

# 4D4Life



## Distributed Dynamic Diversity Databases for Life

Deliverable 7.3

# Report on the implementation and testing of operative prototypes of the new architecture

Work package 7

Jonathan Giddy, Francisco Quevedo,  
Richard White, Andrew Jones, Alex Hardisty

7 November 2011

Capacities Programme of Framework 7: EC e-Infrastructure Programme - Scientific Data Infrastructures INFRA-2008-1.2.2

|                       |   |
|-----------------------|---|
| Grant Agreement No:   | 238988  |
| Project Co-ordinator: | Professor Frank Bisby                                     |
| Project Homepage:     | <a href="http://www.4D4Life.eu">http://www.4D4Life.eu</a> |
| Duration of Project:  | 36 months   |
| Start Date:           | 01 May 2009   |
| End Date:             | 30 April 2012   |

# Report on the implementation and testing of operative prototypes of the new architecture

This is 4D4Life Deliverable 7.3, version 1.0, 7 November 2011. The latest version of this document will be found on the 4D4Life project wiki at <http://wiki.4d4life.eu/wiki>

## Contents

|     |   |    |
|-----|---|----|
| 1   | Introduction.....   | 2  |
| 1.1 | Work package 7 objectives .....                                 | 3  |
| 1.2 | A service-based architecture to meet WP 7 objectives .....      | 3  |
| 1.3 | Terminology and abbreviations.....                              | 5  |
| 2   | Development, deployment and testing of the e-2 prototypes ..... | 5  |
| 2.1 | e-2 components.....   | 6  |
| 2.2 | Unified Assembly Workbench.....                                 | 7  |
| 3   | Issues arising from testing .....                               | 11 |
| 3.1 | Multi-tree and indexed-tree schemas .....                       | 11 |
| 3.2 | Long-running tasks.....   | 12 |
| 3.3 | Rule engine.....  | 12 |
| 4   | Continuing development.....                                     | 13 |
| 4.1 | Further testing.....  | 13 |
| 4.2 | Working with the revised UAW .....                              | 14 |
| 4.3 | Additional components and subsystems .....                      | 15 |
| 5   | Conclusions and recommended practices .....                     | 15 |
| 5.1 | Service-oriented Architecture.....                              | 15 |
| 5.2 | Data integrity, consistency and service abstraction.....        | 17 |
| 6   | Appendices.....   | 19 |
| 6.1 | Appendix I: Components delivered.....                           | 19 |
| 6.2 | Appendix II: The UAW showing e-2 components in use.....         | 21 |

## 1 Introduction

This is the last Deliverable and Report of Work Package 7 (WP 7), which forms part of the 4D4Life project's Theme 4 to introduce a new “e-2” infrastructure for the future support of the Catalogue of Life. Together with work in WP 6 and the specification of future needs for the Multi-Hub Network in WP 4, this will mean that for the first time the Species 2000 Catalogue of Life programme will have an infrastructure which will enable it to give proper attention both to the maintenance and functionality of its current production software and services, and to the design and prototyping for the next generation of production systems.

The role of WP 7 is the scoping, designing, and preliminary testing of the new e-2 distributed system, in coordination with WP 6, which will take over the process of preparing a second, production version of the system to be rolled out across the programme.

## 1.1 Work package 7 objectives

The objectives for Work Package 7 are:

1. To replace the existing distributed model with a purpose-built service-based Architecture.
2. To build the new Architecture around the ‘open ecosystem’ model in which all distributed components are implemented as open services, so that user or partner communities can integrate both low- and high-level services into their own e-Science communities.
3. To build the new Architecture to reflect the recent enlargement and raised objectives of the Catalogue of Life community. This will include replication from one distributed hub to multiple distributed hubs, for the need for the enlarged concept to be managed by an overall management system, and for the management to become a 24/7 operation involving several sites around the world.

These objectives were specified so as to allow flexibility to research and explore how best to design the architecture, introducing novelty as appropriate. It was presupposed that this architecture would need to be service-based in order to achieve decoupling between elements of the system and hence foster maintainability and extensibility.

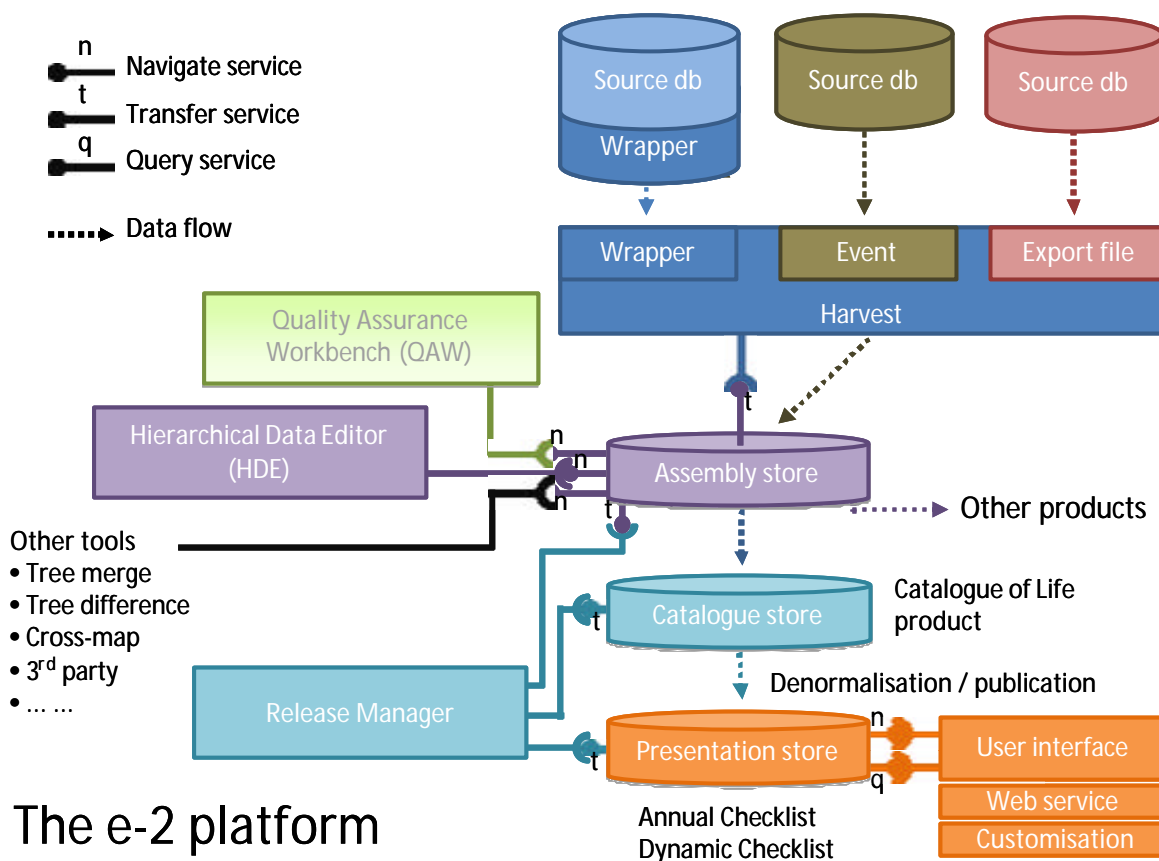
## 1.2 A service-based architecture to meet WP 7 objectives

These objectives are being met by the design and prototyping of a new service-based architecture, known as “e-2”. In this section we provide an overview of this architecture.

The existing software tool chain for the production and distribution of the Catalogue of Life consists of several tools (mainly PHP based), each of which provides web-based control over the contents of a database. Some of these tools have been further developed in Work Package 6 as part of the 4D4Life project. The tools are mostly configured to use a specific database, and are not designed for use without the corresponding web page. There is very little separation between the tool and the database it operates on and between the tool and its user interface. As well as making it difficult to introduce new database schemas, this makes it difficult to coordinate the tools in a more integrated, higher-level fashion. Additionally, adapting the tools to work with new kinds of devices (e.g., mobile devices) is difficult.

The introduction of a service-based architecture is designed to replace the individual tools with a set of components which initially provide an equivalent range of functionality, and which are loosely coupled. It is not assumed that there should necessarily be a one-to-one correspondence between the tools and the components. This loose coupling provides

- a separation of concerns, allowing individual components to provide key capabilities without requiring knowledge of the complete capability set,
- the ability to replace individual components without requiring changes to the entire infrastructure,
- improved maintainability, since each component can be modified and tested in isolation from the other components, and
- flexible orchestration of components, including the ability to combine components in new ways without the user needing to drive the process through multiple web interfaces.



## The e-2 platform

Figure 1: Service-based e-2 version of the CoL assembly process

The breakdown of tools into components also allows the components to be integrated easily into third-party systems. This supports the use of the software at geographically distributed hubs, both for replicating the core editorial and publishing system, and for providing localised variations of the original capabilities. Current design work in WP 4 on support for regional hubs and work in the near future in the “i4Life” project to ensure compatibility of i4Life cross-mapping tools with the e-2 architecture will demonstrate the effectiveness of our approach. A means for interoperability with the i4Life cross-mapping tools at the service level is indicated in Figure 1.

The service-based architecture also ensures the integrity of the data representation by restricting the operations that can be performed to those that are both meaningful and permitted.

During the project, these capabilities were proved through

- access to multiple database schemas via a single user interface,
- use of multiple front-ends to drive similar tasks, including a web interface and non-web clients including the Drools-based Rule Engine and import/export tools that do not have a web interface, and
- modifications to the data representations through an interface that ensures that the underlying data (the taxonomic tree) cannot get into a technically or structurally invalid state, such as including entities that are not properly linked to the tree<sup>1</sup>.

<sup>1</sup> For example, using the service interfaces, the modifications that can be done to the tree are only those that a biologist would consider to be meaningful. He or she might attach a genus to a different family, or rename a species, or add a new common name. These are domain modifications. The biologist would not add a link to the tree that places a higher taxon as a child of a taxon of lower rank, for example, even though this may be possible

### 1.3 Terminology and abbreviations

CoL (Catalogue of Life): both a partnership between Species 2000 and ITIS, and the complete global taxonomic checklist that is its prime product

e-2: the service-oriented architecture for a second generation of Catalogue of Life collaboration and production software

GSD (Global Species Database): a provider of up-to-date checklist data for the higher taxon (taxonomic group) in which its expert editors specialise

HDE (Hierarchical Data Editor): a component for manipulating taxonomic hierarchies, especially the rearrangement of branches

ITIS (Integrated Taxonomic Information System): a partnership of U.S., Canadian, and Mexican agencies, other organisations and taxonomic specialists, which catalogues species of particular importance to North America and dependent territories

QAW (Quality Assurance Workbench): a template-driven environment for ensuring the quality of checklists received from GSDs and converting them to meet the format and taxonomic integrity requirements of the Catalogue of Life

SCA (Service Component Architecture): one of several techniques for obtaining flexibility and consistency in creating SOA components, implemented in our work using Apache Tuscany

SOA (Service-Oriented Architecture): as outlined above and described in WP 7 Deliverables D7.1B and D7.2, also referred to as “service-based architecture”

UAW (Unified Assembly Workbench): a chain of software tools consisting of the QAW, HDE, and other production process components

## 2 Development, deployment and testing of the e-2 prototypes

As reported in the first Periodic (annual) Report (2009-2010), WP 7 staff worked with the project co-ordinator and WP 6 to identify which subsystems of the e-2 design and Enhanced Prototype lie on the critical path for certain tasks in year 3, and which subsystems lie off this path. The tasks on this critical path include those planned for WP 6 (Task 6.20, “Design and planning for production version of e-2 prototype provided by WP 7” and Task 6.21, “Implement production version (+doc), including prioritised enhancements and responses to testing”), and for WP 3 (Task 3.5, bringing selected Global Species Databases (“GSDs”) into the e-2 environment). We have now delivered versions of the components of those subsystems on the critical path.

The e-2 architecture does not itself impose either a specific user interface or a particular data workflow for checklist editors and other users. It provides a framework for the implementation of software that does implement such things. It is for this reason that the prototyping approach that was adopted for implementation of individual components is feasible.

This approach has allowed the flexibility to revise individual components significantly as a result of user feedback, and in particular to permit the following significant change in the order of

---

in the underlying representation (the database schema). By only allowing modifications that make sense in the domain, the service interface prevents the database from getting into an inconsistent structural state.

development of components. WP 6 had to meet the need (described below) in Task 6.10 (“Install QA Workbench for beta test and then production unification”) to put the Hierarchical Data Editor (“HDE”) component from the WP 7 Assembly Subsystem into test and production use earlier than originally foreseen, as part of the Unified Assembly Workbench (“UAW”). This was to permit WP 5 to complete Task 5.4 (“Work with the Systems Design Team and WP 6 to enhance and operate the annual production service for the Annual and Dynamic Checklists, including unification of the admission gate for the two products, and the roll-out of the new e-2 system”). Following discussions with WP 5 and WP 6, the Systems Design Team decided to ask WP 7 to bring forward into Task 7.6 certain aspects previously planned for Task 7.7, in preference to some Task 7.6 elements, in order to offer earlier than planned support to WP 6 for this purpose. We have therefore carried out more work than originally planned by this date on the Assembly Subsystem and correspondingly less on the Control and Harvesting Subsystems of the e-2 architecture; WP 5 and WP 6 have provided components for the UAW which can potentially be adapted for use with the e-2 architecture.

This change has proved timely in the consideration of options for the future deployment of the e-2 architecture. It will facilitate a more gradual migration to e-2, which could be advantageous both by introducing components from the e-2 design to facilitate a more robust, productive and flexible assembly process for the Catalogue, and by lessening the risks arising from an abrupt switch-over from one system to another.

To summarise, during 2011 Work Package 7 adjusted its task prioritisation in order to provide more support for the construction and testing of the UAW. During this process, at the end of April 2011 WP 7 achieved Milestone 7.1, “Prototype handed to Software Services (WP 6) for development of production version”, as a result of actions in Task 7.6 (“enhanced prototype for e-2”) and previous tasks. We have placed a number of software components in the Subversion source code repository hosted by ETI, access to which is described in Appendix I, and others are under development, as described below.

## **2.1 e-2 components**

The development process for software components for the e-2 architecture was described in Deliverable D7.2. In the present deliverable, we describe in more detail the components which have been implemented for the e-2 architecture.

We implemented components following a Service Component Architecture (SCA), specifically using the technology implemented by the Apache Tuscany project. It is one of several SOA (Service-Oriented Architecture) techniques for obtaining flexibility and consistency. SCA does not commit us to a particular method for communicating between components, such as SOAP, REST or RPC, but allows components to be instantiated for use in workflows which adopt any suitable communication protocols.

One of the biggest benefits of adopting SCA is that it provides some degree of future-proofing, as new standards for communicating between components emerge, and it minimises the technology constraints imposed on future providers of components for the e-2 architecture.

### **2.1.1 Data Store services**

We first developed a prototype that allowed access to multiple databases storing representations of taxonomic hierarchies (“trees”) using a set of uniform web service interfaces:

1. Navigate – retrieve data piecemeal
2. Edit – attach data from one tree to another

### 3. Transfer – copy large sections of data from one database to another

Each of these services provides access to a database containing one or more taxonomic trees. Multiple schemas are supported. In particular, the Edit service primarily works on schemas developed within WP 7 (the “Multi-tree Schema”, described below, and its replacement “Indexed-tree Schema”) which allow a single database to store multiple trees. In order to interoperate with other tools in the 4D4Life system, which primarily use the Base Schema, the Transfer service was used to export a tree from one schema and import it into another schema. The Navigate service supports all schemas, and allows taxonomic trees to be browsed, progressively retrieving child taxa from selected higher taxa.

#### 2.1.2 The Hierarchical Data Editor (HDE)

Another component, the HDE, is the hierarchy maintenance component of the Assembly Subsystem of the e-2 architecture. It is the primary client developed to interact with the Data Store services. Managing the multiple taxonomic checklists which make up the Catalogue of Life is critical to the integrity of the data, but the tools used to perform this task in the past have been primitive. The new component has been used both to test and demonstrate the functionality of the e-2 architecture in the prototypes and has been deployed within the existing architecture as part of the UAW (described below) to fill the present gap in this area. It allows browsing of multiple trees, and interactive drag-and-drop editing of trees. In particular, its primary role is to allow an editor to arrange for elements from one tree to be attached to another tree. The use of an early version of the HDE component was described and illustrated with screen images in Deliverable 7.2 (“Proof of Concept demonstrator”). The current version is shown in use in the UAW in Figure 5 in Appendix II.

#### 2.1.3 Data transfer components

To link the HDE to the data coming from the QAW developed in WP6, an additional client (QAW Export) performs a transfer of the data from the QAW Base Schema database to the HDE Multi-tree or Indexed-tree Schema database. To do this, the client contacts the Transfer service for one database, requesting an Export operation, followed by contacting a second Transfer service for another database, requesting an Import operation. A similar process occurs after the editor has finished using the HDE, and the data is transferred to Base Schema again, ready for denormalisation.

Denormalisation is required for the standard CoL web interface to be used. Before denormalisation, the data in the Base Schema database can be viewed using a tree viewer client based on the HDE code, as shown under the Release tab on the portal interface in Figure 6 in Appendix II.

## 2.2 Unified Assembly Workbench

### 2.2.1 Design

Originally, the Catalogue of Life checklist editors used separate manually controlled assembly procedures to assemble the Dynamic Checklist and the Annual Checklists (Figure 2). The editors wanted to avoid the time-consuming and error-prone use of the variety of monolithic legacy tools with which they were familiar, in which each tool has a user interface tightly integrated with the underlying business logic and persistence layer.

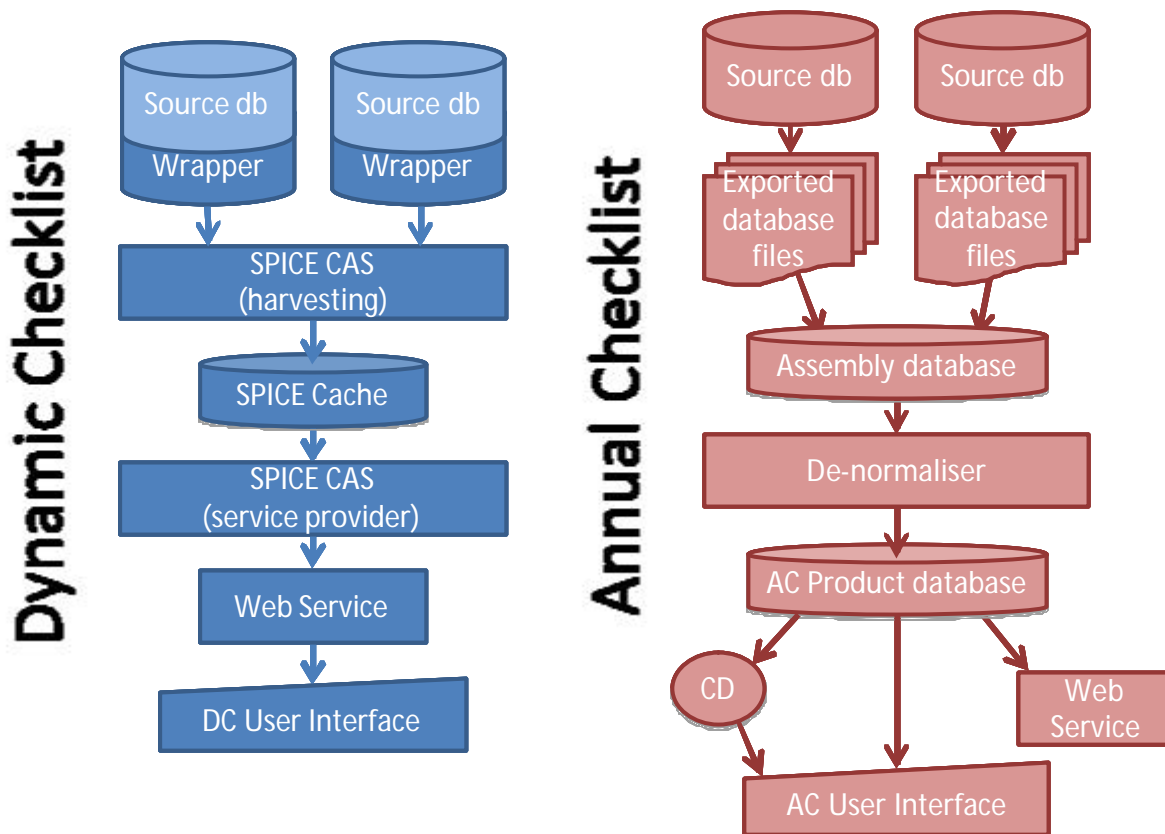


Figure 2: Separate Dynamic and Annual Checklist production

The initial plan by ETI in WP 6 was to integrate their existing tools for harvesting and quality-testing checklist data sets (the Quality Assurance Workbench, QAW, shown in Figure 4 in Appendix II) and for generating and “optimising” (de-normalising) the final databases distributed with user interfaces on DVD and on the Web. The Unified Assembly Workbench (UAW, Figure 3) was conceived at first by WP 5 as a “single tool” for use by the editors who assemble the Catalogue of Life and its Annual Checklists. The UAW was not originally planned as a collection of service-oriented components in the style of the e-2 architecture (Figure 1), but as a means of automating processes already carried out during the assembly process.<sup>2</sup>

<sup>2</sup> The latter was seen as a completely different development which would become available at the end of the project as a replacement for the UAW.

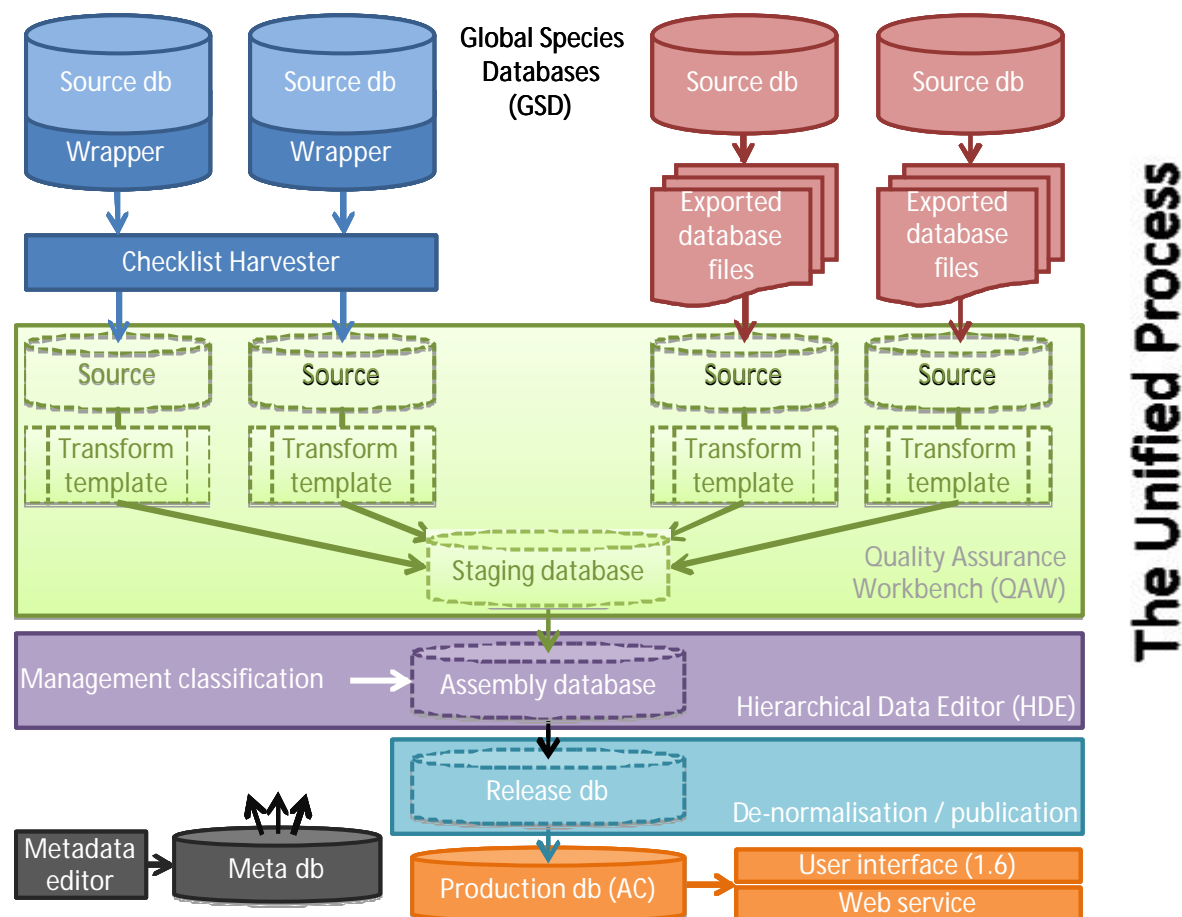


Figure 3: The Unified Assembly Workbench

Because the HDE provided useful capabilities for tree editing that were not available among the tools used for the existing workflow, and because it complemented WP 6 work on the QAW, WP7 was asked to integrate the HDE into the development of the UAW. In this way, our work for Task 7.5 (“To complete and install a test implementation for e-2 with three GSDs as test-beds”) and Task 7.6 (“To respond to testing with an enhanced prototype for e-2”) became closely aligned with the construction of the UAW.

The HDE is a separate component that works in conjunction with the QAW, and therefore contributes to achieving the requirement in WP 6 to streamline the CoL assembly process. It was originally developed for e-2 because it was needed for testing the checklist assembly process in e-2, and there was no existing component to perform manipulations of a taxonomic hierarchy in a reliable manner. However, the service-oriented design and construction details (such as the service component architecture and Java implementation) of the HDE precluded a simple merging of the code into the UAW. A framework with some elements of the e-2 architecture was required to permit the integration of the HDE component originally designed for e-2 into the UAW.

Since some of the other tools to be included in the UAW were monolithic web/database applications whose user interface, business logic and database layers could not easily be separated to isolate components which could be implemented as services, it was necessary to integrate the tools at the user interface level, as described below.

In addition to the QAW and HDE components described above, the integrated UAW user interface has:

- database interface “wrappers” for the harvesting of checklist data from the GSD providers,

- a means to initiate the transfer of harvested data sets from the QAW (including converting Base Schema to HDE Multi-tree or Indexed-tree Schema) and handing control to the HDE component, and
- a means to run a script for database de-normalisation on the Base Schema database after “release” from the HDE, for optimising read access to the database on the DVD and via the web GUI.

### 2.2.2 User interface

The components described above were deployed in a sequence of user interface implementations. The original HDE component was used in the e-2 Assembly sub-system with a tabbed interface to allow users to import checklists, work with the trees and “release” products. This tabbed interface was developed from the HDE into a “mini-portal” to provide the user-interface level of integration for the UAW as explained above. The tabbed interface is designed to separate out the various tasks from a user perspective, but also corresponds to the separation into different components. The interface allowed the editors to move between the web interfaces of the various components, some of which are WP 6 PHP monolithic web tools that accessed the databases directly, others being WP 7 clients that interacted with our Data Store web services. Screenshots of the QAW, HDE, etc., being used in our tabbed interface in the UAW are shown in Appendix II.

### 2.2.3 Deployment

WP 6 has now deployed the QAW at ETI (version 1.0, an earlier version 0.8 was demonstrated at the Third Annual Project Meeting in Prague in March 2011). Both the QAW and the HDE were deployed on servers at the same location to avoid or reduce the need for inter-machine file transfers and reduce transfer and latency times.

WP 7 delivered the HDE and UAW tabbed interface to ETI and deployed it on a server there, where it was used for initial familiarisation and testing by WP 5. The HDE is deployed in Tomcat 6.0. The UAW user interface provided by WP 7 contains several tabbed pages, including one for the QAW, one for the “Assembly” (HDE) and one for “Release”.

The Cardiff team assisted WP 6 (ETI) with deployment issues and operating difficulties revealed during testing. Some problems were recorded in the “Jira” issue tracking system used by WP 6 and WP 7 to coordinate corrections and enhancements to the software. The acceptance trials for the UAW carried out by WP 5 confirmed that the components supplied by WP 7 served their purpose. See a later section for discussion of the implications of recent changes in the UAW being used by WP 5.

The e-2 architecture has been delivered to WP6 for evaluation. Initial indications are that being able to view the database as a tree is invaluable for debugging, but currently requires the database to be denormalised. Using the HDE (with the DataStore services) as a Tree Viewer could allow checklist editors to see problems in the databases more quickly than at present.

### 2.2.4 The Liferay front end

In a further development, the same underlying component implementations used in the UAW with the Cardiff tabbed interface described above were re-deployed using the “Liferay” portal engine to implement an alternative unified user interface. This provides another way to map existing web interfaces to separate tabbed pages, but significantly it allows several editors to work at the same time to assemble the Catalogue, with login authentication and single sign-on.

### 3 Issues arising from testing

Although the initial testing demonstrated the validity of our approach, there were three areas in which it became evident that some further attention needed to be given to specific issues.

#### 3.1 Multi-tree and indexed-tree schemas

The Base Schema was developed by WP 6 as a “canonical form” for holding checklist data. However, during the actual “attachment” of GSDs (positioning them in the taxonomic hierarchy of the CoL Management Classification), the HDE simultaneously manipulates more than one taxonomic sector. Therefore WP 7 developed a “Multi-tree Schema” that allows the storage of multiple sectors within a single database. This was consistent with the concept of the Base Schema as a common reference point for more specialised schemas, but required conversions to take place when data is passed between the different schemas.

The Multi-tree Schema supported significant sharing of data between sectors, especially where a sector exists both independently and as a branch of the CoL. In addition, it recorded modifications of trees as separate versions, allowing the users to keep a historical “trail” of trees, again with many of the common data entries shared between versions. Some other changes were made to the tree structure to provide additional capabilities identified during the requirements phase, such as synonym and common name support for higher-level taxa and separation of the taxon and name hierarchies to handle the exceptional taxa where a species name does not match the taxonomic hierarchy (e.g. hybrids and support for sub-genera).

To support integration of the UAW components, transfer stages between the components were required to move the sector data from one schema to another. This introduced problems, including:

- As the transfers required the combination of a read operation from a Base Schema database, temporary storage in a Darwin Core text file, and a write operation into a Multi-tree Schema database, and then later in reverse, reading from the Multi-tree Schema and writing to the Base Schema, there were many opportunities for data to be lost or misrepresented due to incomplete or buggy code.
- The transfers could take a long time, especially when moving the full CoL (containing over 1 million species) from the HDE to the Base Schema Release database.

To overcome these problems, we have recently moved to using a system where the multi-tree index is in a separate database, and provides the historical trails of different versions of a tree, but a tree in the index database is actually a reference to a database containing a single tree, which may be represented in one of several schemas, typically the Base Schema. This allows the Base Schema databases from the QAW to be used directly by the HDE, and for the CoL to be written directly as Base Schema databases, immediately ready for further processing. The speed and consistency improvements due to reduced transfers and schema changes far outweigh the disadvantages of a slower attachment process (the attachment process still requires a single transfer of the GSD sector).

Significantly, the change from the Multi-tree Schema to an Indexed-tree Schema demonstrates the flexibility of a service-based approach, with the change of database schema requiring no changes to the client, except to support any additional capabilities introduced during the change. This was possible because although the underlying database schema changed, it was not necessary to completely redesign the objects passed between client and service accordingly.

## 3.2 Long-running tasks

In the original version of the HDE, the web portal acts as a Javascript client to the web services (with a simple Tuscany proxy server in between to convert between JSON-RPC and SOAP Web Service protocols). The client typically needs to orchestrate several related sub-tasks, some of which may be long running. For example, to transfer a tree from one database to another, the client first calls an export operation on the source web service, and when that has completed, calls an import operation on the target web service. Other long-running tasks include taxon matching (for allocating persistent resolvable identifiers for use in tracking taxon changes) and de-normalisation (for preparing a checklist database for use with user interfaces on DVD and on the Web).

Major problems occur with the use of the web portal as the web service client for these long-running tasks:

- As many tasks require multiple web service operations to be orchestrated, the web portal (and the Javascript on the web page) need to stay active for the duration of the task. As some of these operations are long-running, this is inconvenient for users who may intentionally or accidentally navigate away from a page, losing information about progress of the task, and in some circumstances, causing the task to fail.<sup>3</sup>
- Due to the behaviour described above, it is not possible for the web browser to be closed during tasks that may take hours, and it is not possible for the user to connect from another browser, say at their home to check progress of the task.
- For transfers, progress information is placed in a database, and can be read by the web client using a service and displayed to the user as progress bars. This was implemented during UAW testing. However, the solution is bespoke for the transfer processes and would require additional work to support other operations. For example, additional steps would be required to add progress bars for the denormalisation process.
- There is no support for cancellation of a running task.

To address this issue, we proceeded on two additional developments:

1. We inserted an intermediate service to orchestrate the multiple services and operations to perform complex tasks.
2. We developed a consistent software interface for starting, monitoring, and cancelling long-running tasks.

The orchestration service allows the web portal to be responsible for starting and monitoring tasks, without the requirement that it exists during the execution of a task. It also makes it possible for a client to connect to the orchestration service to check on the progress of a task started by another client.

## 3.3 Rule engine

One of the plans for the e-2 architecture was to incorporate a rule and process management engine, in order to automate some of the repetitive actions that are performed by the editors

---

<sup>3</sup> The task continues running until it reaches a point where the user interface is intended to control the next step. If the user navigates away during the last step, then the task succeeds (e.g during the import of a transfer process). However, if the user navigates away before the last step, the task fails (for example, in a transfer, the source tree is exported, but the user interface never imports it anywhere).

every time changes occur in the source GSD databases. The idea behind this is to give the editors more time to focus on the more exceptional (and interesting) taxonomic changes that occur in these databases over time.

We have demonstrated a version of this, using the Drools Business Logic Integration Platform (<http://www.jboss.org/drools>), a widely used freely available product that combines business rules and business processes in a single system. We demonstrated a rule triggering a process to harvest a GSD, interact with the QAW, and interact with the Transfer service. We were able to perform a simple operation using the QAW as a service by performing a single operation as a web browser.

However, due to the effort required to make a simple operation (adding a new taxonomic sector) available to non-browser software and the fragility of interacting with a component still under development, it would be more productive to have a properly defined service interface for Drools-to-QAW interaction. Hence we deferred further development of this phase until finalisation of the QAW development. Further Drools development has focussed on the release stage, performing the taxon matching and denormalisation tasks required to create the published CoL. Through this work, we have established recommended practices for the development of new services and for integration of legacy components with no service interface.

The Drools component sits between the user interface and the store services, in the same position as the orchestration service described in Section 3.2. Because of this, we are currently implementing the orchestration service using Drools. This allows us to use the business process management (BPM) component of Drools to specify complex tasks consisting of multiple sub-tasks, and to use the business rules component to manage the interactions of multiple users attempting to perform similar or conflicting tasks concurrently.

Using Drools allows us to move many of the domain constraints up out of the service layer, more clearly separating the structural constraints on the data from the working practices of the data editors. For example, the CoL editors currently replace missing taxonomic ranks with 'Not assigned' pseudo-taxa. However, this is neither a fundamental data schema requirement nor a necessary taxonomic practice (it is done for reasons of presentation in the end-user interface) and the web services and underlying data schemas do not enforce this constraint. At some time, a decision may be made to remove the pseudo-taxa and allow, for example, a genus to appear as a direct descendant of a class without any un-named taxa at intervening ranks. This change can be made in the Drools subsystem without requiring any change at the service layer.

## 4 Continuing development

### 4.1 Further testing

In the recent UAW testing, the editors in Reading tested the harvesting of some new GSDs, their processing in the QAW, their attachment in the HDE to the hierarchy as it appears in the AC 2011, and the export and denormalisation of the product so that it could be examined in the standard GUI as used on the Annual Checklist DVD.

Thorough testing of all aspects of this process, both in the UAW and the new Webmin version of it requires several phases.

The first phase of testing included the successive stages of the harvesting of some trial GSDs, passing them through the QAW, and using the HDE to attach them to an “empty” copy of the

hierarchy as it appeared in the AC 2011 (that is, having all the higher taxa in place, including attachment points, but no GSDs; such a copy was derived from the January 2011 CoL edition and demonstrated at the Third Annual Project Meeting in Prague). This allowed WP 5 to create a subset of the CoL, without the complication of interactions with any other GSDs, and permitted easy inspection of the results in the web-based end-user interface (version 1.6, see Figure 7 in Appendix II). The initial tests were with small amounts of data so that it was possible to inspect what was happening more easily and detect any failures in a methodical manner. This also avoided the delay which can occur before the resulting assembled checklist can be viewed, due to the need for de-normalisation.

In the second phase, when all the stages described above were working satisfactorily, a further series of tests were performed in which the interactions between the new and the existing GSDs were tested. In this case, the other GSDs which make up the CoL (those whose harvesting and QAW processing is not being tested) were already attached to the hierarchy (unfortunately it was not practicable at that time to provide them as separate checklists).

Current plans for testing the UAW and its replacement stop at this point. The first and second phases suggested above do not test the ability to create an entirely new Catalogue free from any residual errors carried over from previous versions. If a previous copy of the CoL product, such as an old Annual Checklist, is imported as a single entity into the UAW, this would be effectively the same as the second phase of testing above, failing to create an entirely fresh Catalogue and potentially carrying over data, editing and assembly errors from the old Catalogue. We believe that the interim UAW was not designed with this problem in mind, and hence full testing of this aspect may not be possible at present.

In a third phase of testing to resolve this issue, the links between the Catalogue product and its GSD sources should be as explicit as possible. One of the roles of the Species 2000 metadatabase is to maintain such links. It is important to demonstrate the traceability of GSDs through the workflow into an assembled Catalogue. The e-2 architecture is designed with this in mind, and hence full testing of this aspect can probably only be achieved later, using the new architecture. This requires:

- that there is a link between the metadatabase and the UAW-based harvesting, editing and assembly processes, and
- that all existing GSDs are available in their original form, as separate GSDs harvested via their wrappers or files transferred to the UAW.

## 4.2 Working with the revised UAW

One change that needs accommodating is the adoption of a Webmin-based Workbench version of the UAW by WP 5 for Catalogue of Life production in the short term. This includes a new Data Exchange Platform, including QAW features for streamlining the scrutiny and data transformations of data sets, where GSDs and other partners can upload and download relevant data sets, not only for Catalogue assembly but also data sets from Regional Hubs and i4Life partners.

The existing e-2 services can be updated to interact with the new Webmin-based version of the UAW being developed in WP 5, to give it better management capabilities and allow the import of data sets using the old “ACEF” format. Our current work on Drools will need to be demonstrated (or at least tested) with components of the above.

### 4.3 Additional components and subsystems

The Cardiff WP 7 team are continuing to work on some aspects of the HDE, while developing other aspects as described below, including architectural features for communication of data and for the development and deployment of new components, including those to handle the steps that occur after data clean-up and tree attachment, including the taxon matching process, since it had been agreed that support for issuing LSIDs<sup>4</sup> would not be included in the recent testing of the UAW by WP 5, but will be provided later. We expect to deploy one or more updated versions of the e-2 software during the remainder of the project as WP 6 prepares it for eventual replacement of the Webmin version of the UAW.

We have commenced Task 7.7 (“Continue to monitor the new architecture, and to assist the Systems Design Team”), in order to work with WP 6 to make the e-2 system available for use (Tasks 6.20 and 6.21) and to prepare this Deliverable 7.3. Note that as previously stated in the first Periodic Report, we are increasing our effort in Task 7.7 by expanding its scope to include the development of the “non-core” but still essential components and to ensure that any issues arising from the hand-over can be dealt with effectively, even if they reveal the need for design enhancements or additional features in e-2 subsystems and components.

We are developing components for event notification, based on the Rule Engine described above, and for quality control and management, using the Rule Engine and addressing issues raised in the Integrity Requirements section below.

## 5 Conclusions and recommended practices

We have addressed the following issues, which we identified and discussed in the specific design areas described above:

- completeness of data schema support and multiple schemas,
- the requirement to support multiple trees,
- performance bottlenecks,
- management of long-running tasks and task orchestration.

We discuss some further principles below.

### 5.1 Service-oriented Architecture

The advantages of SOA in principle are documented elsewhere<sup>5</sup>. From this project, we note the following confirmatory experiences:

- Integrating components that do not specify a service contract was difficult due to the need to reverse-engineer browser-like interactions (in the case of the QAW) or to modify code to replace hard-wired configuration (albeit only the path to a configuration file) to allow

---

4 Globally Unique Identifiers, which can be used in computer systems to refer specifically to the concept associated with each taxon in the Catalogue of Life.

5 See for example the list of references on SOA in Wikipedia at [http://en.wikipedia.org/wiki/Service-oriented\\_architecture](http://en.wikipedia.org/wiki/Service-oriented_architecture)

concurrent executions of multiple instances (in the case of the denormalisation code).

- The service approach naturally allowed us to integrate distributed components, for example databases at ETI and Cardiff, by running multiple Tree Store services integrated initially by the web client, and later by the orchestration service.
- During the project, we demonstrated the flexibility of a service-based approach by replacement of the web services without modifying the client, and vice versa.
  - We replaced a web service based on the Multi-tree Schema with one based on the Indexed-tree Schema (technically, this involved deploying the same service implementation with two different Data Access Objects (DAOs).
  - We replaced the service orchestration part of the web client with an orchestration service, which acts as an alternative client to the Tree Store services (and as a service to the web client).
- We demonstrated the advantages of the service interface by integrating services using the same service interface, but with different underlying schemas. The Tree Store service provided access to the Base Schema, Multi-tree Schema and Indexed-tree Schema, as well as to the native schemas for the Management Hierarchy and the ILDIS database.
- Two developers were involved in WP 7, and both developers worked on all components at various times. This was important as previous experience with fixed-term projects is that certain components can be difficult to maintain if a sole developer moves on at the end of a contract. Version conflicts were mainly avoided by coordinating which components were worked on by each developer during a specific time. This was done fairly informally as the developers were co-located. However, it could have benefited from more preliminary planning to ensure that each developer was not put in a situation of waiting for write authority to a component.
- Choice of an appropriate level of service granularity is an important part of SOA design. Since services may be invoked remotely, there is a compromise between calling small operations more frequently (magnifying the effects of network latency) and calling large operations less frequently (using more bandwidth and possibly making some small tasks unavailable). We found it important to support the most primitive operations that made sense in the domain, in order to maintain flexibility for the client to reorganise the order of tasks. For primitive tasks, a useful way to increase the range of available levels of granularity without adding additional methods was to allow the same operation to be applied to multiple entities using the same web service operation (i.e. replacing single parameters with sequences of parameters)

A possible disadvantage of enforcing access through a service interface is that a user may be restricted to a subset of the valid operations that might be performed on the data representation.

### 5.1.1 Service implementation independence

We chose to use Apache Tuscany as an implementation of Service Component Architecture (SCA). SCA allows different binding, interface, and implementation technologies to be combined in multiple ways while reducing dependence on the underlying technologies.

While it allowed simple creation of web services from implementation classes, we did find some problems with the approach:

- Although SCA is language-independent, only the Java version appears to be under active development. PHP and C++ versions are noticeably falling behind in terms of functionality and interoperability.

- The “wiring” between services is configured at startup, and due to the complexity of the startup protocol, dynamic addition of new services during runtime is not supported. This means, for example, that each Tree Store must be known about at startup. Since each tree store knows about itself at startup, we attempted to use service reference passing to allow Tree Stores to self-publish into a directory, for clients to retrieve the service reference (a classic SOA pattern), but this is a poorly documented feature in Tuscany, and we failed to get it to work.

We found that separation of the technology-independent service implementation classes from the technology-dependent service definition classes worked just as well to allow a degree of technology independence. To be clear, we implemented the services as “plain old Java objects” (POJO), then, in another package, implemented the SCA web services, using subclasses of the POJO classes.

## 5.2 Data integrity, consistency and service abstraction

In the existing CoL editing process, data manipulations have predominantly been performed directly on the database. In an attempt to prevent consistency errors, the original database schema was replaced by a normalised Base Schema. The Base Schema was designed to enforce domain-level rules, such as that a species name always includes its parent genus name. Hence the schema combined the naming hierarchy with the taxonomic hierarchy, making it impossible to insert a name which did not follow this rule<sup>6</sup>. However, normalisation by itself does not completely solve the consistency problem. For example, despite the strict use of foreign keys, “map” tables linking identifiers from two other tables cannot ensure that every item is linked at least once.

Best practice in computer science is to provide a level of abstraction that matches the domain of the user. Providing a service abstraction allows a component to provide arbitrary domain-level transformations which directly model the reasonable transformations a biologist would make to a taxonomic tree, without exposing inconsistent transient states that may exist in the underlying data structures. Each domain-level transformation may require manipulations of multiple tables, or even multiple databases. While these transformations could be implemented using database stored procedures and transactions, this approach, especially with multiple databases, reduces portability and provides limited mechanisms for use in a distributed environment containing non-SQL components.

Hence, a common solution is to use “web service” technologies to integrate the underlying technologies, and provide a domain-level abstraction. By only making available to clients a complete operation containing all the steps to move from one consistent state to another consistent state, services can provide integrity without allowing an inconsistent state to arise. In this situation, database normalisation and referential integrity can be used to confirm the correctness of the service operation, but are not necessary. i.e. a non-normalised schema without referential integrity can be kept consistent by only allowing access through a correctly-implemented service implementation. The benefit of normalisation and referential integrity is

---

<sup>6</sup> Even at the time of designing the Base Schema, there were known valid exceptions to this rule, such as hybrid species, and additional flags and tables were added to represent these. During the project, another requirement arose, to insert a parenthetical subgenus into the name for certain species, without actually adding a new taxonomic rank. Again, this required significant changes to the Base Schema, in complex and non-obvious ways. The WP 7 Multi-tree Schema used separate taxonomic and name hierarchies, and hence supported these requirements simply through the use of simple data elements added to the name hierarchy.

then the additional assurance of the correctness of the service implementation. Furthermore, normalisation can if appropriate be relaxed, as in the case of the denormalised schema used to support the end-user interface to the CoL on the distributed DVD and on the Web.

### **5.2.1 Separation of working practices and fundamental constraints**

There are three levels of constraints that are applied to data transformations:

1. Structural integrity – constraints that ensure the underlying data structures are internally consistent. This includes prevention of missing links (entities missing a required relationship) and dangling links (relationships missing a required entity).
2. Domain integrity – constraints that are fundamental to the domain field, and can never change. In taxonomic checklists, this includes the order of taxonomic ranks.
3. Working practices – constraints that are required by the domain users, but that are not fundamental to the domain. These may differ between groups of users, or within a single group over time.

While services can enforce arbitrary constraints on the operations they perform, it is useful to separate those constraints that are fundamental to the domain data model, including the structural integrity of the data model and fundamental domain integrity constraints, from those constraints that reflect the working practices of the users.

The fundamental constraints should be enforced by the web services and the underlying technologies including database normalisation and referential integrity constraints. In the representation of a taxonomic tree, for example, it doesn't make sense to add a taxon that is not connected (transitively) to the root of the tree. Hence, there is little reason to allow such an operation to be performed using the service interface.

Working practices, on the other hand, should not be enforced by the services and underlying technologies. Instead, they should be isolated to a single higher-level component. This reduces the risk that changes will mistakenly affect the fundamental structural and domain constraints and simplifies deployment of new working practices. It also promotes reuse of the services in other organisations with different working practices.

However, this creates the requirements that at least one set of working practices can be defined and that it can be determined which set of working practices data conforms to, and what transformations are required to meet another set of working practices.

Working practices may be expressed as “business rules”, specifying for example whether a taxon of one particular rank can be added as a child of another specified rank. Some rules are stronger than others. For example, taxonomists never place a genus as a child of a species, but other practices, such as the use of 'Not assigned' pseudo-taxa mentioned earlier, can be altered to suit a change in the policy of the editors of the CoL.

### **5.2.2 Orchestration services**

Where multiple services need to be coordinated over a period of time, it is desirable to have a permanent service to orchestrate the process, in order to remove the responsibility from client tools.

### **5.2.3 Consistent approach for long-running tasks**

Initial development of the operations used a single operation for most tasks, with the web service operation returning when each task had finished. As discussed in this document, this caused

problems both with operation of the tasks and with the availability of information on the progress of long-running tasks. We have developed a uniform protocol for long-running tasks, allowing all tasks to be started, monitored, and cancelled in a consistent manner.

Providing a consistent protocol for managing long-running tasks, allows much of the code to be shared between implementations. It also allows for nesting of long-running subtasks, while passing the status information up to the ultimate user. In the e-2 architecture, all long-running tasks provide progress information in the same structure, allowing higher-level operations to map the progress of lower-level tasks to progress in the higher-level task.

## 6 Appendices

### 6.1 Appendix I: Components delivered

The source code of the components is accessible using a Subversion client from the Subversion source code repository at the URL `svn://dev.4d4life.eu/E2_Infrastructure/trunk/task6`. It is also possible to see the source code using a web browser at the URL

`http://dev.4d4life.eu/websvn/listing.php?reaname=E2_Infrastructure`. The code repository and the document “The e-2 architecture: specification of its software structure” (WP 7, April 2011) provide details of how to build, test and release these components, although the current version of the document is intended to support the current interim use of the components in the UAW rather than in the e-2 environment.

The Assembly and Release subsystems are fully supported in the code repository and fully componentised using the SOA/SCA methodologies described in Deliverables 7.1B and 7.2. Although the WP 6 QAW and denormalisation tools do not currently use the SOA infrastructure, they are integrated through the HDE Project component into the portal as +assembly and release components respectively.

The Control subsystem is demonstrated but not yet componentised using SOA/SCA, but does contain multiple rulesets/ruleflows separated by task.

#### 6.1.1 Application directory

Contains the top-level components developed for the e-2 architecture, provided through Apache Tuscany as Service Component Architecture (SCA) components, allowing protocol-independent linking of the services. Our implementation uses SOAP and JSON-RPC web services.

This provided components including

- the Administration service, which stores exception and timing information of interest to the administrators of the other services,
- the Tree Store service, which wraps each database with a service interface providing Navigate, Edit and Transfer capabilities,
- the Decision Manager service, which wraps the Drools-based rules and process management for orchestration of long-running tasks potentially involving multiple Tree Stores, and
- the Hierarchical Data Editor (HDE), which provides a web-based client for navigating, editing and transferring trees stored in the various Tree Store services.

### 6.1.2 Core directory

Contains the service implementation classes, without any reference to service technologies, including SCA annotations. In particular, this includes the Administration, TreeStore, and Decision Manager implementations. The HDE is not included here, since it is a JavaScript tool reliant on the Tuscany infrastructure.

### 6.1.3 Common directory

Contains the utility modules that provide useful functionality for the e-2 architecture. In particular, this contains

- the ServiceInterface module, which include the XSD and WSDL files defining the services and the transfer protocol for sending parts of a taxonomic tree between components, used for the Navigate and Edit services. This format is based on our Multitree Schema, but provides a superset of the other schemas used in CoL work (ACAS and Base Schema),
- the Transfer module, which includes the WP 7 code or converting to and from DwC-A format<sup>7</sup>, used in WP7 for bulk transfer of data (the Transfer service), and
- the DAO modules, which provide a mapping between JAXB types (derived from the ServiceInterface XSD files) and the various database schemas in use. In particular, this provides an alternative Java implementation to that developed in PHP for WP6 for accessing and modifying databases represented in the Base Schema.

---

<sup>7</sup> The i4Life project has prepared a standard “profile” for DwC-A, based on the experience of WP 7 and others, but this code does not yet fully reflect that profile.

## 6.2 Appendix II: The UAW showing e-2 components in use

These figures show the HDE and transfer components in use in the UAW tabbed interface.

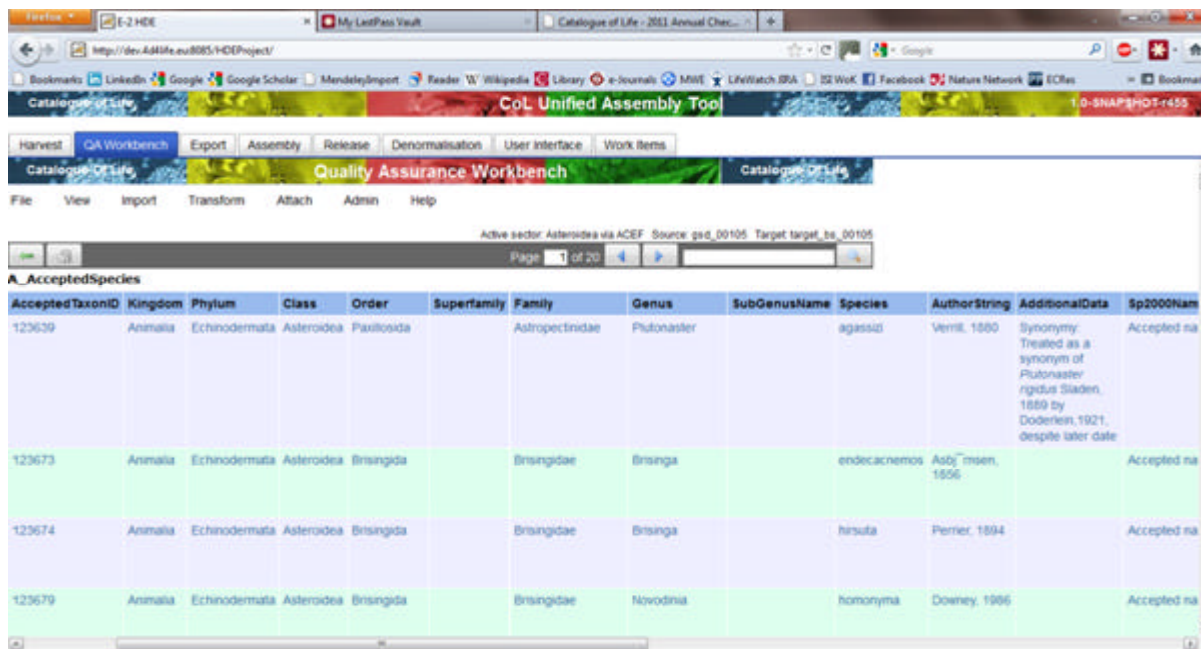


Figure 4 (above): The QAW in use, with imported taxa being viewed and checked for correct importation

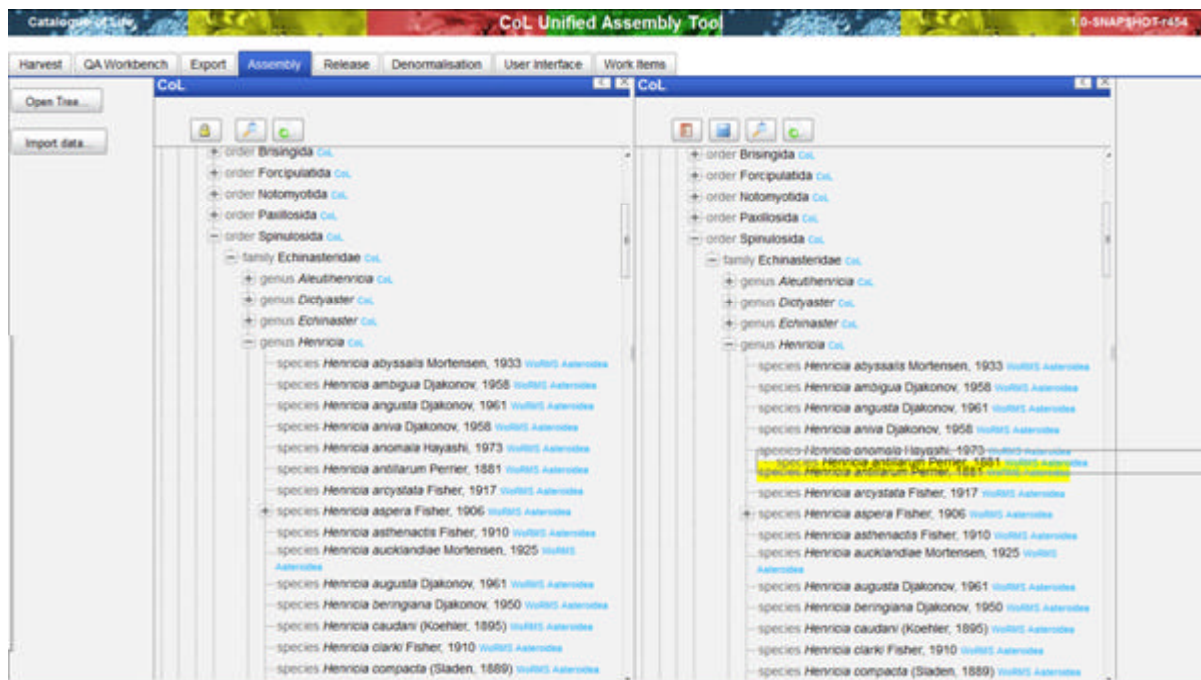


Figure 5 (above): The HDE in use, with a taxon being dragged to a new position in the CoL tree

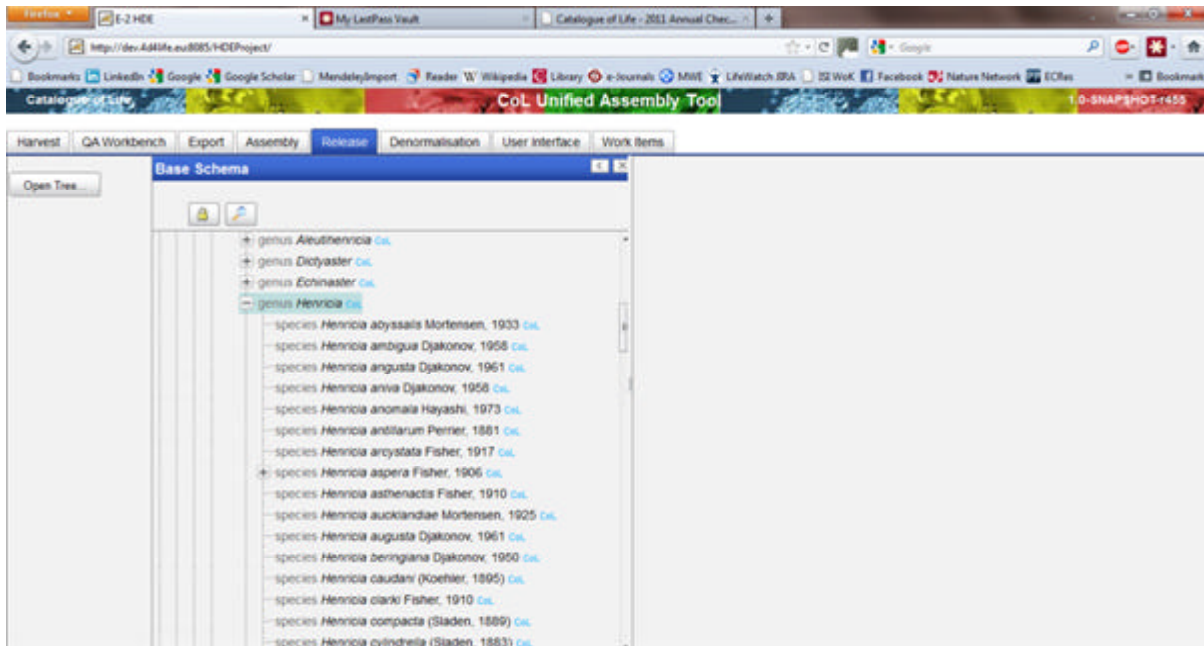


Figure 6 (above): A completed CoL tree being inspected in the UAW Release tab using the Tree Viewer component



Figure 7 (above): A completed and denormalised CoL product being tested before publication using the standard web end- user interface