Release of Personalisation Features and Inquiry Learning Apps – Initial Dissemination Level Public Status Final

Sten Govaerts

▶ **To cite this version:**

Sten Govaerts. Release of Personalisation Features and Inquiry Learning Apps – Initial Dissemination Level Public Status Final. [Research Report] Go-Lab Project. 2014. hal-01201989

HAL Id: hal-01201989

https://telearn.hal.science/hal-01201989

Submitted on 18 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Go-Lab

Global Online Science Labs for Inquiry Learning at School

Collaborative Project in European Union's Seventh Framework Programme
Grant Agreement no. 317601

Deliverable D5.3

Release of Personalisation Features and Inquiry Learning Apps – Initial

Editor	Sten Govaerts (EPFL)
Date	29th April, 2014
Dissemination Level	Public
Status	Final

©2013, Go-Lab consortium

The Go-Lab Consortium

Beneficiary Number	Beneficiary Name	Beneficiary short name	Country
1	University Twente	UT	The Netherlands
2	Ellinogermaniki Agogi Scholi Panagea Savva AE	EA	Greece
3	École Polytechnique Fédérale de Lausanne	EPFL	Switzerland
4	EUN Partnership AISBL	EUN	Belgium
5	IMC AG	IMC	Germany
6	Reseau Menon E.E.I.G.	MENON	Belgium
7	Universidad Nacional de Educación a Distancia	UNED	Spain
8	University of Leicester	ULEIC	United Kingdom
9	University of Cyprus	UCY	Cyprus
10	Universität Duisburg-Essen	UDE	Germany
11	Centre for Research and Technology Hellas	CERTH	Greece
12	Universidad de la Iglesia de Deusto	UDEUSTO	Spain
13	Fachhochschule Kärnten - Gemeinnützige Privatstiftung	CUAS	Austria
14	Tartu Ulikool	UTE	Estonia
15	European Organization for Nuclear Research	CERN	Switzerland
16	European Space Agency	ESA	France
17	University of South Wales	USW	United Kingdom
18	Institute of Accelerating Systems and Applications	IASA	Greece
19	Núcleo Interactivo de Astronomia	NUCLIO	Portugal

Contributors

Name	Institution
Sten Govaerts, Adrian Holzer, Andrii Vozniuk, Na Li, Wissam Halimi, Denis Gillet	EPFL
Lars Bollen, Jakob Sikken, Anjo Anjewierden	UT
Sven Manske	UDE

Legal Notices

The information in this document is subject to change without notice. The Members of the Go-Lab Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the Go-Lab Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material. The information and views set out in this deliverable are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

Executive Summary

This deliverable presents the initial release of the personalisation features and the inquiry learning apps developed following the specifications presented in D5.1. In terms of personalisation, we present the first release of internationalisation features in Go-Lab. The language of the ILS platform (Graasp) is now propagated to the inquiry learning spaces (ILS) and apps when the apps support the language. To achieve this, the apps make use of the OpenSocial internationalisation specification (see D5.1). Furthermore, through an early prototype of the App Composer, missing translations of apps can be added. This app composer prototype has been integrated with Graasp and is currently being tested before evaluating it with teachers. The design of the recommender system has started, but the implementation did not commence yet. We focused on internationalisation in this early state of the project, since it affects both students and teachers involved in the Phase A evaluations.

Next to the work on personalisation, several Inquiry Learning Apps have been designed and implemented, of which some were specified in D1.1 by the pedagogical cluster and in D5.1. We functionally and technically describe the following apps: the Concept Mapper, the Hypothesis Scratchpad, the Questioning Scratchpad, the Experiment Design Tool, the Data Viewer, the Drop File tool, the Resource View app and the Wiki app. Additionally, work has started on the Conclusion Tool. WP3 is currently evaluating several of these apps and is expected to provide insights and feedback on their design and usefulness in D3.2 in M24. Moreover, the pedagogical cluster is refining the theoretical definitions of scaffolding and guidance apps (documented in the internal deliverable G1.3) and together with the pedagogical cluster we are designing new inquiry learning apps that will be documented in D5.5.

In order for these apps to communicate with each other we have extended the OpenApp library to enable drag and drop between apps and have implemented the Vault space to allow data storage and exchange between apps. Access to the Vault space is enabled using a JavaScript library that also supports functions for easy user activity logging using the ActivityStreams format (see D5.1). Finally, to integrate Learning Analytics services with the inquiry learning apps, we have devised an activity logging architecture and library. The work of described in this deliverable will continue and the release of the personalisation features and inquiry learning apps will be finalised in D5.5 (M32).

Table of Contents

1 Introduction

This deliverable presents the initial release of the personalisation features and the inquiry learning apps developed following the specifications presented in D5.1. The main product of this deliverable is the developed software. The accompanying text in this deliverable describes the software development related to personalisation and inquiry learning apps. This deliverable is structured as follows.

Chapter 2 presents the implemented internationalisation features. The language of the ILS platform (Graasp) is now propagated to ILS and apps when the language is supported. In the future, we want to allow teachers to configure the language on ILS level. This will be useful in schools that teach in multiple languages and will not require a translation of the ILS platform. Furthermore through an early prototype of the App Composer, missing translations of apps can be added. Translations in the App Composer can be done not only in different languages, but also in different language levels to support varying levels of language proficiency among students (see D5.2). Currently, we are testing this early app composer prototype and after stress tests, we plan to evaluate its usability with teachers before releasing a production version. Further details on the app composer will be documented in D5.4 (M24). In this deliverable, we have focused on internationalisation, since this affects both teachers and students and is essential for conduction Phase A evaluations in schools around Europe. The design and implementation of the recommender system will be documented in D5.5 (M32). Then, Chapter 3 describes the mechanisms used to support inquiry learning apps, such as inter-app communication, the Vault and user activity logging for integration with the Learning Analytics infrastructure (see D4.2). Chapter 4 describes the inquiry apps that have been designed and implemented, namely the Concept Mapper, the Hypothesis Scratchpad, the Questioning Scratchpad, the Experiment Design Tool, the Data Viewer, the Drop File tool, the Resource View app, the Wiki app and the Conclusion Tool. Finally, Chapter 5 wraps up with a conclusion and points to the future deliverables following this document. Appendix B, see Section 7, provide a snapshot of the documentation of the developed libraries. The live library documentation can be found on GitHub.[1]

[1] Go-Lab GitHub repository, `https://github.com/orgs/go-lab/`

2 Personalisation

Hereafter, we present the first release of the personalisation features in Go-Lab focusing on internationalisation. Search and recommendation will be included in the final release of the personalisation features on M32 in D5.5. Currently, finding relevant material on the Go-Lab portal[1] can be done using keyword filtering through the metadata fields defined in D2.1 and D5.2.

Since the audience of Go-Lab crosses language borders, internationalisation is a key issue. We have integrated internationalisation into apps and ILS, and have implemented the translation module of the App Composer.

2.0.1 Apps

As described in D5.1, apps in Go-Lab are implemented using the OpenSocial Gadget specification. OpenSocial provides an internationalisation specification that is used in the Go-Lab apps and the translator component of the App Composer (see D5.2). Essentially, the developer of the app provides a list of key-value pairs for each language. The keys are used in the source code of the app where a text string is needed in the UI. The OpenSocial container then replaces these keys with the values for a specific language.[2] These key-value pair-based language lists can be described in the app source code or separate XML files per language can be referenced in the source code. Most of the inquiry learning apps described in Section 4 have already implemented OpenSocial-compliant internationalisation.

2.0.2 Inquiry learning spaces

To enable internationalization of inquiry learning spaces, we use the built-in locale feature of OpenSocial [3]. From the implementation perspective, this is done through three steps. First, the Shindig container in Graasp configures the language based on the choice of the ILS author. Second, the ILS metawidget includes the language bundles in its ModulePrefs. Each language bundle is a XML file that contains the translated strings for a given language, as mentioned above. When the ILS metawidget is rendered, it will automatically use the strings available in the language bundle corresponding to the language configured by the Shindig container. Third, as the ILS metawidget contains other apps inside, it should pass the current language to the apps inside so that they can also be internationalised in the same way as described in the second step.

Currently, to choose a language for an ILS, the teacher selects from the language list in the top right corner of Graasp page, as shown in Figure 1. In the future, we plan to add the language option in the ILS creation window to improve the usability for teachers teaching in multiple languages and to limit the translation work only to the ILS UI and requiring a full translation of Graasp. Furthermore, at ILS creation time, the teacher can enter the target age group of his

[1] Go-Lab portal, `www.golabz.eu`

[2] see D5.1 or `http://opensocial.github.io/spec/2.5.1/Core-Gadget.xml#rfc.section.C.13` for more details.

[3] OpenSocial `/ModulePrefs/Locale`, `http://opensocial-resources.googlecode.com/svn/spec/2.0/Core-Gadget.xml#Locale`

students to indicate their language proficiency. After choosing a language, the ILS metawidget will be rendered in the corresponding language. Figure 2 shows an ILS in French. Note that we only translate the strings of the user interface of the ILS (e.g. the tool bar and the welcome message), the content of the ILS will be translated by its creator (i.e. the teacher).

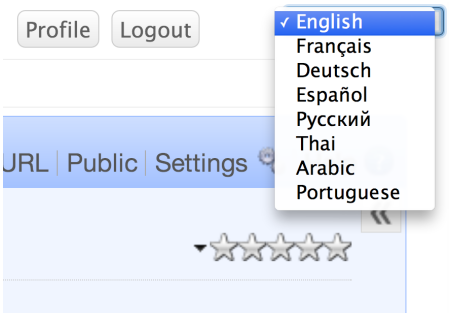

Figure 1: Screenshot of setting the language for an ILS in the top right corner of the Graasp user interface.

2.0.3 Integration with the App Composer Prototype

We have implemented a first prototype of the App Composer of which the specifications were described D5.2. Although the prototype is premature, a basic version of the translator component is working and has been connected with Graasp. A major design choice was to store the translations of an app in the app composer. Since inserting a translation in the source code of an OpenSocial gadget requires to download the gadget XML file, to change the XML file and to host the changed file. And because OpenSocial gadgets can consist of different files, it is complicated to find the path to all these files and download them, because files hosted on different domains can conflict with the same-origin browser security policy.[4]

Figure 3 describes the architecture of this integration in more detail. When an ILS is run on Graasp, each app in this ILS is rendered by Shindig. The language of Graasp is passed to Shindig together with the target group (as described in D5.2). Shindig evaluates whether the app contains the required language, if this is not the case, then the app composer is contacted to check if someone has translated the app for the required language and target group. If there is such a translation, it is returned to Shindig and is dynamically added to the app logic. After this, Shindig can render the app and Graasp can show it to the user.

We allow languages for different target groups to support different levels of language proficiency (see D5.1 & D5.2). In case that the app composer does not have a translation for a specific target group, but does have a general translation of the requested language, the general language is returned. Note, we are currently testing the app composer and therefore this mechanism is not yet available on the production server of Graasp.

[4] Same-origin policy, `http://en.wikipedia.org/wiki/Same-origin_policy`

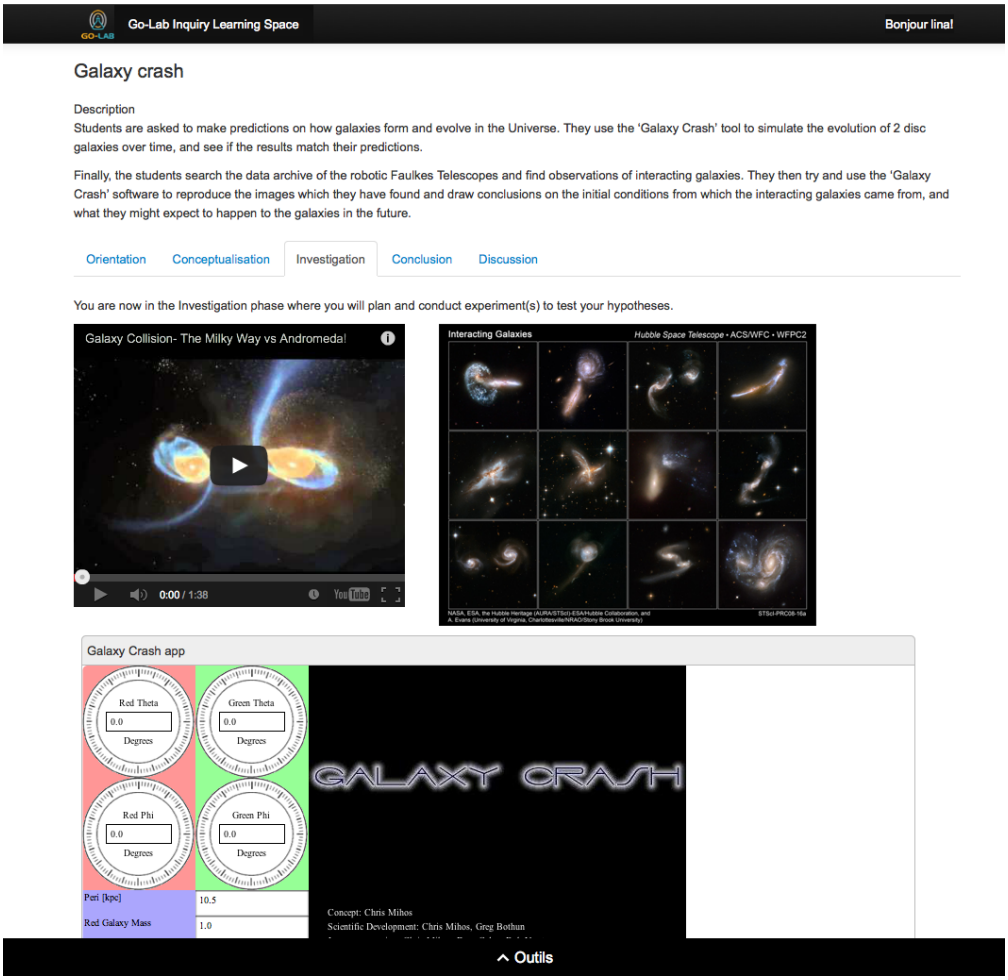

Figure 2: Screenshot of an ILS in French.

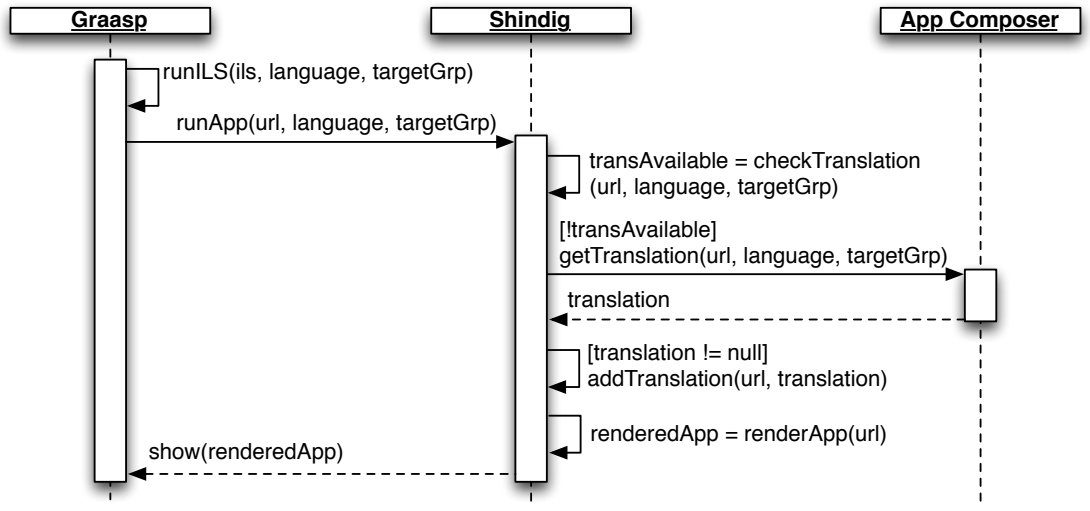

Figure 3: Sequence diagram of the translation retrieval process of the App Composer.

3 Supporting inquiry learning apps

As mentioned, inquiry learning apps are implemented as OpenSocial gadgets. Such apps provide limited interaction possibilities with other apps. In this section, we highlight the work done to improve data communication and exchange between different apps and the ILS platform.

3.1 The Inter-app Communication library

URL of the source code repository: `https://github.com/go-lab/iwc`

Library documentation: `https://github.com/go-lab/iwc`

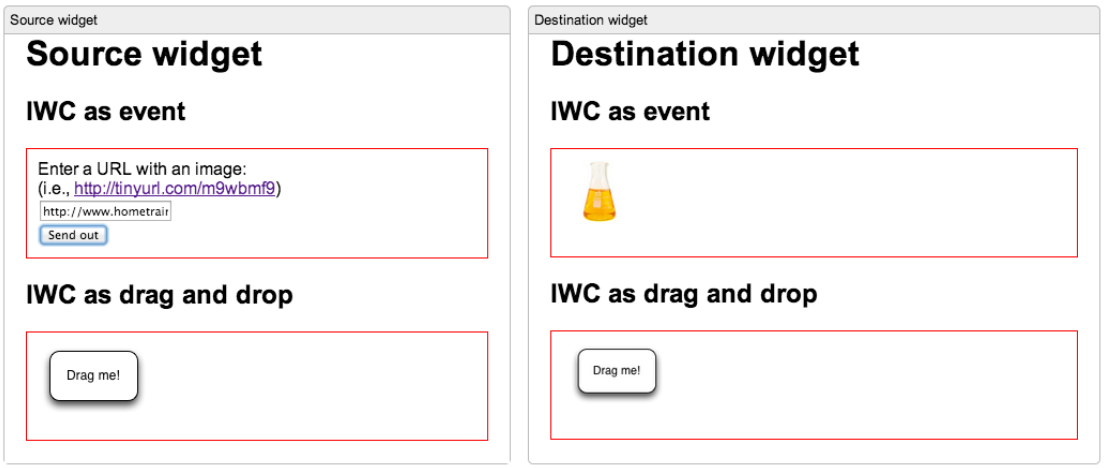

Figure 4: A prototype implementation of the drag and drop inter-app communication. The 'Drag me!' button can be dragged from the Source to the Destination widget or by pressing the 'Send out' button, one can send a regular event.

Different approaches to achieve inter-app communication exist. We use the OpenApp approach introduced by (Isaksson & Palmér, 2010) for app-to-app communication, as described in D5.1. However, we have extended the OpenApp library to enable drag and drop between apps within an app container. More specifically, the library enables the user to drag an object from one app and drop it in another app within the same web page. In the back, the library broadcasts messages to other apps on the same web page when an object is dragged from one app to the other.

To demonstrate and test the implementation, we have created two testing apps[1], illustrated in Figure 4. These two apps allow the user to send an image as an event by entering the URL of the image and pressing the *'Send out'* button. This functionality was already implemented by Isaksson and Palmér. The lower part, *IWC as drag and drop*, showcases the new drag-and-drop functionality. One can drag the *Drag me!* image from the *Source widget* to the *Destination widget*.

[1] This prototype can be used in the following Graasp space: `http://graasp.epfl.ch/#url=iwc`

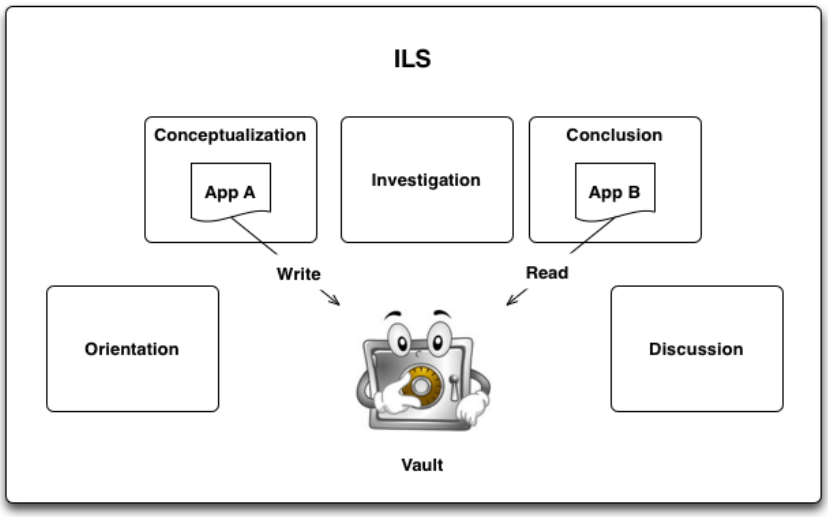

Figure 5: The vault in an ILS.

3.2 The ILS library

URL of the source code repository: `https://github.com/go-lab/ils`

Library documentation: `https://github.com/go-lab/ils/wiki/ILS-Library`

The Inter-app Communication library allows apps to communicate with each other within only one space. In most situations this is sufficient. But sometimes, apps in a given space might want to communicate with apps in different ones. In the case of an ILS, this becomes important, since an ILS consists of multiple phases, which are modelled as spaces (see D5.2). The same app can be used in different phases and data exchange between those two app instances might be a desirable functionality. For instance, the Hypothesis Scratchpad (see Section 4.2) can be added to the Conceptualisation phase to create a hypothesis, and can be added to the Conclusion phase as an aid to compare the hypotheses and the empirical data. Furthermore, the data collected by different apps in the Vault can be very useful to be exploited by the learning analytics backend services (see D4.2).

The Vault is a hidden subspace of an ILS (this structure is described in detail in D5.2). The main functionality of the ILS library is to allow apps to save files to, and to read files from the Vault subspace, as shown in Figure 5. To support this, the library provides four basic functions: (i) to create a new resource (i.e. a file), (ii) to read a resource, (iii) to update an existing resource and (iv) to list all resources in the Vault. When a resource is created some metadata is attributed to it, which could hold information specifying which app and which user has saved this resource from which space. These functions provide a basic interface for apps to interact with the Vault. Those basic functions could be extended in the future to support more advanced requirements of app developers, who wish to gain more exposure to the learning data kept in the Vault.

In addition to this data IO functionality of the Vault, the library also contains some help functions to expose information about the ILS and its phases. Currently, it includes a function to retrieve information about the current user, the current inquiry phase, and the current ILS. In the near future, access to the notification client and action logger of the ILS will be added. Note that having a common action logging client and a notification client on ILS level, lowers the burden for developers to create these clients and can improve performance.

Having some user info of the students who are anonymously logged in with a nickname (without having a personal Graasp account (see D5.2)) is important for many apps and learning analytics (see D4.2), especially those asking for input from the student user. For example, apps can provide a better user experience by saving personal data in the Vault per student, when they have access to the user info.

Apps might also behave differently based on the inquiry phase they are in. For instance, the Hypothesis Scratchpad might not allow the editing of the hypotheses if it is in the Conclusion phase (note: this is an example and such functionality is not implemented).

The library requirements can be perceived as very simple, but they were complicated to handle with the Graasp privacy management, which made it a more laborious task then initially estimated. To be more specific, users should be able to write and read data from the Vault, while the Vault is a hidden subspace of the ILS. Because it is a hidden subspace, any access or modification of the properties/content of the Vault is denied, since this is the privacy policy for hidden spaces in Graasp. The solution to this problem, should respect the Graasp privacy policy for hidden spaces and should satisfy the Vault requirements. So the following modifications were done:

- *Spaces API:* We exposed through Graasp's implementation of the OpenSocial API the retrieval of a space that is of the type Hidden-Vault, and including a Hidden-Vault subspace in the list when getting an enumeration of the subspaces of an ILS.
- *Documents API:* We allow through Graasp's implementation of the OpenSocial API to retrieve a resource, and create a resource in Hidden-Vault subspaces of an ILS

Additionally, to identify students who do not have a Graasp account but log in with their nicknames, we had to extend the user management in Graasp to support temporary users. Please note that for the moment, the library allows saving metadata (student name only for now) per resource only. The storage of the app metadata will be implemented in the next version.

To sum up, the list of available methods of data read/write and some helper methods that are implemented are provided below.

- `readResource`: reads a resource in the Vault
- `createResource`: creates a resource in the Vault

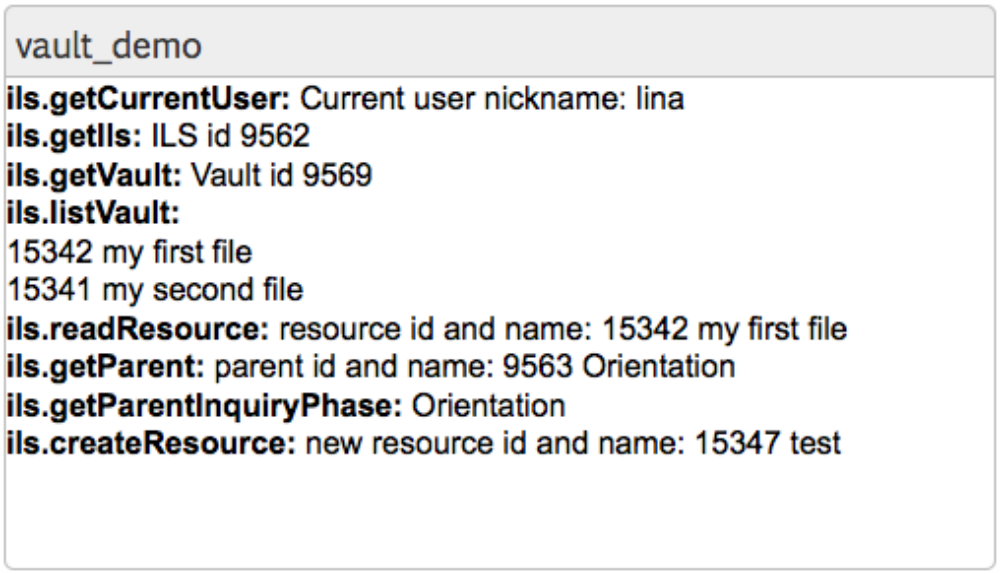

Figure 6: A demo widget showing how to use the library.

- `listVault`: lists all resources in the Vault
- `getVault`: get the Vault of the ILS in which the widget is running
- `getIls`: get the ILS in which the widget is running
- `getParent`: get the parent space in which the widget is running
- `getParentInquiryPhase`: get the type of phase in which the widget is running
- `getCurrentUser`: get the nickname of the current student who is using the ILS

A demo widget showing how to use the library is implemented, see Figure 6 (available at `http://graasp.epfl.ch/gadget/prod/ils_lib/vaul_demo.xml`). In this widget, all the available methods in the library are invoked and the returned results are displayed. The objective of this widget is to demonstrate an example for app developers of how to use this library and it can be used as aid while developing apps that make use of the Vault.

The library is still under development and additional help functions will be added based on the requirements of the Go-Lab developers. The Vault functionality might in the future also support the App Composer Adaptor module (see D5.2) to configure apps in an ILS. Furthermore, more high-level functionality to read and write to the Vault subspace can be added. For example, higher-level functions could be added to write key-value pairs to share data between apps or help functions to deal with configuration data of apps can be useful.

3.3 User activity logging library

URL of the source code repository: `https://github.com/go-lab/ils`

Library documentation: `https://github.com/go-lab/ils/wiki/ActionLogger` and `https://github.com/go-lab/ils/wiki/MetadataHandler`

To integrate the Learning Analytics services (see D4.2) with the inquiry learning apps, it is vital to provide a user activity logging architecture. As an example, with the help of the Learning Analytics services, a learner might be hinted to return to his concept map in the Conceptualisation phase, after he is stuck with the experiments in the Experimentation phase. This hint would be deduced from the learner's activities in the various phases, where he uses different tools and creates multiple artefacts. This example scenario is pictured in more details in *D4.2 - Specifications of Learning Analytics, Scaffolding and add-on services*. To this end, the inquiry learning apps have access to a JavaScript Activity Logging library, that allows app developers to integrate an easy but flexible logging of user actions. As described in D4.2, the activity logging in Go-Lab is based on OpenSocial's ActivityStreams approach, which is based on an `actor-verb-object` approach, describing who (actor) did what (verb) with which object. Additional information describes the `target` of an action (which can be seen as the context of an object), the "generator" (which denotes the application that generated the action) and the `provider` (which describes the context of the application, i.e. the Inquiry Learning Space).

The Activity Logging API tries to relieve the app (developer) from keeping track of the information mentioned above, by providing a so-called MetadataHandler. This MetadataHandler administrates and provides access to this (mostly static) information, like target, generator and provider, and is used to enrich the activity log entries with this contextual information. More details on the API and the implementation of the ActionLogger and the MetadataHandler can be found in the source code repository and its documentation.

As an example, if the Concept Mapper application wants to publish the user action of a newly created concept, it is sufficient to create the information of the object and to provide a verb:

```
logObject = {
   "objectType": "concept",
   "id": "4e0eb010-9d65-11e3-a5e2-0800200c9a66",
   "content": "new concept"
  };
  actionLogger("add", logObject);
```

The Action Logging API propagates this action log item to the OpenSocial ActivityStream API, where it is relayed to the Learning Analytics backend.

3.4 Artefact storage library

URL of the source code repository: `https://github.com/go-lab/ils`

Library documentation: `https://github.com/go-lab/ils/wiki/StorageHandler`

The so-called StorageHandler provides a wrapper to access different storage

implementations for client-side (guidance) apps in Go-Lab. Currently, a MemoryStorageHandler (storing resources in memory), a LocalStorageHandler (storing resources in the browser's local storage) and the Vault storage (see above, storing resources in a hidden space in an ILS) are implemented. A resource in Go-Lab consists of a metadata section, which is based on the metadata specified in the OpenSocial ActivityStream API, and a content section, which stores arbitrary content (in particular for the various inquiry learning apps, see below) in JSON. To this end, the StorageHandler makes use of the MetaHandler as well, comparable to the ActionLogger. More details on the API and the implementation can be found in the source code repository and its documentation.

4 Inquiry learning apps

Along the inquiry learning process, inquiry learning apps can be used to support learners. Hereafter we present several guidance apps, which are currently available on the Go-Lab portal, namely the Concept Mapper, the Hypothesis Tool, the Experimental Designer, the Data Viewer app, the Drop File app, the Wiki app and the Resource View app. An additional app, the Conclusion Tool, which displays input form the other tools is currently in early development stage as it is dependent on various apps in this initial release and shall be described only conceptually here.

4.1 Concept Mapper

URL on the lab repository: `http://www.golabz.eu/content/go-lab-concept-mapper`

Supported languages: English, German

Predominant phase(s): Orientation, Conceptualisation

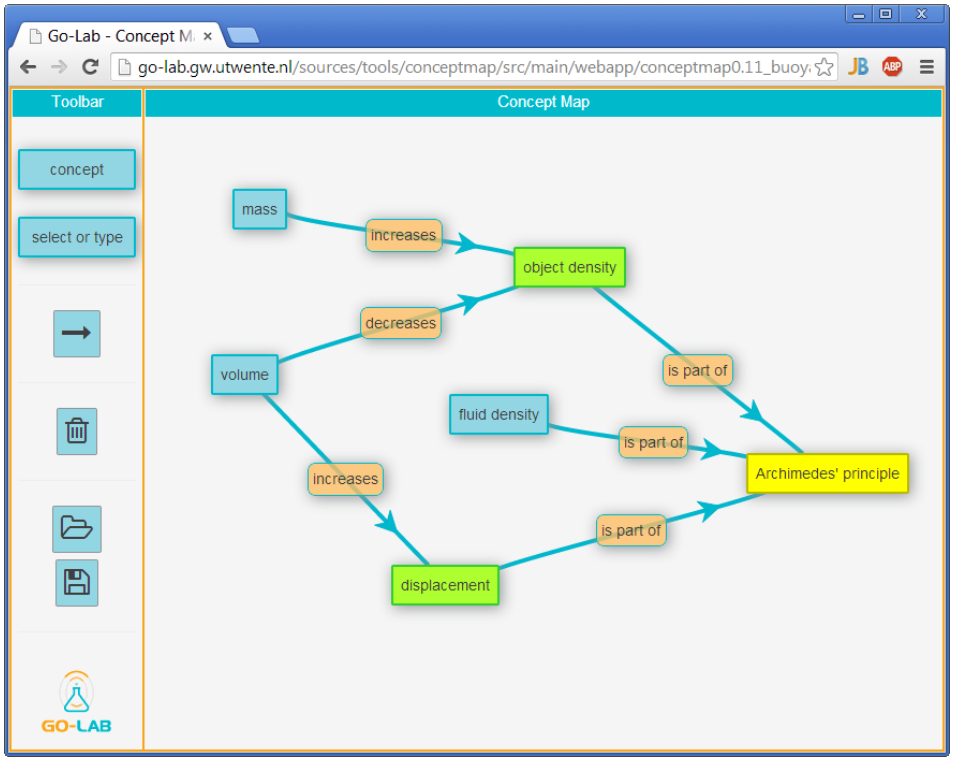

Figure 7: Initial release of the concept mapping tool.

Short description: The Concept Mapper provides features to view, create and edit concept maps with the help of a drag-and-drop user interface. The user can create concept nodes, edit them and create labelled relations between the concept nodes, see Figure 7. The Concept Mapper and its integration as a guidance tool have been inspired by previous experiences (de Jong et al.,

2010).

Usage: Concepts can be created by dragging a template from the toolbar to the map. There, a concept can be changed through free-text editing or by choosing a label from a given set of terms. A concept can be deleted by dragging it to the trashcan icon or through a context menu. Alternatively, concepts can be created by selecting and dragging terms from arbitrary application into the concept map, e.g. from a Wikipedia page in a browser or from a Word document. After clicking the link icon, new relations between concepts can be created through dragging from the source to the target node, or by clicking the source and target node (which is more convenient for touch-based interfaces). A relation can be deleted by creating the same relation again, or by deleting attached concepts. Concept maps can be stored and retrieved using the buttons in the toolbar, where the user will be prompted to select from existing concept maps for retrieval or to name his new concept map for storing.

Integration: The Concept Mapper can be configured, while creating an ILS, with the help of the AppComposer Adaptor. The set of pre-defined terms for the concepts and the labels of the relations can be configured, as well as the availability of a number of features, e.g. creating concepts by dragging terms from external applications.

The Concept Mapper is fully integrated with the Learning Analytics backend, by providing detailed user action logs and reacting to notifications. The user action logging tracks each modification of the concept map, i.e. the creation, modification and deletion of concepts and labels. The Learning Analytics backend can control the Concept Mapper through the Notification Service, by changing its configuration at runtime, similar to the configuration through the AppComposer Adapter at authoring time (see above), or by sending textual messages which are displayed in the form of prompts to the user.

The Concept Mapper stores its artefacts through the Vault in a plain JSON format, making it easily accessible and readable for other tools, e.g. the Hypothesis Scratchpad (to propose concepts as parts of hypotheses) or to the Conclusion Tool (to include a concept map as part of the report). A stored artefact is enriched with metadata information about the author and contextual data, e.g. about the Inquiry Learning Space in which it has been created.

4.2 Hypothesis Scratchpad

URL on the lab repository: `http://www.golabz.eu/app/hypothesis-tool`

Supported languages: English, German

Predominant phase(s): Orientation, Conceptualisation

Short description: The Hypothesis Scratchpad, see Figure 8, helps learners to create and manage their hypotheses. It provides textual building blocks, like

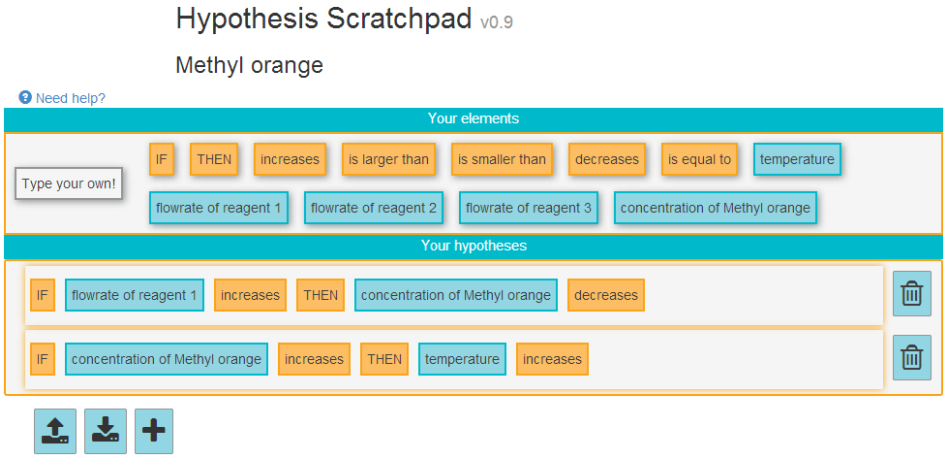

Figure 8: Initial release of the Hypothesis Scratchpad.

'If', 'then', 'increases' etc., as well as domain-dependent terms that are related to the topic of the current learning activity (e.g. 'mass', 'volume', 'density'). These building blocks can be used in a drag-and-drop fashion to create and edit hypotheses. The Hypothesis Scratchpad has been inspired by preceding works (van Joolingen & de Jong, 1991).

Integration: Similar to the Concept Mapper tool (see above), the Hypothesis Scratchpad is integrated with the Learning Analytics backend by providing user action logs and by accepting notifications to change the configuration (e.g. to change the given domain terms) or to display prompts to the learner. Created research questions and hypothesis are stored through the Vault in JSON format, enriched with metadata information as described above.

4.3 Questioning Scratchpad

URL on the lab repository: `http://www.golabz.eu/apps/questioning-scratchpad`

Supported languages: English, German

Predominant phase(s): Orientation, Conceptualisation

Short description: The Questioning Scratchpad (see Figure 9) is a variant of the Hypothesis Scratchpad that allows a more flexible way of creating research questions and hypotheses. The building blocks and domain terms can be used in a similar way as mentioned above, but the learner can freely edit the questions and hypotheses in a text field.

Integration: The implementation is identical to the Hypothesis scratchpad, but the user interface differs.

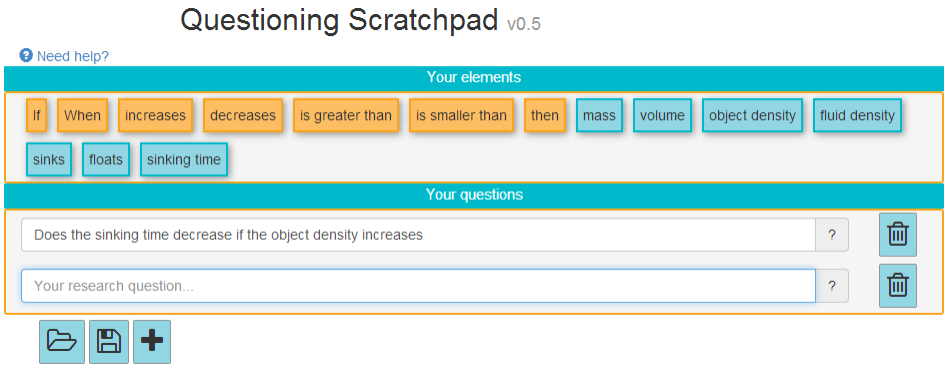

Figure 9: Variant of the Hypothesis Scratchpad: the Questioning Scratchpad.

4.4 Experiment Design Tool

URL in the lab repository: `http://www.golabz.eu/apps/experiment-design-tool`

Supported languages: English, Dutch

Predominant phase(s): Conceptualisation, Investigation

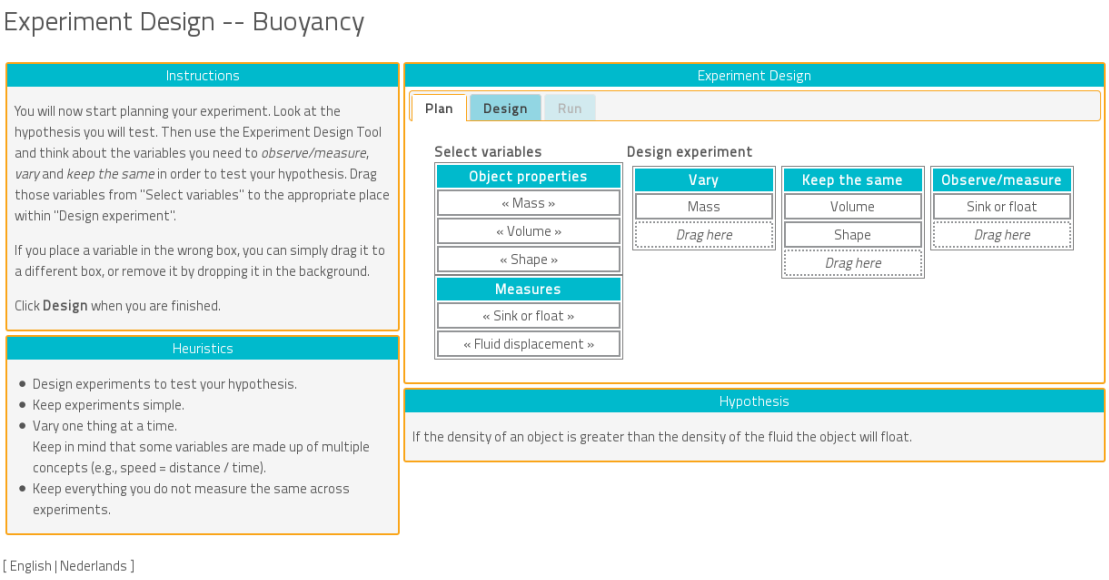

Figure 10: Planning a set of experiments in the Experiment Design Tool. The student has dragged the properties on the left to their roles in the experiment plan (vary, keep the same, observe.

Short description: The Experiment Design Tool, EDT for short, (see Figure 10 & 11) helps students to plan experiments in a laboratory. Normally,

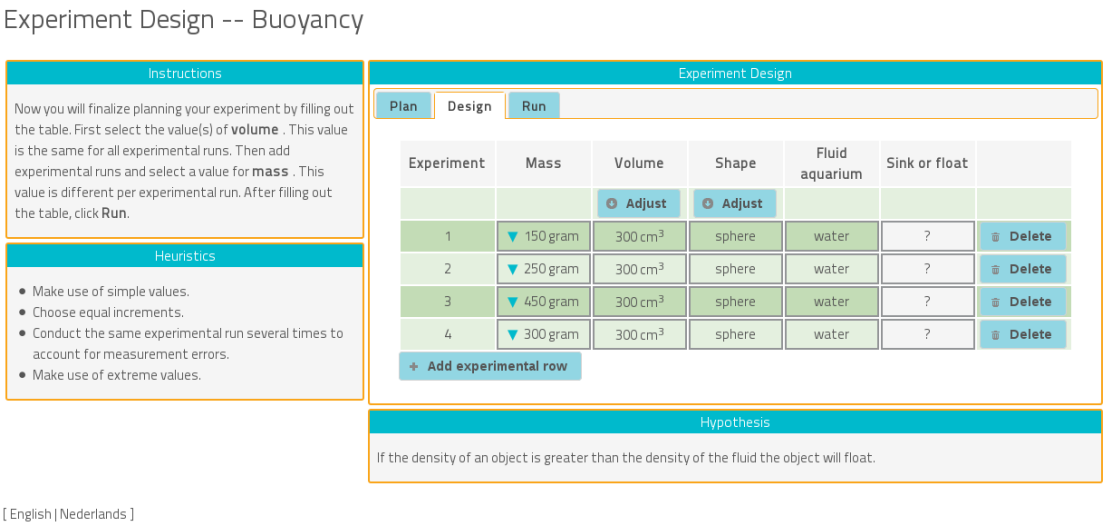

Figure 11: Creating a detailed design of the experiments. The student has selected a single value of the "keep the same" variables and multiple values for mass.

the EDT is started after the student has defined a hypothesis using the Hypothesis Tool. The main purpose of the EDT is to encourage students to think about designing a set of experiments that lets them test the hypothesis. There are three tabs in the EDT for planning experiments, creating a detailed design of the experiments and running and recording the results of the experiments, respectively.

Usage: In the planning tab students select, for each of the properties in the domain of experimentation, which role the property plays in the experiment. Properties can act as a dependent, independent or control variable, or be an observable in an experiment. During *planning* the student is presented with a list of properties relevant for the domain. For a given hypothesis the student has to decide for each property whether it plays the role of dependent variable (vary), independent variable (keep the same), or as an observable. For example, in the buoyancy domain, if a student hypothesises that objects sink because they are heavy, the student could select the *Volume* and *Shape* object properties as independent variables, *Mass* as a dependent variable and *Sink or float* as an observable. Once all properties have been assigned a role, the student moves to the design tab. The *design* tab, is used to specify the detailed design of the experiments. For the independent variables the student selects a single value, *Volume* is 300 cm^3, and *Shape* is *sphere* for example. And for the dependent variables the student selects several values. The EDT shows a row for each experiment specification corresponding to the values selected for the variables. The row possibly also includes control variables the student cannot change, for example the fluid in the aquarium, and a column for the outcomes

of the experiment (the observables). The *run* tab is mainly used to record the outcome of the experiments by filling in the observed value of the property to be measured (e.g., sink or float). The experiments themselves are conducted in the laboratory.

Further technical details are available in Appendix A, see section 6.

4.5 Data Viewer

URL on the lab repository: `http://www.golabz.eu/apps/data-viewer`

Supported languages: English, Dutch

Predominant phase(s): Investigation, Conclusion

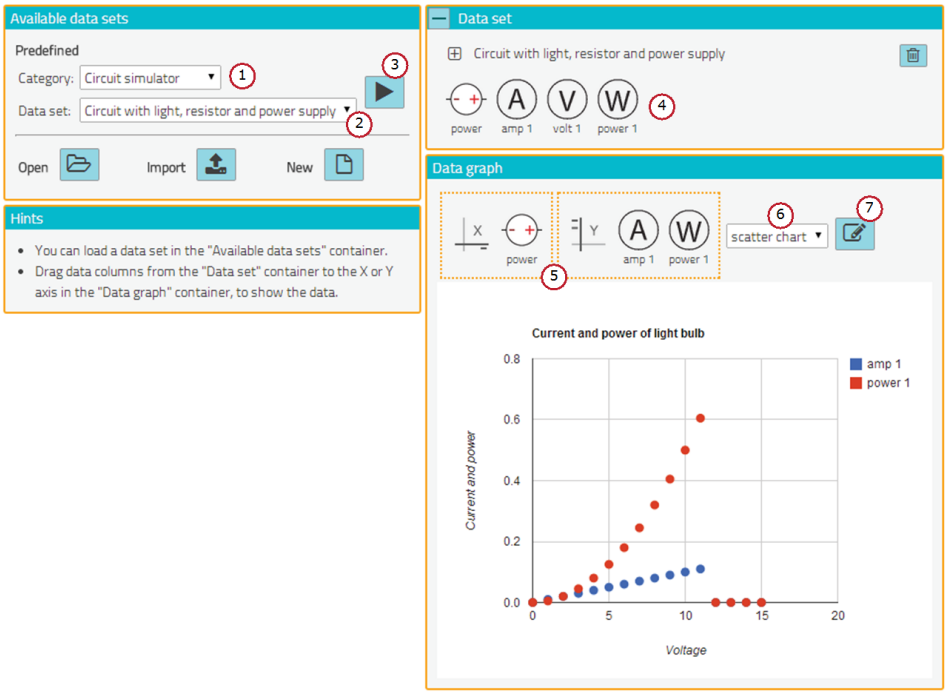

Figure 12: Initial version of the Data Viewer

Short description: The Data Analyser (see Figure 12) allows students to create various types of visualisations of data produced by remote (physical) or virtual online labs. Visualisations may consist of e.g. (numerical) tables, graph plots, bar charts or scatter plots. With the help of the Smart Device integration of remote labs (see deliverable D4.1 for more details on the Smart Device and Smart Gateway paradigm), the Data Viewer constitutes a generic approach to data visualisation in Go-Lab.

Usage: The student can use the Data Viewer to create graph type visualisation of the data sets. These data sets can have been created in real or virtual labs. The Data Viewer is then used independent of the lab. The Data Viewer

can also be used in direct connection with a lab, to show a real time visualisation of the data produced by the lab.

The Data Viewer make use of drag and drop to let the user select what part of the data should be shown. The student can drag variables of the data in the data sets to the X and Y axis and use a simple pop-up menu to select the desired visualisation or use the Chart Editor select an other visualisation and/or the customize the visualisation.

As a detailed example, the following steps will create a data visualisation as shown in the screenshot above:

- First, select a category of available data sets (1). Then, select one of the data sets (2), and click the button (3) to load the dataset. (In this initial release, you can choose from a set of pre-defined data sets. In future, these data sets will be created from the experiments learners conducted in a lab.)
- The data set is depicted by showing the available variables (4), in this example “power”, “amp 1”, “volt 1”, and “power 1”. These variables correspond to what has been measured and recorded in a lab.
- Available variables from the data set can then be dragged to the data graph (5), where they will be plotted automatically. You can drag only one variable to the X-axis, but multiple variables to the Y-axis.
- You can choose from different types of visualisation (6), e.g. a bar chart, a scatter chart or a line chart. Choosing Â´Â´table” will display the values of the selected variables in a data table.
- Details of the visualisation (such as labels, ranges, colors) can be configured in the graph editor dialog (7).

Further technical details are available in Appendix A, see section 6.

4.6 Drop File Tool

URL on the lab repository: `http://www.golabz.eu/apps/file-drop`

Supported languages: English

Predominant phase(s): Discussion, Conclusion

Short description: The Drop File tool, illustrated in Figure 13, allows students to upload their documents into the ILS by drag-and-drop. It also enables the teacher to download and view the uploaded files.

Usage: Typically, a teacher can add the Drop File tool in the Discussion or Conclusion phases of the ILS. Then, the students will be able to upload their assignment reports into the ILS. The teacher can download student reports through this tool as well.

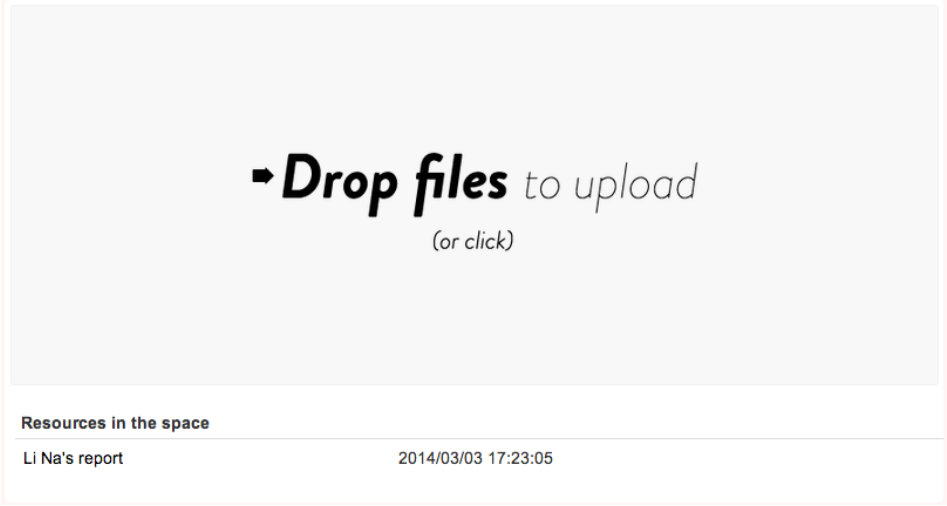

Figure 13: The screenshot of the drop file tool

Technical details: This app uses the document APIs of OpenSocial to create resources in the space where the app has been added. To allow drag-and-drop on the user interface, it uses the Dropzone.js javascript library.[1]

Integration: Currently the Drop File tool stores the uploaded files in the ILS in the name of the ILS creator (i.e., the teacher). In the future, we plan to store the files in the Vault, making it accessible by other apps using the ILS library. Additionally, the student's nickname will be saved in the metadata of the file that she uploads so that the apps in the same ILS will be aware of the author of each report.

4.7 Resource View app

URL on the lab repository: `http://www.golabz.eu/apps/resource-view-app`

Supported languages: English

Predominant phase(s): Orientation, Conceptualisation, Investigation, Discussion & Conclusion

Short description: This app allows to view the resources in a space as a list, as shown in Figure 14. Resources can be files (e.g. slides or a video) but also links. It is possible to download individual resources as well as all of the space resources as a single zip archive. Additionally, a user can enable previews to see content of the resource before downloading.

Usage: A teacher can add the Resource View tool into the inquiry phase, where she has few pictures and reading material. Students can use the Re-

[1] Dropzone.js, `http://www.dropzonejs.com/`

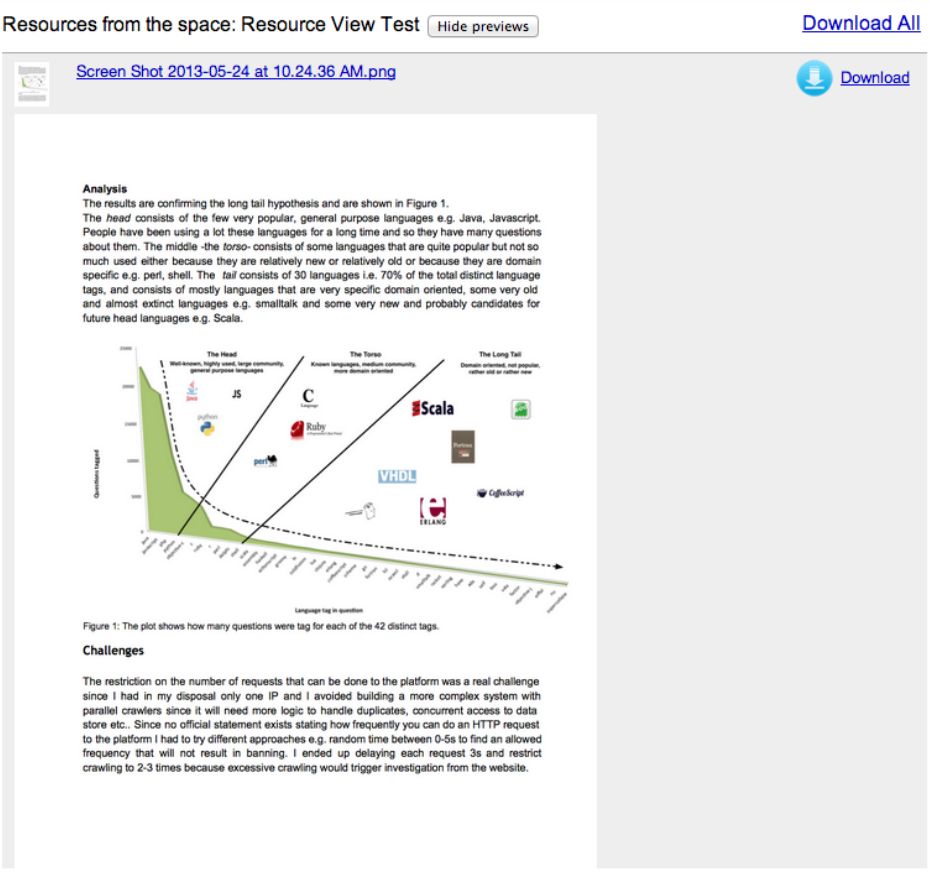

Figure 14: The screenshot of the resource view app

source View tool to preview and download the pictures and the material.

Technical details: The app uses the space API of OpenSocial to get a list of resources in the space where it has been placed.

4.8 Wiki app

URL on the lab repository: `http://www.golabz.eu/apps/wiki-app`

Supported languages: English, German

Predominant phase(s): Conceptualisation, Investigation, Discussion & Conclusion

Short description: This app allows writing or editing texts in the common notion of wiki tools. Students can create new documents or links to other documents. Figure 15 shows a screenshot of an edited page with mark-up.

Usage: The main use of this app is to serve as a text editor, where students create text based learning objects. Students can use this free-notation app for

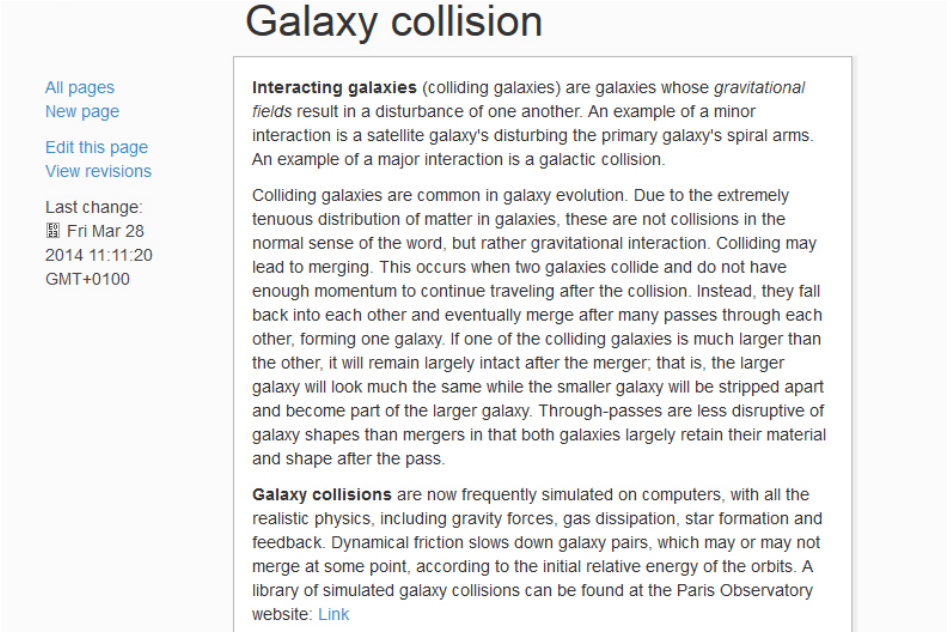

Figure 15: A document with mark-up created through the wiki app.

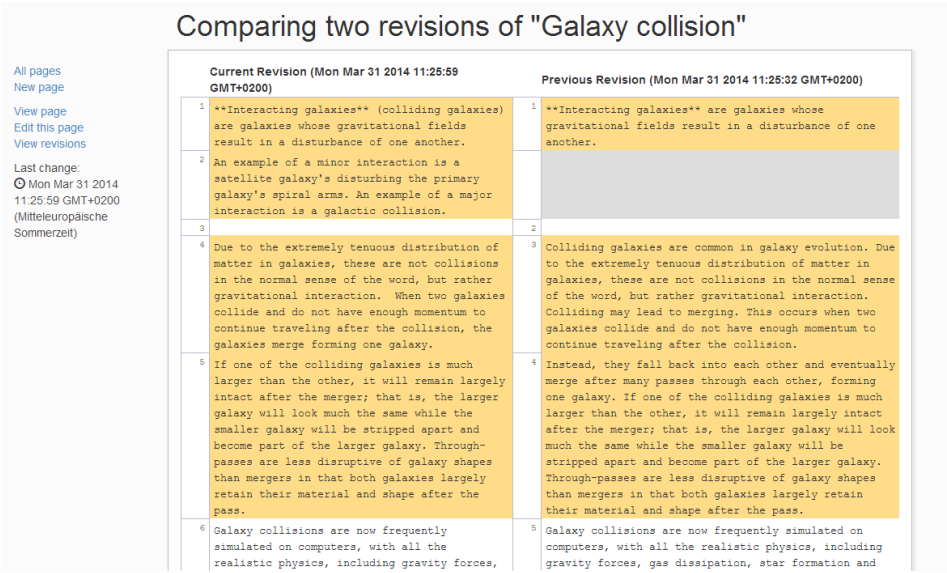

Figure 16: Comparision of two revisions of a document.

several purposes, e.g. answering teacher questions, writing summaries of observations or as a simple reporting tool. Thus, it can be used as an alternative to some more dedicated apps, such as the Hypothesis Scratchpad or the Conclusion tool. This is obviously needed in such inquiry-based learning scenarios in Go-Lab.

Besides the capabilities of a text editor, this tool can be used for reflection and awareness. Wikis create an explicit history of revisions of a document,

which can be used in a learning scenario to reflect on the learning process and progress. On the one hand, a history of revisions can be viewed as a list, on the other hand, different revisions can be compared in the sense of a diff view, which is a tool to visualise the differences of two files graphically. Figure 16 shows the comparison of two revisions in the diff view of the wiki app.

Integration: The wiki app is integrated into the Go-Lab infrastructure: it uses the Go-Lab JavaScript libraries for handling storage, metadata, notifications and logs actions to the back end services. Besides the communication with the back end services it uses the proposed way of internationalisation.

4.9 Conclusion tool

URL of the mockup: `http://go-lab.gw.utwente.nl/experiments/2014-04-D5.3/conclusionTool/build/conclusionTool.html`

Supported languages: English

Predominant phase(s): Conclusion

Short description: The Conclusion tool serves to collect, summarise and present the results and findings from an inquiry learning activity. It is integrated with the previously described tools and can include results from other inquiry phases, e.g. a concept map, hypotheses, or data plots. Learners can elaborate on their results, approve or reject hypotheses and discuss their research questions. Teachers can configure the Conclusion tool by preparing guiding questions to the students.

At the moment of writing this document, only a mock-up version of the Conclusion tool is available. Figure 17 serves to demonstrate the basic idea.

Usage: The student can load the hypothesis he or she wants to validate, along with data from conducted experiments. These experiment data is created by the Data Viewer and can be visualised as a graph plot. The student can enter why the experiment result supports or falsifies the hypothesis or why it does not support or falsifies the hypothesis. When all arguments have been collected, the student must decide what the experiment result says about the hypothesis and enter the result.

After all available experiment results have been scored, the student must draw the final conclusion about the hypothesis. The student gets the summed score of all experiment results and can now enter the arguments for supports, falsifies and unclear. When all arguments have been entered, the final validation result of the hypothesis can be made and entered.

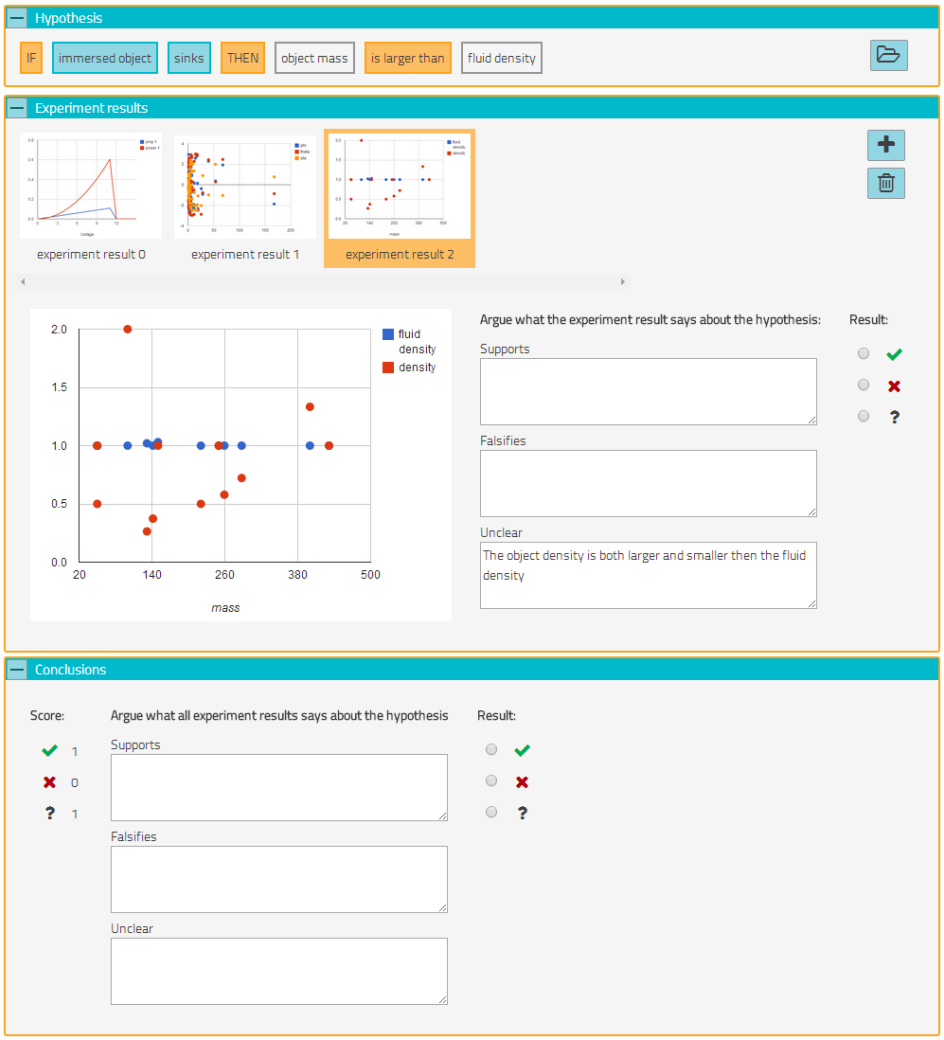

Figure 17: Mockup of the Conclusion tool

5 Conclusion

In this deliverable, we have presented the implementation work completed for the initial release of the personalisation features of the Go-Lab portal and the inquiry learning apps. The requirements for this initial release were based on supporting the Phase A evaluations. Therefore, for personalisation, internationalisation had a high priority.

We elaborated on the implementation of internationalisation in apps and ILS and the procedure of how new translations made on the app composer can be used by the ILS. This is an experimental feature and will be further documented in D5.4.

Additionally, we have described the different mechanisms that allow communication between apps to create a richer user experience. The main mechanisms are inter-app communication, which now also works with drag and drop of objects between apps, and the Vault space for data exchange between apps. Additionally, the user activity logging library is described that enables apps to easily save user activity traces to the learning analytics service.

Finally, the deliverable presented the current state of development of several inquiry-learning apps, namely the Concept Mapper, the Hypothesis Tool, the Experimental Designer, the Conclusion Tool, the Data Viewer app, the Drop File app, the Wiki app and the Resource View app. WP3 is currently evaluating several of these apps and is expected to provide insights and feedback on their design and usefulness in D3.2 in M24.

The release of the personalisation features and inquiry learning apps will be finalised in D5.5 (M32).

References

de Jong, T., van Joolingen, W., Giemza, A., Girault, I., Hoppe, U., Kindermann, J., ... van der Zanden, M. (2010). Learning by creating and exchanging objects: The SCY experience. *British Journal of Educational Technology*, *41*(6), 909–921. doi: 10.1111/j.1467-8535.2010.01121.x

Isaksson, E., & Palmér, M. (2010, September). Usability and inter-widget communication in PLEs. In *Proceedings of the 3rd workshop on mashup personal learning environments.*

van Joolingen, W., & de Jong, T. (1991). Supporting hypothesis generation by learners exploring an interactive computer simulation. *Instructional Science*, *20*, 389–404. doi: 10.1007/BF00116355

6 Appendix A

This appendix provides some more technical details of certain inquiry learning apps, namely the Experiment Design Tool (see Section 4.5) and Data Viewer (see Section 4.4).

6.1 Technical details of the Experiment Design Tool

The experiment design tool is itself domain independent and therefore has to be configured for each domain or sub-domain. A specification of such a configuration for the floating and sinking sub-domain of buoyancy is given in the figure below (using JavaScript object literal notation). So far, specifications have been constructed for several other domains (e.g., Methyl orange, Archimedes' principle, and Electricity). The AppComposer (see D5.2) can assist ILS authors to create and edit the domain specifications.

```
{ domain: "buoyancy",
  name: "Floating or sinking,
  description: "Simulation-based version of the buoyancy experiment",
  object_property_selection: ["mass", "volume", "shape"],
  object_measure_selection: ["sink_or_float", "water_displacement"],
  system_property_selection: ["fluid_aquarium"],
  object_property_specification: [
      { property: "mass",
        initial: 300,
        unit: "gram",
        range: { minimum: 50,
                 maximum: 500,
                 increment: 50
               }
      },
      { property: "volume",
        initial: "200",
        unit: "cm_3",
        range: { minimum: 50,
                 maximum: 500,
                 increment: 50
               }
      },
      { property: "shape",
        initial: "sphere",
        values: ["sphere", "cube"]
      }
    ],
    system_property_values: [
        { property: "fluid_aquarium",
          value: "water"
        },
        { property: "density",
          value: 1.0
        }
    ]
}
```

6.2 Technical details of the Data Viewer

The Data Viewer works independent of the labs, it only visualises data. The Data Viewer is not aware of the meaning of the data or any possible relations in the data.

The Data Viewer uses Google Charts[1] as visualisation engine. Google Charts was chosen because it makes use of HTML5 and SVG (Scalable Vector Graphics), which results in high quality graphics for all devices and all screen resolutions and pixel densities. And Google Charts also offers a format to store data sets, including an interface (Chart Tools Datasource protocol) to access data from outside (such as Google spreadsheets). The data set format makes use of the Google Charts data format and adds a thin wrapper with additional information. Here is an example of a data set:

```
"dataSourceColumns" : [
   "imageInformation": "type": "commonImage", "value": "DC.png", "unit": "V",
   "imageInformation": "type": "commonImage", "value": "A.png", "unit": "A",
   "imageInformation": "type": "commonImage", "value": "A.png", "unit": "A"
],
"dataTable" :
   "cols" : [
      "id": "dcVoltageSource", "label": "power", "pattern": "", "type": "number",
      "id": "amp1", "label": "amp 1", "pattern": "", "type": "number",
      "id": "amp2", "label": "amp 2", "pattern": "", "type": "number"
   ],
   "rows" : [
      "c": [
         "v": 0, "f": "0.000 V",
         "v": 0, "f": "0.000 μA",
         "v": 0, "f": "0.000 mA"
      ],
      "c": [
         "v": 1, "f": "1.000 V",
         "v": 0.01, "f": "10.00 mA",
         "v": 0.02, "f": "20.00 mA"
      ],
      ...
   ],
   "p" : null
```

The Data Viewer is created in a modular way, in order to make it easy for the labs to use the Data Viewer. The Data Viewer has a framework independent data set model, which labs can use create data sets. There is also a simple protocol to transfer data "real-time" from the lab running in the browser to the Data Viewer. Currently the Data Viewer can be used together with the Circuit Simulator. The Circuit Simulator makes use of model components of the Data Viewer. As both

[1] Google Charts, `https://developers.google.com/chart/`

the Data Viewer and the Circuit Simulator are using AngularJS[2], the Circuit Simulator is actually using HTML/JavaScript code from the Data Viewer.

[2] AngularJS, http://angularjs.org/

7 Appendix B

This appendix describes the documentation of the software libraries presented in Section 3. The following sections contain a copy of the documentation available on GitHub to reflect the development stage in the Go-Lab project at the time this deliverable was submitted to the European Commission.

7.1 Documentation of the Inter-app Communication Library

This section contains the documentation of the Inter-app Communication Library available on `https://github.com/go-lab/iwc` *on April 30, 2014.*

Enables the inter widget communication and drag and drop between widgets within a widget container.

7.1.1 How to start

- See two widgets in action here: `https://graasp.epfl.ch/#url=iwc`
- Source code of these widgets: `http://graasp.epfl.ch/gadget/iwc/src.xml` and `http://graasp.epfl.ch/gadget/iwc/dest.xml`

7.1.2 Description

Widgets are often rendered as iframes inside a widget container. The library allows a widget to broadcast messages to other widgets open on the page. In addition, an object can be brought from one widget to another with drag and drop.

7.1.3 Example 1: send data from one widget to another

```
// require the iwc library in both src and dest widgets
<script type="text/javascript" src="http://graasp.epfl.ch/gadget/
libs/iwc.min.js"></script>

// ------- Source gadget --------
// send some data
iwc.publish({
  event: "select",
  type: "json",
  message: {
    data: "some text"
  }
})

// ------- Destination gadget --------
// start listenning on incoming events
iwc.connect(function (envelope, message) {
  var data = message.data
  console.log(data)
}
```

7.1.4 Example 2: drag&drop from one widget to another

```
// require the iwc library in both src and dest widgets
<script type="text/javascript" src="http://graasp.epfl.ch/gadget/
libs/iwc.min.js"></script>

// ------- Source gadget --------
// 'dragme' - id of the DOM node that can be dragged
iwc.draggable('dragme', {
  // function should return the data that you want to transfer
  dragstart: function () {
    return "my data"
  }
})

// ------- Destination gadget --------
// 'droparea' - id of the DOM node that accepts draged elements
iwc.droppable('droparea', {
  drop: function (data) {
    // data - that was passed during the drop
  }
})
```

7.1.5 APIs

Inter Widget Communication

```
// start listenning on incoming events
iwc.connect(function (envelope, message) {
  // envelope - contains extra info about the event
  // message - object passed from one widget to another
})

// broadcasts event to other widgets
iwc.publish({
  event: "select",       // type of event: select, click, etc.
  type: "json",
  message: {             // message object passed
    data: "some text"
  }
})
```

Drag and drop

```
// turns a DOM element into a draggable object
iwc.draggable(elemId, opts)

opts.dragstart = function (drag) {
```

```
  // drag - object on which dragstart is executed
  return {data: "data"} // this data passed to droppable object
}

// turns a DOM element into a droppable object
iwc.droppable(elemId, opts)

opts.drop = function (data, drop, drag) {
  // data - that was passed to droppable object
  // drop - object on where the drop occured
  // drag - object that was dropped
}

opts.dragover = function (drop) {
  // drop - object on where the drop occured
}
```

7.1.6 Thanks

The initial code of inter-widget communication is based on the openapp library by Erik Isaksson and Matthias Palmér.

7.2 Documentation of the ILS library

This section contains the documentation of the Inter-app Communication Library available on `https://github.com/go-lab/ils` on April 30, 2014.

This javascript library provides APIs to developers, allowing apps to access info and data in the ILS they are running in. ### The Inquiry Learning Space Structure An ILS contains by default five phases: Orientation, Conceptualisation, Investigation, Conclusion, Discussion, in addition to a Vault space, and an About space. Each phase could contain a few apps, and the Vault is used for data exchange between apps. For instance, one app could save a data file in the Vault, and another app could read this file from the Vault. This library will allow apps to exchange data via the APIs.

7.2.1 API

getCurrentUser(callback: function(result: string): void): string Gets the nickname of the current student, and returns it in the callback. callback: function(result: string):void a callback function to handle the returned result result: a String object containing the nickname of the user of the user

Result example on success:

```
"Mario"
```

Result example on failure:

```
{
```

```
  "error": "cannot find the nickname"
}
```

getParent(callback: function(result: object): void): object Returns the parent space of the widget in the callback. callback(result): a callback function to handle the returned result result: a JSON object containing the parent space properties

Result example on success:

```
{
  "description": "some text",
  "displayName": "Orientation",
  "id": "9563",
  "metadata": {"type":"Orientation"},
  "objectId": 9563,
  "parentId": 9562,
  "parentType": "@space",
  "profileUrl": "http://localhost:3000/#item=space_9563",
  "spacetype": "folder",
  "updated": "2014-03-17T12:43:28+01:00",
  "visibilityLevel": "public"
}
```

Result example on failure:

```
{
  "error": "cannot get the parent"
}
```

getIls(callback: function(result: object): void): object Returns the current Ils in the callback. callback(result): a callback function to handle the returned result result: a JSON object containing the ILS space properties

Result example on success:

```
{
  "description": "some text",
  "displayName": "ils_example",
  "id": "9562",
  "metadata": "",
  "objectId": 9562,
  "parentId": 28,
  "parentType": "@person",
  "profileUrl": "http://localhost:3000/#item=space_9562",
```

```
  "spacetype": "ils",
  "updated": "2014-03-17T12:43:28+01:00",
  "visibilityLevel": "public"
}
```


Result example on failure:


```
{
  "error": "cannot get the Ils"
}
```


getParentInquiryPhase(callback: function(result: object): void): object Returns the type of the current phase. callback(result): a callback function to handle the returned result result: a String object containing the name of the phase the app is running in

Result example on success:


```
"Orientation"
```


Result example on failure:


```
{
  "error": "cannot get the parent inquiry phase"
}
```


getVault(callback: function(result: object): void): object Returns the Vault space of the current Ils in the callback. callback(result): a callback function to handle the returned result result: a JSON object containing the Vault space properties

Result example on success:


```
{
  "description": "some text",
  "displayName": "Vault",
  "id": "9569",
  "metadata": {"type":"Vault"},
  "objectId": 9569,
  "parentId": 9562,
  "parentType": "@space",
  "profileUrl": "http://localhost:3000/#item=space_9569",
  "spacetype": "group",
  "updated": "2014-03-17T12:43:28+01:00",
  "visibilityLevel": "hidden"
}
```

Result example on failure:


```
{
  "error": "cannot get the vault"
}
```


listVault(callback: function(result: object): void): object Returns all the resources in the Vault in the callback. callback(result): a callback function to handle the returned result result: an array of JSON objects, each JSON object containing properties of the respective resource in the Vault

Result example on success:


```
[
  {
    "id": 15312,
    "displayName": "example",
    "name": "example",
    "objectId": 15312,
    "parentId": "9569",
    "parentType": "@space",
    "profileUrl": "http://localhost:3000/#item=asset_15312",
    "thumbnailUrl": "/asset/picture/15312/thumb/image-0.jpg",
    "updated": "2014/03/26 13:26:51 +0100",
    "mimeType": "txt",
    "metadata": {"username": "Mario"},
    "data": "",
    "attachment": {}
  },
  {
    "id": 15313,
    "displayName": "example2",
    "name": "example2",
    "objectId": 15313,
    "parentId": "9569",
    "parentType": "@space",
    "profileUrl": "http://localhost:3000/#item=asset_15313",
    "thumbnailUrl": "/asset/picture/15313/thumb/image-0.jpg",
    "updated": "2014/03/26 13:26:51 +0100",
    "mimeType": "txt",
    "metadata": {"username": "Luigi"},
    "data": "",
    "attachment": {}
  }
]
```

Result example on failure:


```
{
  "error": "cannot get the resources in the vault"
}
```


readResource(resourceId: number,callback: function(result: object): void): object Returns the resource with the resourceId in the callback. callback(result): a callback function to handle the returned result result: a JSON object containing the properties of the resource references by resourceId

Result example on success:


```
{
  "id": 15312,
  "displayName": "example",
  "name": "example",
  "description": "some text",
  "objectId": 15312,
  "parentId": "9569",
  "parentType": "@space",
  "profileUrl": "http://localhost:3000/#item=asset_15312",
  "thumbnailUrl": "/asset/picture/15312/thumb/image-0.jpg",
  "updated": "2014/03/26 13:26:51 +0100",
  "mimeType": "txt",
  "metadata": {"username": "Mario"},
  "content": {
       "concepts": [
        {
            "x": 297,
            "y": 188,
            "content": "energy",
            "id": "7f800d79-cd66-2167-724c-6c1cda7abc5e",
            "type": "ut_tools_conceptmapper_conceptSelector",
            "colorClass": "ut_tools_conceptmapper_blue"
        },
        {
            "x": 652,
            "y": 238,
            "content": "thermodynamic temperature",
            "id": "a1ad6ace-c722-ffa9-f58e-b4169acdb4e3",
            "type": "ut_tools_conceptmapper_conceptSelector",
            "colorClass": "ut_tools_conceptmapper_blue"
        }
    ],
    "relations": [
```

```
        {
            "source": "7f800d79-cd66-2167-724c-6c1cda7abc5e",
            "target": "a1ad6ace-c722-ffa9-f58e-b4169acdb4e3",
            "id": "con_71",
            "content": "influences"
        }
    ]
  }
  "data": "",
  "attachment": {}
}
```

Result example on failure:

```
{
  "error": "cannot get the resource"
}
```

createResource(resourceName: string, content: object, callback: function(result: object): void): object Create a resource in the Vault, and returns the new resource just created.

resourceName example:

```
"example file"
```

content should be any JSON, see example below:

```
{
    "concepts": [
        {
            "x": 297,
            "y": 188,
            "content": "energy",
            "id": "7f800d79-cd66-2167-724c-6c1cda7abc5e",
            "type": "ut_tools_conceptmapper_conceptSelector",
            "colorClass": "ut_tools_conceptmapper_blue"
        },
        {
            "x": 652,
            "y": 238,
            "content": "thermodynamic temperature",
            "id": "a1ad6ace-c722-ffa9-f58e-b4169acdb4e3",
            "type": "ut_tools_conceptmapper_conceptSelector",
            "colorClass": "ut_tools_conceptmapper_blue"
```

```
        }
    ],
    "relations": [
        {
            "source": "7f800d79-cd66-2167-724c-6c1cda7abc5e",
            "target": "a1ad6ace-c722-ffa9-f58e-b4169acdb4e3",
            "id": "con_71",
            "content": "influences"
        }
    ]
}
```

callback(result): a callback function to handle the returned result result: a JSON object representing the creating resource in the Vault

Result example on success:

```
{
  "id": 15312,
  "displayName": "example",
  "name": "example",
  "objectId": 15312,
  "parentId": "9569",
  "parentType": "@space",
  "profileUrl": "http://localhost:3000/#item=asset_15312",
  "thumbnailUrl": "/asset/picture/15312/thumb/image-0.jpg",
  "updated": "2014/03/26 13:26:51 +0100",
  "mimeType": "txt",
  "metadata": {"username": "Mario"},
  "data": "",
  "attachment": {}
}
```

Result example on failure:

```
{
  "error": "cannot create resource in the vault"
}
```

7.2.2 How to Use

This library relies on Jquery, so Jquery should be included in the widget headers before the use of this library. Below is an example of how to use the getIls API. For more information on how to use this library, please refer to the the demo widget: https://github.com/go-lab/ils/blob/master/demo/vault_demo.xml

```
//include jquery library in your widget source
<script type="text/javascript" src="http://graasp.epfl.ch/gadget/
libs/jquery-1.8.0.min.js"></script>
//include the ILS library in your widget source
<script type="text/javascript" src="http://graasp.epfl.ch/ils_lib/
main/ils.js"></script>

//example of calling the getCurrentUser api
ils.getCurrentUser(function(current_user){
  // write your code here to use the current_user
  // this line simply prints the current user's nickname
  console.log(current_user);
});

//example of calling the getIls api
ils.getIls(function(ils_space){
  // write your code here to use the ils_space
  // this line simply prints the ILS space JSON representation
  console.log(ils_space);
});

//example of calling readResource api
ils.readResource(15339, function(resource){
  // write your code here to use the resource
  // this line simply prints the returned resource's
  //JSON representation
  console.log(resource);
});

//example of calling createResource api
var example_content = {
   "concepts": [
      {
        "x": 297,
        "y": 188,
        "content": "energy",
        "id": "7f800d79-cd66-2167-724c-6c1cda7abc5e",
        "type": "ut_tools_conceptmapper_conceptSelector",
        "colorClass": "ut_tools_conceptmapper_blue"
      },
      {
        "x": 652,
        "y": 238,
        "content": "thermodynamic temperature",
        "id": "a1ad6ace-c722-ffa9-f58e-b4169acdb4e3",
        "type": "ut_tools_conceptmapper_conceptSelector",
```

```
        "colorClass": "ut_tools_conceptmapper_blue"
      }
   ],
   "relations": [
     {
        "source": "7f800d79-cd66-2167-724c-6c1cda7abc5e",
        "target": "a1ad6ace-c722-ffa9-f58e-b4169acdb4e3",
        "id": "con_71",
        "content": "influences"
     }
   ]
};

// create a resource in the Vault
ils.createResource("test", example_content, function(resource){
  // write your code here to use the resource
  // this line simply prints the created resource's
  // JSON representation
  console.log(resource);
});
```

7.3 Documentation of the User Activity Logging Library

This section contains the documentation of the User Activity Logging Library available on `https://github.com/go-lab/ils` *on April 30, 2014.*

The ActionLogger provides an easy mechanism for logging students' activities in Go-Lab. Since the Go-Lab portal is built on top of Apache Shindig, an OpenSocial container implementation, and since OpenSocial makes us of the Activity Streams specifications for activity logging, user action logging in Go-Lab is handled through Acitivity Streams as well. The ActionLogger requires a MetadataHandler at construction time, and can be configured towards various "logging targets", i.e. endpoints to receive the occurring action log information. The actual use of the ActionLogger at runtime mostly consists of repeatedly calling the function log(verb, object) which creates an ActivityStream object from the given information, and adds information from the MetadataHandler. The ActivityStream object is then relayed to the specified logging target. The Activity Streams format utilises an "actor-verb-object" metaphor, with optional additional elements like target, generator, and provider (see also `MetadataHandler`). During typical tool usage in Go-Lab, most of these elements remain static (over the course of an activity), leaving only the "verb" and the "object" to be used as a parameter in the ActionLogger's function calls. Typically, each tool instance creates and uses its own ActionLogger instance, along with its own MetadataHandler. An example action log object can be found here.

7.3.1 Creating an ActionLogger

7.3.2 Logging Targets

In order to conveniently change the loggers behaviour (e.g., for development, testing or production use), the action logger can be configured towards various 'targets' (see API below). Currently, five logging targets are implemented: * "null": Discards all incoming action logs, doing nothing. * "console": Prints the full Activity Streams object to the JavaScript console. * "consoleShort": Prints only the verb, object and object identifier to the console. * "dufftown": Relays the Activity Streams objects directly to a Learning Analytics backend server of one the project consortium's member. * "opensocial": Relays the Activity Streams object to the underlying Go-Lab platform, where it can be processed and relayed further.

7.3.3 API

setLoggingTarget(loggingTarget: function(activityStreamObject: object): void): void Sets the logging target in the ActionLogger. The logging target is a function that processes (and typically relays) an Activity Streams object. `loggingTarget: function(activityStreamObject: object): void` A function that accepts and processes an Activity Streams object. Available implementations are e.g. consoleLogging() or opensocialLogging(), but you can also pass custom functions.

setLoggingTargetByName(loggingTargetName: string): void Sets the logging target in the ActionLogger, using a string representation of the implemented logging targets (see above). This function is convenient to use when the setting for the logging target is e.g. read from a configuration file. `loggingTargetName: string` The name of the logging target to be set. Currently, accepted values are "null", "console", "consoleShort", "dufftown", and "opensocial". Any other value will be interpreted as "null".

log(verb: string, logObject: object): void Creates an Activity Streams object from the given verb, object and from the information in the MetadataHandler, and calls the logging target function with this Activity Streams object. `verb: string` A textual representation of the activity, in the fashion of a verb. Typical examples would be "create", "update", or "remove". `logObject: object` A JSON object describing the "object" of an activity, following the Activity Streams specifications. Typical examples would be a concept in a Concept Map, or a hypothesis from the Hypothesis Scratchpad tool.

7.4 Documentation of the Artefact Storage Library

This section contains the documentation of the Artefact Storage Library available on `https://github.com/go-lab/ils` on April 30, 2014.

The StorageHandler provides a wrapper to access different storage implementations for client-side (guidance) apps in Go-Lab. Currently, a MemoryStor-

ageHandler (storing resources in memory) and a LocalStorageHandler (storing resources in the browser's local storage) are implemented; a "vault" storage (storing resources to Graasp) will follow.

7.4.1 What is a resource?

In Go-Lab, in the context of the StorageHandler, a resource is a data artefact created by a guidance app or a lab. Examples are a set of hypotheses, a concept map, or a numerical dataset. A resource has a unique identifier, a metadata section (based on information from the MetadataHandler), and a content section, which contains arbitrary, tool specific data in JSON format. An example resource can be found here.

Using the StorageHandler is closely connected to the `MetadataHandler`, which stores and gives access to information like username, object name and type, id of the current ILS etc., which is needed by the StorageHandler to create the object's metadata.

7.4.2 Creating a StorageHandler

When creating a StorageHandler object, you can choose between a MemoryStorageHandler and a LocalStorageHandler (and a VaultStorageHandler in future). All three handlers require a `MetadataHandler` at construction time.

or

7.4.3 API

getMetadataHandler(): MetadataHandler Returns the MetadataHandler.

readResource(resourceId: string, callback: function(error: string, resource: object): void): void Returns a resource through a callback, identified by its resourceId. `resourceId: string` The (unique) identifier of the resource to be retrieved. `callback: function(error: string, resource: object):void` A callback function to return the result. On success, 'error' is undefined, else 'resource' is undefined.

createResource(content: object, callback: function(error: string, resource: object):void): void Creates a resource by bundling the given content and metadata, and stores it. `content: object` An arbitrary JSON representation of the data to be stored. `callback: function(error: string, resource:object):void` A callback function to return the result. On success, 'error' is undefined, else 'resource' is undefined. The 'resource' bundles the 'content', metadata from the MetadataHandler, a new resource identifier (a version 4 random UUID, accessible through resource.id, see example below)and the creation date.

updateResource(resourceId: string, content: object, callback: function(error: string, resource: object): void): void Updates a resource with new content.

`resourceId: string` The identifier of the resource to be updated. `content: object` The new content of the resource. `callback: function(error: string, resource: object): void` A callback function to return the result. On success, 'error' is undefined, else 'resource' is undefined.

listResourceMetaDatas(callback: function(error: string, metadatas: object[]): void): void Returns an array with all available resource metadata (in an ILS). `callback(error: string, metadatas: object[]): void` A callback function to handle the returned result. On success, 'error' is undefined, else 'metadatas' is undefined. The returned array contains objects of the form {id: "...", metadata: {...}}, where the metadata sub-objects correspond with the metadata sections of the available resources.

listResourceIds(callback: function(error: string, ids: string[]): void): void Returns an array of all available resource identifiers. `callback: function(error:string, ids: string[]): void` A callback function to handle the returned result. On success, 'error' is undefined, else 'ids' is undefined. 'ids' contains all available resource identifiers of the current storage.

resourceExists(resourceId: string, callback: function(error: string, exists: boolean): void): void Checks if a resource with a given identifier exists. `resourceId: string` The identifier of the resource to check. `callback: function(error: string, exists: boolean): void` A callback function to handle the returned result. On a successful call, 'error' is undefined, else 'exists' is undefined. 'exists' is 'true', if a resource with the given resource identifier exists, 'false' otherwise.

readLatestResource(resourceType: string, callback: function(error: string, resource: object): void): void A convenience function that returns the latest stored resource of a given type. This function searches for resources where 'metadata.target.objectType' equals the parameter 'resourceType', and returns the latest matching resource. If 'resourceType' is left 'undefined', any type will match. `objectType: string` The type of the resource (cf. metadata.target.objectType). `callback: function(error: string, resource: object): void` A callback function to return the result. On success, 'error' is undefined, else 'resource' is undefined.

7.4.4 Usage example

7.4.5 Notes

A tool using the StorageHandler is responsible to propagate potential changes in the metadata to the MetadataHandler. E.g., if the document name (metadata.target.displayName) changes during a save operation, the tool needs to update this information in the MetadataHandler. If a resource is retrieved from the StorageHandler, and if this resource will be the main working document for

a tool, the retrieved metadata has to be set in the MetadataHandler. This is necessary for a consistent handling of resource metadata (which is also being used in e.g. the ActionLogger).

7.5 Documentation of the Metadata Handler Library

This section contains the documentation of the Metadata Handler Library available on `https://github.com/go-lab/ils` *on April 30, 2014.*

The MetadataHandler encapsulates and provides access to a set of metadata items needed by client-side apps in Go-Lab. In particular, these metadata items are also required by the `ActionLogger` (to add it to the action log) and by the `StorageHandler` (to add it to stored artefacts). Typical metadata items would be the name and identifier or the current user, name and identifier of the current working document, the current ILS etc. Every instance of an app creates its own MetadataHandler, and passes it on to the ActionLogger and StorageHandler. An app is responsible to update the information in the MetadataHandler in case of changes. At the moment of writing this, this is only necessary when an artefact is stored and given a new name (cf. setTargetDisplayName()). ### What's in the metadata? The metadata is based on the Activity Streams specification, since this is used for action logging as well and thus brings consistency by design. The metadata consists of the following sections: * actor: holds information about the current user * target: holds information about the current working document * generator: holds information about the current app * provider: holds information about the current context (i.e., the ILS)

7.5.1 Technical representation

In the current implementation, the metadata is represented as a JSON object. Using JSON integrates well with other technologies for the development of web-based applications, in particular with JavaScript. In addition, JSON can be serialised and parsed easily, while it still remains human-readable. The following MetadataHandler API returns and accepts the metadata items or metadata sections in JSON. An example can be found here.

7.5.2 Creating a MetadataHandler

Currently, the MetadataHandler is created with a given set of metadata items. In future implementations, an increasing number of items will be automatically set through the ILS Metawidget API. For this reason, the construction of the MetadataHandler is handled asynchronously, requiring a callback function at construction time. The following code fragment illustrates this approach:

7.5.3 API

setMetadata(newMetadata: object): void Sets a new, complete set of metadata in the MetadataHandler. `newMetadata: object` The new metadata JSON object to be set in the MetadataHandler

getMetadata(): object Returns the complete JSON metadata object.

getActor(): object Returns the 'actor' sub-section of the metadata.

getGenerator(): object Returns the 'generator' sub-section of the metadata.

getProvider(): object Returns the 'provider' sub-section of the metadata.

getTarget(): object Returns the 'provider' sub-section of the metadata.

setTarget(newTarget: object): void Sets a new 'target' sub-section in the metadata object. `newTarget: object` The new 'target' sub-section JSON object to be set in the MetadataHandler Note: Typically, only the 'target' metadata (describing the current working document) changes during a tool's lifecycle. If other metadata sections change dynamically in future, more functions like this will be added.

getTargetDisplayName(): string A convenience function that directly returns the displayName of the 'target' sub-section (i.e. the current document name).

setTargetDisplayName(newName: string): void A convenience function to directly set a new displayName of the 'target' sub-section (i.e. setting a new document name). `newName: string` The new target's displayName to be set in the MetadataHandler

7.6 Documentation of the Metadata Handler Library

This section contains the documentation of the Metadata Handler Library available on `https://github.com/go-lab/ils` on April 30, 2014.

The MetadataHandler encapsulates and provides access to a set of metadata items needed by client-side apps in Go-Lab. In particular, these metadata items are also required by the `ActionLogger` (to add it to the action log) and by the `StorageHandler` (to add it to stored artefacts). Typical metadata items would be the name and identifier or the current user, name and identifier of the current working document, the current ILS etc. Every instance of an app creates its own MetadataHandler, and passes it on to the ActionLogger and StorageHandler. An app is responsible to update the information in the MetadataHandler in case of changes. At the moment of writing this, this is only necessary when an artefact is stored and given a new name (cf. setTargetDisplayName()).

7.6.1 What's in the metadata?

The metadata is based on the Activity Streams specification, since this is used for action logging as well and thus brings consistency by design. The metadata consists of the following sections:

- actor: holds information about the current user

- target: holds information about the current working document
- generator: holds information about the current app
- provider: holds information about the current context (i.e., the ILS)

7.6.2 Technical representation

In the current implementation, the metadata is represented as a JSON object. Using JSON integrates well with other technologies for the development of web-based applications, in particular with JavaScript. In addition, JSON can be serialised and parsed easily, while it still remains human-readable. The following MetadataHandler API returns and accepts the metadata items or metadata sections in JSON. An example can be found here.

7.6.3 Creating a MetadataHandler

Currently, the MetadataHandler is created with a given set of metadata items. In future implementations, an increasing number of items will be automatically set through the ILS Metawidget API. For this reason, the construction of the MetadataHandler is handled asynchronously, requiring a callback function at construction time. The following code fragment illustrates this approach:

7.6.4 API

setMetadata(newMetadata: object): void Sets a new, complete set of metadata in the MetadataHandler. `newMetadata: object` The new metadata JSON object to be set in the MetadataHandler

getMetadata(): object Returns the complete JSON metadata object.

getActor(): object Returns the 'actor' sub-section of the metadata.

getGenerator(): object Returns the 'generator' sub-section of the metadata.

getProvider(): object Returns the 'provider' sub-section of the metadata.

getTarget(): object Returns the 'provider' sub-section of the metadata.

setTarget(newTarget: object): void Sets a new 'target' sub-section in the metadata object. `newTarget: object` The new 'target' sub-section JSON object to be set in the MetadataHandler Note: Typically, only the 'target' metadata (describing the current working document) changes during a tool's lifecycle. If other metadata sections change dynamically in future, more functions like this will be added.

getTargetDisplayName(): string A convenience function that directly returns the displayName of the 'target' sub-section (i.e. the current document name).

setTargetDisplayName(newName: string): void A convenience function to directly set a new displayName of the 'target' sub-section (i.e. setting a new document name). `newName: string` The new target's displayName to be set in the MetadataHandler.

7.7 Documentation of the Notification Handler Library

This section contains the documentation of the Notification Handler Library available on `https://github.com/go-lab/ils` on April 30, 2014.

7.8 NotificationClient

Through the notificationClient, ILS apps can register with the Learning Analytics Backend Services to receive notifications sent by the Notification Broker.

The connection happens over Websockets via socket.io. A combination of the provider id, actor id and generator id, taken from the metadataHandler you pass on instantiation identifies the notification client instance on the server.

7.8.1 API

NotificationClient(metadataHandler)

Instantiates the notification client. A websockets connection to the server will be established. The metadataHandler supplies the data (actor id, generator id and provider id) that enables the server to identify the socket connection to the specific instance of the notification client uniquely. Actor is the current user. Generator is the app. Provider is the ILS in which an app lives.

notificationClient.register(premise, handle)

Registers listeners with the notificationClient to receive notifications.

`premise` is a function that receives the notification and returns `true` if the notification is of interest, and `false` if it is not.

`handle` is a function that receives the notification if `premise` returned `true`. This is where you put the logic that takes action when a notification arrives. If `false` is returned from the function, the notification will be passed on to other listeners registered after this one. If `true` is returned, the listener is considered greedy, and the notification is not passed on to following listeners.

notificationClient.processNotification(notification)

The method is called if the client receives a notification. All registered listeners will be iterated and if the notification is of interest for a listener (determined by the premise-function as mentioned before), the registered callback will be performed. The notification will be passed as a parameter to the callback.

7.8.2 Example notification

```
{
    type : "prompt",
    // other possible types are "configuration" or "resource."
    importance : "8",
    // importance level with range [1, ..., 10].
    target : {
        type : "app",
        id : "provider_id-actor_id-generator_id"
        // unique id to address a particular app.
    },
    content : {
        text : "This is an example message"
        // message content if notification type is "prompt".
        url : "http://..."
        // url if notification type is "resource".
        configuration: { App configuration as property-value list }
        }
}
```

7.8.3 Usage Example

```
var metadata = {
  actor:     { id: '123' },
  generator: { id: 'abc' },
  provider:  { id: 'xyz' },
};
var metadataHandler    =
    new golab.ils.metadata.GoLabMetadataHandler(metadata);
var notificationClient =
    new ude.commons.NotificationClient(metadataHandler);

var premise = function(notification) {
  // depends on the format of the notification. This is just an example.
  return (notification.target === xyz-123-abc)
           && (notification.importance > 5);
};

var handle = function(notification) {
  alert('The notification I care for:' + JSON.stringify(notification));
};

notificationClient.register(premise, handle);
```