Project no. 034595

SELF
Science, Education and Learning in Freedom

Instrument: SSA - Specific Support Action
Thematic Priority: IST-2005-2.5.5 – Software and Services

Deliverable D5
SELF Platform Development:
System design and representation and beta release

Period covered:
From July 1st, 2006 to 31th May 2007

Date of first submission: May 31st 2007
Date of revision: September 25th 2008

Start date of project:
July 1st, 2006

Duration:
2 years

Organisation name of lead contractor for this Deliverable
Homi Bhabha Center for Science Education
Nagarjuna G.

Version 0.7

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Acknowledgements

This document has been collaboratively written by the SELF Project team. The Team has held many discussions on various alternatives of shaping the Platform, and implementation details. The team mainly consisted of: Federico Heinz, Meena Kharatmal, Dragoslava Pefeva, David Jacovkis, Nagarjuna G., Prasanta Barua, Bipin Apandkar, Debarshi Ray, Rakesh Pandit, Wouter Tebbens, and David Megias. All participants are gratefully thanked for their valuable inputs, in particular we thank Vedran Vucic and Machtelt Garrels for their active participation. A student of Symbiosis Institute of Computer Studies & Research, Pune, who worked with Nagarjuna as an intern during the last winter, Ms. Divya, who did the import and export module of SCORM into GNOWSYS knowledge base, must be made a special mention. She painstakingly studied the SCORM specification and did a pilot version of the scripts, which are being updated for the SELF project.

Document History

This document has been presented to the European Commission on April 29th 2008. As the development of the platform has progresses, several bugs have been eliminated, etc. The main results of the Platform are represented in the updated version of this document.

Table of change history

Date	Version	Comments
31 May 2007	0.6	First version sent to the EC for review
25 September 2008	0.7	Updated version

1 Introduction

One of the most important functions of the SELF Platform is to facilitate collaborative course development in an online environment. It is an author-centric platform for developing learner-centric content. While translating the high level specifications[1] of the platform into an implementable design, one of the main factors kept in mind is to keep the authors in the centre of the platform and to think around them. The SELF Platform is like a stage, on which the main actors are authors, and they together collaboratively create courses.

The SELF Platform's development was concieved at a time when so much of experience was already gained within the free software community in building such web based applications. The existing free e-learning applications, Moodle and Atutor are centered around the student, even though theyhave an authoring system in place. Keeping this experience in mind, if there is any reason to develop another application, instead of using them directly, it is in order to provide some features that the authors need. Free e-learning environments have already matured as delivery applications, and are increasingly being used around the world. They are, in a technical sense, good free run time environments (RTE) for LOM, SCORM and IMS compliant, in various degrees, electronic course materials. The focus of the SELF project is not a RTE, but an easy to use, community based portal for harvesting free content from all possible sources, and collaboratively author and develop the existing content, and become a resource base for course material. Although our initial focus is to develop content for Free Software and Open Standards, the SELF Platform is independent of content, and therefore can be used for any subject.

Having made the implementation focus clear, the next step is to learn from the existing state-of-the-art. Among the most successful collaborative authoring experiments in the history of Internet is obviously the Wikipedia. To risk a parallel, the SELF Platform is to FreeCourseWare, as Wikimedia is to Wikipedia. As platform to share knowledge, and collaboratively author content, the wikipedia project demonstrated to the world not only how technology can be shaped together, but also how technology can be shaped to support the culture of sharing. Simplicity of authoring, ability to track each individual change, negotiation among the authors, made Wikipedia a very successful project. While making the SELF Platform, we keep this goal in mind: shape the technology to support the collaboration among authors, teachers and students.

Other examples that come to mind are Gforge[2], free software similar to the software project development of sourceforge[3]. Gforge is used, for example, by the GNU project at Savannah[4] - the free software repository. Such projects are very good examples of supporting collaborative software development, integrating the entire lifecyle of an application. We intend to do something similar for managing the course authoring projects.

We kept these major legacy application design in mind while implementing the Platform.

In this document we will describe the main architecture of the SELF Platform and the implementation of the SELF Platform Definition (the high-level specs). The current release of the system can be found as a registered project in Savannah[5]. An online demonstration[6] of this release is set up and will be updated as soon as newer releases come out.

1 See the SELF Platform Definition as submitted to the European Commission for review and published on SELF Project: http://selfproject.eu/en/system/files/SELF_Platform_Definition_20070213.odt
2 http://gforge.org/
3 http://sourceforge.net/
4 http://savannah.gnu.org/
5 The SELF Platform is accepted as a non-GNU project on Savannah and can be found here: http://savannah.nongnu.org/projects/self-platform/
6 Demonstration version of the SELF Platform: http://dev.selfplatform.eu.

2 Software Selection for the Platform

The SELF Platform has the following software requirements:

1. A Secure Web Server
2. A Web Application Framework
3. A Database
4. A Content Management System
5. A Version Control System
6. A search engine
7. An agent-oriented and semantic-web enabled platform
8. An Authoring System for creating packages that work in a LMS (Learning Management System)

It is very unlikely that we can find one single application among the free software that is already available, which will perform all the above mentioned tasks. However, there do exist application development frameworks that support extensions to existing products so as to obtain all the features easily packaged into one. Keeping this in mind we selected ZOPE (see the following section for details) to be the main web publishing platform. This choice, among other reasons, is also based on the development team's knowledge and experience.

2.1 ZOPE

ZOPE (Zee Object Publishing Environment) provides a mechanism for publishing objects (instances of Classes usually written in Python) as web services. It is widely used all over the world and is actively developed and maintained. ZOPE provides the following features:

1. web based management
2. an object oriented database
3. all major web service protocols (HTTP, FTP, WebDav, XMLRPC, SOAP etc.)
4. inbuilt HTTP Server (can be integrated with Apache or other HTTP servers)
5. powerful templates for dynamic content (ZPT, DTML)
6. Connectors to Relational Database
7. Powerful Content Management Framework

By inheriting the existing features and products of ZOPE, extending some other applications, it is indeed possible to develop the SELF Platform with all the major features mentioned above as a product of ZOPE. Thus in a sense, several of the required features come to us almost ready to use on top of the ZOPE Platform.

ZOPE's web service infrastructure for publishing the knowledge base generated by the SELF Platform is very useful in implementing the distributed knowledge base (see the section on GNOWSYS). Every published object of the knowledge base is provided by a URL. This helps in using an object independent of the others from any where on the Internet.

2.2 Plone

Plone is arguably the most popular product of ZOPE, which is widely used all over the world for content management[7]. It is an extension of the CMF (Content Management Framework) offered by ZOPE. We use the powerful Archetypes tool of Plone to define two main content objects in the SELF Platform (learning object type and its instnce learning object) for managing the content.

7 http://plone.org/about/plone

Another feature that we will use extensively from Plone is the integrated CSS rendering.
SELF Platform therefore depends on the availability of Plone, and in turn on ZOPE.

3 GNOWSYS

GNOWSYS (Gnowledge Networking and Organizing SYStem) is a generic specification for a knowlede representation scheme. GNOWSYS, being generic, can comply with multiple knowledge representation standards such as OWL, XTM, CL, etc. The generality of GNOWSYS enables easy data exchange from one standard model to another. The current specification is implemented as a product of ZOPE (Zee Object Publishing Environment). GNOWSYS is free software and an official GNU project[8]. Its development is currently supported by the Homi Bhabha Centre for Science Education, TIFR.

Since GNOWSYS will keep the entire storage (both data and metadata) of the content , and is not widely used application, we will give the details of how the network storage is managed below.

3.1 Technical specification of GNOWSYS

The GNOWSYS specification[9] suggests to have 20 storage classes (tables) of which 19 of them provide core knowledge representation scheme. But as a distributed networking database, all the 19 core classes are specified as nodes of the networking model. For the implementation of the SELF Platform, we use only ten of the regular classes, since the data model of the SELF Platform does not need the procedural representation and process modelling. These additional classes can be used optionally if necessary, say if SELF Platform is extended in future to include the Run Time Environment, so that it supports delivery of the courses to students. In this document we will not make any further mention of the classes other than the ten required classes.

The knowledge base can be implemented in any database, for GNOWSYS is an implementation-independent specification. For the SELF Platform we will use Zope for Phase I. And as the specification of the Platform freezes, the storage will be made in PostgreSQL, an object-relational database, which will also ensure stability and scalability of large size knowledge bases.

The core classes of GNOWSYS are organized in three layers as shown in the Illustration below.

8 http://savannah.gnu.org/projects/gnowsys/
9 http://www.gnu.org/software/gnowsys/GNOWSYS_files/cpv2/index.html

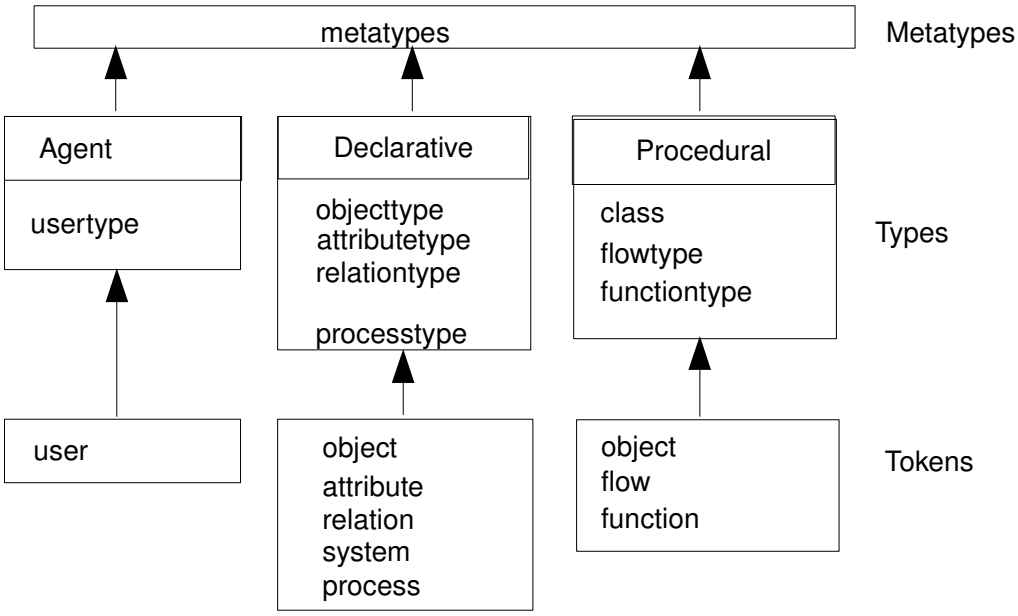
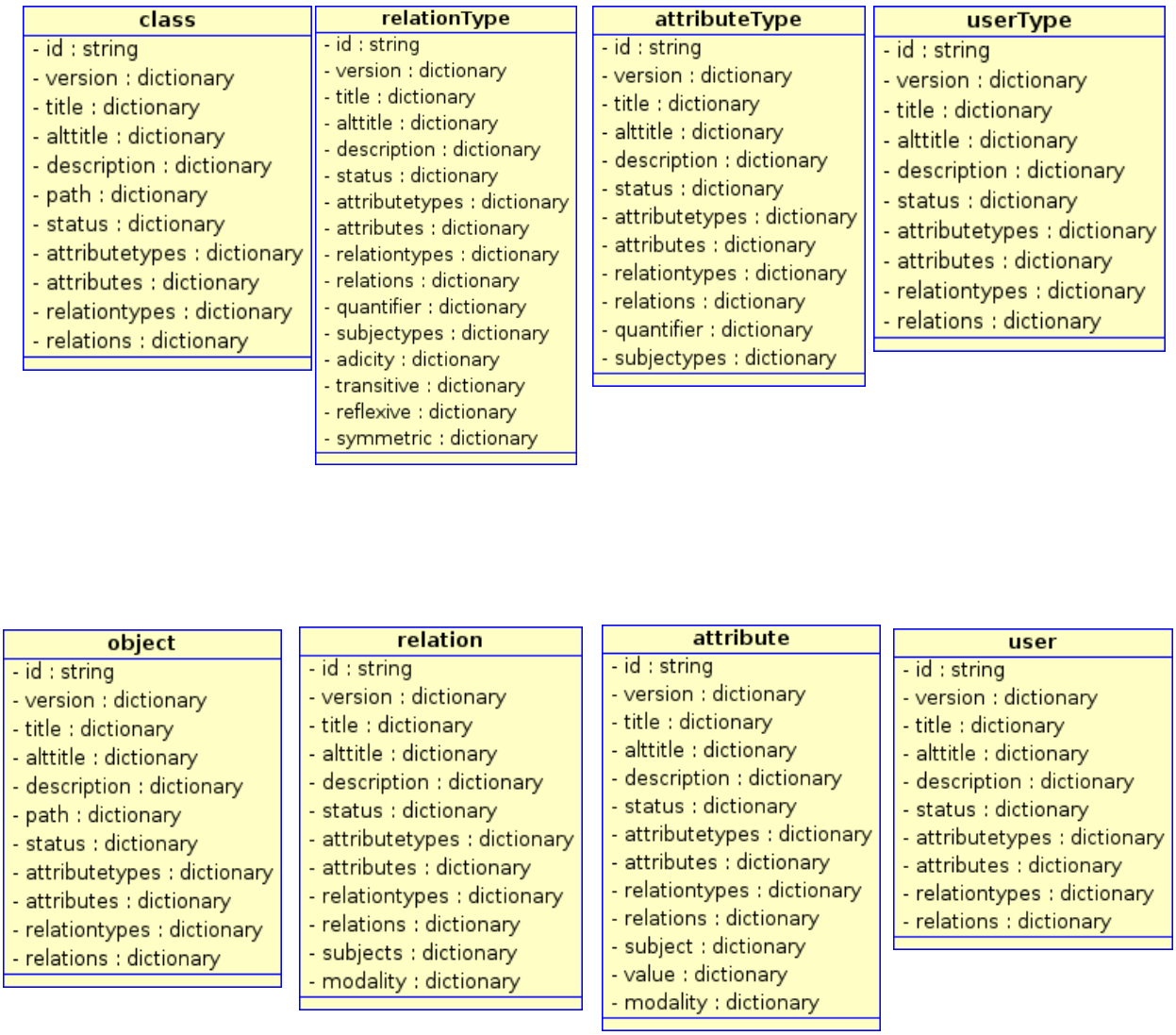


Illustration 1: The twenty classes (node types) of GNOWSYS which are used to represent knowledge. Instances of each class are the nodes which get linked to each other to form a network. These are grouped, as shown in the illustration as tokens, types and metatypes. Based on what each of them stand for, they are further classified as agentive, declarative and procedural.

The nodes are networked among three kinds of nodes: declarative, procedural and agent. Agent objects are further divided into User Type and User. User Type is used to categorize its User instances, which act as proxy objects for real users. Declarative objects are frames that store the metadata of the objects. Procedural objects are proxy objects which when invoked execute the specified procedures.

The following figures describe the structure of the classes (Node Types). Using this specification, we can implement them as object oriented classes, or since declarative nodes do not have any methods associated with them, they can be implemented as tables in a relational database.



llustration 2: Most widely used GNOWSYS Node Types for the SELF Platform. Another upper ontology Node Type, called Metatype, is not included in the illustration, for its structure is similar to Class. All the above types can be made instances of the Metatype. The bottom row shows the instances of the Node Types. Each Node acts as a frame keeping its neighborhood information to generate the organization and networking of the distributed elements of a knowledge base. System and System type will be used for organizers, sequencing, defining adaptive navigation, rollup conditions, sequencing actions etc. These two node types will have an additional field called 'structure', in addition to what the node types class and object have.

3.2 Change Management in GNOWSYS

GNOWSYS is made for collaborative web based authoring of knowledge applying with semantics. Unlike collaborative software development, where hackers check out a development version from change management systems like cvs, svn, GNU arch, and make changes in a standalone editor, and frequently makes committs once they have reached a landmark, in this situation, the version control system is required to have only the versions of various committs that each user makes. However, in the case of a collaborative web based authoring, the authors are generally using a client application (normally a browser), and each modification made can also be part of the history. This is a special requirement of the version management of web applications.

The knowledge base of GNOWSYS is complex, but follows a specified (predefined) structure. One aspect of the specification is that each node has a fixed number of fields in any given release of GNOWSYS. This feature enables us to define a version management scheme that manages to keep every change economically. The structured granularity will also be very useful in mirroring different instances of the knowledge base.

Each node in the network has a neighbourhood of nodes, and the specification clearly defines what kind of node will have what kind of objects in the neighbourhood. For example, an object node will have the following fixed kind of objects in the neighbourhood: relations, relation types, attributes, and attribute types. Maintaining versions of each node, and the metadata associated with each is a challenging task. The relevance of this feature is very high if the application is to be used as a community portal with collaborative work cycles. Since GNOWSYS is particularly meant for sharing, collecting and distributing free knowledge, the following version management model emerged during its evolution.

1. Each node will have a version field in the database.
2. The version is stored as a dictionary, key and value pair. After migrating the storage from ZODB to Postgresql dictionaries will be translated into separate value tables and field tables.
3. The key of the version includes two fields separated by a dot (“.”).
    1. the first field will increment whenever a commit is made by the user.
    2. the second field will increment whenever a change is made by the user. For example, after 3 commits and 8 changes the key of the version number will be 3.8.
4. The value of the version key will be a list with three items
    1. the first item of the list stores the id of the user who made the change
    2. the second item of the list is an ordered list of pairs, implemented as a tuple. For each metadata field of the object, a commit number followed by a period and a change number will be stored. For an object with six metadata fields the tuple will be as follows:

        ```
        (0.0,1.4,2.3,0.0,0.0,0.1)
        ```
    3. the third item of the list is a list of field positions where the change took place, indicated by field number. e.g: [1,2] suggesting that the second and third metadata fields are modified.
    4. Thus, the entire key value pair of the version can be entered in Python language as follows:

```
version = {3.8:['userid34', (0.0,1.4,2.3,0.0,0.0,0.1), [1,2] ]}
```

The complex version number is therefore represented by a special datatype for each nodetype of the network. Each object will have a unique id, and hence this id is never modified. For each change, the version number becomes a primary key for that state of the object. Thus every state has a unique identifier. Each changed version in the database field will be stored as a list of changes. Thus, given the key of a version number, we know which user made the change, what is the snapshot of the object (stored as the complex ordered tuple) at that version, and which fields of the database modified.

When we want to know what is the nature of the change and extent of the change, we can ascertain by computing the differences. We will also be in a position to know, what kind of changes a

particular user does. E.g,, a user may contribute more by making appropriate changes to the metadata, while another user is keenly making changes to the most commonly visible fields of a learning object, namely description and content.

To illustrate the above model, please see the following illustration and the explanation that follows.

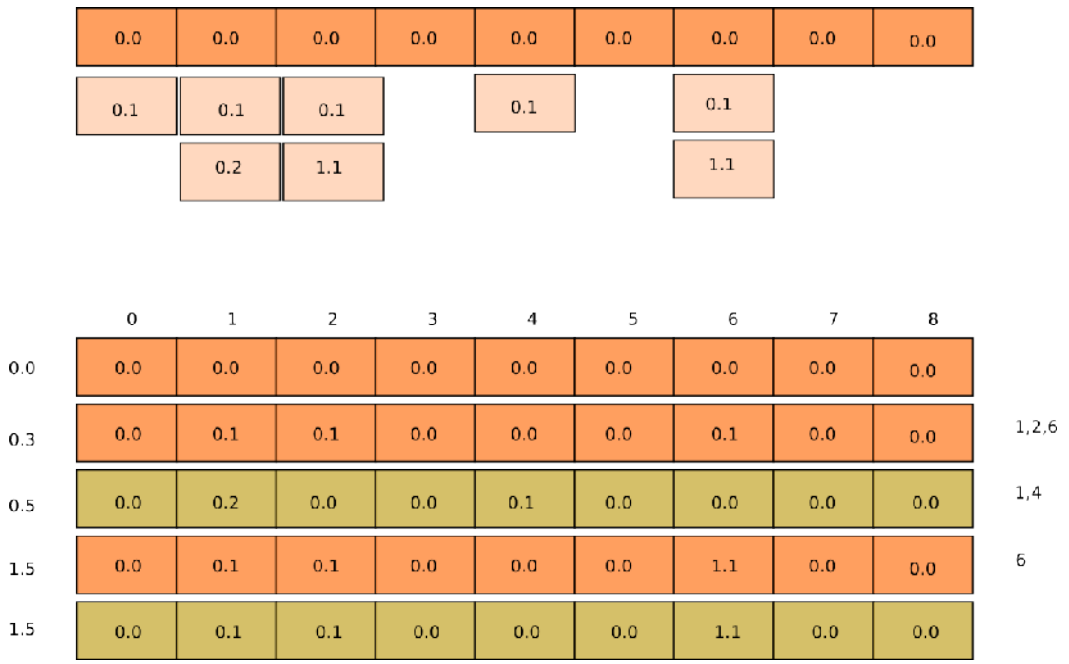


Illustration 3: Version Control Model in GNOWSYS. The top horizontal set of boxes represent the nine fields of an object and the version number corresponding to each at the time of creation. The boxes beneath in lighter color are the changes made to each field. The bottom part of the illustration displays the five snapshots of the object. Three of them are changed by the creator of the object, while two others in the alternate color are by another user.

The Illustration 3 is an example of an object's history with about five snapshots of the object. The object underwent one committ and five changes. In the second transaction the user changed the fields 1,2, 6 of the object, hence the change number then is 0.3. Later another user makes a change to the object at fields 1,4, making the change number 0.5. Since the first user did not make any commits after creating the object, the second user does not see 0.3, but only 0.0. Later the first user makes a commit at the field 6, to make the version 1.5. The second user now sees the changes committed by the first user. Each of the snapshots will be stored in the form of a list, in the list field of the object.

Keeping such detailed snapshots of each node of the network provides a unique mechanism to see the various changes. We can get detailed state changes of the object, and its history, and can retrieve given the change number the snapshot of the object at any time.

The third item in the value field of the version number keeps the field numbers modified. This information will come in very handy under three situations. One, for highlighting the part of the data modified. Two, when a version of an object should be deleted, it can be achieved by removing only those fields that are indexed in the third item. Third, if we wish to inspect each state of the object, during the course of its history, the values of the changed fields only need to highlighted, producing a visual graph displaying the modifications.

Another important consequence of this model of representing the changes is its economy of space required in keeping backups. If, for example, an 80MB document is modified, and the modification consists in merely a change in the title, keeping the backup and the recent version is very expensive. Since the knowledge in GNOWSYS is highly granular and all the granules are packed together as a

package, the entire package need not be taken a backup, but only the modified field.

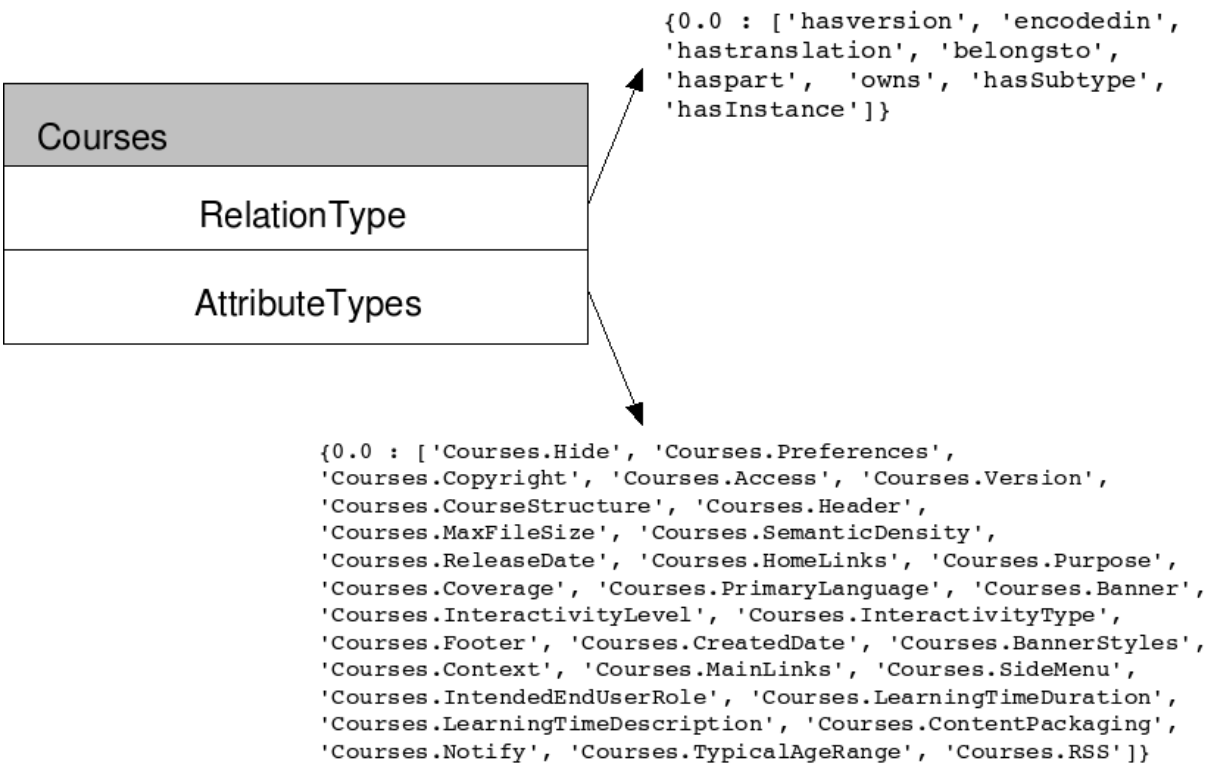


Illustration 5: The class node 'Courses' is made with, among other metadata, relation types and attribute types. The illustration shows how the data fields of relation type and attribute type are encoded as a dictionary data type.

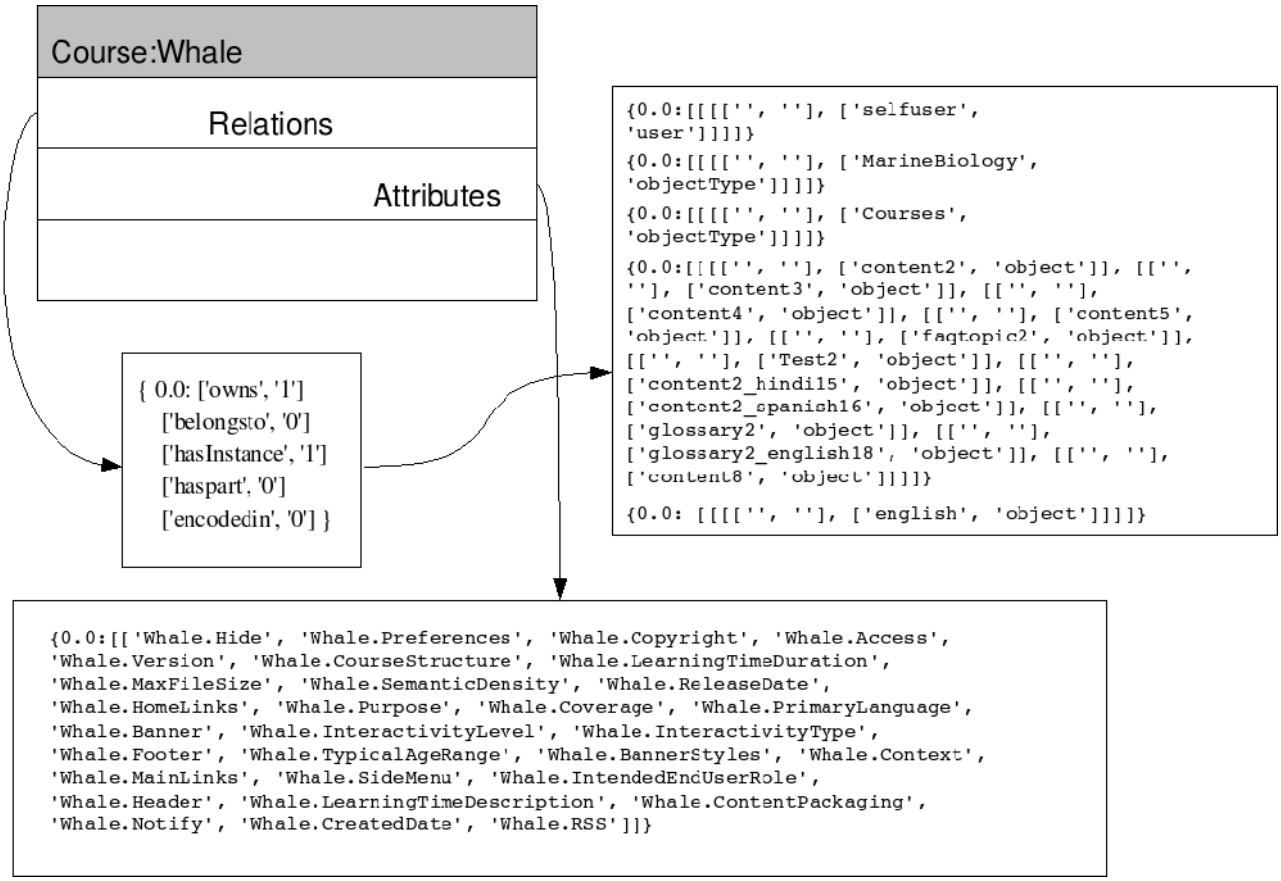


Illustration 6: The class node 'Courses' is made, among other metadata, with relation types and attribute types. The illustration shows how the data is stored in the knowledge base defining the schema of a Course.

3.3 GNOWSYS for the SELF Platform

In this section, we list the special features specified in the Platform Definition that need a distributed networking system like GNOWSYS.

1. Distributed Knowledge Base
2. Semantic Web Ready
3. Flexible, Arbitrary (User) Organization
4. Semantic Search
5. Contextual Menu
6. Reuse of a lessons, questions, FAQs, etc. from one course in another.
7. Collaborative web based authoring
8. Version control of both data and metadata.

4 Translating Platform Definition into Implementation Design

Based on the high level specification of the platform definition (D5)[10] we can draw a list of features and the required implementation design. The presentation below follows a style of formatting to make the purpose of each section clearer. Each section below describes a feature in a summary form, followed by a paragraph describing the design and implementation, which is displayed in Type Writer font.

4.1 Creation of the Initial Knowledge Base and Basic Data Model

As defined above in the section, the knowledge base of SELF Platform will be implemented using the nine node types of GNOWSYS. At the time when the SELF Platform is initialized, the data model of the application (schema) will be constructed with almost no data. This data model in a sense constitutes the basic ontology of a teaching and learning environment. To ensure that the knowledge base of the Platform can be ported to other elearning environments, we followed, as specified in the earlier documents as elaborated by the Learning Standards Expert Group[11], we follow Dublin Core Metadata for content objects, LOM for learning objects, and SCORM for the interrelationships between the learning objects. The following table specifies the objects to be created when an instance of a Platform is made.

Users initiate the creation of several elements into the knowledge base through the Platform's user interface: Course Categories, Lessons, various kinds of Part materials, comments, relations between the learning objects, translation relations, language encoding, creating an account for themselves, changing passwords etc. We are not giving an exhaustive list here, but more detailed descriptions of what happens to the knowledge base when each action is taken are elaborated while describing the implementation details.

10 See http://selfproject.eu/en/system/files/SELF_Platform_Definition_20070213.odt

11 See the Learning Standards Expert Group (LSEG) and the selected standards for use in SELF: http://selfproject.eu/en/LSEG and http://selfproject.eu/en/system/files/ir3.1_0.pdf

Node Types	*Objects in the Knowledge Base*
Metatypes	Course Type, Lesson Type, Question Type, Test Type, FAQ Type, Glossary Type, Part Material Type, News Type, Bibliography Type, Role Type, Hyperlink Type, Language Type, Comments Type, BookShelf Type, Collaborator Type, Message Type
Classes	Lessons, Questions, Tests, FAQs, Comments, UserType, References, Links, BookShelf, Collaborator, SCO, Course Material, Course, Course Enrollment, Part Material, Content, FAQ Topics, FAQ Entries, Glossary, Link Categories, Links, Tests, Test Questions, Test Results, Test Answers, Test Groups, Test Questions Associations, Objective Questions, Multiple Choice Questions, True False Questions, Descriptive Questions, Languages, Language Text, Feeds, Poll Members, Member Track, Forums, Forum Subscription, Forum Thread, News, Related Content, Themes, Resource Categories, Resource Links, Backups, Messages, Polls, Course Stats, Handbook Node,
Objects	No factory objects created
Relation Types	part of, sub type of, author of, translation of, prerequisite of, instance of, has License, shares, question of, FAQ of, lesson of, test of, collaborator of, message to, message for, message cc,
Relations	Several relations are automatically created, this list is not exhaustive.: Author is an instance of Role Type, Student is an instance of Role Type; Urdu is an instance of Right-to-Left Language; Book is an instance of Bibliography Type; Events is an instance of News Type; MCType question is an instance of Question Type; Activitities, Narratives, Experiments, Illustrations, etc. are instances of Lesson Types, and so on.
Attribute Types	A Large Pool of reusable Attribute Types are created: Date, File Size, System File Name, File Name, Contents, Ordering, Last Modified, Revision, Formatting, Release Date, Keywords, Content Path, Text, Inherit Release Date, Interactivity Type, Interactivity Level, Semantic Density, Learning Time Duration, Learning Time Description, Content Packaging, Access, Created Date, Notify, Max Quota, Max File Size, Hide, Preferences, Header, Footer, Copyright, Banner Text, Banner Styles, Primary Language, RSS, Home Links, Main Links, Side Menu, Release Date, Banner, Interactivity Type, Inter activity Level, Semantic Density, Intended End User Role, Context, Typical Age Range, Learning Time Duration, Learning Time Description, cost, Coverage, Version, Purpose, Course Structure, Name, Revised Date, Approved, Question, Answer, Language Code, CharSet, Direction, Reg Exp, Native Name, English Name, Owner Type, Name, Url, Link Name, Approved, Submit Name, Submit Email, Submit Date, Hits, Website, first Name, middle Name, last Name, DPB, Gender, Postal, Province, Preferences, Creation Date, Language, Inbox Notify, Private Email, street Address, phone, email, city, country, Type, Feedback, question, Choice0, Choice1, Choice2, Choice3, Choice4, Choice5, Choice6, Choice7, Choice8, Choice9, Answer0, Answer1, Answer2, Answer3, Answer4, Answer5, Answer6, Answer7, Answer8, Answer9, Properties, Weight, Ordering, Format, Start Date, End Date, Randomize Order, Num Questions, Instructions, Result Release, Num Takes, Anonymous, Out Of, Random, Difficulty, Word Definition, Option1, Option2, Option3, Option4, Option5, Answer

Node Types	*Objects in the Knowledge Base*
Attributes	No factory attributes created
User Type	Manager, Author, Editor, Student, Anonymous
User	Only the default user called 'admin' will be created.

Table 4.1: Node Types of GNOWSYS used in SELF Platform, and some of their instances made at the time of making an installation. These are usually called Factory Objects. They provide the schema (data model) of SELF Platform.

Node Types	*Attribute Types for Node Types*
Backups	Date, File Size, System File Name, File Name, Contents
Content	Ordering, Last Modified, Revision, Formatting, Release Date, Keywords, Content Path, Text, Inherit Release Date, Interactivity Type, Interactivity Level, Semantic Density, Learning Time Duration, Learning Time Description
Courses	Content Packaging, Access, Created Date, Notify, Max Quota, Max File Size, Hide, Preferences, Header, Footer, Copyright, Banner Text, Banner Styles, Primary Language, RSS, Home Links, Main Links, Side Menu, Release Date, Banner, Interactivity Type, Inter activity Level, Semantic Density, Intended End User Role, Context, Typical Age Range, Learning Time Duration, Learning Time Description, cost, Coverage, Version, Purpose, Course Structure
FAQ Topics	Name
FAQ Entries	Revised Date, Approved, Question, Answer
Language	Language Code, CharSet, Direction, Reg Exp, Native Name, English Name
Languages	Language Code, CharSet, Direction, Reg Exp, Native Name, English Name
Language Categories	Owner Type, Name
Links	Url, Link Name, Approved, Submit Name, Submit Email, Submit Date, Hits
Members	Website, first Name, middle Name, last Name, DPB, Gender, Postal, Province, Preferences, Creation Date, Language, Inbox Notify, Private Email, street Address, phone, email, city, country
Test Questions	Type, Feedback, question, Choice0, Choice1, Choice2, Choice3, Choice4, Choice5, Choice6, Choice7, Choice8, Choice9, Answer0, Answer1, Answer2, Answer3, Answer4, Answer5, Answer6, Answer7, Answer8, Answer9, Properties, Weight, Ordering
Tests	Format, Start Date, End Date, Randomize Order, Num Questions, Instructions, Result Release, Num Takes, Anonymous, Out Of, Random, Difficulty
Glossary	Word, Definition
Objective Questions	Option1, Option2, Option3, Option4, Answer
Multiple Choice Questions	Option1, Option2, Option3, Option4, Option5, Answer
True False Questions	Answer

Table 4.2: List of Attribute Types assigned for the Node Types of GNOWSYS used in SELF Platform.

Left Role	Relation Type	Right Role
Learning Object	is part of	Course Materials
Learning Object	is based on	Learning Standards
Learning Object	is written in	Language

Left Role	Relation Type	Right Role
Learning Object	is assigned by	Assigned Role
Learning Object	is contributed by	Contributor Role
Learning Object	is targeted for	User Role
Learning Object	is aimed at	Context
Learning Object	has requirements	Technical Requirements
Learning Object	has requirements	Other Platform Requirements
Learning Object	holds	Copyright
Learning Object	is released under	Licences
Learning Object	has references	Reference Materials
Learning Object	is a kind of	Learning Resource Type
Learning Object	is tagged for	Purpose
Learning Object	is classifed as	Taxon
Courses	has part	Course Material
Courses	belongs to	Course Category
SCO	has translation	SCO
Course Material	has references	Part Material
SCO	encoded in	Language
Content	has version	Content
Lesson	subtype of	Course
Author	author of	Lesson
Learning Object (spanish)	translation of	Learning Object (English)
Learning Object	prerequisite of	Course
Student	instance of	Role Type
Test Question	question of	Course / Lesson
Learning Object	FAQ of	Lesson
Test	test of	Lesson
user	collaborator of	Lesson
Learning Object	owned by	user
courses	announced as	news
courses	conducted by	members
courses	encoded in	language
content	generated by	member
members	broadcast	news
members	provides	course statistics

Table 4.3.: List of Relation Types assigned for the Node Types of GNOWSYS used in SELF Platform. Each Relation Types are associated to the Node Types as shown in Left Role and Right Role.

4.2 Authoring System

Since the Platform is meant for authors, teachers, learners, publishers, lecturers, professors, the authoring system is one of the core functions of the SELF Platform. Authors can do several actions within the Platform. Creation and validation of content are two most general categories of action. But, these two processes in a collaborative authoring system may have several other derivative actions such as commenting, discussing, rating, translating etc.

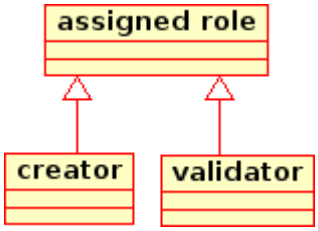


Illustration 7: Two main roles authors can be either of the two in an authoring system. The arrow indicates that the bottom class is an instance of the metatype <Role>..

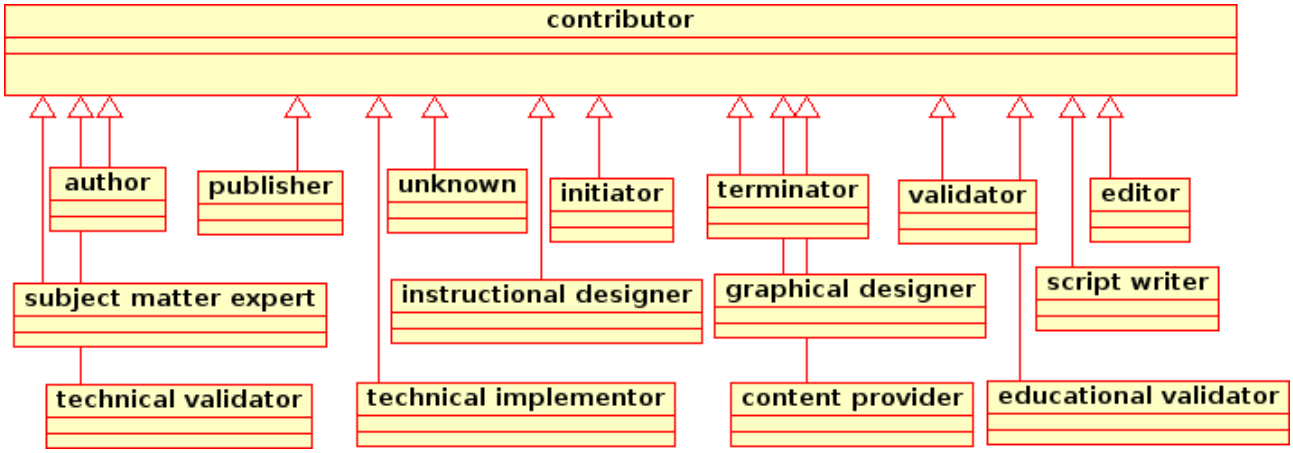


Illustration 8: Multiple Roles Contributors can play in the system. The arrow indicates that the bottom class is an instance of the metatype <contributor>.

4.2.1 Creating Content Items

The following procedure provides an example of how each of the items are created and networked with each other within the knowledge base.

1. course category: create a <class> node by collecting the <title> and <description>; create a <sub-type-of> relation if another course category is selected; with the <attribute node> <creation time>,

   create an <author-of> relation with all the objects and relations that get created during this transcation; set the <language encoding> relation based on the default user's preferences; set the <media-type> of the content; set the <version> to <key> 0.0, and <snapshot value> to each field as 0.0; set creation date-time; (The italicized portion describes the kind of actions to perform while creating any objects.)

2. courses: create an <object node> with an attribute type <structure>; create an <instance of> relation with the selected or context defined class node #Course; create a relation node <belongs to> with #courseCategory;

3. lessons: create an <object node> with a <relation node> <part of> set with one or more courses selected; with a <relation node> <instance of> set with <class node> lesson.

4. FAQ: create an object node with relation node <instance of> set with class node <FAQ> with the selected FAQ category class node; create a relation node <faq of> with selected object learning object/s.

5. glossary: create an object node with relation node <instance of> set with class node <glossary>; with selected glossary category class node; create a relation node <glossary of> with selected object learning object/s.
6. test: create an object node with relation node <instance of> set with class node <test>; with selected test category class node; create a relation node <test of> with selected object learning object/s.
7. question: create an object node with relation node <instance of> set with class node <question>; with selected question category class node; create a relation node <question of> with selected object learning object/s.
8. comments: create an object node with <instance of> relation with <comments>; create a relation <comment of> with the object selected or context defined in the content type object;

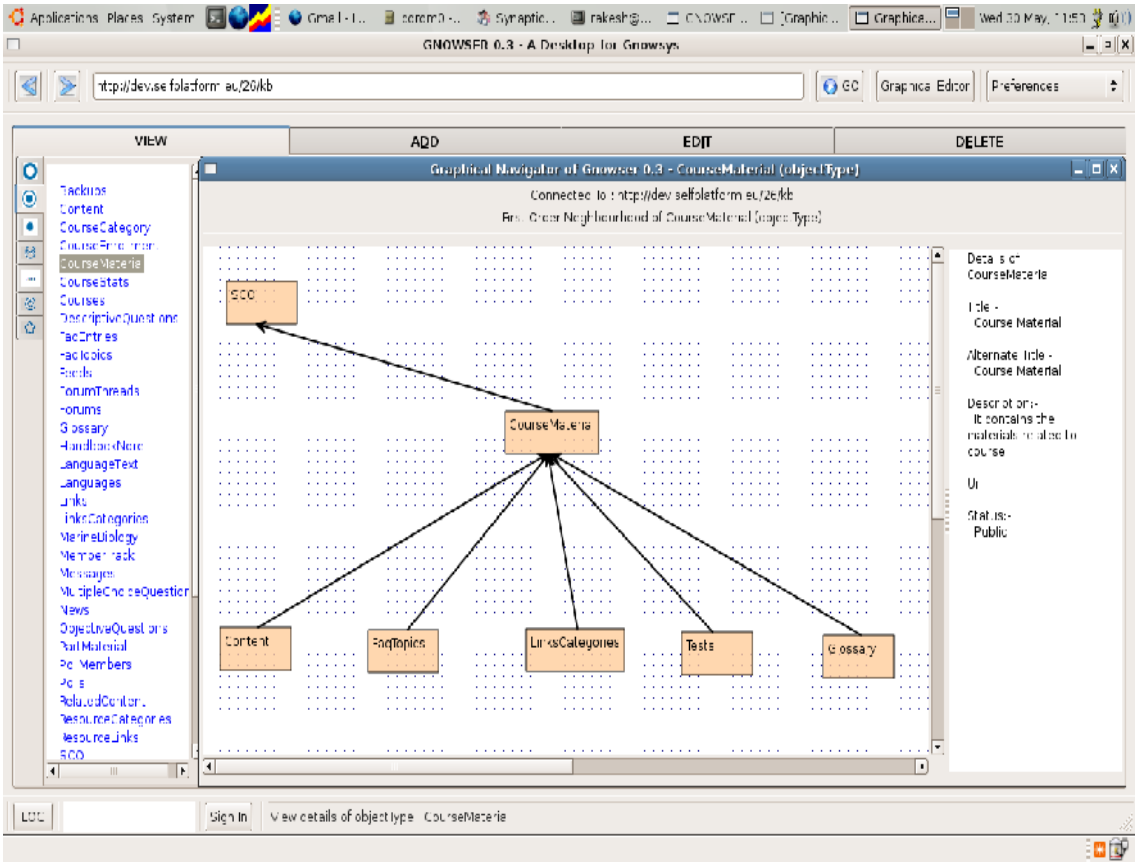


Illustration 9: Screenshot of the application Gnowser, which connects to the knowledge base of GNOWSYS, and displays graphically the network nighbourhood of the selected node. In this case it displays the the five different kinds of 'course material' as subtypes, and the class node 'course material' is a subtype of the class node 'SCO'. Gnowser is emerging as a gnowledge browser meant to navigate all the distributed knowledge bases in the cyber space, and network, manage, and organize the distributed resources.

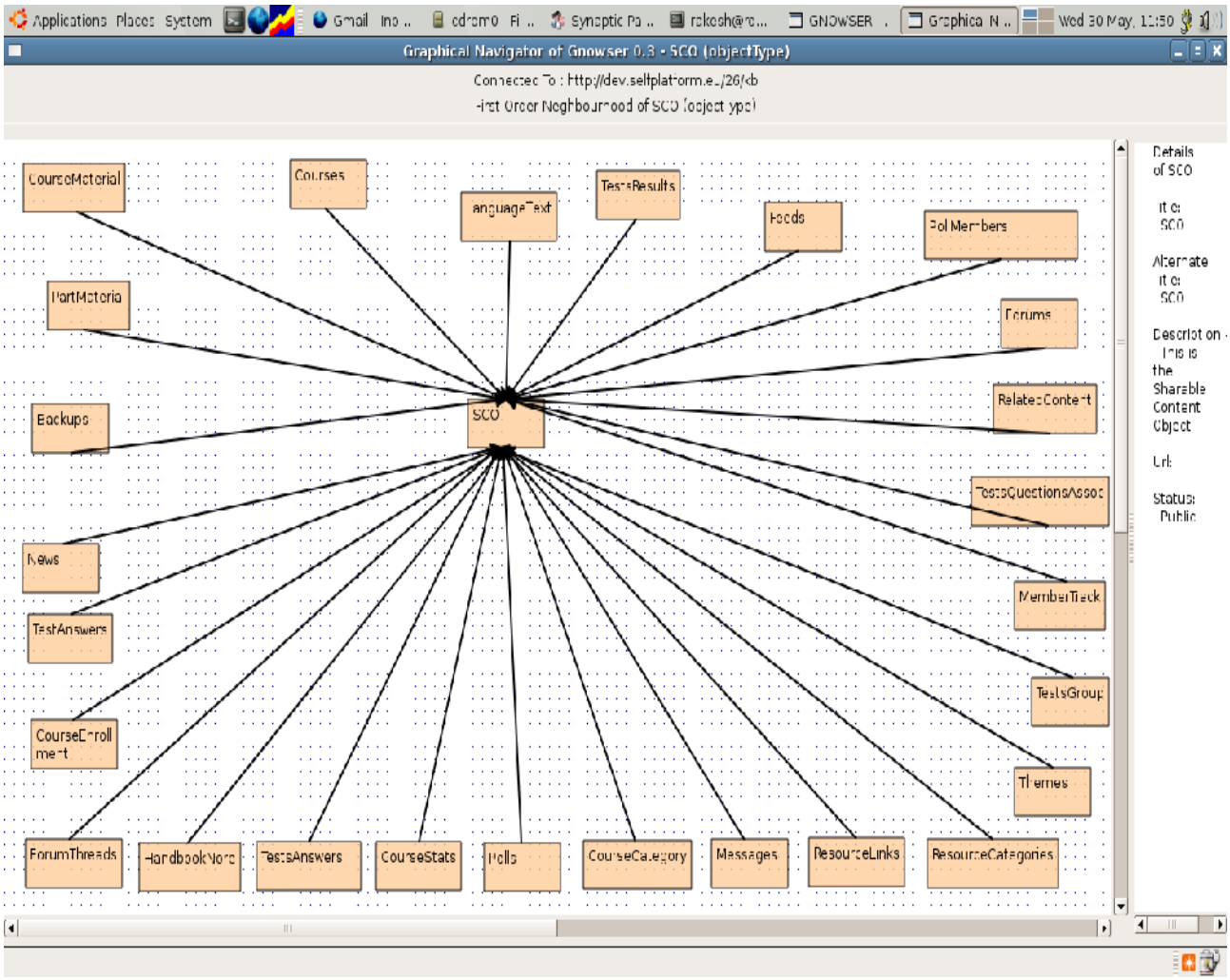


Illustration 10: Screenshot of Gnowser connected to http://dev.selfplatform.eu/26/kb/, displaying the network nighbourhood of the main class node SCO.

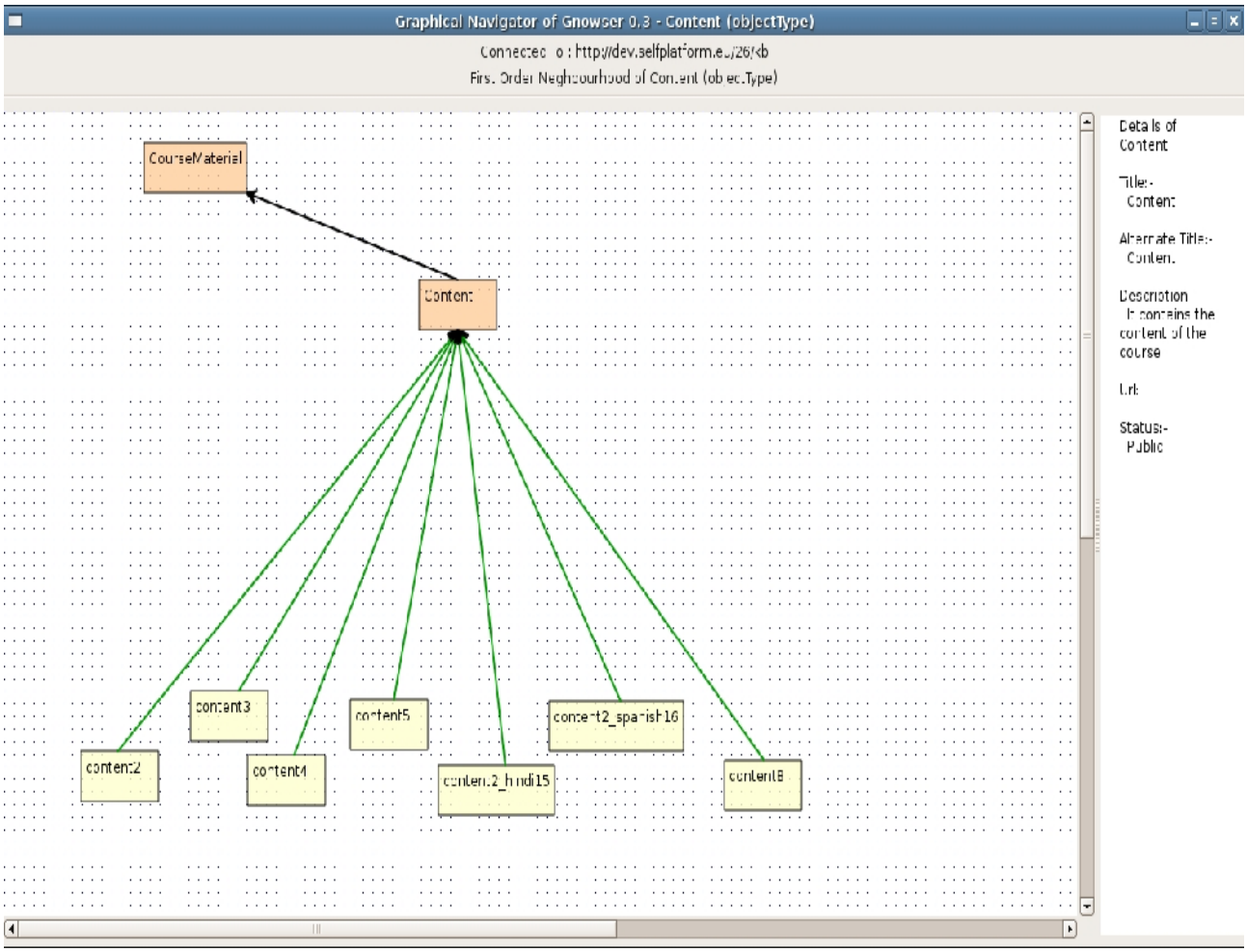


Illustration 11: Screenshot showing the seven instances of the class node 'content', which is a subtype of 'Course Material'.

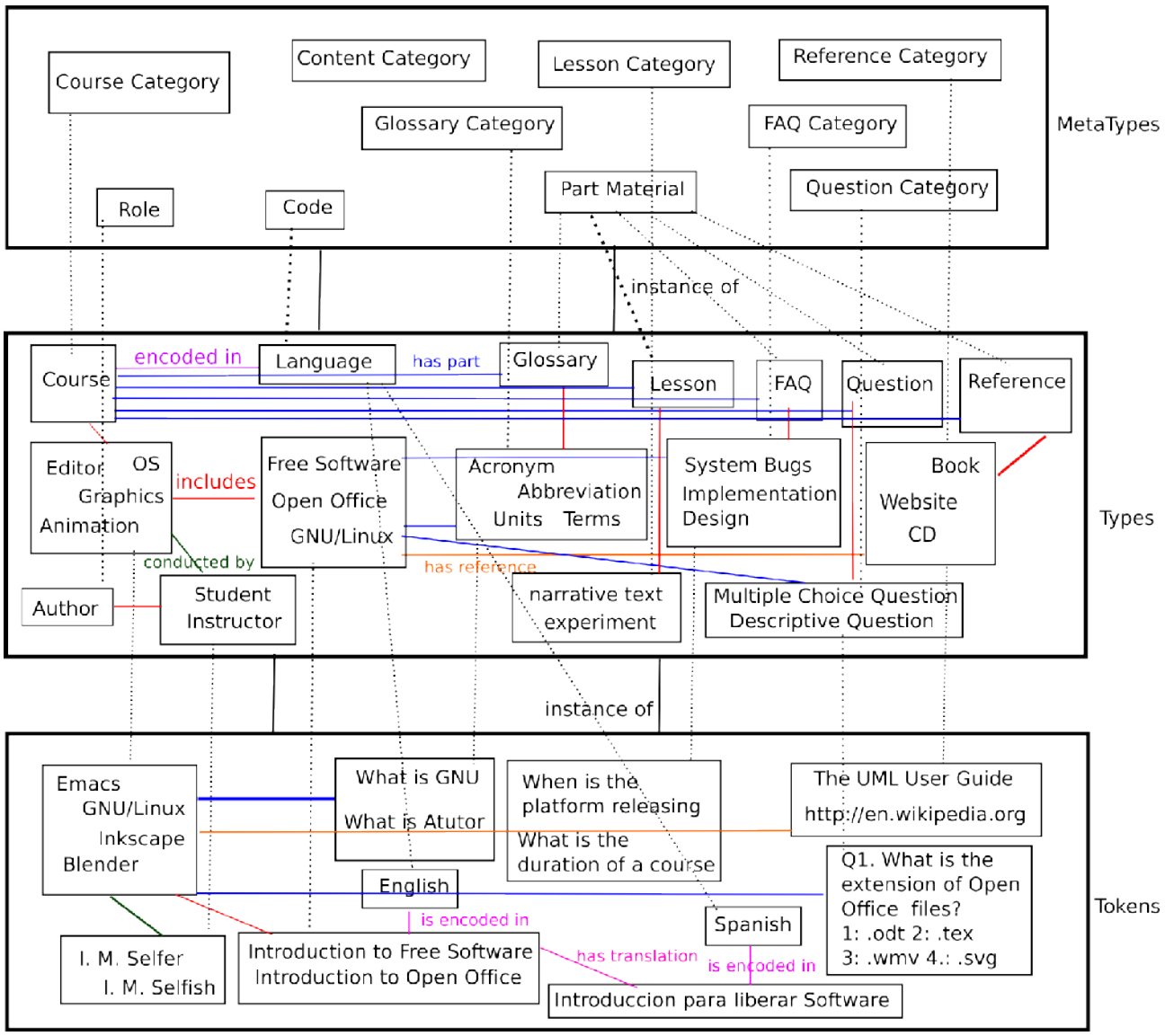


Illustration 12: The illustration depicts how the various kinds of objects in the knowledge base of the platform are grouped and related according to the GNOWSYS specification. The dotted lines represent 'instance of' relation. Metatypes help in organizing the various kinds of courses and their part material shown in the middle type layer. The bottom layer shows the actual data.

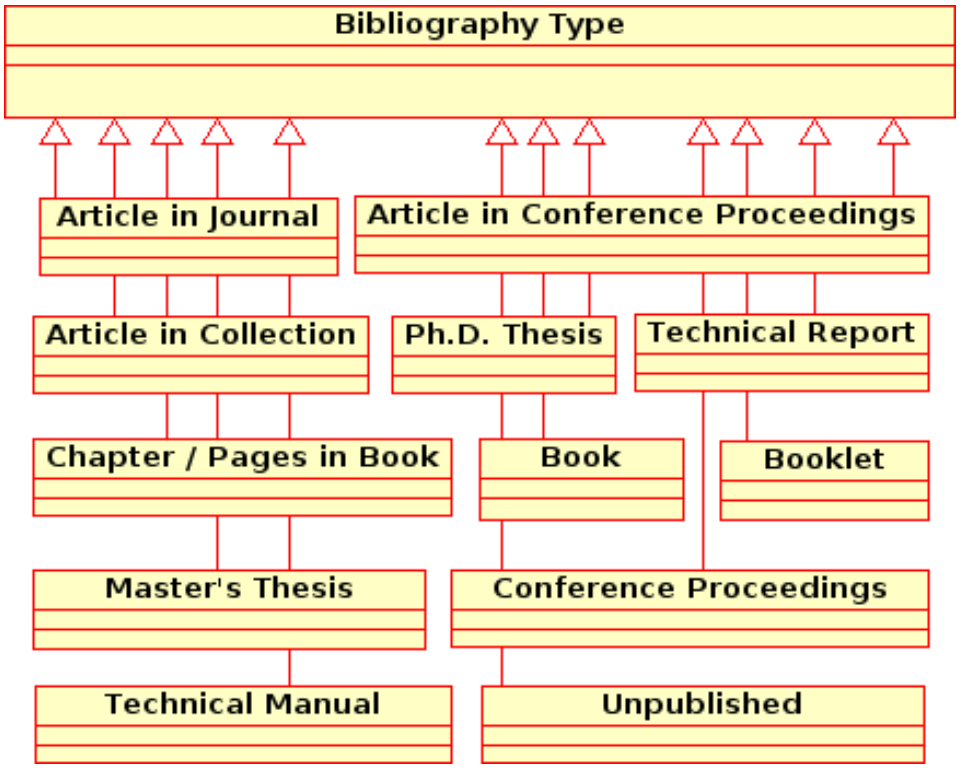


Illustration 13: The part material references can be further classified into various kinds of documents. The instances of these type of bibliography types will be networked with different learning objects by the relation node <has reference>. These instances can be reused across courses, that a book *Emacs Manual* may be referred in the course on *Programming Environments,* as well as *Editors*. can have a reference relation with the book. The arrow indicates that the bottom class is an instance of the metatype.

4.3 Shelf Management

A Shelf is to help the author to keep the selected objects from the knowledge base, for editing, composing a course, traslating, collaborating etc. SELF Platform proposes to have multiple shelfs. Users can add a Shelf, by giving it an arbitrary name (#bookShelf).

1. `Create a class node <bookShelf> with relation node <instance of> <ShelfCategory>;`
2. `Create an object node #bookShelf with relation node <instance of> with class node <bookShelf>; with relation node <has Shelf> with the user node #user;`
3. `Create a relation node <item in BookShelf> with subjects object node #learningObject and object node #bookShelf;`
4. Create a relation node <bookshelf of> with user and bookself

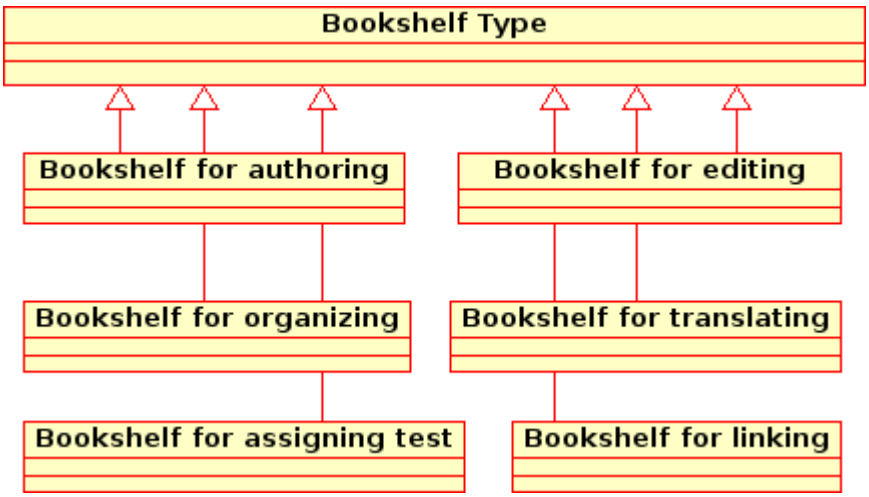


Illustration 14: Authors can add arbitrary bookshelfs, but desirably classified into one of the above kinds. The arrow indicates that the bottom class is an instance of the metatype <BookShelf>.

4.4 Harvesting

Harvesting is the process of collecting useful free knowledge from other sources of the cyber 'field', and filling the knowledge base with them. Objects collected by this process will be maintained separately on another instance, and will be retrieved by the authors through the search interface. If authors are interested in the harvested objects, they will be added to the Shelf, and integrate with the courses. SELF platform provides two ways of collecting them: semi-automatic and manual harvesting.

The harvesting process identifies a portion of the source document, and designates it as a learning resource, provides a title, and organizes it either as a module, lesson, or lesson component or part material. This applies to both kinds of harvesting.

4.4.1 Semi-automatic Harvesting

There are a large number of structured knowledge bases available from other portals like TLDP (The Linux Documentation Project), Wikipedia, Wordnet etc. TLDP documents are also available in LaTeX as well as SGML markup (DocBook format). Similarly Wikipedia articles are also structured into sections and subsections, taxonomy, attributes, language information etc. Software documentations like Unix manual pages and texinfo documentation pages are highly structured documents. The structure in them provides sufficent semantics so that their components can be inserted into the knowledge base by scripts by parsing the documents. Gnowledge.org already contains harvested content from Wikipedia, which will be available to authors for adoption.

Automatically harvested objects may not be usable directly in the SELF Platform. The automatically harvested content will be kept separately from what the authors actually make. This content will act as seed content for authors, or reference material. Transforming them into usuable course form requires contribution by authors. Further processing happens through atomiser, described in section.

4.4.2 Manual Harvesting

Unstructured free knowledge available in the form of text files, HTML files, PDF files, ODT files, cannot be automatically parsed for semantics. Such files will be uploaded to the SELF platform through the regular file upload function. The information in such files can be used for course construction only after breaking the document into meaningful parts, and also adding the semantics

manually. We call this process “atomising”.

4.5 Atomizing and Organizing

The authoring environment of SELF Platform is designed to encourage the creation of small learning objects. It is desirable that each learning activity meets a single objective. Transforming existing course content into small content objects will be one of the activities that the authors will be engaged in. atomised resources also enhance the possibility of reusability in several other courses.

The atomization supported by organizer works as follows:

1. An author chooses one of the source documents that requires to be broken down into usable learning experiences with specified learning objectives.
2. The author locates most likely a small portion, a sub-sub-subsection from the source documnet, and selects the text area. The selected area will be added into the course as a lesson. This process continues till all the components are harvested.The author then organizes the learning objects (atoms) and gives it structure similar to the following:

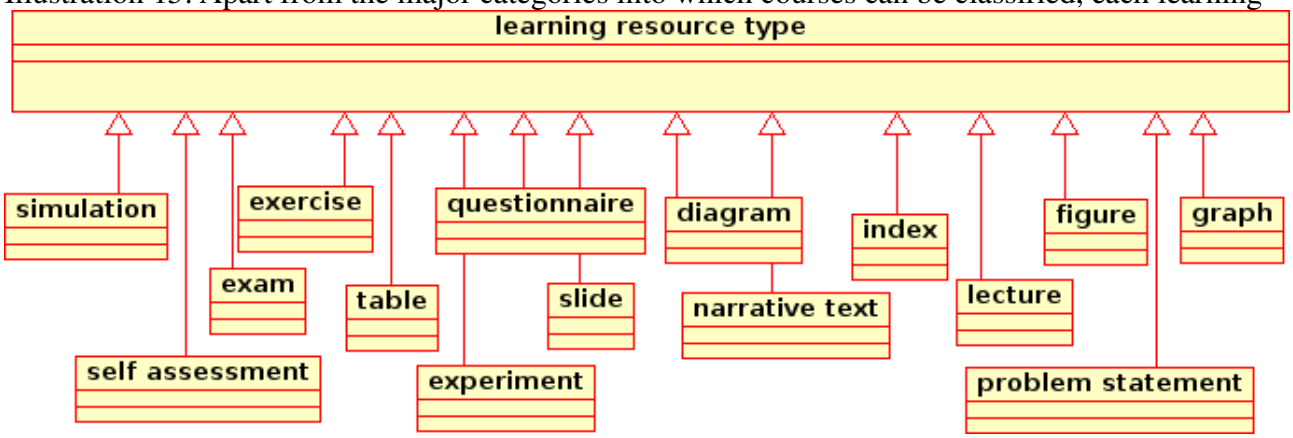
```
Module 1
|-Section 1
|--Subsection 1.1
|--Subsection 1.2
|-Section 2
|--Subsection 2.1
|---Subsubsection 2.1.1
|---Subsubsection 2.1.2
|--Subsection 2.2
|--Subsection 2.3
@
```

The leaf nodes of this heirarchy usually contain the learning experiences. The structure among the items will be collected as the default navigation and sequencing order and will be stored on the system node. The default navigation sequence will be stored in the structure field of the system node in the form of a sequence of *ids*. For the above example, the structure will be as follows:

```
[Module 1
     [Section 1
          [Subsection 1.1
           Subsection 1.2]]
     [Section 2
          [Subsection 2.1
                [Subsubsection 2.1.1
                 Subsubsection 2.1.2]]
          Subsection 2.2
          Subsection 2.3]]]
```

Each node of the hierachy will be added as a system node, and all the nodes will be serialized into a sequence and will be stored in one system. Such as master system that holds references to other systems within, will have a relation <basicSequence of> a course. Please see sequencing details in the section 4.6.

Illustration 15: Apart from the major categories into which courses can be classified, each learning



experience is often named according to the kind of resource. One of the guiding principle while atomising a course can be to mark the resource appropriately. Guiding principle for each atom is that it provides a unique learning experience, and must have a clear objective. The arrow indicates that the bottom class is an instance of the metatype.

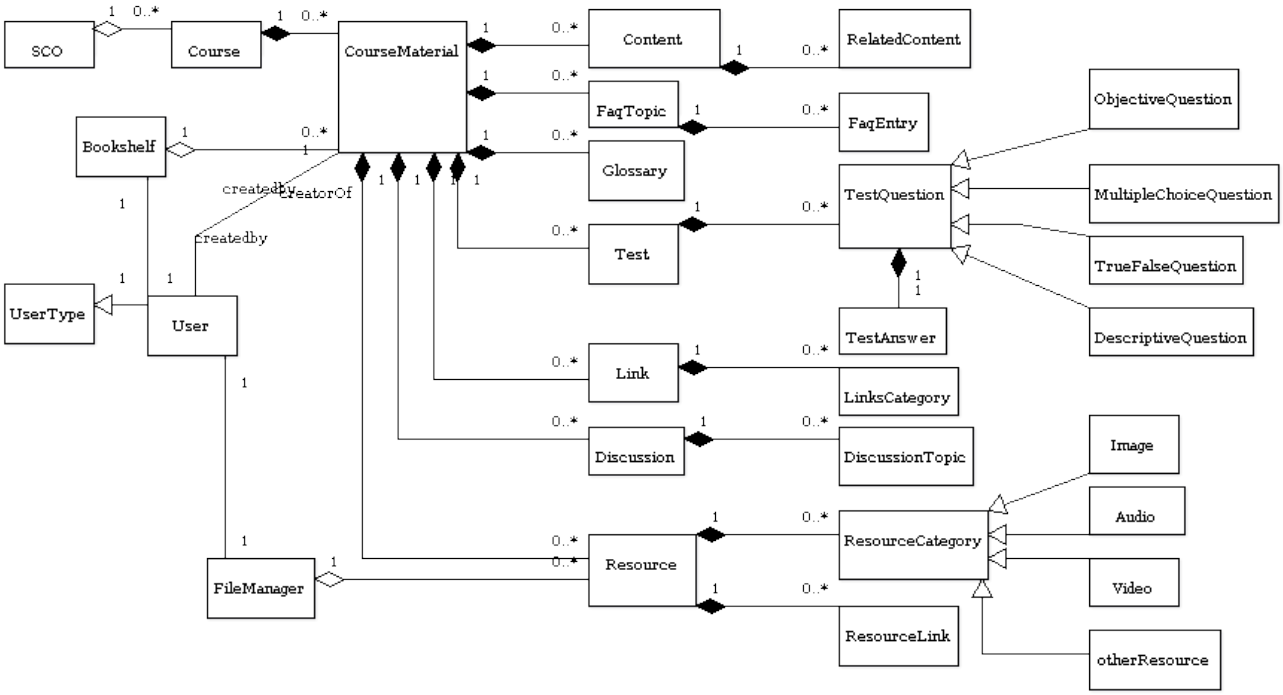


Illustration 16: A consolidated UML diagram showing the relations between the various components of the SELF Platform.

4.6 Course Organization, Sequencing and Navigation Definition For the SELF Platform

The Platform Definition provided only a high level specification that the SELF Platform will have a facility to organize course trajectory. However, since the specification demands that the SELF Platform be LOM[12] and SCORM[13] compliant, the detailed specification is drawn from these standards. The following details are essentially implementation details of the SCORM 2004 standard.

4.6.1 Default Sequence and Navigation

The default course navigation is the basic content organization as described in the section above on the atomizing and organizing. In the absence of advanced or adaptive sequence relations this structure will be taken as a guide while navigating the course.

Since the authoring environment in the SELF Platform is a crucial part, the main course navigation support will be the default sequence.

12 Final 1484.12.1-2002 LOM Draft Standard: http://ltsc.ieee.org/wg12/20020612-Final-LOM-Draft.html
13 SCORM® 2004 3rd Edition Overview Version 1.0 DRAFT: http://www.adlnet.gov/scorm/20043ED/Documentation.aspx

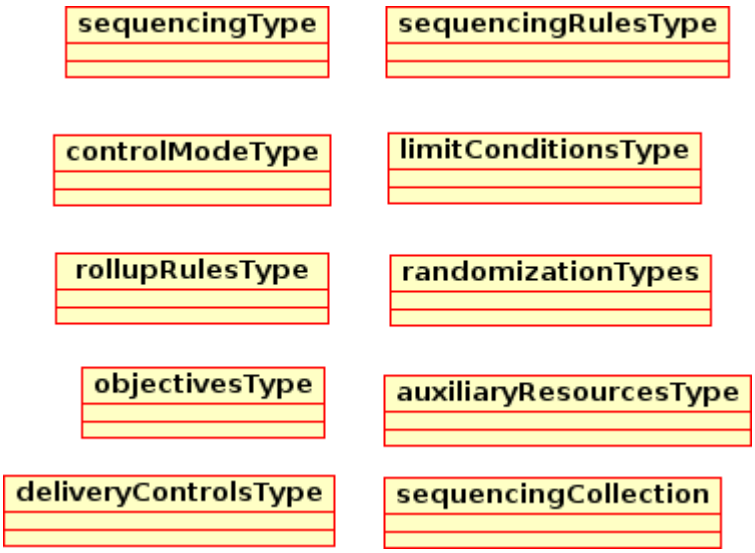


Illustration 17: Adaptive sequencing, rollup and navigation requires the above mentioned nodes. These are as per the SCORM 2004 specification.

4.6.2 Advanced (Adaptive) Sequence, Navigation and Rollup

Adaptive or advanced sequence definition is one of the most complex, but very useful set of metadata applied to an activity. A Learning activity, according to SCORM 2004 specification can be described as a "meaningful unit of instruction" any collection of that can be together grouped, given the basic sequence, and is intended to meet one of the course objectives. An activity may also be comprised of other sub-activities in a hierarchical manner.

Illustration 18: An example of a learning activity. Source from ADL, SCORM SN Book 2, Figure 2.2a.

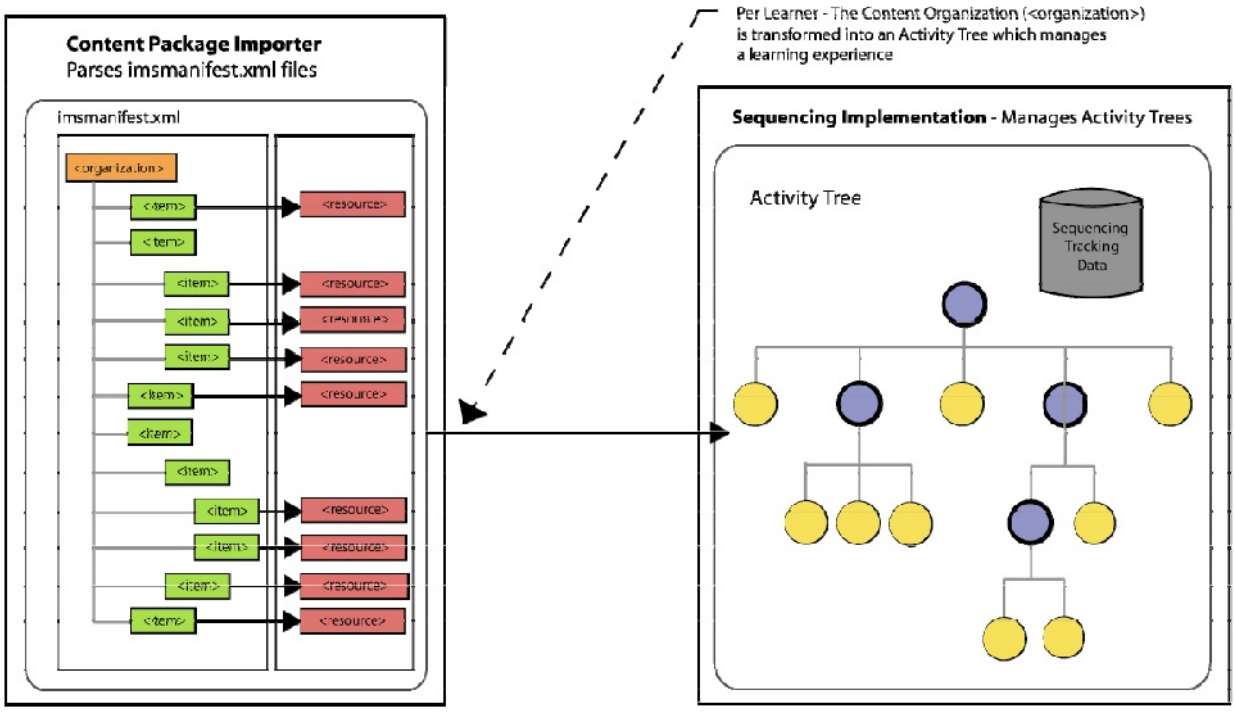


Illustration 19: The relationship between the basic default sequencing and the activity tree. Source from ADL, SCORM SN Book2, Figure 2.1.1a. Copyright ADL 2004.

In the SELF Platform, each parent node will be implicitly taken to be an Activity, unless otherwise specified. Thus for each parent node, the author can apply the adaptive sequencing metadata.

The rules defined to each activity will form the navigation model for each course. The entire definition will be collected as in the nodetype <system> of the GNOWSYS knowledge base.

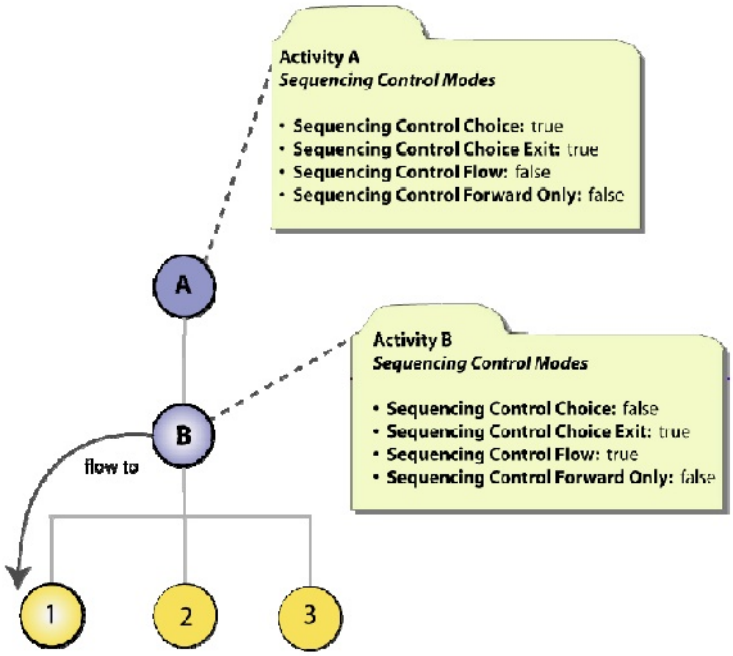


Illustration 20: Activity node will have the metadata suggesting 'Sequencing Control Flow = true', therefore the learner will be delivered the objects in the sequence. If it is set to false, the learner has the choice to do any of them in any order. Figure source from ADL, SCORM SN Book2, Figure 3.2.1b. Copyright ADL 2004.

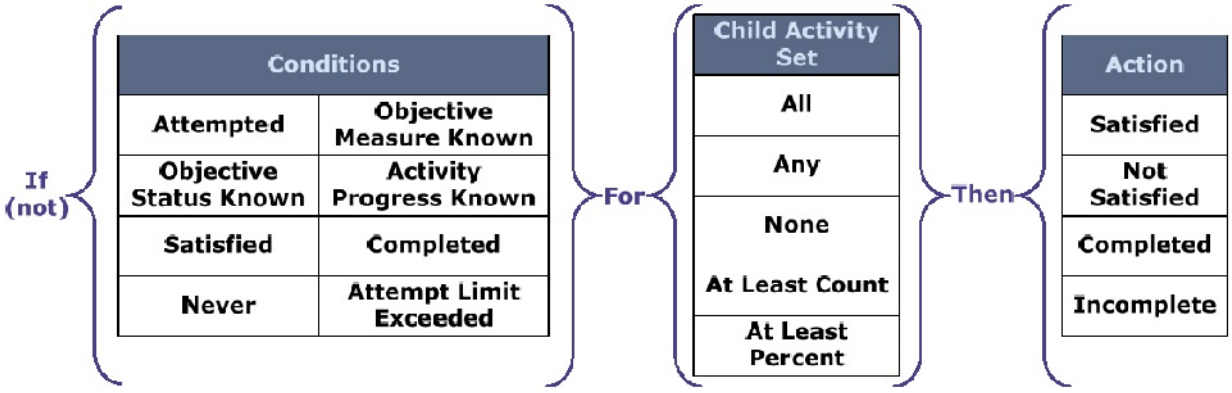


Illustration 22: The Rollup Rule Conditons Child Activity Set and Actions. ADL, SCORM SN Book2, Figure 3.7a. Copyright ADL 2004.

Illustration 23: Sequencing Rule Conditions and Actions. ADL, SCORM SN Book2, Figure 3.4a. Copyright ADL 2004.

will be disabled by the LMS. ADL, SCORM SN Book2, Figure 3.2.3a. Copyright ADL 2004.

A system node is an organizer node. GNOWSYS contains a special field called 'structure', which holds essentially the sequence of nodes. One of the nodes in this seqeunce can also be a system node. Thus nested structure is obtained. A hierarchical structure can be defined using this organizer node.

Illustration 24: The illustration shows the special datatype classes useful for defining the advanced sequencing rule conditions, rollup actions, rollup conditions, condition combinations, etc. All these are used as special attribute values while defining the sequencing and rollup rules.

Embedded within this sequence some of the nodes will specify a sequencing rule, sequencing rule action or a rollup rule. Sequencing rule objects are described in the illustration 45. The order is very similar to the way an XML file serialises different elements and embeds sub-elements within. Structure node in GNOWSYS will be isomorphic to the standard XML definition of each document encoding. It can be SCORM, ODT, OWL, or any such document encoding specification.

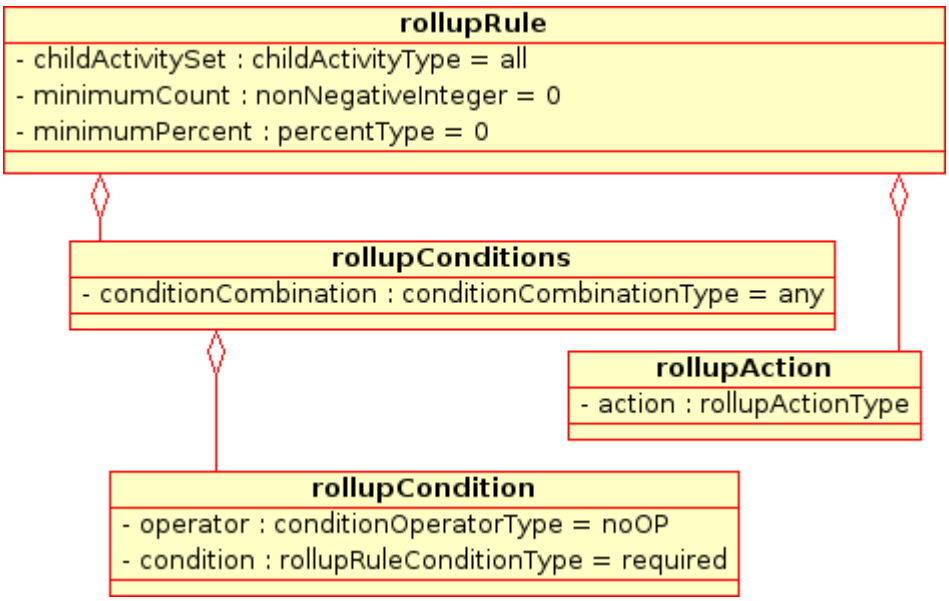


Illustration 25: Course Rollup rules specifiable by the authors. Rollup rules help in aggregating the learners behavior and adaptively modifying the navigation. As the learners are experiencing, the course, their actions and results are tracked, and passed on to the parent nodes, informing the navigator the results of taking a course. When all the activities are experienced by the learner the course sort of rolls up. In the SELF Platform, these conditions are applied to the parent nodes of activities. Rollup actions are also classified as pre, post and exit as in the sequencing rules.

4.6.3 Classification of Resources

Classification of the resources can be made according to the standard schemas suggested by LOM and SCORM, as well as user defined categories. Network model of SELF platform is eminently suitable for organizing the same resources in multiple ways.

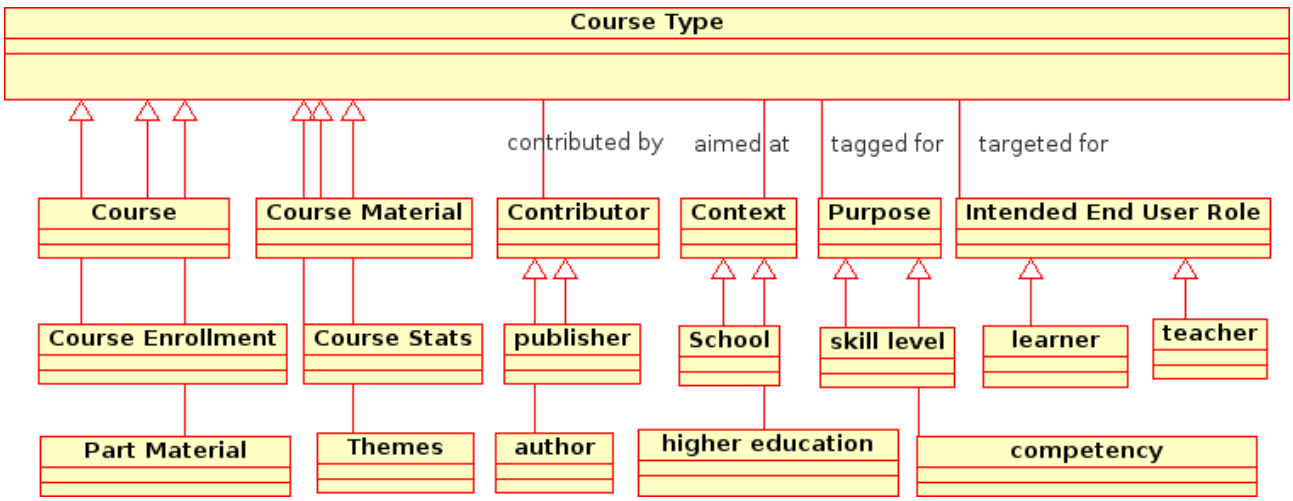


Illustration 26: Courses can be classified according to predefined ways, as well as arbitrary tags the users of the platform suggest. All these will be implemented by defining the appropriate relations. The arrow indicates that the bottom class is an instance of the metatype <Course Type>.

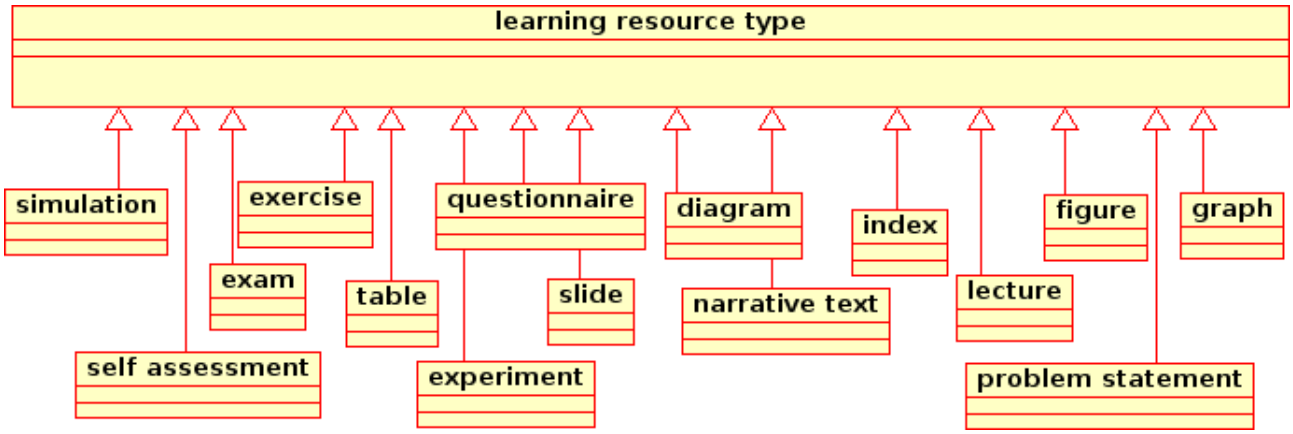


Illustration 27: Shareable Learning Objects can be classified as above. A course depending on the kind of objectives will have a few of the above kinds of learning objects. These will be implemented by establishing the relation node <subType> between the above types, and node type <instanceOf> will be used between the specific learning object and the type. The arrow indicates that the bottom class is an instance of the metatype <learning resource type>.

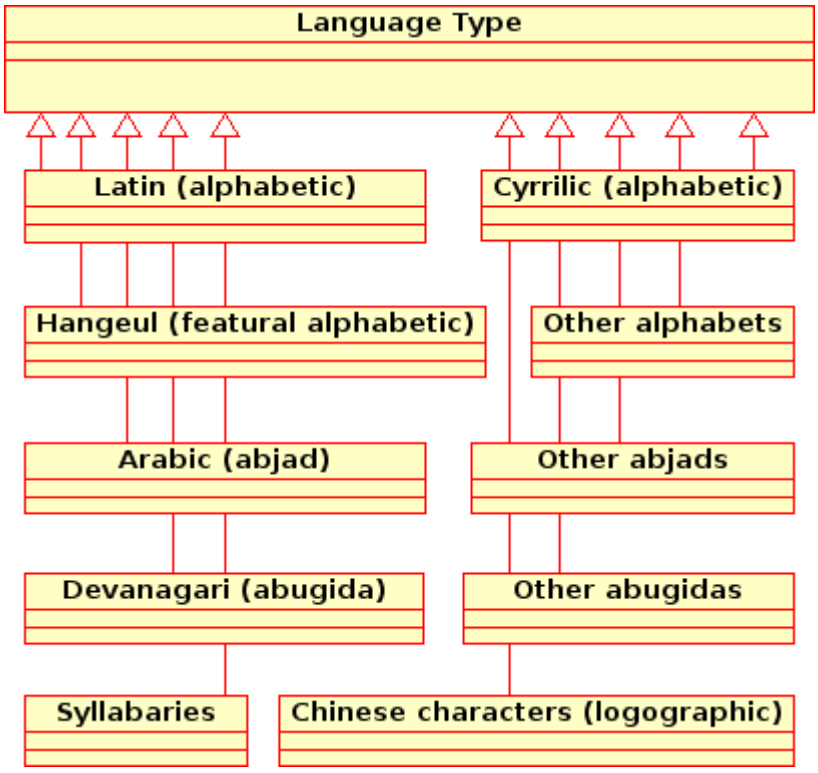


Illustration 28: Specific languagess will be made as instances of the above language types. The arrow indicates that the bottom class is an instance of the metatype <Language Type>.

4.7 Version Control

All changes to the knowledge base will be marked either as changes or as committs. Unless

modifications are committed, other users will not see them. The changes and committs will be recorded in the knowledge base by the version control enabled instance of GNOWSYS. The default change management policy will be applied to all metadata, as explained in section 3.2.

The main body of the lessons or courses, and other file types like images etc. are all stored external to GNOWSYS. For the Phase I of the SELF Platform, such files will be saved in Plone. Each modified version of the file's URI will be entered in the 'Path' field of the nodes in GNOWSYS, with the number computed as mentioned in section 3.2 For the Phase II of the SELF Platform. All the media files other than text will be stored in a file system. . This will reduce the load on the database, as the platform begins to unfold and becomes more active.

4.7.1 Keeping history

All committed changes from the day of the final launch of the portal will be maintained. Users can view the older version of an object by selecting it. This will trigger a retrieval call of the version by using the unique version number in the form of a complex structured sequence of numbers picking the exact snapshot from the history. Since this is like any normal retrieval of content from the knowledge base, it does not involve any special computation, and will not take more than normal time.

4.7.2 Tracking and Record Keeping

A special class will be maintained in the knowledge base that will keep the use, change numbers, objects changed, for each transaction. Since, the knowledge base uses a networking model, usually every transaction changes more than one node in the network. Thus, whenever an object is modified, user records are also modified.

Only the modifications are tracked for the authoring system of the SELF Platform. Retrieval does not trigger any records to be kept in the knowledge base. Thus each access of the knowledge base does not modify the nature of the knowledge base.

However, when SELF Platform evolves later as a Run Time Environment delivering courses, then each and every access is required to be tracked. SCORM 2004 specification provides the schema for implementing such a feature. These records are used in tracking the progress of learners.

4.7.3 Comparing versions

The users of the SELF Platform can select any two versions of the same object, and compare the changes.

4.7.4 Branching

Branching happens implicitly whenever an older snapshot of a node is modified inplace of the recent snapshot.. When a translation is attempted, or when changes of educational context, level, purpose are made, a new trunk is created.

4.7.5 Final Storage Design

During the Phase II development, the storage will be implemented in Postgresql and filesystem, leaving nothing in the Zope database. Zope however will continue to provide the web application framework and web services, and will contain the UI templates.

- **Nodetype, NID and SSID**: Every node is classified according to any of the 9 types, and all nodes irrespective of type are provided an unique id called node ID, NID. First time a node is made the first snapshot of the node will be created, and will be given a snapshot ID, SSID.

Given an NID or an SSID the system can determine what type of node it is by making a simple query.

- **VID and Value Tables**: All values are stored in value tables which are as many as the datatypes supported by Postgresql, currently about 105. As sson as a value ID, VID, are generated, its reference will be placed in the field tables. This storage eliminates the need to store a value twice in case it occurs in a node. Since every snapshot of a node is stored as a separate rwo in the snapshot tables, most of the values are repeated except the modified value. This way the underlying storage mechanism of GNOWSYS does not bloat despite keeping all changes in the database.
- **FID and Field Tables**: The field tables contain reference of which VID is used by which node.
- **Snapshot Table**: One snapshot table each for each nodetype stores the references of values for each field defined for every nodetype. Each snapshot table contains foreign keys to FIDs, and field tables in turn contain foreign keys to the VIDs generated by the value tables.
- **Auxiliary tables**:
    - **Nodetype Tables**: A reference table providing a unique NTID for each Node Type.
    - **Special Tables for Attribute Types**:
        - **regular expressions**: A table that stores regular expressions helping in validating different types of values. Since each regular expression restricts the number of possible values of a domain of values, it is used to define what possible values a given attribute can have.
        - **vocabulary table**: Some of the attributes need one or more of a given list of values to be selected. Whenever the system requires such an attribute, a vocabulary will be created, often from a standard vocabulary list available from best practices. These lists can be reused across applications. E.g., a list of languages, countries, levels recognized in a curriculum etc.
        - **value restriction table**: This is a table that helps in defining the various ways in which the universe of values can be devided into smaller well defined values reusable by a name.
- **Views**: The storage spread out in various tables is collected by complex join queries into view tables. These tables will be used for acessing all the values of each node. Search operations on the knowledge base will work on these dynamically generated views.
- **Storing in the Filesystem**: For each node a folder is made on the server with a NID as its name. Within this folder a number of snapshot files will be generated for faster retrieval of the information of each node in the form of memory dumps. These memory dumps are in the pickles (marshalled) which can be directly fed into the main name space depending on the context. This makes retrieval of all information very easy and fast, and reduces load on the database. The structure of these pickles is uniformely encoded as Python dictionaries as key and value pairs.

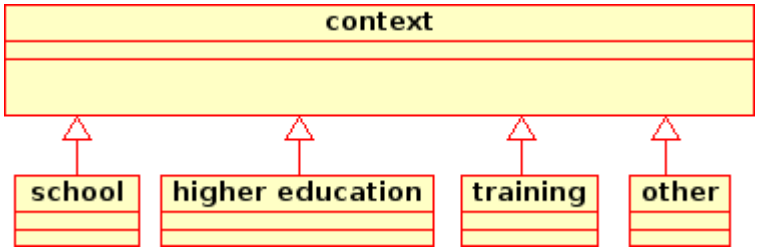


Illustration 29: Each content object must be authored keeping in mind the educational context. The platform supports branching a course from one context to the other, so that by making suitable modifications, the content can be adapted to other educational goals.

4.8 Rating and Crediting System

User and content rating and user rating will be implemented based on a number of measures that the knowledge base accumulates. Since there are several ways of computing, this will remain an open ended problem. As and when we find a useful model of measuring, an implementation will be attempted. Currently our focus is on trying to keep all the useful records which can be used to compute the rating and credits.

The user rating model in SELF Platform mostly happens by what an author does in the system. Though user rating can also be made by user's feedback, more reliable rankings can be generated by constructing the profile of the users. This kind of passive rating happens due to highly structured semantics of the knowledge base. First, SELF Platform keeps track of every change made to the knowledge base. Second, the change is always made to a category of an object, say to atomiser, editing, creating new objects, adding categories, translating, organizing, sequencing, editing metadata etc. Third, the change is made to a particular kind of field of the knowledge base, say to relations, attributes, or content etc. All this information will be very useful in generating a profile of each user, and such user profiles can be used to rate the users. A user may be good in writing good content, but may not be very good in defining the sequence, navigation controls, and rollup rules. Another user may be good in doing the latter and not the former. Based on the creator of such data, we can distinctly credit users. Thus, even though a author does not explicitly say that he is a translator, by the nature of the changes the system will infer as a translator.

SELF Platform's rating is not confined to users, but also of the content. When an object created by a user, is not used in any course, the rating will fall not only of the users, but also of the content. If a learning object is used in more than one course, its rating will be high. In a network based storage, computing the number of relations with other objects, and to which kind of objects is straight forward. If the reputation of a content is high, obviously the reputation of the authors will also be.

Volume of each user's contribution will be computed by measuring the amount of change. Amount of change can also be on the bases of the size of change, the number of words, and the number of insertions. Most direct calculations will be updated for each commit/change.

In a collaborative environment, more than one user often contributes to an object or to a course. So SELF Platform keeps the record of who created the object, who contributed to the object. All the users will be credited by announcing them to authors.

Based on the time the user logs in and logs out we will record the duration of each session. And the changes made in a given duration can be added up to assess the kind of usage of the system.

The Shelfs in the platform also provide useful information. How many times a user enters into the

User Shelf of a course will tell the user's collaboration profile. And how many of the user's articles find their way in other users's book shelf.

All such computations, and possibly several others can be carried out. As mentioned earlier, as and when a clear model to compute is known, we will make all efforts to implement.

4.9 Translation

One of the main activities envisaged on the SELF Platform is translation of content from one language to another. Each user can select one main language and a few other languages as secondary. This stored configuration will help us to compute the list of languages for selection when a user intends to translate an object.

Translation is required for three different kinds of layers of the Platform.

4.9.1 Content

Every content object has two language specific relations.

```
#content object <encoded in> #langauge
#content object <translation of> #content object
```

The former is made at the time of creating the object, while the latter is made only when an author saves a transaltion. The following procedure is followed for this step:

1. create a new sibling object node with relation node <translation of> with the source object.
2. warn the user if the object is a course component and if the translation version of the course does not exist, prompting the user to create one.
3. if the course object is not created by the user despite the warning, add the translated object into the TranslatorShelf.
4. if the course object is created or if already exists, establish the regular <part of> relations with the translated course object of the current langauge.
5. each translation creates a new object, therefore will have the version key set to 0.0, and create a relation node <branched from> with the source version of the object.

4.9.2 Metadata

The metadata refers to the names of attributes and relations, such as <branched from>, <part of>, <translation of>. All such metadata used for SELF Platform will be collected into a POT file (Portable Object Template[14]), as per the free software internationalization guidelines. A translation interface for them will be available so that PO (Portable Object) for each language will be created, and will be used for each instance of SELF Platform. Though initially seven languages will be supported, eventually other translations will be applied to SELF platform as and when community contributions are committed to the source tree.

14 Portable Object Template files are explained in the GNU Gettext utility, designed to handle translation files: http://www.gnu.org/software/gettext/manual/html_mono/gettext.html

4.9.3 User Interface

Similar to the metadata, all messages of the SELF platform will be collected as another POT file, and will be translated to generate PO files for each language.

4.10 Search

Searching the knowledge base of the varioius kinds of information is part of any web application. The SELF Platform search is enabled by maintaining two catalogs.

One catalog keeps all the natural language content and the other catalog mantains all the metadata of the content. The input string is matched with objects's containing the string, and prioritizes them on the basis of percent of relevance. The input strings are dynamically expanded while typing, and the objects of higher relevance are shown immediately.

It is thought often that a kind of blind matching of a string strategy is not effective. The SELF Platform provides another search (semantic or advanced) which shows results based on the data model of the SELF Platform. Since the knowledge base of the SELF Platform is implemented in GNOWSYS, which is a semantic web enabled server, users can query based on the semantics used in the data model of the SELF platform.
Traditional databases are accessible only through a CGI application, which in turn makes the queries to the database. This does not expose the data model to the users. Through the advanced search, SELF Platform will let the users look at the knowledge base by the data model.

Advanced search behavior can be as follows: when users do not specify any specific category, the input string will be matched with only the title, description and keywords fields of the knowledge base. This is the default behavior of the advanced search. Later the user can specify, search only among the courses, or only among the Spanish courses, or only among the part material, or among those objects present in the book shelves of the current users, sort objects according to user rating, etc. This search gives endless possibilities to see through the database.

4.11 Delivery

The main objectives of the Platform is to facilitate collection (harvesting) of the available free knowledge resources, create fresh resources, organize them into packages. The packages thus obtained can be distributed in various ways.

4.11.1 Default

Being a web based platform the default delivery of the packages will be an online delivery. A basic default navigation allows the users to go through the content and its organization.

4.11.2 HTML

Each package can be downloaded as a single or a bundle of navigable content in the form of HTML, and other media files that are internally linked.

4.11.3 SCORM

Each course package can be exported in the standard SCORM 2004 format. These files can be played in any standard compliant LMS, such as Moodle, Atutor etc.

4.11.4 LaTeX

Each package will also be exportable as a LaTeX file, or a bundle of LaTeX files.

4.11.5 PDF

The packages will be delivered as a PDF file after processing them from either a LaTeX source of the package, or the HTML source of the package.

4.11.6 ODT

As ODT is emerging as a widely used free document standard, it is highly desirable to have an export mechanism for this format as well. However, users can take the HTML file and then later save them as ODT files, till we find a way of direct conversion.

4.11.7 Docbook

The package content will be exported into Docbook standard format, which can be used by publishers for reuse of the courses distributed at SELF Platform.

4.12 Distributed knowledge base

As stated earlier, the knowledge base of the SELF Platform needs not keep all the data in one server. The data can be physically located anywhere on the Internet. A SELF Platform instance can be configured to search the catalogues of a list of servers specified in a list published as instance's property. When a user makes a query, the results will show the objects of all the servers. Since each result (object) has a unique URL, users can access the object, adopt it in the current instance, or make relations with the remote object etc.

4.13 p2p service architecture

Exploiting the distributed knowledge base feature, each server acts as a peer to other servers. The servers are configured to each other either as primary or secondary, similar to the P2P architecture of DNS servers. This service works as follows: if a server SP-1 is marked as primary to another server SP-2, whenever commits are made to the knowledge base, the primary server sends a message to the secondary server, and propagates the changes to the secondary server. If the secondary server does not want the entire knowledgebase, the server can be specified to get only courses, say of “computer science”. Since changes are highly granular, it does not create heavy traffic between the servers.

If SP-3 is located in a remote area, and is not regularly connected to the main server SP-1, people can contribute directly to SP2, and when the connection is established, changes from SP-3 will be propogated back to SP-1. The same feature can be used for making backups of knowledge base on a regular basis without any human intervention.

The implementation is a feature available in GNOWSYS, which provides functions like populatePeer, initializePeer, and addPeer.

4.14 Accessibility

SELF Platform inherits several good practices of a web application from Plone, which conforms to Section 508 (reference), W3C AA compliance (reference). One of the developers of the SELF Platform is a visually impaired programmer, who will be testing the platform to check accessibility.

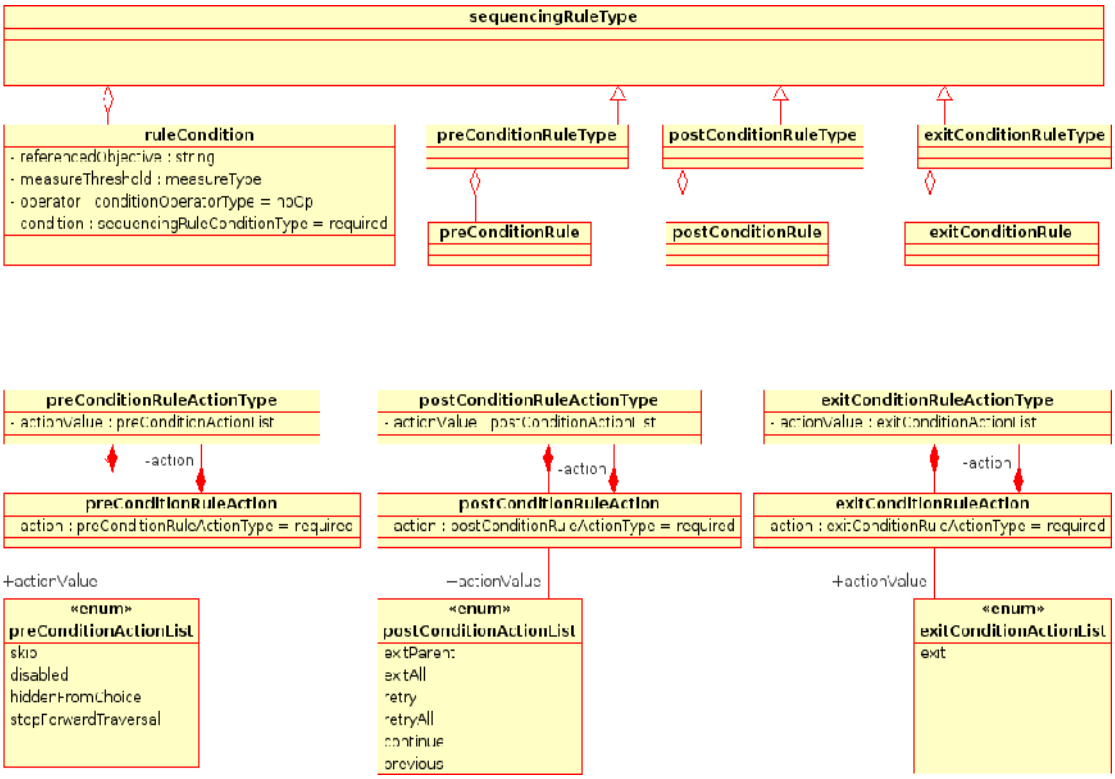


Illustration 30: Advanced Sequencing Model of SELF Platform conforming to SCORM 2004 Sequencing rules are defined by setting preconditions, post conditions and exit conditions. Each of the rules inform the RTE, how to behave with the learner. And each of the conditions are mapped to the corresponding actions. The possible actions are enumerated in the 'three separate lists in the illustration marked as 'preConditionActionList', 'postConditionActionList', and 'exitConditionActionList'.

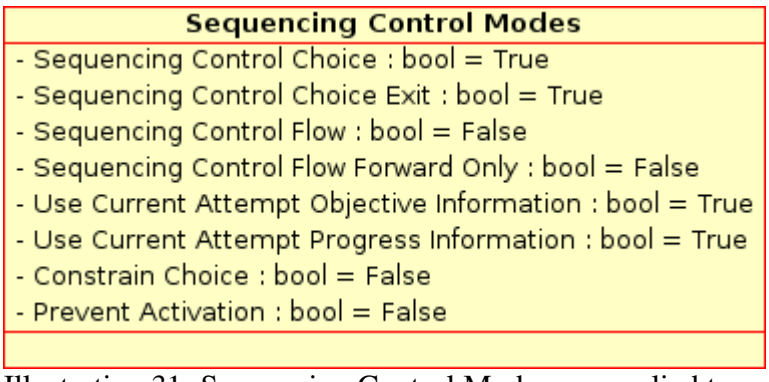


Illustration 31: Sequencing Control Modes are applied to activities. These will not have any effect if applied to the leaf node of the basic sequence defined by the organizer.

5 Glossary

Term	Description and/or references to the terminology
ADL	Acronym for “Advanced Distributed Learning”, an Initiative sponsored by the OUSD P&R of the USA Department of Defence. ADL maintains the SCORM collection of specifications.
CL	Acronym of “Common Logic”, a formal language based on first-order logic, intended to facilitate the exchange and transmission of knowledge in computer-based systems. http://en.wikipedia.org/wiki/Common_logic
CMS	Acronym for “Content Management System”, a system designed to help users publish content on the web, usually embodying editorial policies and workflows, and unifying layout and design.
CVS	Acronym for “Concurrent Versions System”, CVS is a widely spread version control system published under the GPL license.
DAISY	Acronym for “Digital Accessible Information System”, a standard for Digital Talking Books. http://www.daisy.org
Distributed Database Management System	A distributed database is a database that is under the control of a central database management system in which storage devices are all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. http://en.wikipedia.org/wiki/Distributed_database
DocBook	A markup language for technical documentation. It was originally intended for authoring technical documents related to computer hardware and software but it can be used for any other sort of documentation. http://en.wikipedia.org/wiki/DocBook
DTML	Acronym for “Document Template Markup Language”, a server-side scripting language introduced with early Zope versions.
Dublin Core Metadata	The Dublin Core metadata element set is a standard for cross-domain information resource description. http://en.wikipedia.org/wiki/Dublin_Core_Metadata_Initiative
FTP	Acronym for “File Transfer Protocol”, it is one of the first and most popular high-level Internet protocols, often used to publish large files for public access. http://en.wikipedia.org/wiki/FTP
GNU	Recursive acronym for “GNU's Not Unix”, a project launched by Richard Stallman in 1984 with the aim of creating a complete computing environment comprised exclusively of free software. http://www.gnu.org/
GNU arch	A software revision control system that is part of the GNU Project. Since May 2006 the project is maintained but not under active development
HTTP	Acronym for “HyperText Transfer Protocol”, it is the network protocol uses to deliver most of the World Wide Web's content. http://en.wikipedia.org/wiki/HTTP
IMS	Acronym for “Instructional Management Systems”, a non-profit standards organization concerned with establishing interoperability for learning systems and learning content and the enterprise integration of these capabilities. http://en.wikipedia.org/wiki/IMS_Global
LaTeX	A document preparation system based on Donald Knuth's TeX typesetting system. It is very popular and widely used for scientific papers in the so-called “hard” sciences. http://en.wikipedia.org/wiki/LaTeX

LMS	Acronym for “Learning Management System”, a software package that enables the management and delivery of online content to learners. http://en.wikipedia.org/wiki/Learning_Management_System
LOM	Acronym for “Learning Object Metadata”, a data model, usually encoded in XML, used to describe a learning object and similar digital resources used to support learning. http://en.wikipedia.org/wiki/Learning_object_metadata
ODF	Acronym for “Open Document Format”, a open standard document file format used for describing electronic documents such as memos, reports, books, spreadsheets, charts, presentations and word processing documents.
ODT	Acronym for “Open Document Text”, the ODF format for storing word processing files.
OWL	Somewhat contrive acronym for “Web Ontology Language” is a language meant to describe and instantiate web ontologies. http://en.wikipedia.org/wiki/Web_Onology_Language
P2P	A peer-to-peer (or P2P) computer network relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. http://en.wikipedia.org/wiki/P2P
PDF	Acronym for “Portable Document Format”, a file format used for representing two-dimensional documents in a device independent and display resolution independent fixed-layout document format. Each PDF file encapsulates a complete description of a 2D document that includes the text, fonts, images, and 2D vector graphics that compose the document. http://en.wikipedia.org/wiki/PDF
PLONE	A free CMS system based on Zope. http://www.plone.org/
PostgresSQL	PostgreSQL is a free software object-relational database management system (ORDBMS), released under a BSD-style licence. http://en.wikipedia.org/wiki/Postgresql
RTE	Acronym for “Runtime Environment”, a configuration of computer programs that enables the execution of a given process, as opposed to its development.
SCO	Acronym for "Shareable Content Object" as used in SCORM.
SCORM	Acronym for “Sharable Content Object Reference Model”, a collection of standards and specifications used to encode content for computer-aided instruction systems. http://en.wikipedia.org/wiki/SCORM
Section 508	Section 508 requires that all content and services provided by the USA Federal Agencies be accessible by individuals with disabilities. It includes a collection of standards for software applications, web-based content and other ICT products. http://www.section508.gov
SELF	You really shouldn't be reading this... http://selfproject.eu
SELF Platform	SELF Platform is an online learning environment with educational and training materials about Free Software and Open Standards. http://dev.selfplatform.eu
SGML	Acronym for “Standard Generalized Markup Language”, a data representation metalanguage designed to be able to specify data representation languages for a wide variety of applications. HTML, XML and DocBook are instances of languages defined in SGML. http://en.wikipedia.org/wiki/SGML
SOAP	Acronym for “Simple Object Access Protocol”. a protocol for exchanging XML-based messages over computer networks, normally using HTTP. http://en.wikipedia.org/wiki/SOAP
SVN	Subversion, also referred to as SVN, is a revision control system designed to be a modern replacement for CVS.

URI	Acronym for “Uniform Resource Identifier”, is a structured string that is used to identify a specific resource, such as a file on a web server(http://en.wikipedia.org/wiki/URI), a telephone (sip:+1615555123), a file on the local disk (file:///etc/passwd), etc. Most people call URIs “web addresses”. It is oten used interchangeably with “URL”.
URL	Acronym for “Uniform Resource Locator”, an URI which specifies the means and location of the resource, as opposed to an URN. It is mostly used interchangeably with “URI”. http://en.wikipedia.org/wiki/URL
URN	Acronym for “Uniform Resource Name”, an URI which may not specify a location where the object can be retrieved. http://en.wikipedia.org/wiki/Uniform_Resource_Name
W3CAA Compliance	Content is considered W3CAA compliant when it follows the Level Double-A Conformance to Web Content Accessibility Guidelines 1.0 published by the W3C. http://www.w3.org/WAI/WCAG1AA-Conformance
WebDAV	Acronym for “Web-based Distribution, Authoring and Versioning”, a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on remote web servers.
XML	Acronym for “eXtensible Markup Language”, a language designed to represent structured data in a standardized and platform-independent fashion. http://en.wikipedia.org/wiki/XML
XML-RPC	Acronym for “eXtensible Markup Language – Remote Procedure Call”, a protocol used to access different kinds of services delivered by a server process. http://en.wikipedia.org/wiki/XMLRPC
XTM	Acronym for “XML Topic Maps”, an abstract model and XML grammar for interchanging Web-based Topic Maps. http://www.topicmaps.org/xtm/
ZOPE	A free application server for building content management systems, intranets, portals, and custom applications. http://www.zope.org/
ZPT	Acronym for “Zope Page Templates”, a web page generation tool to help programmers and designers collaborate in producing dynamic web pages for Zope web applications.