

# 4D4Life



## Distributed Dynamic Diversity Databases for Life

Deliverable 7.1B

# Design of the e-2 architecture

Work package 7

Jonathan Giddy, Richard White,  
Alex Hardisty, Andrew Jones

31 August 2010

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

## Design of the new e-2 architecture

### Status

This is 4D4Life Deliverable 7.1B, version 1.1, 31 August 2010. The latest version of this document will be found on the 4D4Life project wiki at <http://wiki.4d4life.eu/wiki>

The original Description of Work called for a single deliverable D7.1. This was split into two deliverables, D7.1A previously submitted on 30 April 2010 and the present D7.1B. This deliverable D7.1B does not replace D7.1A, but contains new material on the e-2 architecture design, and should be read as a continuation of D7.1A.

The contents of both deliverables will be updated as necessary and maintained as a complete integrated design reference document containing all the parts of D7.1A and D7.1B, maintained on the WP7 wiki and Design Team web site to reflect subsequent changes, thus in due course superseding both D7.1A and D7.1B.

### Table of contents

1.	Introduction .....	3
1.1.	Catalogue of Life context .....	3
1.2.	Scope of the e-2 architecture design .....	4
2.	Design .....	4
2.1.	Design principles .....	4
2.2.	Design outline.....	5
2.3.	Design process and methodology .....	5
2.4.	The e-2 platform.....	6
3.	Technical specifications .....	9
3.1.	System overview .....	10
3.2.	Harvesting subsystem.....	12
3.3.	Assembly subsystem.....	15
3.4.	Release subsystem.....	18
3.5.	Customisation subsystem .....	22
3.6.	User Management subsystem .....	23
3.7.	User Components .....	24
3.8.	Cross-mapping subsystem .....	25
3.9.	Control subsystem .....	25
4.	Data schemas .....	26
4.1.	Service interfaces.....	29
5.	Appendix I: Phases of development .....	33
5.1.	Proof of concept (PC).....	33
5.2.	Test implementation (TI).....	34
5.3.	Enhanced prototype (EP).....	34
5.4.	Later “non-core” additions (NC) .....	34

# 1. Introduction

This document represents the results of our research into the requirements for the new “e-2” architecture for the Catalogue of Life, and our proposals for the specification and design of the new architecture.

The initial Work Package 7 Deliverable D7.1A (“Report on Requirements, Specification and Design of new e-2 architecture”, 30 April 2010) provided detailed requirements for the architecture. It included in its Appendix I a large number of “user stories”, which documented tasks which the new architecture should support and on which the requirements and design were largely based. D7.1A also provided a specification of the main features of the new architecture, and a high-level discussion of the design. These main features and design principles described in D7.1A are summarised below for the convenience of the reader.

We describe the design of the system in sufficient detail that the initial phases of the e-2 architecture development will be implementable directly from the design. Our next deliverable in WP 7 (D7.2) is a “proof of concept” of this design, consisting of several software components operating together to demonstrate the working of the new architecture. The development of this proof of concept will involve some experimentation, may reveal requirements which it might not be possible to meet, and might result in concomitant changes to some of the details of the design. Any such changes and improvements will be recorded in future versions of this document. Because of the need to maintain an up-to-date set of documentation about the e-2 architecture design, a complete design reference containing all the parts of D7.1A and D7.1B will be maintained on the web.

## 1.1. Catalogue of Life context

Although not restricted in its application to the Catalogue of Life (CoL), the e-2 architecture must meet the needs of the CoL, which provides a mapping of names to species (and higher-level taxa), and the information is provided in a tree structure reflecting the traditional hierarchical way of classifying organisms. Because of the key role of the taxonomic hierarchy in organising this information, we frequently use the word “tree” as a concise term to refer to a set of taxa defined by membership of a higher taxon, consisting of a checklist of species, including synonyms and other CoL data elements, arranged in a single taxonomic hierarchy.

A tree represents a taxonomic model. In the existing architecture, there is a single “consensus” tree which defines a widely-accepted taxonomic model based on expert opinion contained in the Global Species Databases (GSDs). Each GSD is curated by experts in the taxonomic groups represented in the GSD. The e-2 architecture will support the representation of alternative taxonomic models, to allow users to work with other models and to compare multiple models.

A node of the tree represents a set of scientific names (or taxon concepts), which refer to the same taxon according to the taxonomic model represented by the tree. Each node in a tree has a single accepted name, which is one of the scientific names (or taxon concepts) chosen as the preferred name, again, based on the expert opinion contained in the GSDs.

## 1.2. Scope of the e-2 architecture design

As described in D7.1A, the e-2 architecture will provide a new framework for the Catalogue of Life software system. Key enhancements will be:

- renewing the existing architecture to minimise the load on data providers (typically GSDs) by providing a variety of options for data provision and updating, while at the same time improving data currency,
- providing better availability and scalability of services, using a distributed architecture,
- placing greater emphasis on the specification and use of automated rules for checking and maintaining the consistency of the Catalogue, to reduce the workload of experts and editors and allow them to concentrate on the difficult cases,
- a more flexible query mechanism providing generic querying over all the attributes stored in the Catalogue, not just names,
- a more fully service-based architecture allowing components to be reused in different configurations and allowing better integration with client systems, and
- the adoption of an ‘open platform’ approach to cater more easily for future needs and interactions with third parties.

The e-2 architecture, as used by the Catalogue of Life, does not:

- insist on real-time access to source data (and not all GSDs may be online), but will allow the Catalogue to stay up-to-date and minimise the number of out-of-date entries,
- impose a single view of taxonomic names or hierarchical positions on species, but it provides a means of presenting and delivering information supplied by expert taxonomists, including the development of tools to monitor quality, identify taxonomic conflicts, support navigation within multiple hierarchies, etc.,
- provide a complete catalogue of all attributes available in the GSDs, but uses a small number of key attributes to allow users to restrict which data sets need to be interrogated further, or
- provide individual occurrence data (but it may provide regions of species distribution).

## 2. Design

### 2.1. Design principles

A Service-Oriented Architecture (SOA) has been adopted for the design (report D7.1A -- Section 3.2.1). This means that appropriate SOA protocols can be used, and replaced in future as necessary with minimal impact on the rest of the system. Specification of the protocol and operations supported in the “Proof of Concept” demonstration (see below) is the subject of current activity. The SOA approach is complemented in the design by a Business Process Modelling (BPM) approach (Report D7.1A -- Section 3.2.2) which is fundamental to the management framework because it supports flexible co-ordination of management and other tasks. Specifically, also, event notification will support the detection of abnormal

situations that need remedial action (Report D7.1A -- Section 3.2.4).

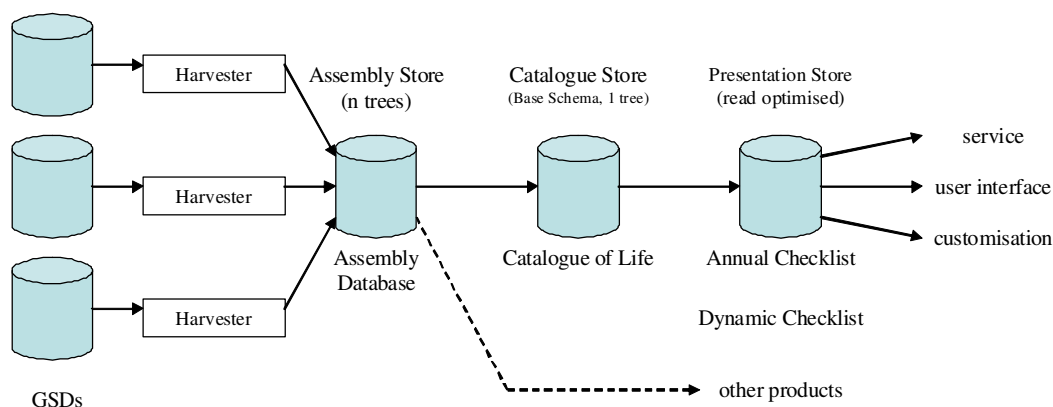
For components that are externally accessible, we plan to provide service-based messaging, based around a popular technology, such as SOAP Web services, REST Web Services, or JSON RPC-style services.

Internally, components may use more specialised protocols, controlled by SCA (Service Component Architecture).

## 2.2. Design outline

The design is based on the standard data flow or business model for the Catalogue of Life, which consist of three main stages: (i) **harvesting** data from data providers (at present mostly GSDs, Global Species Databases), (ii) **assembling** a single integrated Catalogue of Life, and (iii) **publishing** this in various products including the Annual Checklist and Dynamic Checklist.

The following diagram introduces some key components of the e-2 architecture design, illustrating these three stages (many of the components defined later have been omitted for simplicity at this point):

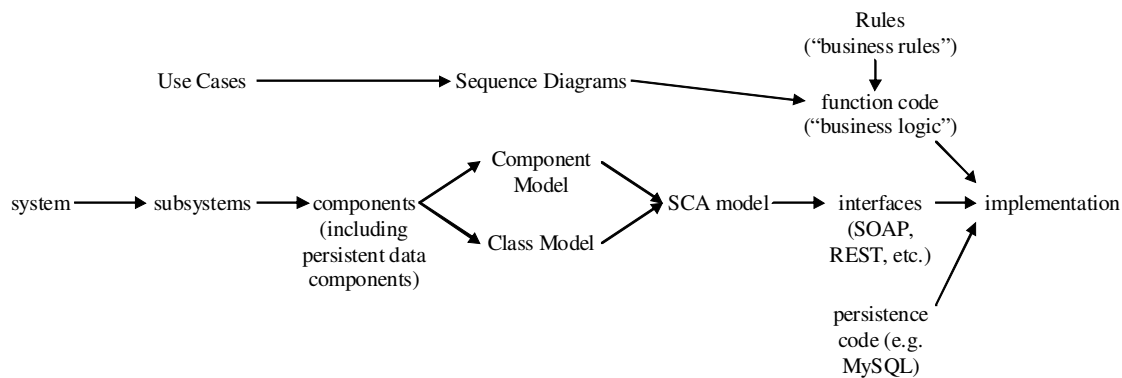


**Fig. 1 Data flow and main components**

## 2.3. Design process and methodology

Since we are designing an architecture, the key issue is to identify the components of the architecture and their interactions. We have taken three complementary approaches:

1. Identify components and their inputs and outputs, based directly on our understanding of tasks to be performed (the “Subsystems and components” section)
2. Identify sets of components relating to specific responsibilities of the architecture, and identify the kinds of interfaces needed by each (the “Component diagrams”)
3. Identify the sequence of tasks that will be performed in specific scenarios (the “Sequence diagrams”), in order to help us determine (a) whether there are further important components that are needed and (b) the specific operations that will be used in the components’ interactions (as opposed to the general interfaces that will be provided and required).



**Fig. 2 Design process**

The use of the methodology above does not commit us directly to specific technologies. However, it is our view – as explained in D7.1A – that a Service Component Architecture is appropriate as the basis for implementing the design. This provides us with a distributed, modular basis for the architecture, readily extensible and interoperable with other systems.

Furthermore, the information entered and stored during the construction of the component diagrams provides for the automated generation of some of the skeleton program code for implementing the components.

The sequence of steps in the modelling and design process is therefore:

1. High level UML design: Component diagram for overall system and its sub-systems, including review
2. Mid level UML design: Individual component diagrams for each sub-system (7-10), leading to perhaps 25-40 components overall; review
  - Assign user stories as use cases to each component; identify missing use cases; create a subset of these (the rest can be done by WP6 after handover if required).
  - Create sequence diagrams (similarly, only a subset needed for WP7 tasks)
  - Create interface specifications
  - Validate model, review and correct inconsistencies
3. UML Model to code transition:
  - Class diagrams
  - Automatic generation of framework and stubs, including validation and corrections
4. Implementation of business logic – what each component actually does that's unique: the framework and stubs still require implementation of the domain model, which is where the real coding work is needed. SCA allows one to focus on developing the model, with the framework being automated. Not all the business logic needs to be implemented by WP7.
  - alpha testing
5. Deployment
  - beta testing

## 2.4. The e-2 platform

This section provides a summary of the description given previously in D7.1A.

The Catalogue of Life is currently developed using a process whereby data is retrieved from the GSDs, through an on-line service (“wrapper”) or an off-line dump. The trees from multiple GSDs are attached to a base hierarchy to create a single tree, which is published as the Catalogue of Life. Users can search the Catalogue using scientific and common names and matching entries from the Catalogue are returned as a list.

This architecture suits users looking for a particular single species, to check the accepted name or correct spelling, or to link to the source GSD. However, retrieving large portions of the taxonomic tree is inefficient using the existing interface.

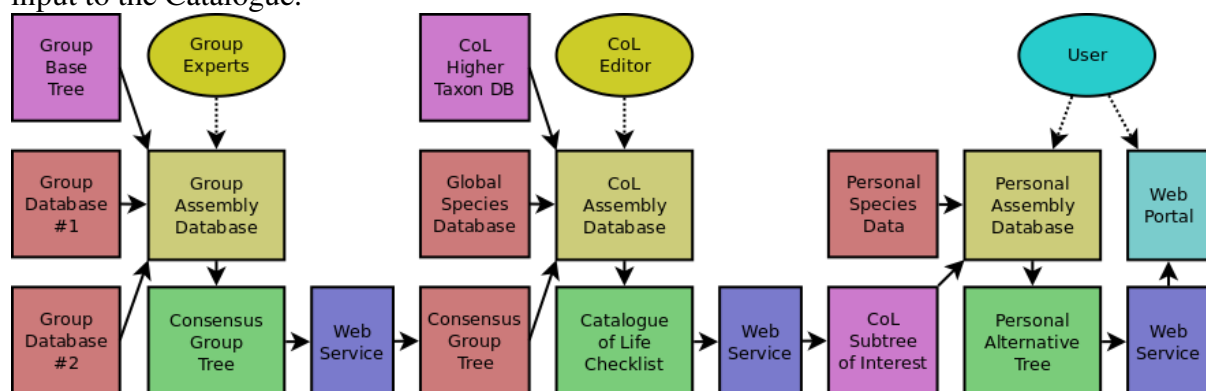
Many users wish to process the search result further, either to deal with advanced queries that return large amounts of data, or to process the tree directly using their own algorithms. In general, users want (i) to access outputs represented using the full CoL tree, or a subset still represented as a tree, and (ii) to modify tree nodes and links, and attach data or alternative subtrees, i.e. to perform similar processing to the editing stage.

To this end, the new architecture exposes the data structures of the editing process and makes the tools available for general use. The architecture changes from providing a *product* that provides read-only access to data to providing a *platform* that can be used to process data, to generate new data, and to share the generated data.

The e-2 architecture, while retaining the basic structure of the existing architecture, enables the provision of a general platform where users can edit multiple taxonomic trees, for example to create a single tree. This tree can then be searched and navigated as in the existing architecture, but in addition, an enhanced web service allows the release of the generated tree in a form suitable for further processing. This makes it possible to use the output of an instance of this model as input to another instance of the model, allowing the chaining of several processing stages, which may be performed by different users at different sites.

In the e-2 architecture, the Catalogue of Life becomes a single instance of the model, with experienced editors combining a number of quality input sources to create a consensus hierarchy for general use. Users may then choose the Catalogue as a comprehensive base hierarchy on which to build further trees, enhancing it with their own data or building an alternative taxonomic model.

Alternatively, the e-2 platform may be used without referring to the Catalogue of Life. For example, species databases from two different sites documenting species from related taxonomic groups may use an instance of the model to combine their taxonomic trees into a new checklist curated by experts from both sites. The generated output tree may become an input to the Catalogue.



**Fig. 3 Example of a system containing repeated instances of the e-2 model**

In order to implement the e-2 platform, we are using leading-edge, standardised technologies and best practices, including the following:

### **2.4.1. Service oriented architecture**

As shown above, the e-2 Architecture plans to support components that can be reused in many scenarios. To support this, the e-2 architecture is based on a Service Oriented Architecture (SOA).

SOA supports the development of coarse-grained components defined by their external behaviour, with replaceable implementations, which provide a well-defined interface that allows external clients to interact with them over a network.

A service component architecture (SCA) provides flexible composition of components into higher-level components that reflect the tasks to be performed in the problem domain. It also provides a great deal of flexibility to update the actual mechanism for publishing a component as a service.

Service Data Objects (SDO) provides an open flexible data transfer mechanism supporting disconnected operation and change sets. SDO ensures that the service parameters do not expose the underlying implementation of a service, supports a disconnected programming model, allowing it to be useful even when the services providing data are not always online, and supports the representation of change-sets, allowing smaller messages to be sent when only parts of a large data structure have changed.

### **2.4.2. Business process modelling**

Often, the same tasks will be performed in the same sequence over different data sets. Documenting these “business processes” reduces errors in running repetitive tasks and promotes adaptability of processes to new situations.

Business Process Management (BPM) provides an environment for specifying and executing these sequences of tasks. BPM tools generally provide graphical user interfaces allowing non-programmers to connect services together to represent a sequence of tasks. The tools encourage the replacement of repetitive error-prone manual tasks with automated processes, but supports situations where a human must make an expert decision or deal with an exceptional case.

### **2.4.3. Rule management and execution**

Much of the correctness of taxonomic trees can be encapsulated in constraints that can be expressed quite simply. Checking the large amounts of data in the Catalogue of Life to ensure satisfaction of all these rules for all entities, in the face of continuing updates, can be facilitated by a rule engine in order to identify broken rules that need further attention. When problems outside the existing rule-set are found, additional rules can be created to ensure that these problems are not repeated.

Furthermore, making edits to combine trees and make other changes can be manually intensive. When changes in source trees are received from data providers, it is not desirable for the editor user to have to repeat previously applied checks and corrections or to delay updates because of the effort of re-applying these edits.

A solution is to encode the edits as rules, which can be re-applied automatically. Editors can

be notified of individual data element updates only if a rule fails.

#### **2.4.4. Event notification and message queues**

In the Test Implementation phase, we will experiment with improved techniques for the retrieval of data from large remote databases, using a more scalable and loosely-coupled model which supports low-bandwidth fast notification of changes and reliable messaging. This will support additional capabilities such as informing users when changes occur to parts of the taxonomic tree in which they have an interest.

#### **2.4.5. Customisation and sharing**

The existing architecture provides only for anonymous access with support for limited customisations of results restricted to a single query. The e-2 architecture is designed with support for more complex customisations, such as annotations and alternative trees, and for personal user accounts as a repository for persistence of these customisations.

In some cases it may be useful to share these customisations with others. Examples may include annotations which record errors in the data or alternative trees which represent a widely used taxonomic model. Making customisations public requires a mechanism to identify these additions as originating from an alternative source, particularly where unmoderated annotations may be mixed with authoritative data supplied by a GSD.

### **3. Technical specifications**

The original word “specification” used in the 4D4Life Description of Work referred to the process of establishing what the broad scope of the new architecture should be. That “specification” consists of fixed objectives and design principles and processes (service-oriented, open to the community, able to replace the existing Annual and Dynamic Checklist assembly and publication processes, able to support multiple regional hubs, etc., see D7.1A section 3), whereas the Technical Specifications in this document (below) describe the details which a person developing a system component would need to know, including definitions, diagrams, and documents, and which may change in the light of experience with the prototypes.

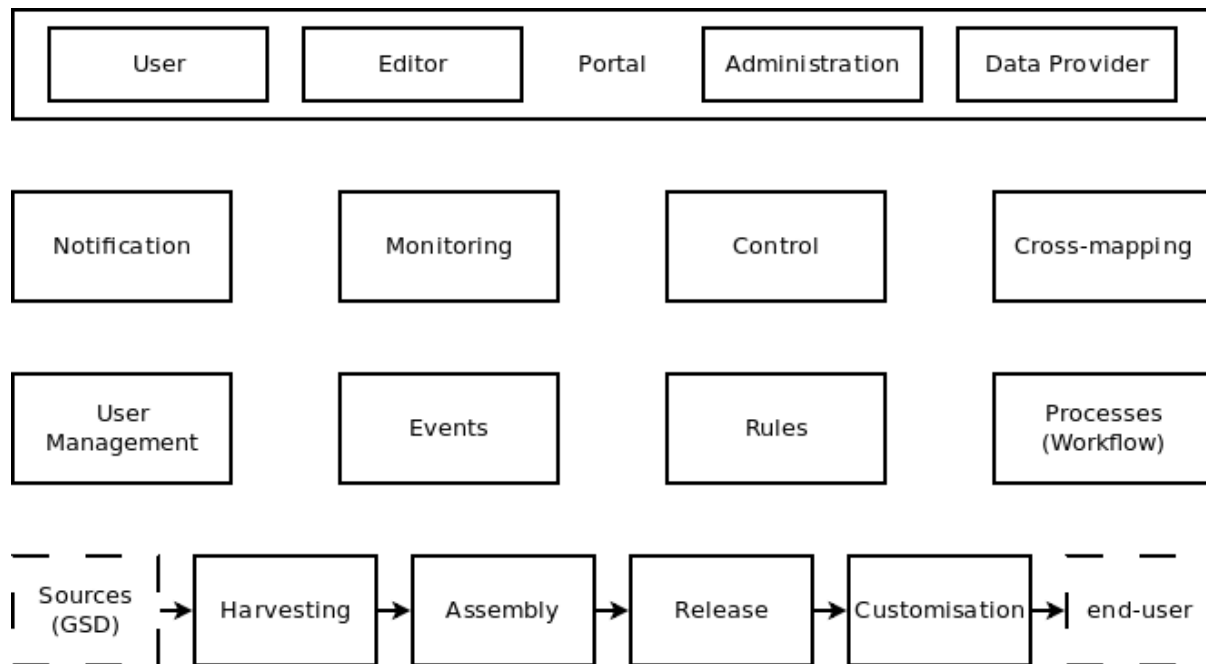
1. Class diagrams define the interfaces of the various components, i.e. the operations which they support.
2. Some of the components are primarily state machines. Definitions of system states and the events which can be viewed as causing changes of state will be documented. State diagrams will capture an alternative view of some of the processes described.
3. A set of “business rules” will be defined, and will evolve as the project proceeds, which will analyse and act on the states and other data to call services and execute tasks, for example to enforce appropriate procedures for catalogue updates, manage notification of events, etc.

The Specifications will be developed continuously during the next phase of the project, and will eventually contain an inventory of components and definitions of the data schemas and

service interface definitions, including explanations of the semantics of the data fields, service methods, parameters and events.

### 3.1. System overview

Figure 4 shows an overview of the subsystems planned for the e-2 architecture.



**Fig. 4 Overview of subsystems**

Starting from the bottom of the diagram, the main dataflow of the system shows data going through several subsystems between the source database and the end-user.

These subsystems are:

1. Harvesting, which retrieves data from the sources (GSDs);
2. Assembly, which assembles multiple hierarchies into a single hierarchy;
3. Release, which stores released versions of the single hierarchy, including archives of historic versions; and
4. Customisation, which adds user-provided additional information to the data.

The second layer contains the overarching internal support subsystems, including user management (authentication, access control, and group and role management), and systems to scale and automate processing, using events, rules and processes (workflows).

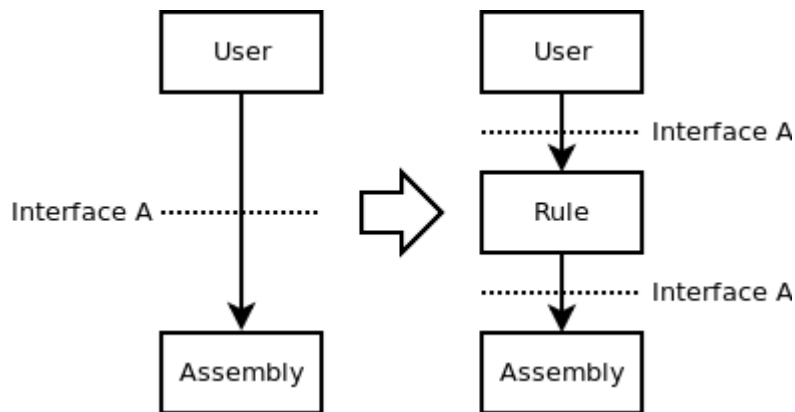
The third layer contains external-facing subsystems, with which users, data providers and system administrators interact directly.

The top layer shows a portal for interaction with stakeholders, including users, data providers, editors, and system administrators.

The use of a Service Oriented Architecture to connect these subsystems (and the components within these subsystems) allows the extension of the core subsystems through the insertion of

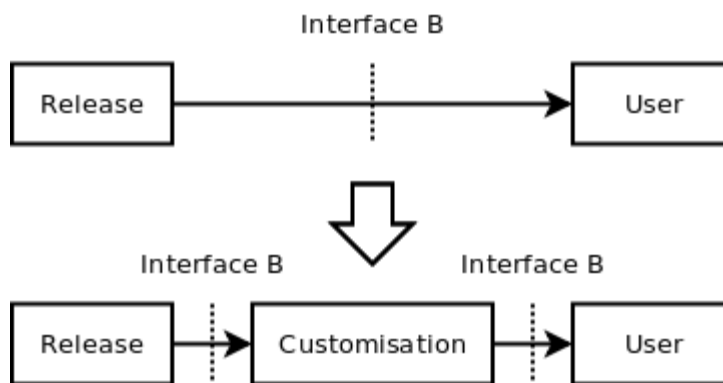
additional subsystems with minimal changes to service interfaces, and hence to the components being used to build the system.

For example, an initial design where an assembly user interface connects to the Assembly Subsystem using interface A to attach a node in one hierarchy to a node in another can be replaced by a design where the assembly user interface connects to the Rule Subsystem using interface A to create a rule to attach a node, and the Rule Subsystem connects to the Assembly Layer using interface A to attach the node. This insertion of an additional component allows automated repetition of the attachment process, for example when a GSD is re-harvested.



**Fig. 5 Insertion of Rule subsystem**

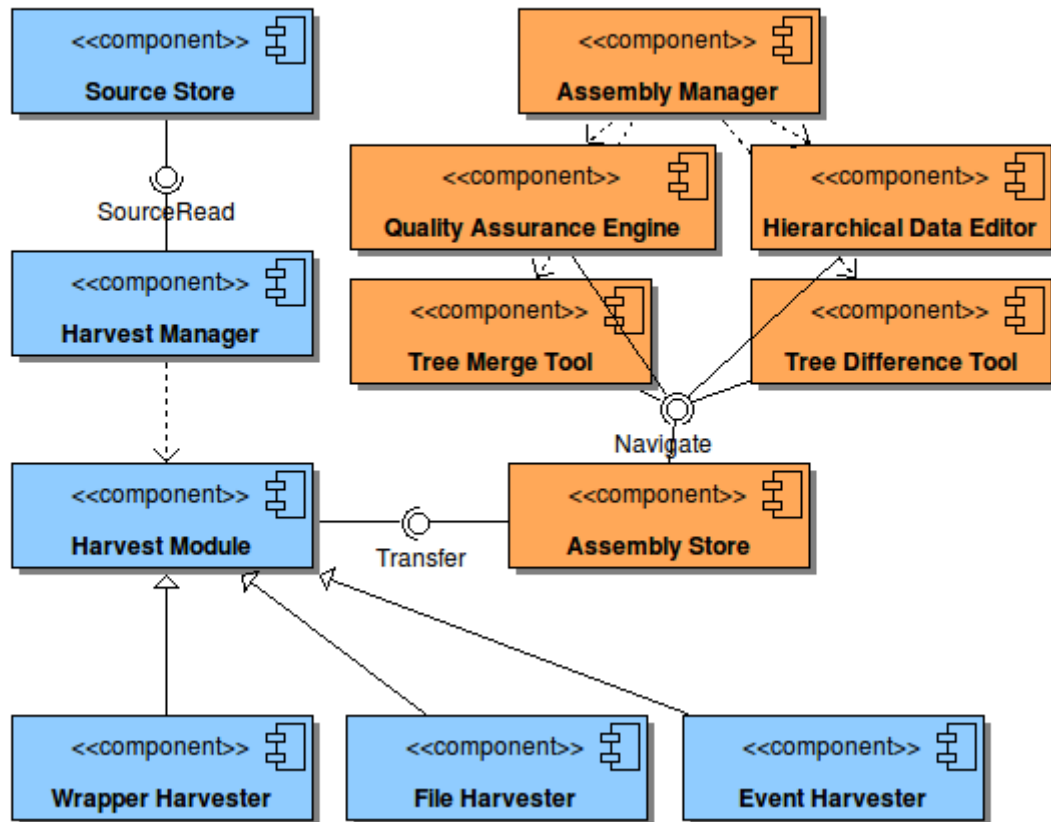
As another example, the Release Subsystem sending a subtree to an end-user interface using interface B, can be modified by the insertion of the Customisation Subsystem to add annotations to the data in the subtree. The Release Subsystem continues to send the subtree using Interface B and the end-user interface continues to receive the subtree using Interface B.



**Fig. 6 Insertion of Customisation subsystem**

This repeated use of the same interface allows the flexibility to compose the systems using user-driven workflows.

## 3.2. Harvesting subsystem



e-2 Harvesting and Assembly Subsystems

This diagram shows the components in the Harvesting (blue) and Assembly (orange) subsystems. The Transfer service interface is documented in section 4.2.

### 3.2.1. Component Source Store

The Source store provides information about the GSDs and other sources. Information includes:

- a unique name for the source (e.g. ILDIS, AlgaeBase)
- the method for access to data (legacy wrapper, file upload, event watcher)
- endpoint (e.g. a URL or network connection)
- administrative and technical contacts

Services provided: SourceManage, SourceRead, SourceWrite

### 3.2.2. Component Harvest Manager

The Harvest Manager controls the execution of multiple Harvester Modules, which each retrieve data from a single source, hiding the specific protocol being used by a data provider from the rest of the system (“Harvesting” refers to any method to ensure that the Assembly Database contains the latest version of a data provider’s data set).

Depends on Harvest Module

Services required: SourceRead

### 3.2.3. Component Harvest Module

A Harvest Module obtains updated taxonomic information from source databases (GSDs). Harvester Modules use a variety of legacy and novel protocols to retrieve the data. The exact process involved depends on the type of Harvest Module supported by the GSD. There are 3 types of Harvest Module: Event Harvester; File Harvester; and Wrapper Harvester.

Services required: Transfer

#### 3.2.3.1. *Component Event Harvester*

The Event Harvester is a new component that monitors a GSD for event notification messages indicating that the database has changed. The module will clone an existing tree in the Assembly Store and update it to create a new tree.

#### 3.2.3.2. *Component File Harvester*

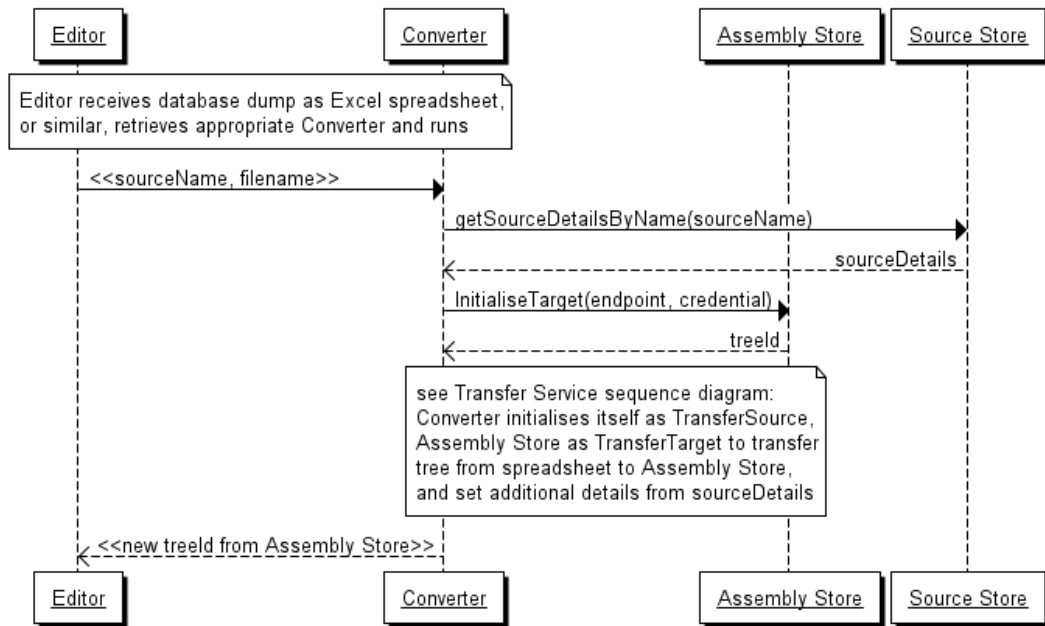
The File Harvester allows submission of a formatted file (e.g. spreadsheet) from the source. The module will transfer the new tree to the Assembly Store.

#### 3.2.3.3. *Component Wrapper Harvester*

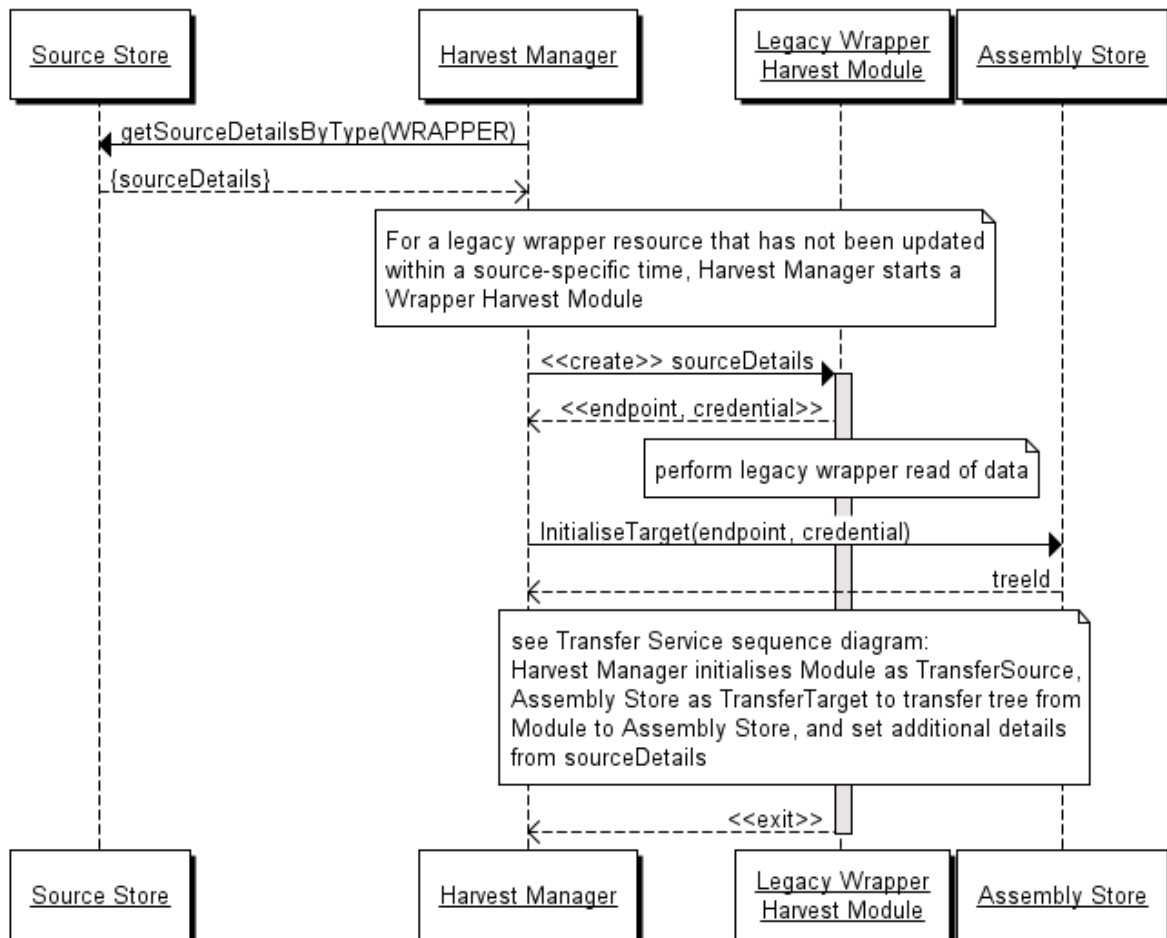
The Wrapper Harvester runs periodically and uses the legacy CoL Web Service to retrieve information from the source. The module will transfer the new tree to the Assembly Store.

### 3.2.4. UML sequence diagram: File Upload Harvesting

This one does not use the Harvest Manager and Module, since the use case is that an editor receives a file, and runs a tool to import it into the Assembly Database. The Converter is similar to a Harvest Module but is started explicitly by the editor.

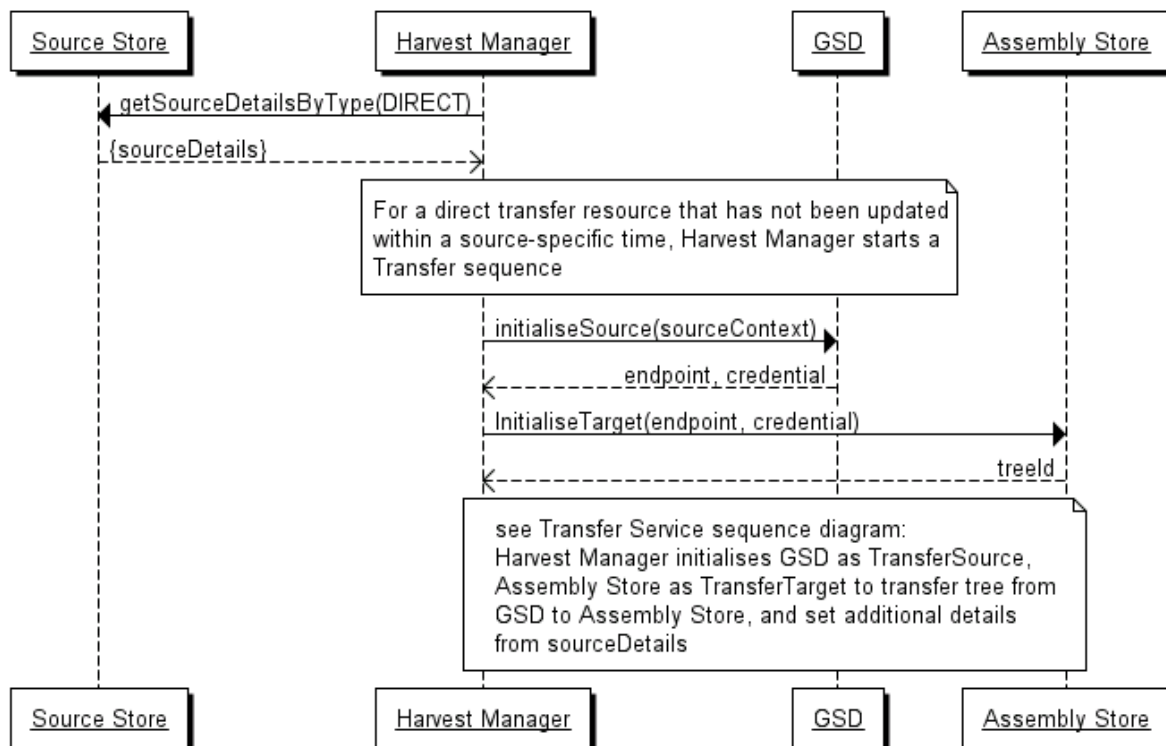


**3.2.5. UML sequence diagram: Legacy Wrapper Harvesting**



### 3.2.6. UML sequence diagram: Direct Transfer Harvesting

Direct transfer harvesting is a proposal for a new wrapper-style approach. Since we are creating a well-defined data transfer schema and a Transfer Service interface for bulk transfer of a taxonomic hierarchy, we can use that schema for the GSD → CoL transfer.



## 3.3. Assembly subsystem

### 3.3.1. Component Assembly Store

The Assembly Store provides a data store for multiple checklists and taxonomic hierarchies to be edited and attached to create a single consensus hierarchy or “tree”. The Assembly Store schema is optimised for the editing process.

Services provided: Navigate, Transfer

### 3.3.2. Component Assembly Manager

The Assembly Manager provides overall control of the assembly process from the submission of a tree into the Assembly Store through to the publishing of a tree to the Catalogue Store. This process can be managed manually, or could map to a workflow process:

- QAW on input tree
- connection of tree to base hierarchy

- QAW on base hierarchy
- publish to Catalogue Store

Depends on Quality Assurance Engine, Hierarchical Data Editor, Tree Merge Tool, Tree Difference Tool

Services required: SourceManage

### **3.3.3. Component Hierarchical Data Editor**

The Hierarchical Data Editor provides an environment for linking trees together and editing individual nodes.

Services required: Navigate

### **3.3.4. Component Quality Assurance Engine**

The Quality Assurance Engine validates the structure of a taxonomic hierarchy to ensure it complies with various rules, including the appropriate International Codes of Nomenclature.

Services required: Navigate

### **3.3.5. Component Tree Difference Tool**

Tests two trees for similarity and reports differences

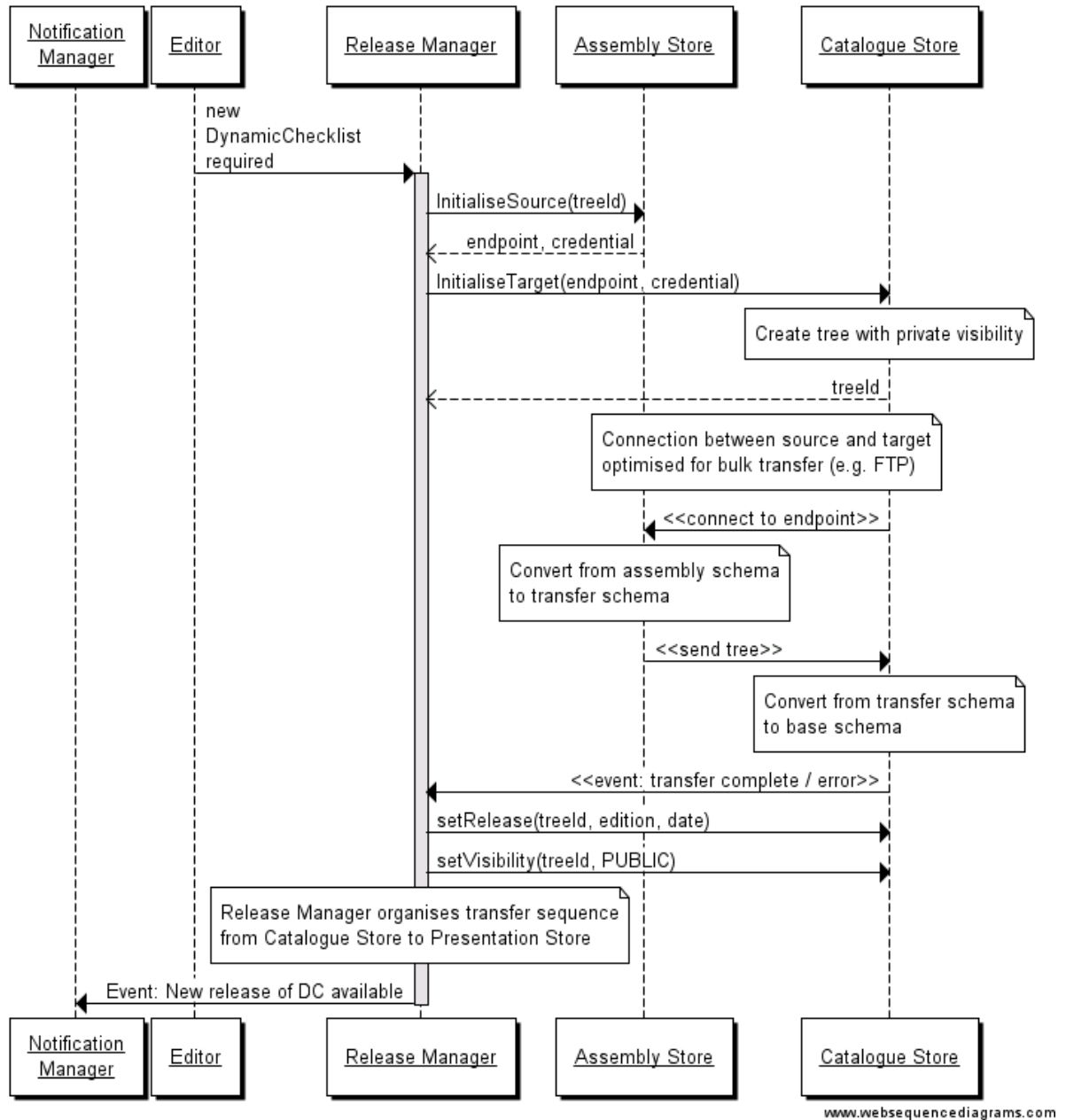
Services required: Navigate

### **3.3.6. Component Tree Merge Tool**

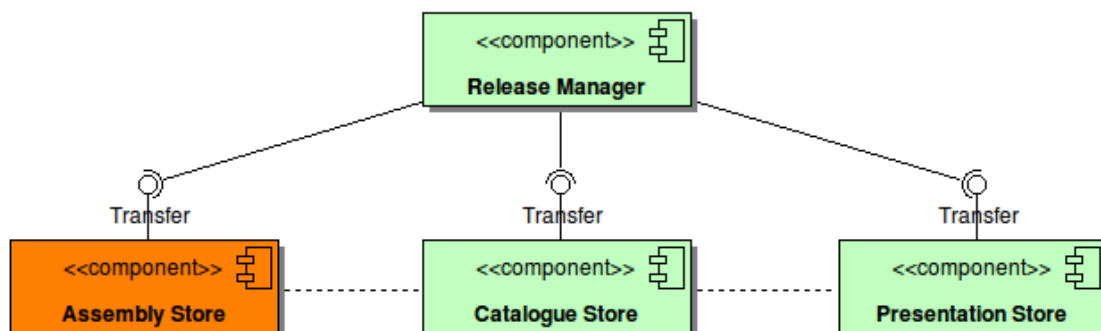
Merges two trees, possibly after cross-mapping or difference detection

Services required: Navigate

### 3.3.7. UML sequence diagram: Update Dynamic Checklist



### 3.4. Release subsystem



e-2 Release Subsystem

The Release subsystem provides access to a single consensus tree. The Release Manager transfers the consensus tree from the Assembly Store to the Catalogue Store, providing it with an edition number to allow users to distinguish it as a particular version of the tree. The Catalogue Store is primarily used for batch retrieval of the tree or sub-trees, for example to take a snapshot of the tree at a particular time. For typical user queries, the data from the Catalogue Store is converted to a form optimised for search and retrieval of smaller results. This optimised form is stored in the Presentation Store. Various user-level tools can query the Presentation Store to provide information to users.

#### 3.4.1. Component Catalogue Store

The Catalogue Store provides a data store for the reference version of a released edition of a taxonomic hierarchy, but is typically not used for user queries. The Catalogue Store schema is optimised for long-term archiving, and generation of user services and other products.

Services provided: Transfer

#### 3.4.2. Component Presentation Store

The Presentation Store provides a data store for a taxonomic hierarchy, optimised for search and other read operations.

Services provided: Navigate, Query, Transfer

Services required: Query

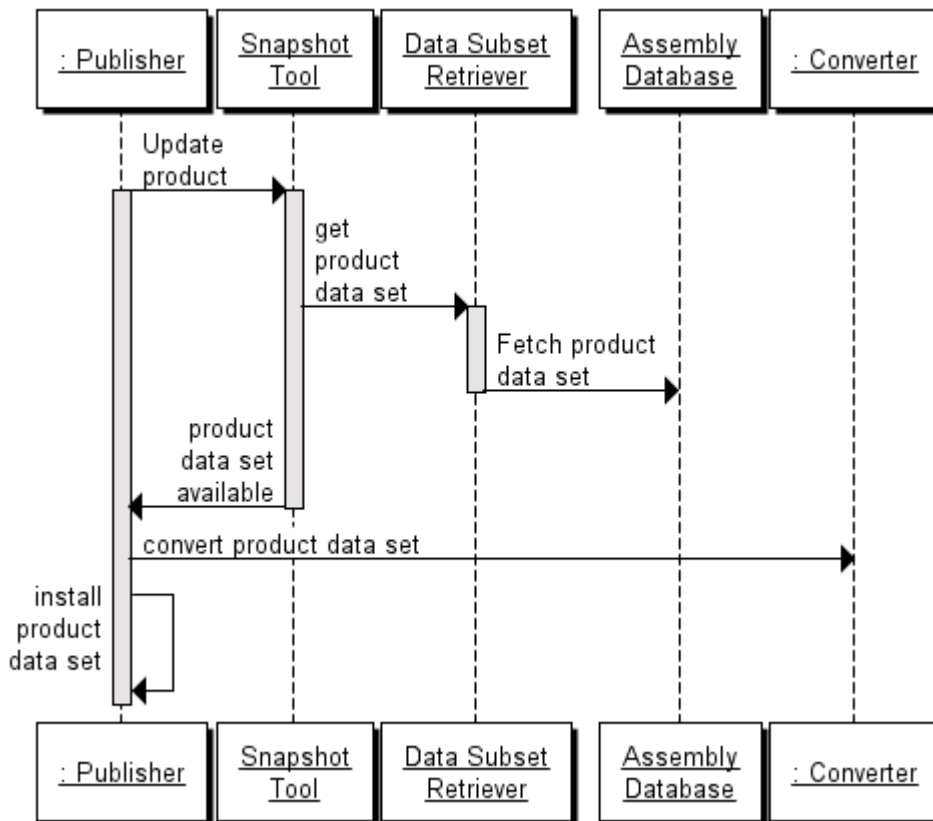
#### 3.4.3. Component Release Manager

The Release Manager component transfers a taxonomic hierarchy from the Assembly Store to a Catalogue store. This step is intended to indicate the publication of a new edition of the hierarchy. Apart from the hierarchy data, the Product Manager adds version metadata to

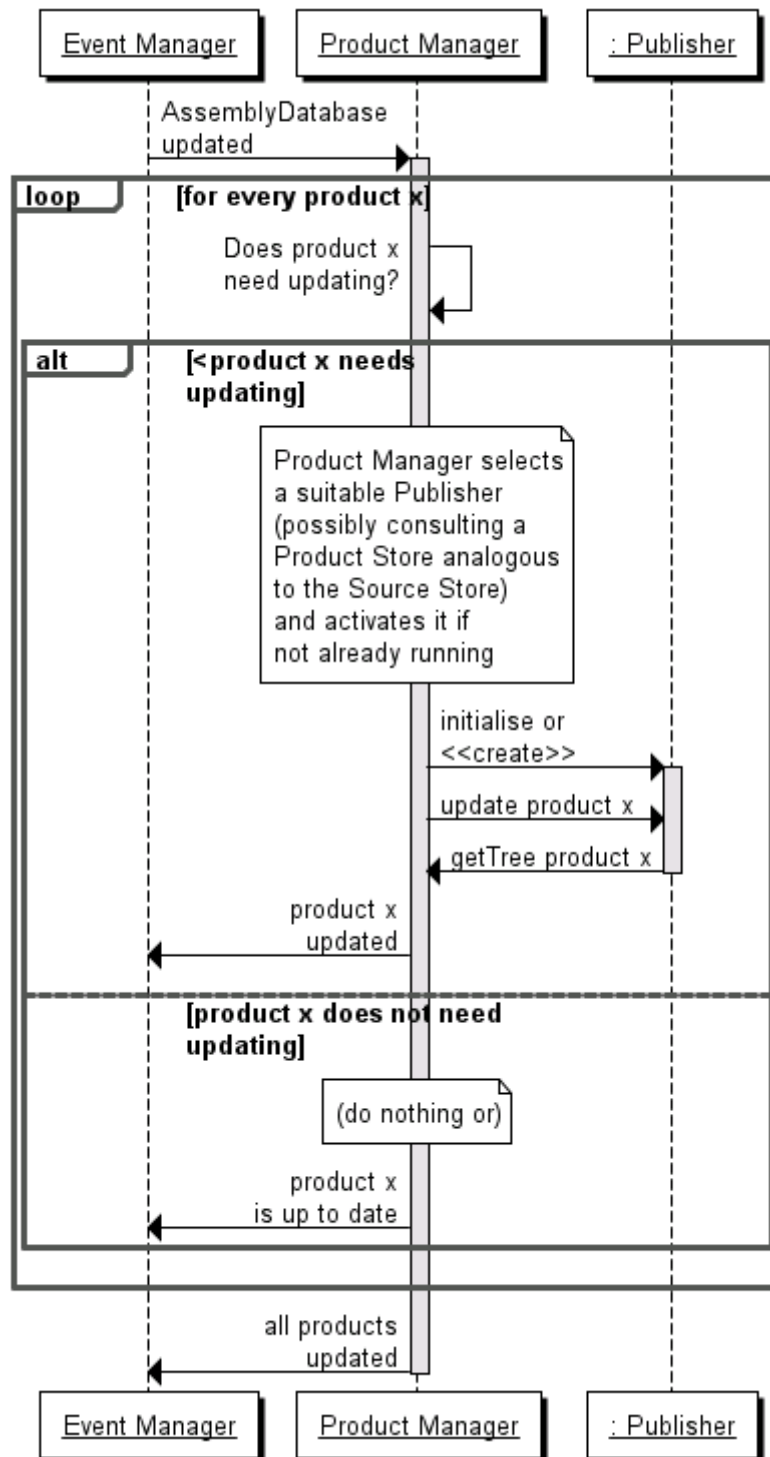
uniquely identify the new hierarchy (e.g. a date and an “edition”). The hierarchy then continues to be propagated to additional stores, in particular, the presentation store, which holds the hierarchy in a schema optimised for read operations.

Services required: Transfer

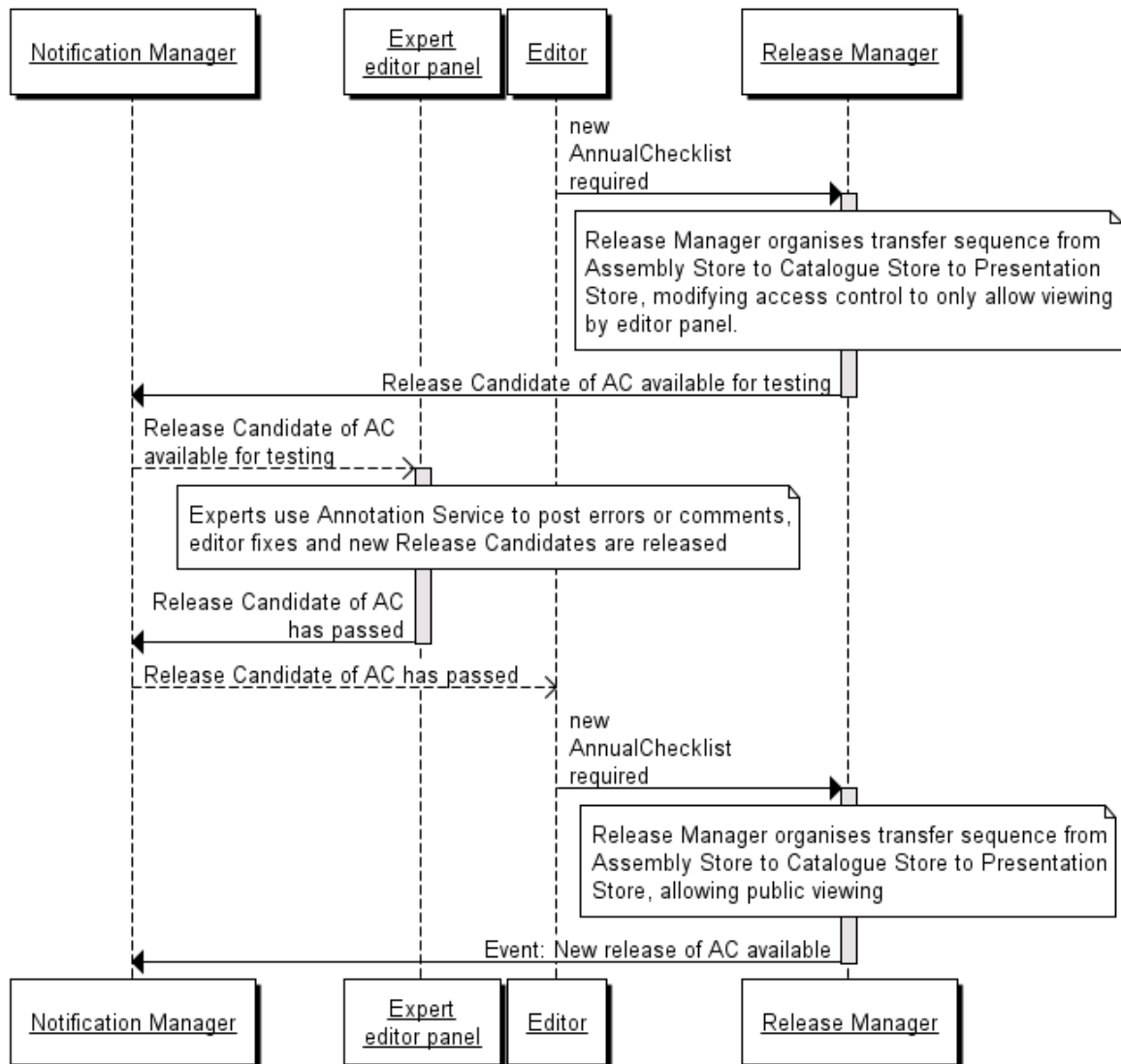
### 3.4.4. UML sequence diagram: Update a product



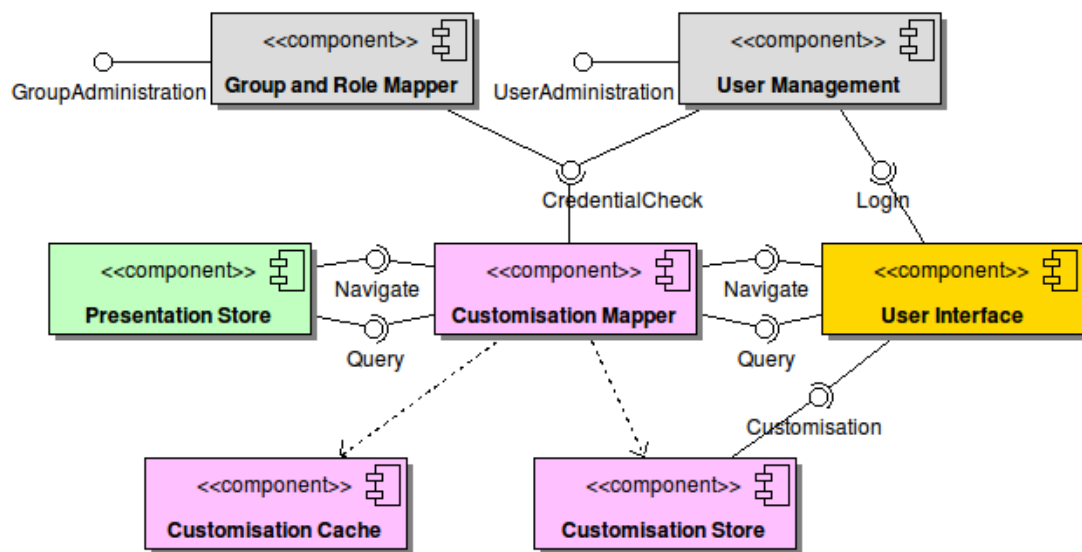
3.4.5. UML sequence diagram: Update all products



### 3.4.6. UML sequence diagram: Create Annual Checklist



### 3.5. Customisation subsystem



**e-2 User Management and Customisation Subsystems**

The Customisation Mapper is transparently inserted between the User Interface and the Presentation Store to annotate the results with additional customisations. Customisations are stored permanently in the Customisation Store. For efficiency, data combined with customisations may be stored temporarily in a Customisation Cache.

#### 3.5.1. Component Customisation Mapper

The Customisation Mapper modifies query results to include annotations, or other customisations, that are marked as visible to the user.

Depends on Customisation Store, Customisation Cache

Services provided: Navigate, Query

Services required: CredentialCheck, Navigate, Query

#### 3.5.2. Component Customisation Store

The Customisation Store provides storage of user customisations, including annotations

Services provided: Customisation, Query

Services required: CredentialCheck, Query

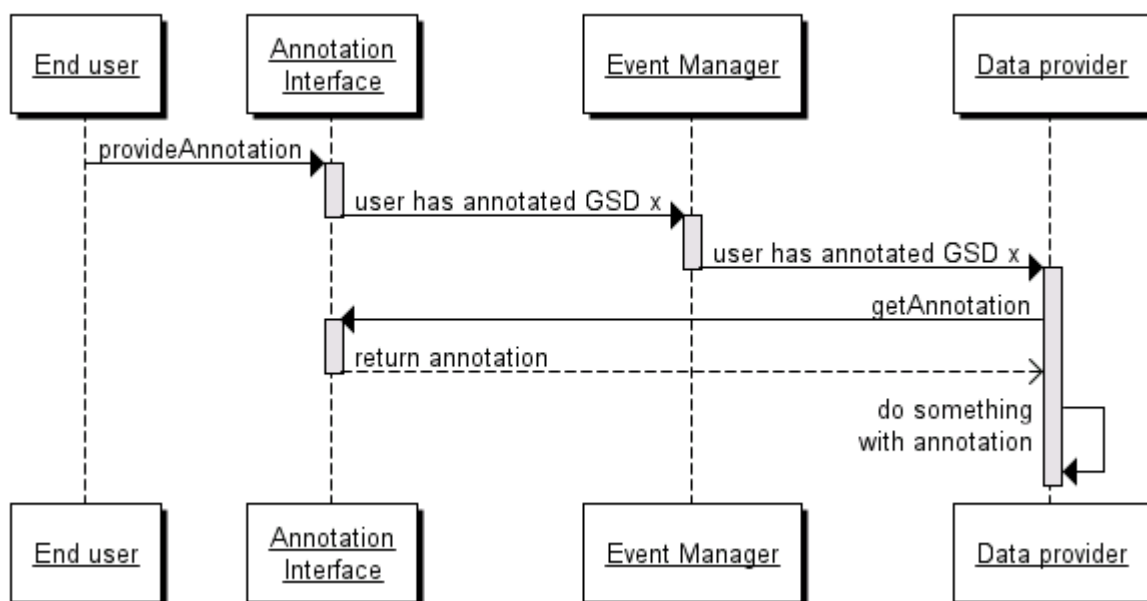
### 3.5.3. Component Customisation Cache

The Customisation Cache provides a mechanism for caching data modified by user customisations, to reduce the load on the Customisation Store. This is an implementation detail, but we expect it will be required since mapping of customisations may slow down presentation of data significantly.

Services provided: Query

Services required: Query

### 3.5.4. UML sequence diagram: Annotation and feedback



## 3.6. User Management subsystem

The User Management components provide tools for managing users and groups, allowing control of attribution and visibility of data and customisations.

### 3.6.1. Component User Management

The User Management component provides user account and login services.

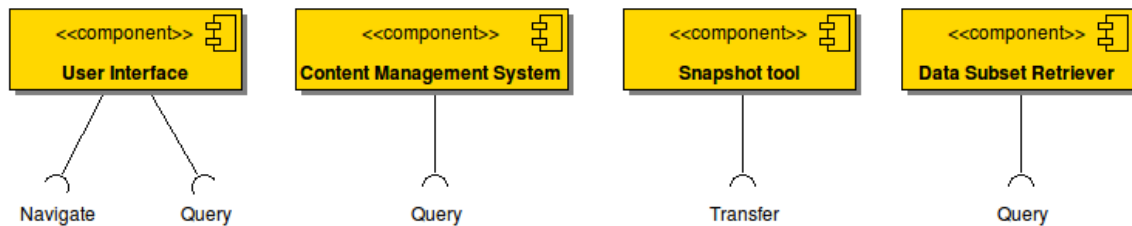
Services provided: CredentialCheck, Login, UserAdministration

### 3.6.2. Component Group and Role Mapper

The Group and Role Mapper provides control and checking of access control permissions.

Services provided: CredentialCheck, GroupAdministration

### 3.7. User Components



User components

#### 3.7.1. Component User Interface

The User Interface provides a web-based interface to navigate and query the data.

Services required: Customisation, Login, Navigate, Query

#### 3.7.2. Component Content Management System

The Content Management System provides static and dynamic content, which some of the dynamic content being retrieved from the Catalogue of Life stores.

Services required: Query

#### 3.7.3. Component Snapshot tool

The Snapshot tool allows a user to obtain a complete copy of the Catalogue of Life database for local use.

Services required: Transfer

#### 3.7.4. Component Data Subset Retriever

The Data Subset Retriever allows a user to retrieve a subset of data based on arbitrary queries which the user can then format appropriately, e.g. to create a checklist of endangered plants for a particular region.

Services required: Query

### 3.8. Cross-mapping subsystem

(To be developed in collaboration with WP4)

### 3.9. Control subsystem

#### 3.9.1. UML sequence diagram: Control and Management

A variety of workflows involve the activity of (human) managers, here designated as simply “Manager”.

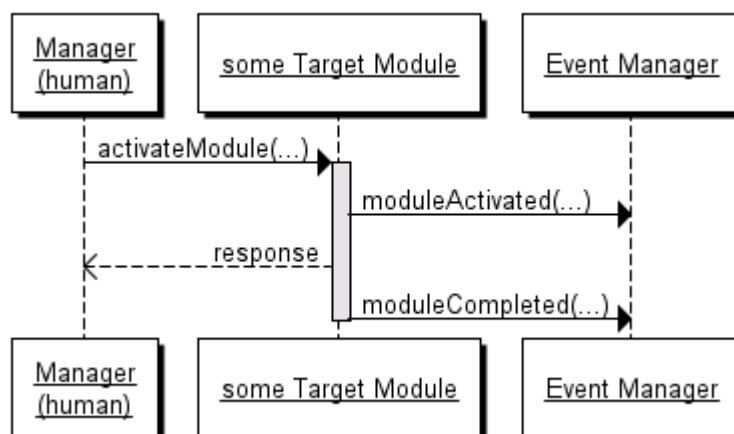
##### 3.9.1.1. Status tracking

The human manager needs to be aware of the current status of the system, hence the first necessary task is to keep track of the system's status.



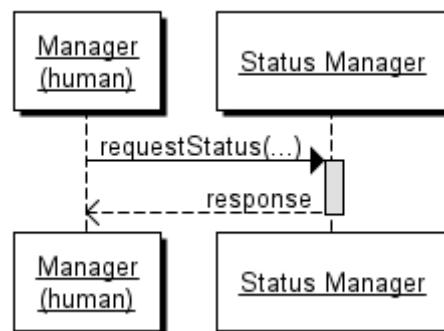
##### 3.9.1.2. Manager initiates action

The human manager may wish to initiate some activities manually, for example:



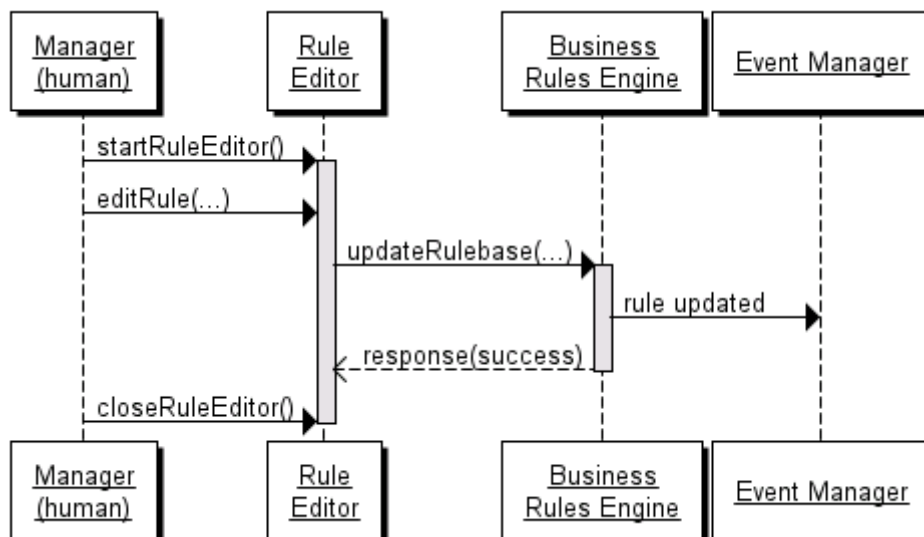
### 3.9.1.3. Request status report

The human manager may wish to initiate some activities manually, for example:



### 3.9.1.4. Edit rules

The human manager may wish to edit the rules which control the automated parts of the system.



## 4. Data schemas

WP6 is defining a data storage schema, called the Base Schema, as a standard schema for storing data.

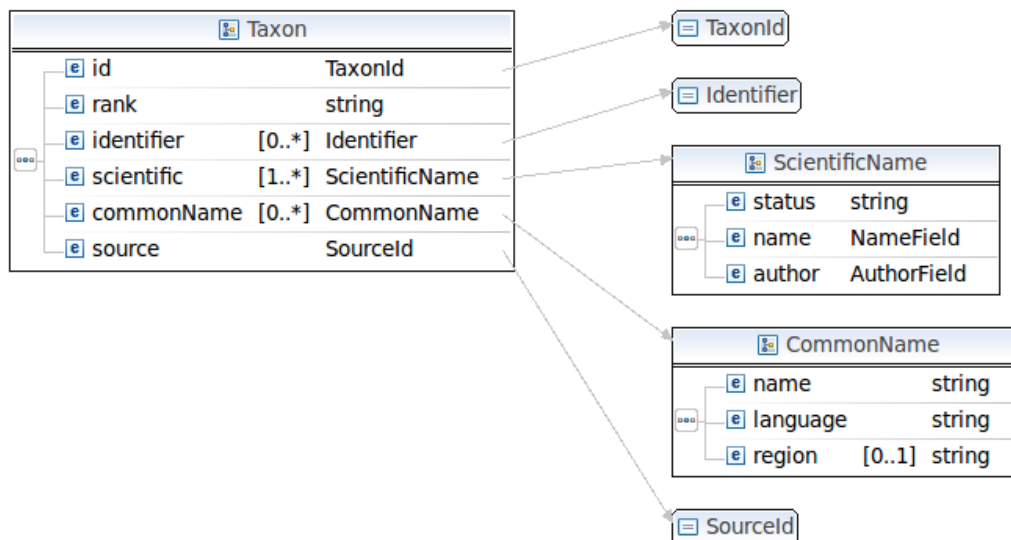
In WP7, our main concern is with providing a uniform data transfer schema compatible with the base schema, but appropriate for service-oriented operations. In particular, in a service-oriented architecture, it is usual to provide coarse-grained operations, often working on aggregate data. This is different to the approach typical of object-oriented architecture, where a single data object is manipulated by each operation.

The data transfer schema reuses some basic data types throughout the system, in order to

improve reuse and reduce translation of datatypes.

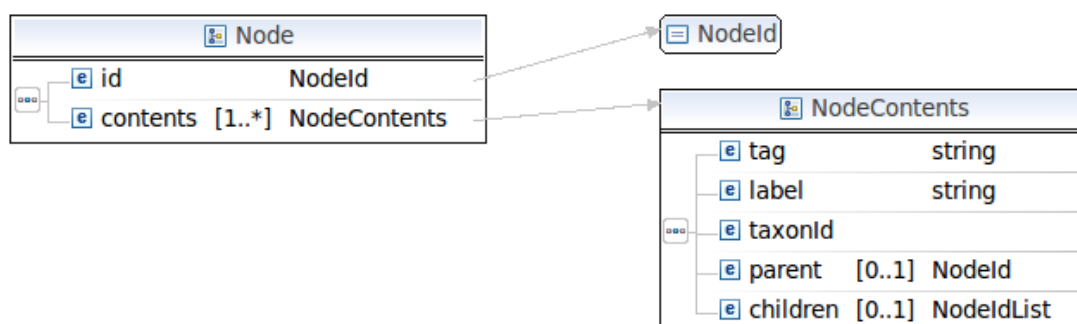
The basic “domain type” is a Taxon, which represents an entry in the catalogue. This datatype contains all the information that is represented by a single entry in the catalogue. However, it does not contain information about the relationship of one entry to another.

Note that the type represented here contains a subset of the information contained in the Base Schema, enough to test our system. We expect this structure eventually to be expanded to contain all the information in the final Base Schema. This structure can be made arbitrarily complex without affecting the other structures within the data transfer schema, particularly the Tree and Node structure defined below.



A container type called Node provides the structure for representing the relationships between Taxon elements. A node can contain multiple Taxon objects, differentiated by a tag. This may occur when a single concept has different names. For example the family-rank taxon for legumes is commonly known as both Fabaceae and Leguminosae. A single Node containing two Taxon references can represent this.

As well as the taxon represented by this node, the node may also contain information about



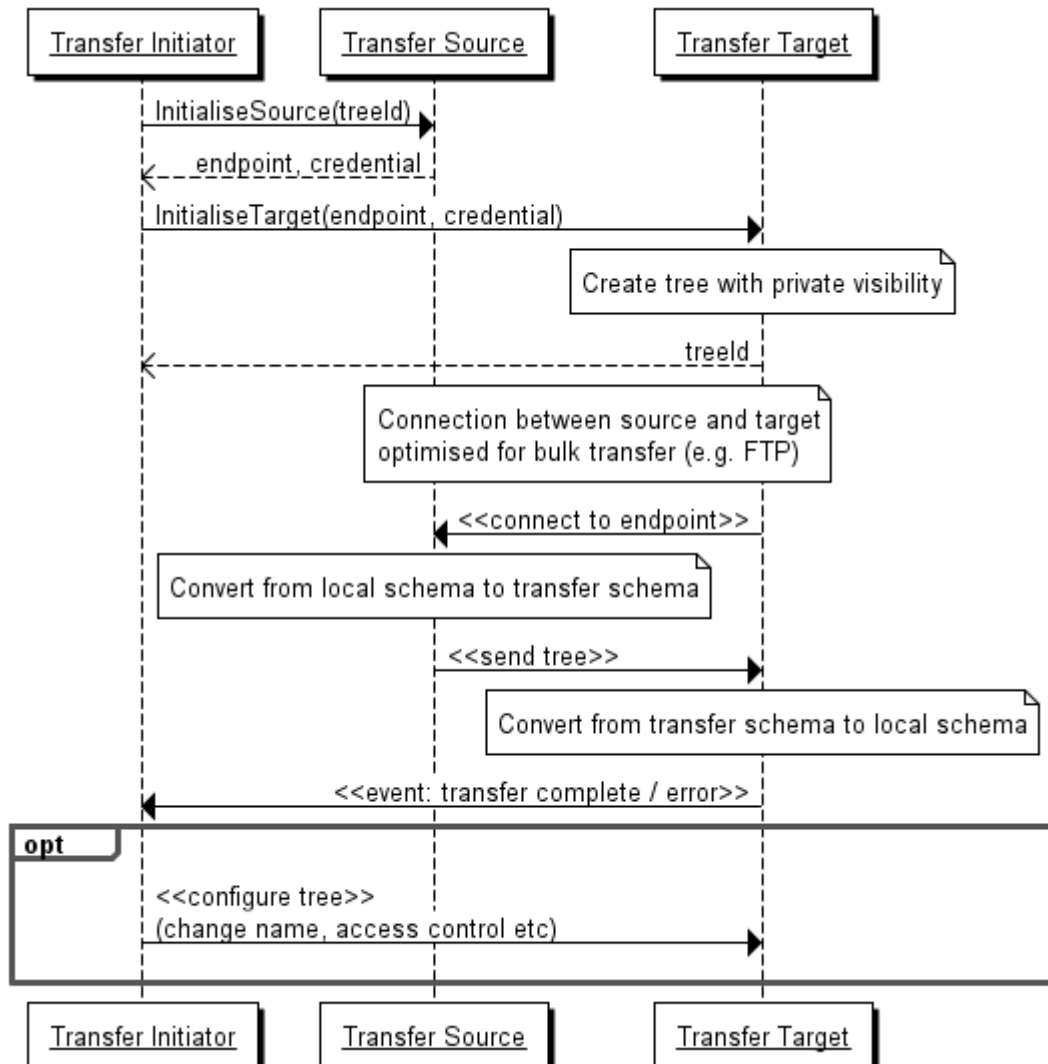
the parent and children of the taxon. If present, these elements contain references to the parent and child nodes, which can be retrieved through further operations.

A Tree object contains an identifying name for a taxonomic hierarchy, and a list of root nodes in the tree.



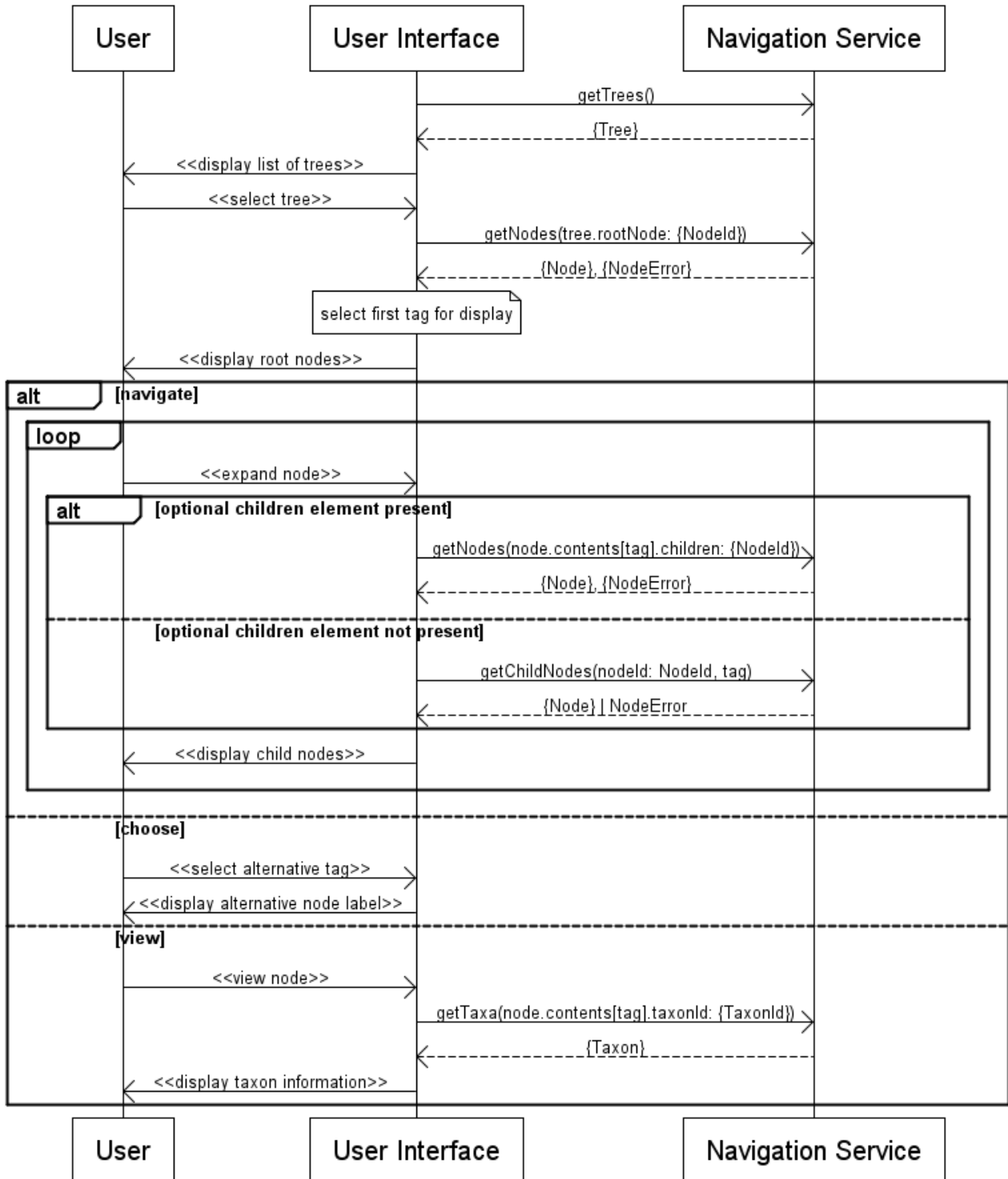
## 4.1. Service interfaces

### 4.1.1. UML sequence diagram: Transfer



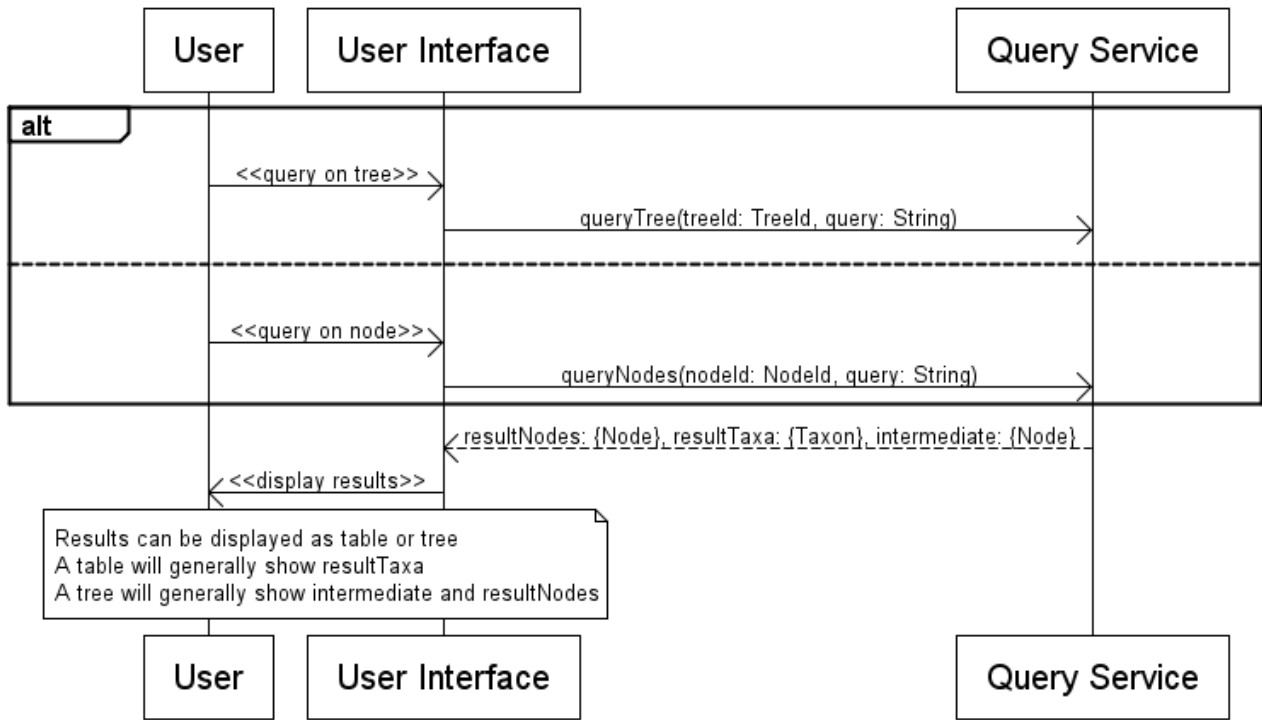
### 4.1.2. Tree Navigation Service

User Stories: 110, 122 (where intermediate non-primary ranks may be selected using a tag)



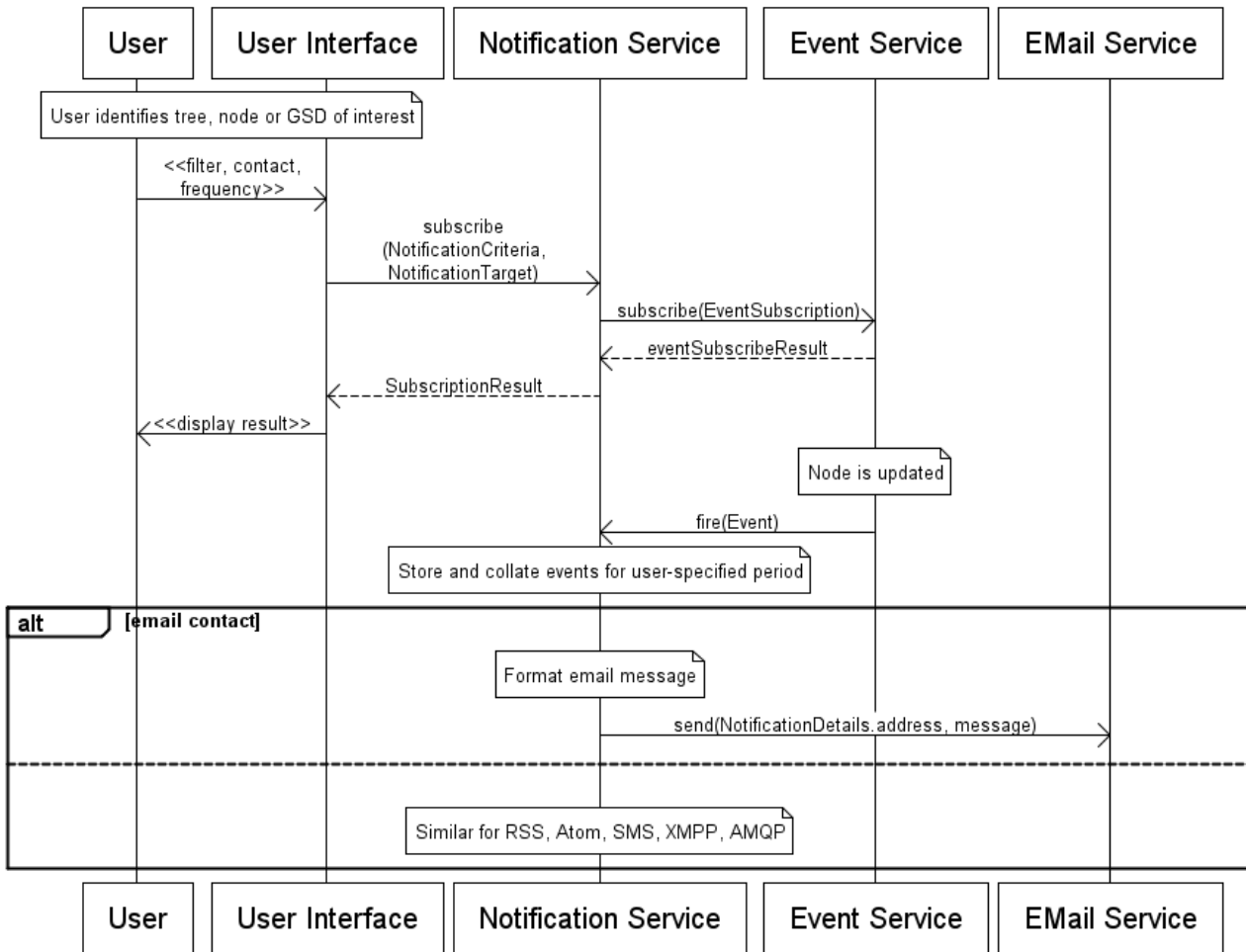
### 4.1.3. Tree Query Service

Use cases: 100, 101, 111, 120, 130, 131, 132, 140, 143, 145



### 4.1.4. Notification

Use cases: 190, 191



## 5. Appendix I: Phases of development

The priorities for the components are based on the following proposed scenarios for the four phases of development of our prototypes for the new architecture. (*We refer to “phases” of system development, to be distinguished from the harvesting, assembly, release and customisation “stages” of the data work-flow.*)

The following four phases are anticipated:

- Proof of Concept (demonstrating the feasibility of the proposed architecture)
- Test Implementation (demonstrating the architecture in use in realistic scenarios)
- Enhanced Prototype (including prototypes of all essential components needed for a “working” system)
- Non-Core (post handover) activities (extensions to support additional behaviour which, though important, is not absolutely essential to the basic operation of the system)

### 5.1. Proof of concept (PC)

This will demonstrate the three key stages of Harvesting, Assembly and Publishing. However, it will not be a fully “joined up” system - some steps will need to be manually activated, which in later phases would be automated.

- The data sources to be harvested would include both GSDs already wrapped and data files already exported from GSDs' off-line databases (perhaps one legacy Spice wrapper (ILDIS, also Hierarchy), one offline data file obtained from Luvie and Yuri, and one source with update events).
- The HDE (Hierarchical Data Editor) will demonstrate the feasibility of assembling the Catalogue from these pre-existing data sources.
- A product will be published (on a server) automatically as the Assembly Database is updated. This product could be the Catalogue of Life, with the advantage that the WP6 GUI newly modified for the Base Schema could then be used. (Alternatively it might not be a full copy of the CoL; it could be some summary or status report, perhaps including the number of species in the major taxa.)

The status of at least one of the data sources and of the Assembly Database would be monitored and cause event messages to be issued.

The demonstration and operation of the Proof of Concept will allow us to

- determine whether we agree that the tools are adequate and appropriate, and
- flesh out the components with an eye to these tools, especially relating to the use of workflow and rules.

In particular, we will be able to

- demonstrate the applicability of Apache Tuscany, JBoss Drools and current HDE

(Hierarchical Data Editor) development, and

- research and test preliminary plans for the event-handling and rule-based business logic and work flow subsystems.

## 5.2. Test implementation (TI)

This phase will

- add proper support for real data sources (at least three were promised)
- support production and publishing of the Annual and Dynamic Checklists, thus replicating the current system's functionality
- be a more robust fully "joined up" system

For example, adding a species to one of the data sources would result in the issuing of an event message. This would trigger harvesting, updating of the Assembly Database, and warning the expert editor that a change had occurred. When the expert acknowledges the change, the products are updated. When the product server fails (or is "accidentally" switched off for test purposes), another event message is generated. The events triggered by the Assembly Database update and the loss of the server result in SMS messages being sent to the demonstrator's mobile phone.

## 5.3. Enhanced prototype (EP)

This phase will

- add support for multiple hubs, including metadata to keep track of them and their data sets, their replication and mirroring, and user interface enhancements such as the choice of regional hub to view and the use of cross-maps to navigate between alternative trees – tools to support these activities may be implemented by work package 4, 6, or 7, or in the i4Life project (month 12, = month 30 of 4D4Life)
- introduce a basic rule editor
- add more data providers for testing from the GSD pilot projects

## 5.4. Later "non-core" additions (NC)

Certain features such as user authentication, annotation and feedback, while desirable, are not critical to the operation of a system with the desired core functionality. Their implementation will therefore be completed after the core functionality has been delivered in the Enhanced Prototype.