13 - Notifications

A user can view and delete automated notifications from the WCT. The WCT sends notifications to users via their preferred delivery method(s). Delivery methods for internal notifications include email and via the messages section of the users In Tray. Notifications can also be sent to external individuals and organisations via email, fax or print.

3.2.14 UC14 - Reporting

A user with the appropriate report privilege can request a report from the WCT, print, or email the report and/or export the report to the local file system in an open format.

3.3 Use-Case Realizations

For more information on the use case realizations, refer to each of the use case realization documents.

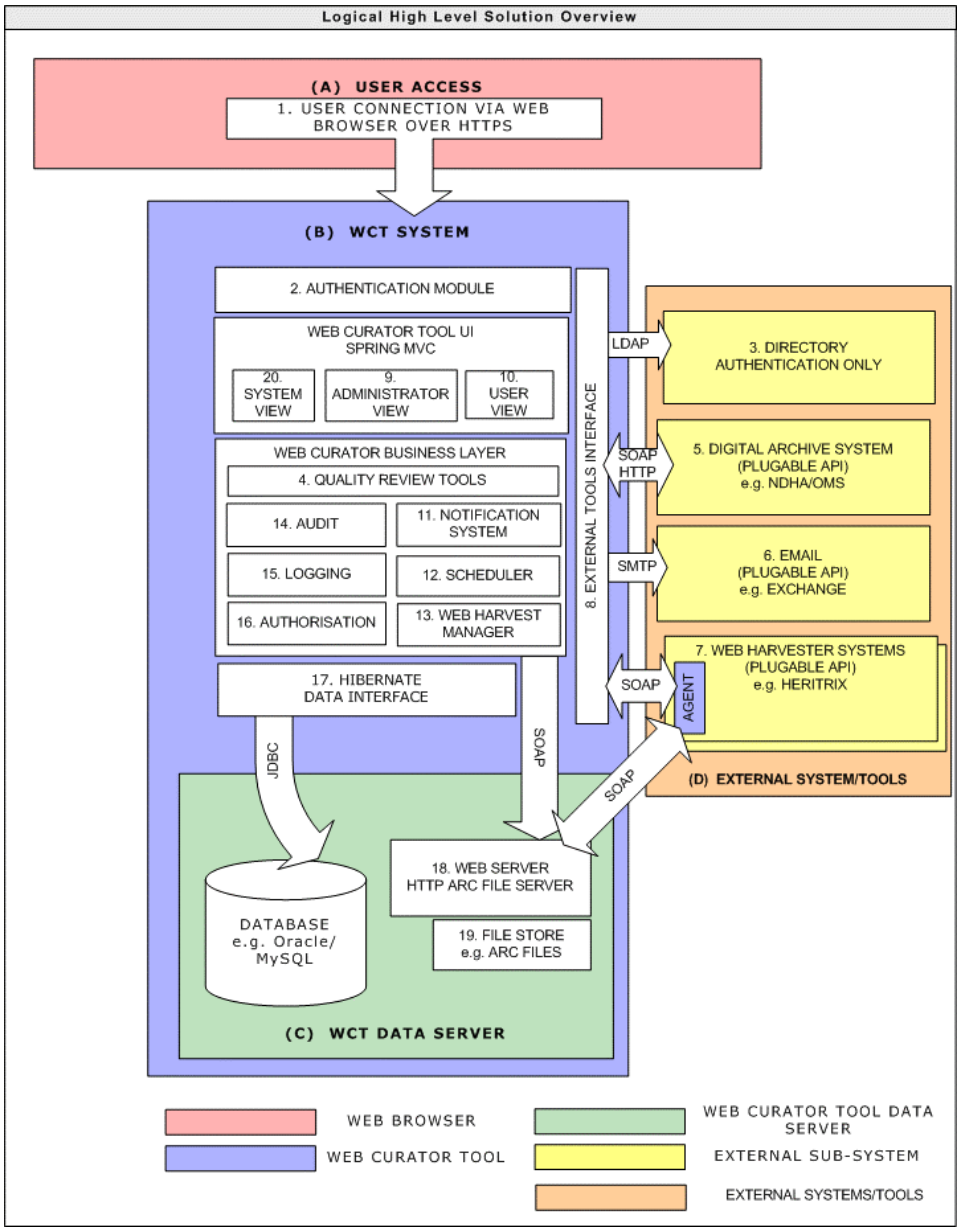
4. Logical View

This section of the document discusses the important logical components of the Web Curator Tool. It also touches on the packaging and communication paths for key components of the system.

4.1 Overview

The following diagram provides a high level overview of the components of the Web Curator Tool. The diagram represents the four main components of the system:

- Users who access the Web Curator Tool via a Browser (A)
- Web Curator Tool Application Server (B)
- Web Curator Data Server (C)
- External Systems/Tools (D)



All system access will be carried out by the end-users from a web browser over HTTPS

(encrypted connection) in order to protect the Logon credentials of the user. Based on the user's privileges they will be presented with the appropriate menu options within the Web Curator Tool. There are three main views of the system, the:

- System Administrator view (20)
- WCT Administrator view (9)
- WCT User view (10)

The System Administrator view (20) provides functions and features to manage the Web Curator Tool users and system specific settings.

The WCT Administration view (9) provides functions for managing aspects of the Web Harvester System and the configuration of harvest profiles.

The user view (10) provides an in-tray view of work to be completed by the individual and provides integration with external components.

The External Interface Tools layer (8) provides access to all external systems and tools required by the Web Curator Tool. These include Interfaces for:

- Directory Access for user authentication (3)
- Digital Archive System storage (5)
- Web Harvester Sub-system for distributing harvesting activity (6)

Component (C) of the diagram provides the ability to break out the data server onto different physical hardware, for load and performance reasons. The Data server is responsible for holding all of the Web Curator Tool data, including the ARC files (19) themselves. The ARC files are to be stored on the physical disk.

4.2 Package and system decomposition

The following section describes the packaging of the Web Curator Tool java components into logical groupings. The base package will be `org.webcurator`

The Core packages of the web curator consist of all the central components of the system, that are essential to its functioning. These include the scheduler, notification system, harvester coordinator and harvest agents and the classes used to communicate with the internal WCT digital asset store.

```
org.webcurator.core.common
org.webcurator.core.exception
org.webcurator.core.harvester
org.webcurator.core.harvester.coordinator
org.webcurator.core.harvester.agent
org.webcurator.core.profiles
org.webcurator.core.scheduler
org.webcurator.core.scheduler.servlet
org.webcurator.core.notification
org.webcurator.core.store
org.webcurator.core.store.arc
org.webcurator.core.store.tools
org.webcurator.core.util
```

The External package holds all the java components used to communicate with external system like the digital archive systems used to hold the end result of a harvest.

```
org.webcurator.external.archive
org.webcurator.external.archive.oms
```

The Authentication package holds all the components required for the authentication of users into the WCT. The WCT has the ability to communicate with both a directory server and/or the

WCT internal database for user authentication.

```
org.webcurator.auth
org.webcurator.auth.ldap
org.webcurator.auth.dbms
```

The Common package holds all the components needed by multiple parts of the system. In particular this includes the Audit component and utilities for manipulating ARC files.

```
org.webcurator.common
org.webcurator.common.arc
org.webcurator.common.audit
```

The Domain package holds all the domain layer objects, those that represent the business objects that make up the web curator system. This layer is also where hibernate is used to automatically manage and persist these objects to the database.

```
org.webcurator.domain
org.webcurator.domain.model.audit
org.webcurator.domain.model.auth
org.webcurator.domain.model.core
org.webcurator.domain.model.dto
```

The User Interface package hold all the classes related to rendering and managing the User Interface of the web application. The classes are broken into command objects, controllers, validators. These represent the model and controller objects that form part of the MVC pattern. The view component of the MVC pattern is represented by JSP's that do not have a package. The xxxx in the package structure below represent an occurrence of the MVC pattern. For example the home page would be `org.webcurator.ui.home.controller.`

```
org.webcurator.ui
org.webcurator.ui.util
org.webcurator.ui.xxxx.command
org.webcurator.ui.xxxx.controller
org.webcurator.ui.xxxx.validator
```

4.3 Common Functionality

The following section defines common functional classes within the WCT.

4.3.1 Auditing

The Auditing function is common across almost all use cases of the system. The auditing function will be provided through the `org.webcurator.core.util.Auditor` Interface. A single implementation of the interface will be provided that records all audit information to the database.

4.3.2 Ownable Objects

The security system of WCT relies on the concept of owners for objects within WCT. There are two types of Owner, an Agency or a User. Some objects in the system are owned by Users whilst others are owned by Agencies.

Two Interfaces will be provided to help make ownable object consistent. There will be an Interface for Agency owned objects called `org.webcurator.domain.AgencyOwnable` Similarly there will be an Interface for User owned objects called

`org.webcurator.domain.UserOwnable.`

A domain object in the system that has an owner must implement one of these two Interfaces. This allows the `org.webcurator.auth.AuthorityManager` to determine what functional actions can be taken on the object, based on the users privilege and the privilege scope.

4.3.3 AuthorityManager

The AuthorityManager is responsible for determining what actions can be taken on ownable objects within the WCT. The AuthorityManager is also responsible for determining if a User has the correct privileges to view or act upon screens and/or objects within WCT.
The AuthorityManager makes its decisions based on the privileges and scopes, as defined by the User role associations. Refer to Use Case Realisation 11 for more detail on Users, Roles, Privileges and scopes.

4.4 Architecturally Significant Design Packages

The following are the architecturally significant Use Cases of the Web Curator Tool. They are deemed architecturally significant due to either their complexity, unique requirements or the way they impact the design of the system.

4.4.1 UC4 – Quality Review

Due to the evolution of quality review strategies for large harvests, modularity and extensibility are critical requirements for the Quality Review component. To support extensibility, it is proposes the development of an Application Programming Interface (API) that quality review tools can use to access and manipulate the harvest results in a controlled manner. This allows new quality review tools to be created over time and plugged into the Web Curator Tool with minimum effort. The API will be built to encapsulate all of the ARC file and Harvest Result processing to ensure that the quality review tools only need to deal with the quality review itself, and do not need to manage the harvest results.

In addition to the development of new tools, the availability of a standard API also allows the community to integrate existing tools by building custom adapters. These adapters will be responsible for translating between the Quality Review API of the Web Curator Tool and the interface for the tool.

A limitation of the Quality Review Tool API, is that the API is only available local to the WCT core. There will be no way interface with the API remotely, therefore resulting in Quality review tools only being made available via the browser. Fat client Quality Review tools will not be an option with the WCT. However the API could be enhanced in future releases to allow this.

4.4.2 UC5 – Submit to Archive

The archive submission feature must cater for the fact that different libraries may choose to use different archival systems.

As such the submission system will make use of the adapter pattern. The adapter is a simple design pattern wherein an adapter class translates an external system's interface into the interface expected by the client system. In this case, the WCT will define a very simple interface to allow this. For each Digital Archive System with which the WCT must interact, an adapter class will be written that implements this interface.

Two main Archive adapters will be created, the `FileSystemArchive` that simply takes a `TargetInstance` and the ARC file and creates a Submission Information Package (SIP) and writes it to a local disk. The other will carry out a similar function to the `FileSystemArchive`

but will also export the SIP into the external OMS system.

The functionality to take a `TargetInstance` and an ARC file and turn it into a SIP will be extracted into a utility class to ensure other adapters can reuse it in the future.

4.4.3 UC8 – Monitor & Manage Web Harvester System

Registration and Management

Each harvest process will be required to register itself against the Web Curator server on start-up, identifying itself by host name, process name, and listen port number. These details will provide enough information for both the WCT and users to independently identify the harvester processes. The registration process will ensure that every harvester process has a unique name.

Once the harvester is registered, it will request the WCT to send the appropriate time of day bandwidth restrictions.

Throughput Monitoring

Some status tools only generate the statistics upon request. However, it is usually more useful to see the changes in the statistics over time. Having a history available is also important for tracking down and resolving performance or other issues. As such the Harvester Agents will regularly submit throughput information back to the WCT server.

System Alerts

The Notifications System (UC13) has been designed to support notifications directly to individual users or to roles. The Harvesters will use the notification system to register notices should any of the specified events occur.

Several monitoring requirements, including monitoring of available disk space, and CPU/memory utilisation, cannot be supported directly in Java due to its platform independence. To support these machine-wide monitoring requirements, a machine based monitoring agent will be created. Java interfaces will be developed to allow interchangeable implementations of monitoring technique, allowing different monitoring solutions to be plugged in via a configuration file. For Unix flavoured machines, external system commands such as df (for determining available disk space), and sar (for CPU and memory statistics) can be invoked through the Java Runtime utility. The output of these commands can be read and parsed by the monitor, which can raise alerts if the results become unacceptable. A monitoring agent will be built for the Unix flavoured environments based on the df and sar utilities. No other OS machine based agent will be implemented at this stage.

4.4.4 UC9 – Logon

The proposed architecture contains an Authentication and Authorisation module as a common component. This component will be responsible for authenticating users.

The Authentication and Authorisation module will rely on the Acegi Security System framework. This framework plugs into the Spring Application Framework to provide both page-based and, if necessary, method-based authentication and authorisation.

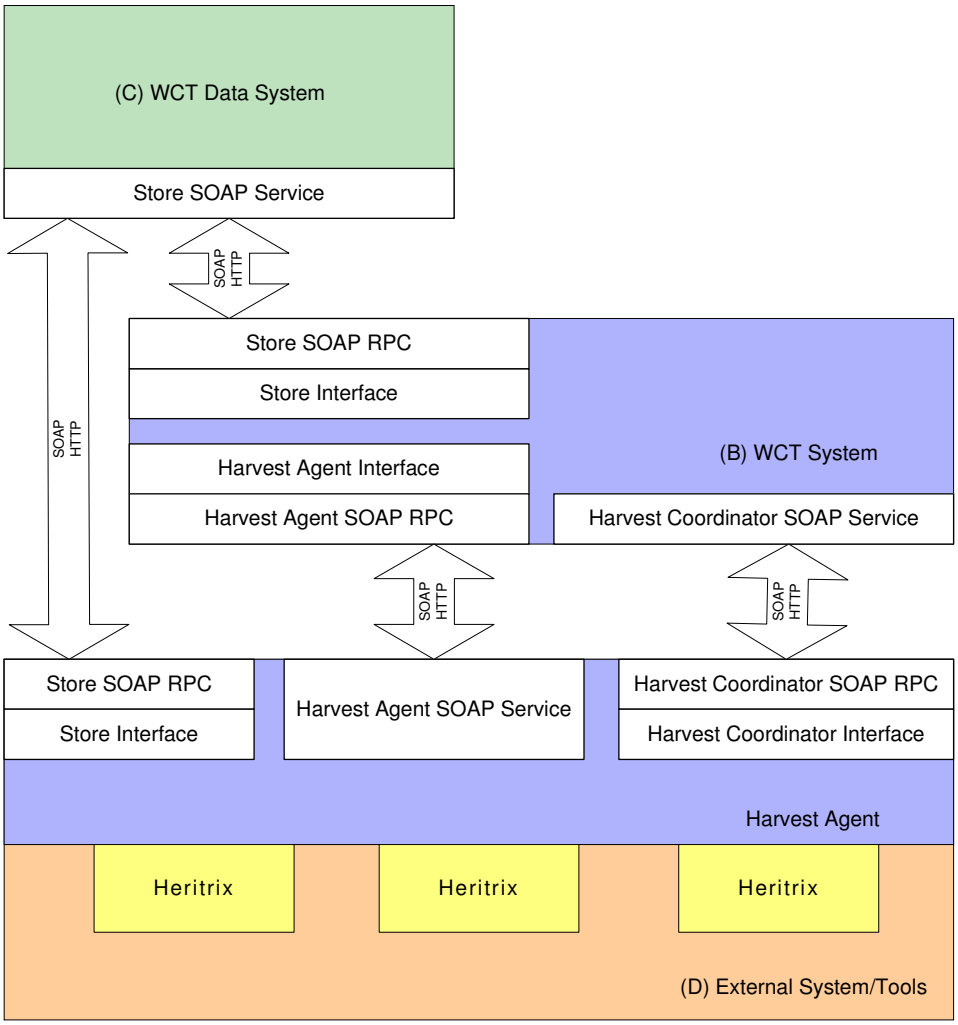
Where passwords are specified in the WCT database, they will be hashed using a one-way hash algorithm (such as MD5 or SHA-1) to protect the password. This ensures that even direct access to the database will not allow passwords to be compromised, as the one-way hash cannot be reversed to reveal the password.

The Acegi framework uses declarative security to define authorisation levels required to access certain areas of functionality. This is useful as it allows the security requirements to be changed without code changes. The privileges assigned to a user via their role will dictate what features and functions are exposed to the end user.

The Reporting Use Case requires the ability to report on the amount of time a user has been logged into the system. This requires that all login/logoff times be recorded. This will be handled by defining a session listener to register all logins and logouts in the database. Because of the stateless nature of the HTTP protocol, users will sometimes close a browser without specifically logging out of an application. To counter this, web applications define a session timeout period that states the period of non-activity before a user is forcefully logged out of the application. The Session Listener interface will allow the application to capture both forced and manual logout times.

4.4.5 UC10 – Scheduler

The diagram below provides a high-level overview of the proposed design for managing the distributed nature of Harvesters as described UC8 and UC10, and the non-functional requirements described in section 6.8.



4.4.5.1 Isolated Communication Strategy

The Harvester Manager and Agent components will isolate all of the distributed communication strategy. This simplifies modifying the communication strategy at a later date should that be desirable. Furthermore, it ensures that the Harvester implementations need to know nothing about how to communicate with the rest of the system.
SOAP over HTTP will be utilised to communicate between the distributed components. Using SOAP over HTTP offers the advantage of being able to stream the harvest data into the Digital Asset Store without having to load the entire archive into memory. Other communication strategies such as RMI do not provide the low level access to allow streaming, while SOAP provides a good, open standard for communication (as opposed to custom socket-based communication).

4.4.5.2 Manageability

Each Harvester will register itself against the WCT on start-up. This simplifies deployment effort and remote management. The registration is also necessary to allow the WCT to actively monitor the harvester throughput as described in UC8.

4.4.5.3 Distributed Harvest Indexing

To support the extensible Quality Review framework discussed in UC4, it is necessary to create an index that specifies how to extract individual resources from the harvest result. Because the creation of such an index is potentially time-consuming and CPU intensive, it has been decided to distribute this task out to the harvester agents.

4.4.5.4 Store File Server

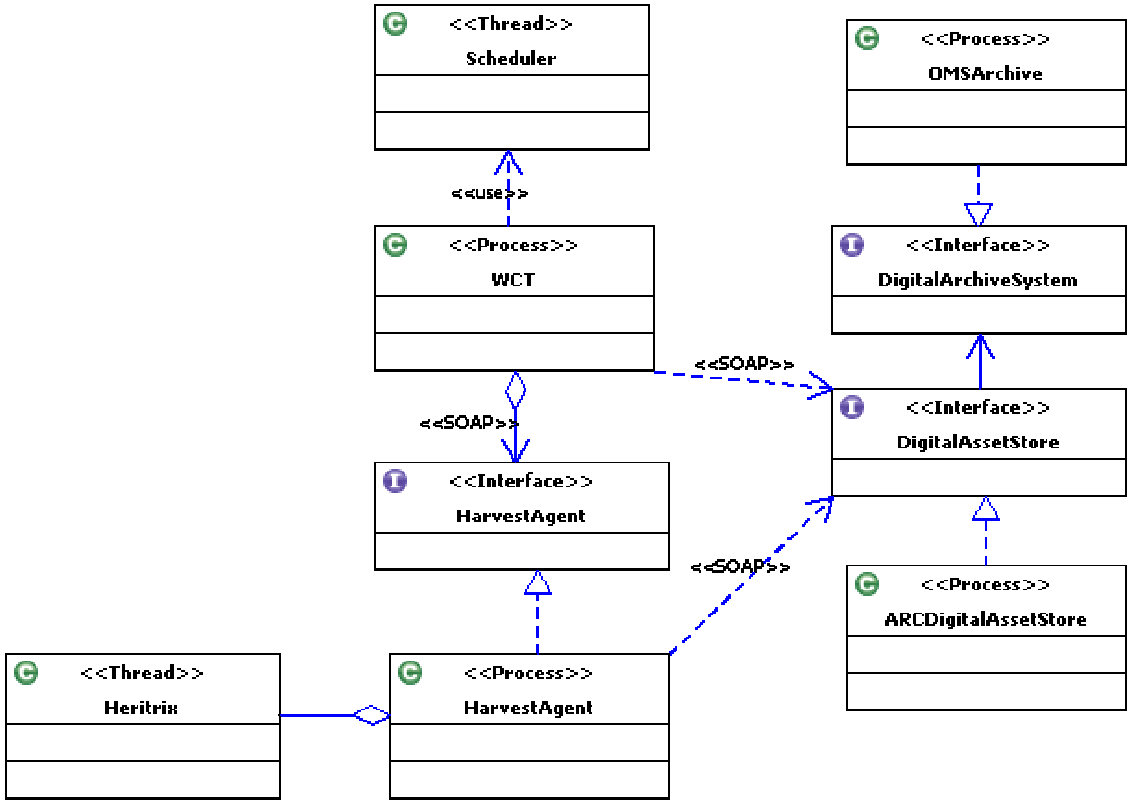
The Store File Server component is a dedicated service for storing and querying digital archive files. It is highly independent of the rest of the Web Curator Tool and is not concerned with the fact that a single harvest may be made up of multiple files. The Store File Server is specifically responsible for the following things:

- Storage and retrieval of full ARC files
- Random access of the ARC files, given a start point and end point.

The Store File Server may be deployed either locally or remotely to the Web Curator Tool. It is desirable to store the ARC File Server remotely to separate the data and business segments of the deployment. Furthermore, the Store File Server will have very large storage requirements. The complete independence of the Store File Server from the WCT allows for enhancements or changes in the storage mechanism to be made without large impact.

5. Process View

The process diagram below shows the main processes and threads of execution in the Web Curator Tool. If this model is used in combination with the deployment view, a picture of how the components interact and are deployed should be gained.



5.1 WCT Process

The central process is the WCT, an application process running on Tomcat. This runs the core processing of the web curator tool and is responsible for the web UI.

5.2 Scheduler Thread

This component handles all the scheduled events in the system, it is a Thread pool responsible for starting scheduled events based on the system clock.

5.3 Harvest Agent Process

The Harvest Agent is a distributable component that can run on one or more servers, to manage the harvesting of Target Instances. Spawning a Heritrix instance starts the Harvest process.

5.4 Heritrix Thread

The Heritrix thread is a single call to Heritrix to carry out a specified harvest of a Target Instance. Each request to Harvest a `TargetInstance` will result in a new Heritrix job thread.

5.5 ARC Digital Asset Store Process

This component is responsible for storing and retrieving ARC files created by the Harvest Agent process. This component implements the `DigitalAssetStore` Interface. In combination with the `DigitalArchive` Interface the ARC Digital Asset Store will also be able to create SIP files for export to an external Digital Archive System.

5.6 OMS Archive Process

This component is an implementation of the `DigitalArchive` Interface for exporting a SIP to an external Digital Archive System, in this case the Object Management System (OMS).

6. Deployment View

6.1 Operating Systems

The Web Curator Tool and Harvester System will be developed in Java. The platform independence of the Java language means that the Web Curator is deployable on any operating system for which a Java Virtual Machine is available. However, despite the platform independence of Java, it is critical that the Web Curator Tool is tested upon multiple platforms. The WCT will be deploy and tested on the following platforms to ensure Solaris and Unix-based compatibility:

- SPARC Solaris 9, 10
- Red Hat Linux Enterprise Linux 3

The Windows platform should also work, but no full end-to-end testing will be completed on the Windows platform. A large proportion of the development will be undertaken on the Windows 2000 platform.

6.2 Database Servers

The Web Curator Tool will rely on Hibernate to provide the persistence layer. Hibernate is an object relational mapping tool that supports multiple database dialects. As such, the Web Curator Tool will be compatible with any database supported by Hibernate. The Hibernate Team officially support and QA against the following databases:

Database	Version
Oracle	8i, 9i, 10g
DB2	7.1, 7.2, 8.1
Microsoft SQL Server	2000
Sybase	12.5 (JConnect 5.5)
MySQL	3.23, 4.0, 4.1, 5.0
PostgreSQL	7.1.2, 7.2, 7.3, 7.4, 8.0, 8.1
TimesTen	5.1
HypersonicSQL	1.61, 1.7.0, 1.7.2, 1.7.3, 1.8
SAP DB	7.3

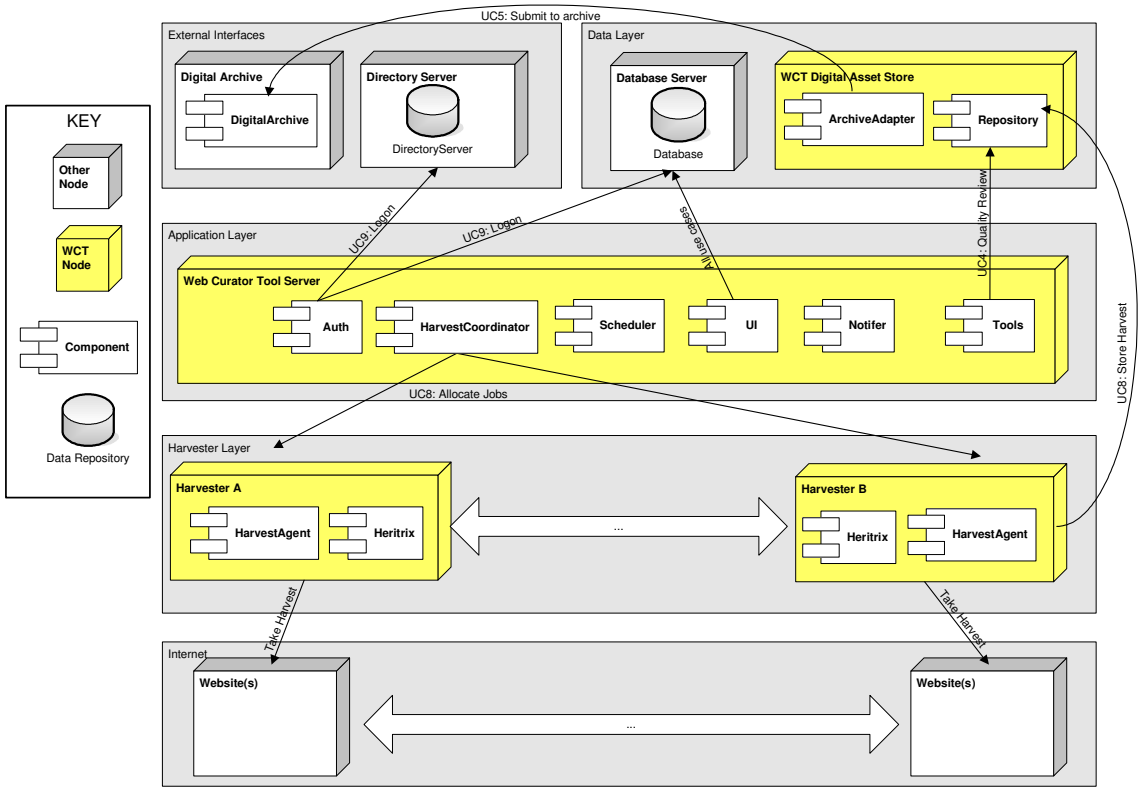
(from http://www.hibernate.org, 19/01/2006)

The Hibernate tool is known to support other databases that are not in the official list.
For development and testing purposes, the Web Curator Tool will be tested against the following databases:

- Oracle 10g
- MySQL 5.0

6.3 Logical Deployment

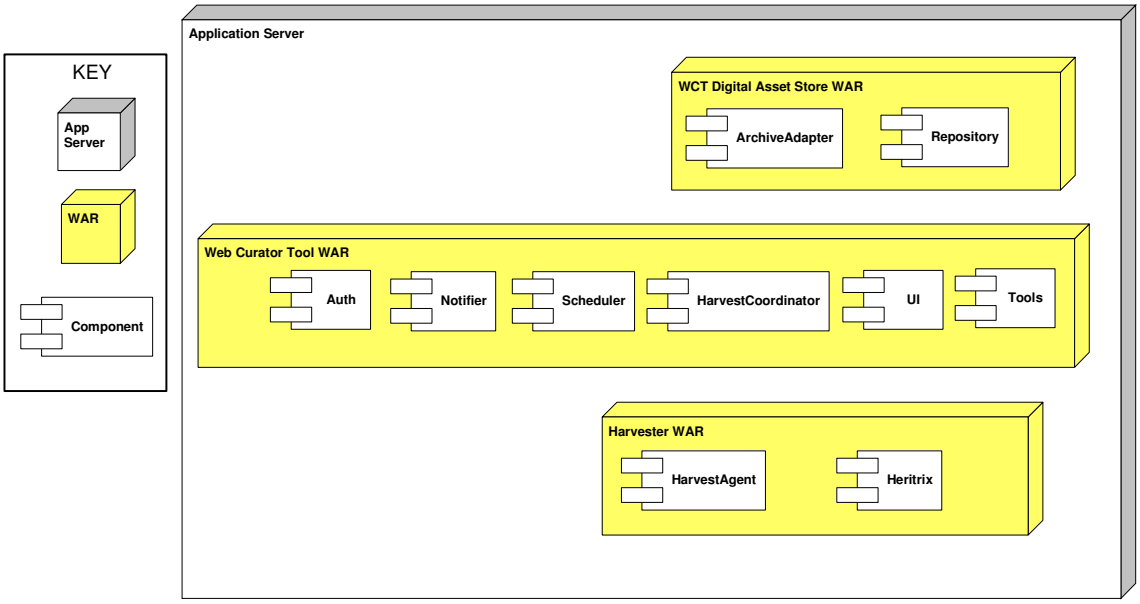
The diagram below shows the logical mapping of components on each of the nodes for the Web Curator Tool. It also identifies the interfaces into external systems like a digital archive.



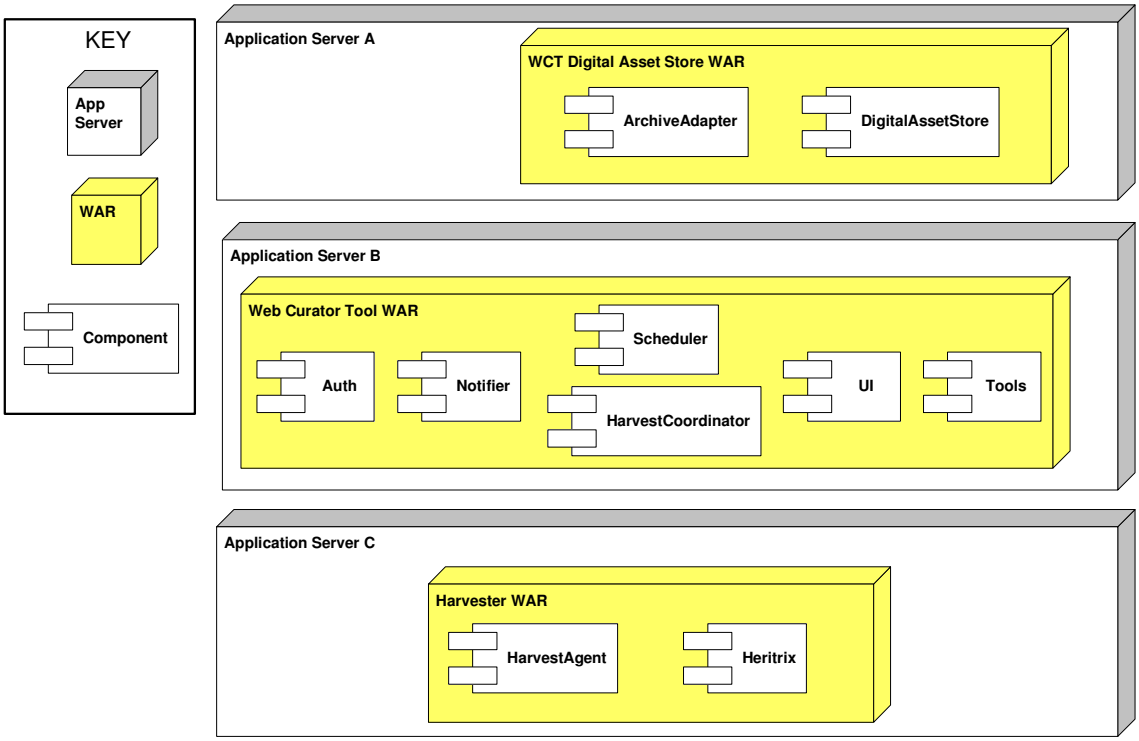
This logical grouping will result in the creation of three deployment packages that make up the Web Curator Tool. The three main components are:

- Web Curator Tool Core
- Web Curator Tool Digital Asset Store
- Harvester Agent

Each of these components will be delivered as a standalone WAR file. This allows the components to be deployed to either a single application server instance as separate web contexts as shown below:



The alternative is deployment to separate application server instances on different servers as shown below:



7. Data View

Hibernate is to be used for all data persistence, so it is not envisaged that the persistence layer be described in much depth. For more detail around key data structures refer to each of the Use Case Realisations. In particular UC1 Manage sites and permissions is a key part of the data model.

8. Size and Performance

8.1 Performance Requirements

The performance requirements of the Web Curator Tool are as follows:

- The WCT must process and respond to login requests in less than 5 seconds 90% of the time when subjected to peak user volumes.
- The WCT must process and respond to report requests in less than 30 seconds 95% of the time when subjected to peak user volumes.
- The WCT must process and respond to page requests in less than 5 seconds 95% of the time when subjected to peak user volumes.

8.2 ARC File Transfer

One of the critical elements of the Web Curator Tool is the collection and submission of large ARC files. These files may consist of one or more websites, so may be very large. SOAP with attachments has been determined an appropriate mechanism for transferring large files as it supports streaming while retaining the flexibility of an application programming interface. The Apache Axis support library has been used due to its support for streaming attachments.

8.3 Bandwidth Conservation

One of the major requirements of the Web Curator Tool is to limit the bandwidth consumed by harvesters. The ability to manage bandwidth is limited by the capabilities of the Heretrix Web Crawler. At the time of writing, the Heritrix crawler only provides support for restricting *average* bandwidth *over time* requirements 6.8.3 – 6.8.6. This strategy does not prevent heavy peaks and troughs in bandwidth utilisation. The Web Curator Tool does not attempt to provide any additional level of bandwidth restriction on top of what Heritrix provides.

9. Quality

9.1 Resiliency

It is anticipated that the Web Curator Tool will perform most of its harvests overnight during off-peak bandwidth periods. The operation of the system will, therefore, be largely unsupervised, leading to a need for the application to be highly resilient.

To meet this requirement, the harvester processes must attempt to recover from exceptions, or halt independent components without halting the full system. Specific resiliency requirements include:

- If the ARC File Server runs out of disk space, any Harvester attempting to store their ARC files must be able to retry saving the harvest at a later point. This is critical because a harvest may have taken many hours to collect, and should not be thrown away due to lack of disk space on the destination server.
- An error from a single harvest thread must not interfere with other harvests.

9.2 Regression Testing

It is always critical to ensure that new development does not break other components of the software. To ensure this, the Web Curator build scripts optionally execute a set of JUnit test cases.

9.3 Load Testing

Load testing will also be conducted on the Web Curator Tool's user interface to ensure that it meets the performance requirements in 6.10.1 – 6.10.3.