The AppComposer Web Application for School Teachers: A Platform for Translating and Adapting Educational Web Applications

Luis Rodriguez-Gil*, Pablo Orduña*, Lars Bollen†,
Sten Govaerts‡, Adrian Holzer‡, Denis Gillet‡,
Diego López-de-Ipiña* and Javier Garcia-Zubia*
*DeustoTech – Deusto Institute of Technology
University of Deusto, Bilbao, Spain
Email: {luis.rodriguezgil,pablo.orduna,dipina,zubia}@deusto.es
†University of Twente, The Netherlands
Email: l.bollen@utwente.nl
‡École Polytechnique Fédérale de Lausanne, EPFL, Switzerland
Email: {sten.govaerts,adrian.holzer,denis.gillet}@epfl.ch

***Abstract*—Developing educational apps that cover a wide range of learning contexts and languages is a challenging task. In this paper, we introduce the AppComposer Web app to address this issue. The AppComposer aims at empowering teachers to easily translate and adapt existing apps that fit their educational contexts. Developers do not need to provide extensive translations and configurations of their apps and can simply follow certain guidelines to make their apps translatable and adaptable by the AppComposer. Since the AppComposer makes use of the standard internationalization specification used by OpenSocial, even external apps can be translated without contacting the original developer.**

I. Introduction

In today's world, science and technology are of utmost importance for the prosperity of any economy. To promote growth, the world requires initiatives that can bring science and technology to students, so that they can learn, be motivated and consider those fields as their career paths [1]. In this context, the European Commission is funding Go-Lab, a large scale research project on federated online laboratories which is meant to promote and support STEM (Science, Technology, Engineering and Mathematics) among young students [2], [3]. Go-Lab is meant to enable inquiry-based learning at schools by offering online laboratory experiments and other support tools, which are to be accessible through OpenSocial web apps on the *Go-Lab Portal* [4].[1] Some online laboratory apps provide access to real hardware (e.g. Faulkes Telescope), others to simulations (e.g. Gear Sketch). Support tools include scaffolds (e.g. Concept Mapper) and more general tools (e.g. InputBox). These apps can then be integrated in an inquiry learning space (ILS) [5], which is an online, collaborative space used for an inquiry learning activity [3].

Figure 1 illustrates an example of an ILS where a teacher used the Concept Mapper and the Gear Sketch for her English speaking class. In order for the Concept Mapper to be used in another context, such as in a class about Electricity in Spain, its user interface would need to be translated in Spanish and its predefined concepts would need to be adapted to fit the domain-specific terminology. Unfortunately, the effort required for an app developer to provide support for a wide range of contexts and languages is substantial and teachers usually lack the coding skills to extend existing apps.

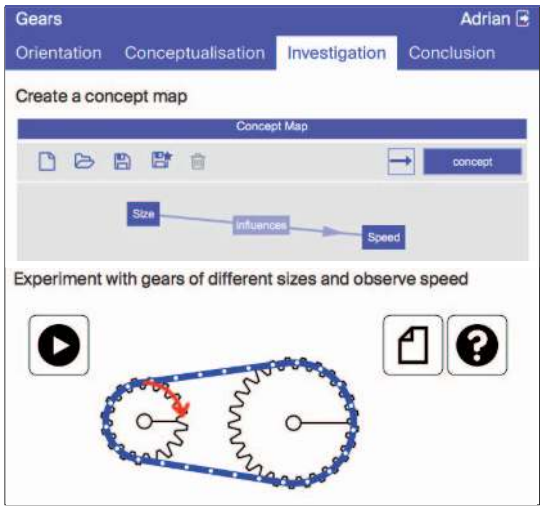

Fig. 1. An example of an ILS with the Concept Mapper and the Gear Sketch.

The AppComposer, which is the main contribution of this paper, precisely addresses this issue by providing the means for non-technical users (teachers) to translate and adapt existing apps to meet their particular contexts. With the AppComposer, developers do not need to provide extensive translations and configurations of their apps and can simply follow guidelines to make their apps translatable and adaptable. Since internationalization is well-specified in OpenSocial, even external apps can be translated without contacting the original developer. The AppComposer consists of a Translator and an Adaptor module. The Translator module, as the name suggests, is used to translate apps in different languages. Figure 2

[1] http://golabz.eu

978-1-4799-1908-6/15/$31.00 ©2015 IEEE
18-20 March 2015, Tallinn University of Technology, Tallinn, Estonia
2015 IEEE Global Engineering Education Conference (EDUCON)

shows the Concept Mapper being translated from English into Spanish. The Adaptor module allows to configure apps for different contexts. Figure 3 shows how the Concept Mapper can be configured so that it can be used in an Electricity class.

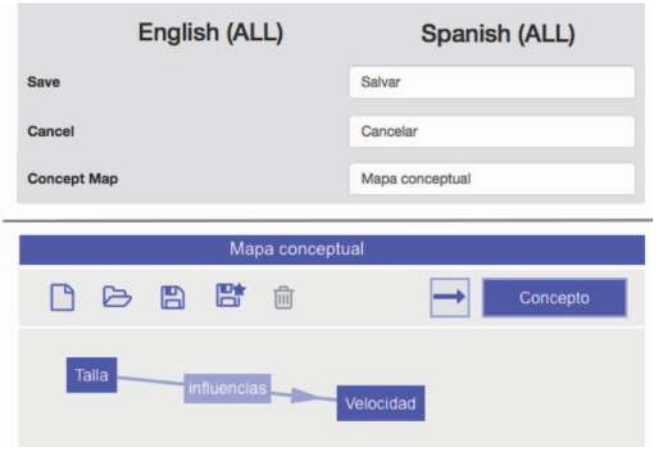

Fig. 2. The Concept Mapper being translated in the AppComposer (above) and the result (below)

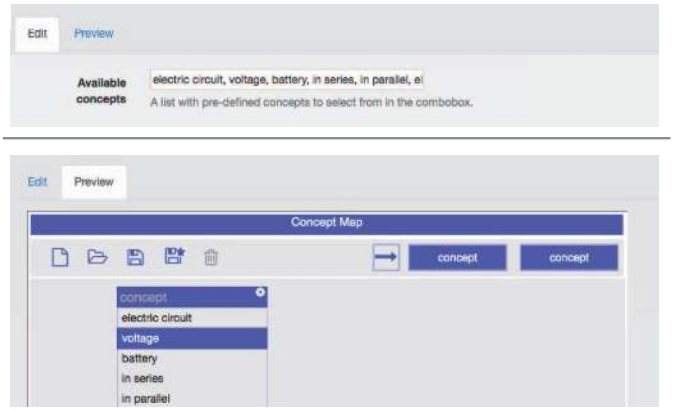

Fig. 3. The Concept Mapper being adapted in the AppComposer (above) and the result (below)

The reminder of this paper is structured as follows. Section II reviews the state of the art and Section III discusses the background. Then Section IV presents the AppComposer, while Sections V and VI present the Adaptor and the Translator modules. Finally, Section VII discusses intellectual property rights and Section VIII wraps up with a conclusion and an outlook on future work.

II. State of the Art

Education is changing, influenced like many other fields by the rapid advances of Information Technologies. Through them, the field of Technology Enhanced Learning (TEL) has been growing and is expected to keep growing in the future [6], [7], [8]. One of the technologies that have surfaced are *online laboratories*. These labs, which will be described in greater detail in later sections, allow users to access either remote or virtual hardware from any part of the world. To apply these new technologies and conceptions developers and teachers require tools to create actual pedagogical content for the students or users. Significant research efforts have already been dedicated to this kind of tools.

The design of hybrid interactive learning environments for children and adults is a source of interest for instructional technologists [9]. School teachers and non-expert programmers make use of a selection of tools to create new learning apps or modify existing ones. In content authoring dynamic elements are mixed smoothly with static documents. Raptivity[2] or Zebrazapps[3] are examples. Support for mobile platforms has also attracted some attention [10].

Nowadays, most software editors generate a low level of interaction in the resulting media. Though UI design for learning is mainly intended to make people think and perform, UI design in general is about efficiency [11]. Also, interactivity is fundamental in order to give students an active role and is a key feature provided by serious games, simulations and labs in e-learning scenarios to reinforce teaching of STEM (Science, Technology, Engineering, and Mathematics) principles [12].

Apps tend to be more interactive than standard Web pages, but currently the technologies are similar. XML and HTML make it possible to store the information, in such a way the final layout can be selected by the user and adjusted to the output device. Internationalization and localization are two other aspects of software that affect the whole creation process [13]. Also, many apps have been developed with Flash and Java. These became the de facto standard for interactive Web content due to the possibility of embedding them and the limitations of HTML. This is no longer necessarily the case [14]. Among the disadvantages of non-standard proprietary components such as these are severe security issues, the need to install them on each device, and the lack of mobile support. Applications based on the modern HTML5 stack are now appearing and generally preferred.

Another trend has been to merge dynamic assets with lessons that where previously static, through slideshows, wikis, Learning Management Systems and custom e-learning solutions. Courses provide and include SCORM packages, and try to make it easy to create and edit content and to not require server-side components. Sometimes they rely on specialized editors. In a recent work [15], an analysis of the existing web application toolkits is provided, and their main features, requirements and issues explored. It highlights the need to provide an authoring tool to let teachers build high-quality interactive applications which are customized to the student's cognitive progress within a particular course or session.

III. Background

The *AppComposer* is developed in the context of the European Go-Lab Project (Global Online Science Labs for Inquiry Learning at School[4]), which aims to make online laboratories more accessible so that they can be used on a large scale in education. This goal is accomplished by providing a technical framework through which teachers across Europe are able to enrich their classroom activities with inquiry-based learning enabled by online laboratories. A first version of the

[2]http://raptivity.com
[3]http://zebrazapps.com
[4]http://www.go-lab-project.eu

AppComposer was presented in [16]. The Go-Lab framework comprises several components which are *loosely coupled* and which are related to the *AppComposer* in different ways. This section will elaborate on the different components related to the *AppComposer*.

A. The Go-Lab portal

The Go-Lab portal [4] aims to supply lab owners with the means to make their laboratories known and to share them. Likewise, it aims to supply teachers with the means to discover useful online labs, supportive apps and ILS published by other teachers to support their classroom activities. Teachers can easily create their ILS from an online lab or repurpose a published ILS. The Go-Lab portal consists of two main platforms, namely the Lab Repository and the ILS Platform.

The ILS Platform, also known as Graasp[5], allows teachers to create and customize the ILSs for their students. The *AppComposer* is integrated with *Graasp*, because it is meant to provide the application customization features that teachers require. This integration includes a single sign-on authentication system which will be described in more detail in the next sections, which enables a straightforward use of adapted and translated applications.

The *Lab Repository*[6] is a repository for online laboratories, applications and ILS. Each resource has associated metadata which makes search, discovery and recommendations possible.

Through the *Lab Repository* API, the *AppComposer* enables teachers to easily find and select applications to translate or to adapt.

B. OpenSocial apps

OpenSocial[7] is an Application Programming Interface (API) designed for creating web-based social platforms. Its primary goal is to provide a common framework that developers can use to ensure interoperability across different social networks on the Internet, which act as containers for each OpenSocial-compliant application. OpenSocial apps must be delivered through an *app container*.

The most common and widely used container is Apache Shindig[8]. The *Graasp* platform relies internally on a modified version of this container, and the AppComposer will rely on Graasp's instance of Apache Shindig to distribute the published translations in an efficient and scalable way. This will be described in more detail in later sections.

IV. AppComposer

The *AppComposer* has been designed as a Web platform.[9] Figure 4 shows a general overview of its most significant components. Some components are mostly generic and are services required for the platform as a whole. The most significant of these ones are:

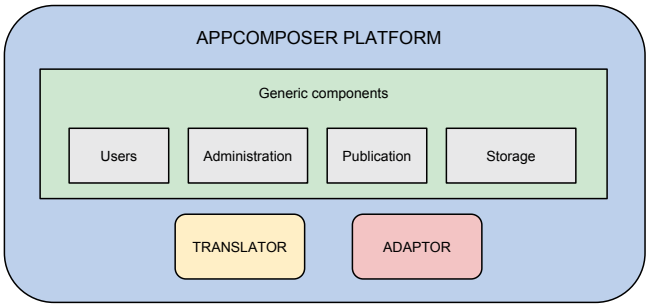

Fig. 4. General Overview of the AppComposer components

- The *users* component provides personal user management or authentication (either through a user and password combination or through Graasp).
- The *administration* component provides privileged features such as user and application management.
- The *publishing* component enables the straightforward publishing of translated and adapted applications.
- The *storage* component provides a storage layer to be used throughout the platform, to store information about users, the application translations, and the application adaptations.

Beyond those generic components, which are present, to some extent, in many Web applications, the *AppComposer* provides its actual functionality through the two different *Composers* that have been described in previous sections. These two composers, known as the *Translator* and the *Adaptor*, will be described in detail next.

V. Translator

The *AppComposer Translator* is the component of the AppComposer that lets teachers find and translate an OpenSocial application for their students to use.

Being available in the local language is often a very strong requirement for an application to be useful, particularly in the context of Go-Lab, because its audience are often young students who do not necessarily understand the language of the original application. Even with some language proficiency, this might often add extra complexity and could harm the learning experience.

Due to practical limitations, it is not possible to create a translator which is simply able to translate any kind of app. Thus, in order to be translated, an application must meet some criteria, which are nonetheless relatively simple:

- It must be an OpenSocial application
- It must conform to the OpenSocial Internationalization API

The OpenSocial Internationalization API is an XML-based API through which OpenSocial developers can specify translation bundles for their applications. Then, in the application's code, they can refer to the messages within these bundles using the API.

[5] http://graasp.eu
[6] http://www.golabz.eu/
[7] http://opensocial.org/
[8] https://shindig.apache.org/
[9] The AppComposer is available at http://composer.golabz.eu

For the *Translator* to work as expected, developers do not need to actually translate their applications (that is the purpose of the *Translator*), though they can do so if they wish to. They only need to use that API instead of hard-coding their strings, so that the *Translator* can locate which strings to translate.

The OpenSocial Internationalization API can translate applications for specific languages and territories. However, in the particular domain of learning it has been noted that the language of the audience, even within the same country and territory, varies significantly. It is hence desirable to be able to also provide different translations to different groups. For instance, a translation for young (10-13 year old) students, and a translation for older ones (14-17 years old). Since OpenSocial can't do that by itself, the *Translator* adds support for it by extending the OpenSocial API and dynamically generating and serving the OpenSocial XML bundles that define the contents of the translation. This has a significant influence in the design of the *Translator*.

To enable reuse of the translations done by other users, mostly teachers, it is necessary to provide the means to publish (host) the translations. For ease of use, especially considering the potentially non-technical background of the teachers, the system must do that transparently.

Every use of a translated application involves several requests. Because thousands of students are expected to use the translations, if the *AppComposer Translator* served the translations itself it would suffer a particularly heavy load. To avoid this issue, the system has been designed in such a way that serving the translations to the end-users is done mostly by the *Graasp* ILS system, which is better suited for that purpose. This scheme is described in the following sections.

A. User Workflow

The goal of the *AppComposer* is to be as simple and straightforward to use as possible. Most users will not have a technical background (and they are not expected to). Thus, to provide an adequate user experience, the workflow tries to be linear and provides a wizard-like style. Because the *AppComposer* is created as a web application, the only technical requirement is a relatively up-to-date browser. Once the teachers access the *AppComposer* website, they can log-in through the *Graasp platform* or other means and access the *Translator*.

First, they are presented with the application finder screen, which can be observed in Figure 5. It contains a dynamic list of applications to translate, which is provided, along with certain metadata, by the *Go-Lab Repository*. Through it, or through the repository itself, the teachers can easily find the applications they are interested in or discover new applications they did not know about. Then they only have to select it to begin the translation process. Alternatively, they can just specify the URL of an existing OpenSocial application, even if it is not present in the *Go-Lab Portal* repository.

Once they have chosen, the *Translator* will create their own *instance* of that application. The system automatically extracts translation information from the application they have chosen, and if there is an existing translation for any language, either in the original application or in the AppComposer, the system will automatically merge the translations.

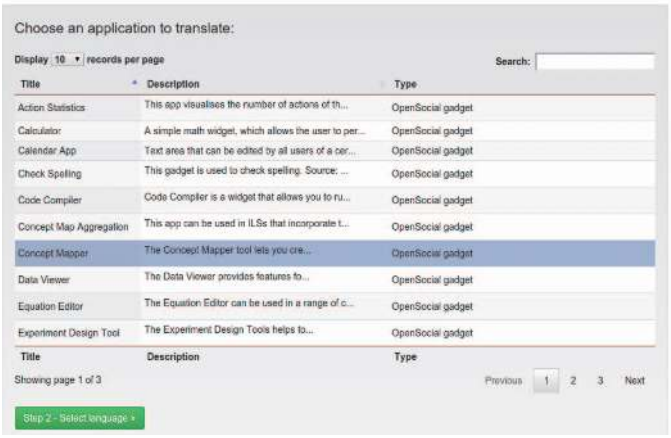

Fig. 5. Application finder screen. Data is provided by the Lab Repository.

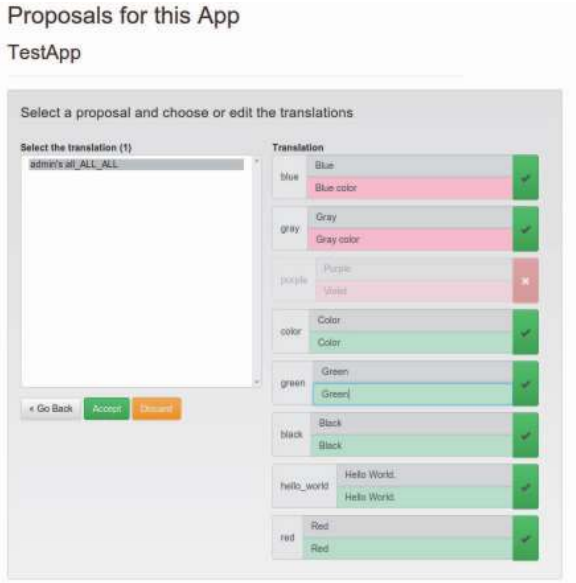

Fig. 6. Merge tool screen, through which language owners for an application can easily review proposed changes

From then on, the teachers have their own instance, and they can choose the source and target languages (and optionally the source and target groups) and start translating. Collaboration between users is supported. Each language translation of a particular application is owned by the first person who translated it. Teachers can automatically submit their translation proposals to that owner, who can then merge the changes after reviewing them, reject them if they are wrong, or even decide to trust all translation proposals implicitly. The merge tool is shown in Figure 6.

By default, when publishing the application, the *default* translation (which is the translation that *owns* a language for an application) will be used. This is the case for Graasp, which will always serve the *owner* translation. However, when teachers, for any reason, do not like the *default* translation, they can easily publish their own version to their students.

It is noteworthy that though the teachers need to log-in to create a translation, students will not need to register or

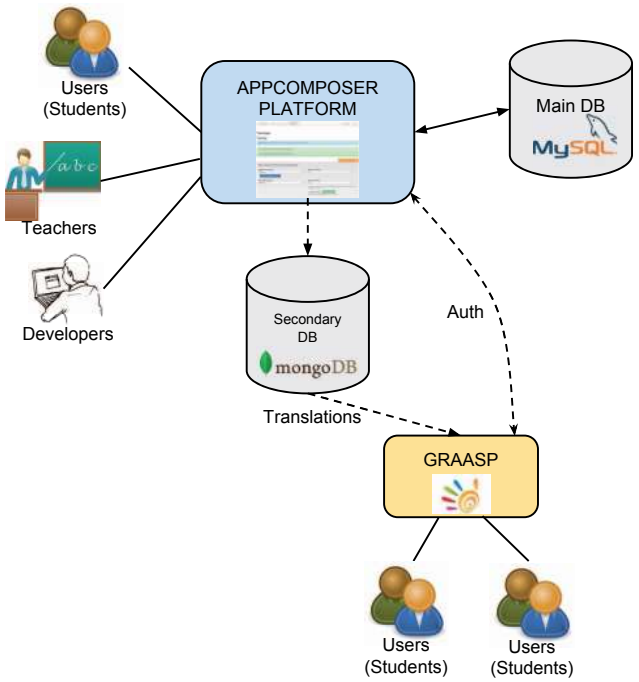

Fig. 7. AppComposer design and integration

log-in at any moment. The experience for them is meant to be as seamless as possible. Once the teachers have translated and published an application they can simply share a link with their students and they will immediately be able to access the translated version of the App through *Graasp*.

B. Architecture Overview

Figure 7 depicts the general architecture of the system. As the diagram shows, students, which are the end-users, will not be in direct contact with the AppComposer. Students will only make use of the translated apps made available by the AppComposer in the Graasp ILS. This way, the user experience is transparent, and end-user requests are handled by Graasp and are cached by a secondary database to ensure scalability for the kind of load that is expected.

Teachers and developers, however, will access the AppComposer directly, even though they will still, in general, authenticate through Graasp.

The diagram includes two different databases. The first one is a MySQL instance, and it acts as a primary database. The AppComposer system relies on SQLAlchemy, a Python-based ORM, so, actually, most Relational Database Management Systems would be supported. This primary database contains all operational information, including the existing users and the applications and their data. The aim of the secondary, MongoDB-based database is to provide an efficient and scalable means to integrate the AppComposer translations with *Graasp* or other external systems. This scheme will be discussed in detail in the *synchronization scheme* section.

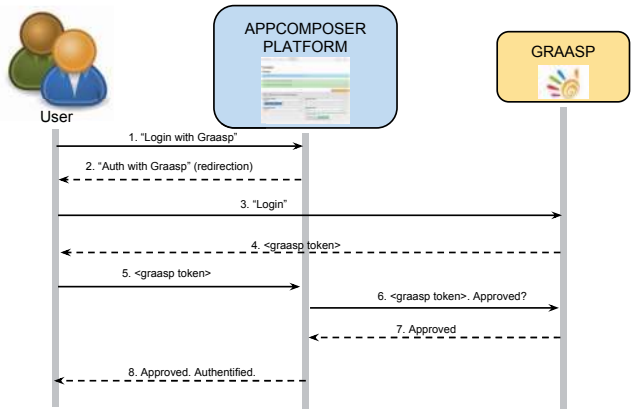

Fig. 8. The authentication protocol for the single sign-on system

C. Graasp Authentication

One of the goals of the *AppComposer* platform is to integrate seamlessly with *Graasp*. For this, a single sign-on system is provided. *Graasp* provides a custom token-based non-standard system which resembles *OAuth*.

Figure 8 illustrates this system. Whenever users wish to authenticate against the *AppComposer* they are redirected to *Graasp*, which gives them a temporal and random *token*. This *token* is then shared with the *AppComposer*. Then, the *AppComposer*, by consulting directly with *Graasp*, verifies that the *token* is indeed valid and belongs to the user. If everything is right, access is granted. Additionally, if it is the first time that the user is accessing the system, then it is automatically registered in the primary database.

In the future, the *AppComposer* will make use of the newly developed authorisation *Graasp* service using the *OAuth 2* protocol.

D. Synchronization scheme

As has been mentioned in previous sections, the *AppComposer* is part of the Go-Lab project. As such, it is expected to support thousands of students throughout Europe, who should be able to access the published translations easily and without delay.

The amount of users and translation requests can be very high, because each use of a translated application involves at least a translation request, and each request, in fact, involves several files. This kind of problem is handled very efficiently by the OpenSocial application server, i.e. *Apache Shindig*, which is used internally by *Graasp* and which is an Open Source project designed specifically for that purpose. However, the *AppComposer* does not have that aim. To reduce the potential load on the *AppComposer* server and to promote scalability, a particular design has been implemented, in which the *AppComposer* provides the translations to Graasp and *Graasp* serves them to the end-users.

This is depicted in Figure 9. A significant aspect of this design is that there is no direct communication between the *AppComposer* and the external system. Instead, that communication takes place through a secondary *MongoDB*. Thus, when

978-1-4799-1908-6/15/$31.00 ©2015 IEEE 18-20 March 2015, Tallinn University of Technology, Tallinn, Estonia

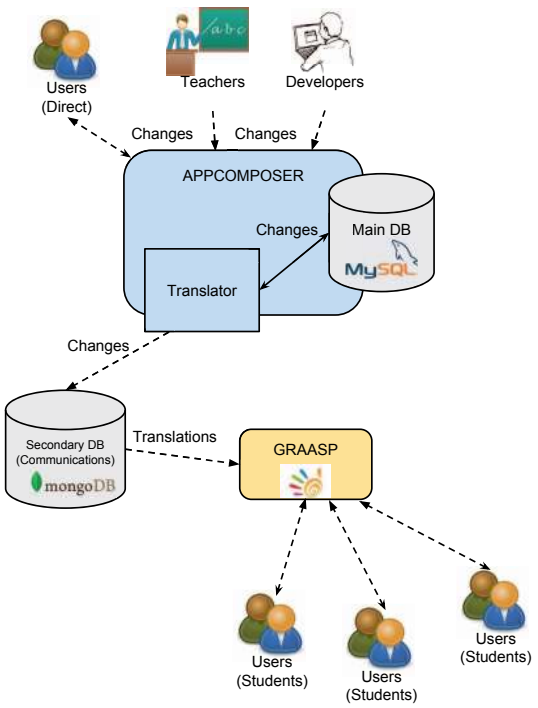

Fig. 9. Architecture of the synchronization scheme for the communication with external systems

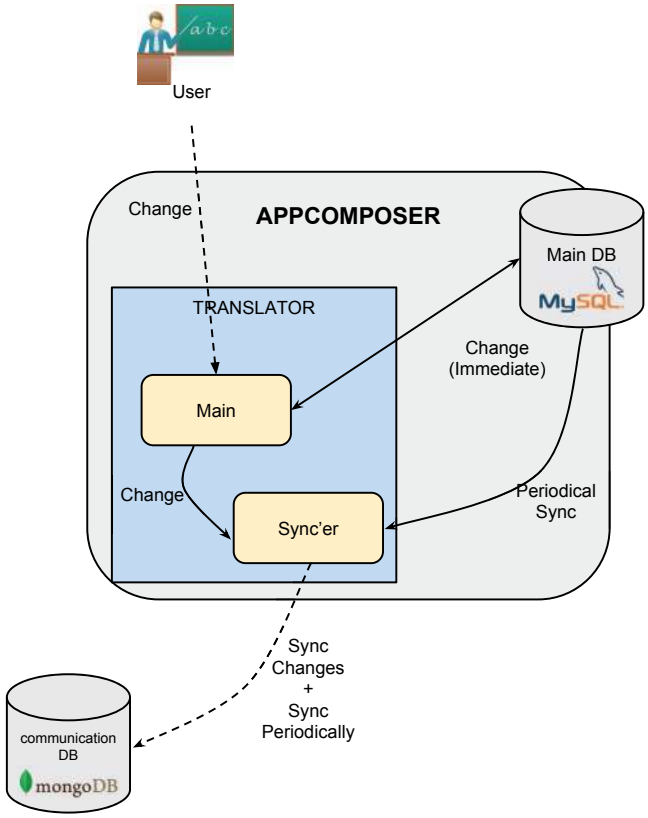

Fig. 10. AppComposer Translator synchronization flow

users ask *Graasp* for a translation, *Graasp* will not ask the *AppComposer*, but will instead obtain it from the *MongoDB* database, similarly to a cache server.

The secondary database does not replace the primary one in any way. The primary MySQL database has at all times the full authoritative information (and in fact, it contains significantly more information than the secondary one). The secondary database only contains the translations, stored in JSON and indexed to provide simple and efficient access.

Though the system is designed so that almost at all times the information in the secondary database is up to date with respect to the primary one, this synchronization is deliberately non-transactional and is not necessarily in real-time. It is synchronized in two ways, which are illustrated in Figure 10.

- When a change occurs (update or removal of a translation) a synchronization request is queued for the specific translation. If the secondary DB is available and ready, the synchronization will be almost instant.
- Periodically a full synchronization will be done. This guarantees long-term integrity and increases reliability and fault-tolerance. If a particular update failed, or even if the whole secondary database were corrupted or exchanged by an empty database, the system would recover on its own in a very short time.

This scheme achieves a very high **scalability**, because the limit will mostly be set by the *MongoDB* instance. Because MongoDB by design supports very heavy loads and can scale horizontally, it could even be transparently replaced by a cluster.

Also, it makes the system more **fault-tolerant** and secure, because it decouples the *AppComposer* from *Graasp*. If one of them failed, the other would still work on its own. For instance, if the *AppComposer* servers failed, end-users would still be able to access the last translations through *Graasp*. Likewise, if *Graasp* was not reachable due to heavy loads, *DDoS* attacks, or any other issue, the AppComposer could still be accessed by teachers and customization developers or even (with limitations) by the students themselves.

VI. Adaptor

The AppComposer Adaptor is a tool for adapting applications. The target is to let web developers create customizable applications that teachers can easily take, adapt, and publish for their students. The way to adapt these applications is centralized through the AppComposer.

Examples of these applications include:

- Customize the domain of certain applications: given a generic application where users have to interact with concepts.
- Customize the behaviour of an application: showing a different set of tabs depending on the purpose of the teacher, or reducing the level of explanation of certain parts.
- Customize the user interface of a virtual or remote laboratory: show only a subset of the experiment for different purposes. In a laboratory like VISIR

[17], teachers might want to reduce the number of electronics components or attached devices (e.g., hide the DC Power or the Oscilloscope).

A. Basic workflow

Teachers the AppComposer, which essentially uses a single sign on mechanism with Graasp (so teachers don't need to be registered multiple times). Then, they select which application they want to adapt. When selected, they can also see existing adaptations for that application, since the adaptation they are going to create could have already been developed by somebody else, and seeing existing adaptations could help in terms of inspiring on new models for adapting the application. Then, an adaptation panel is displayed, which can be a generic one with few settings provided by the developer or it can be completely developed from scratch by the application developer (see next section). Whenever the adaptation is modified it is saved automatically and a URL is provided. Teachers can take this URL and add it as an app on the Go-Lab ILS platform so students use the adapted version of the application.

New versions of the application should not affect the adaptations, as long as the developer manages old settings stored in the AppComposer.

B. Customizable application development

The AppComposer Adaptor provides a Python plug-in development mechanism which relies on the Flask web microframework. This plug-in system provides simplified access to the database, enables the developer to create new URLs, interacting with the disk, etc. On top of it, other approaches can be developed and are presented in this section.

1) Pure web applications: Through one of its plug-ins, the AppComposer enables web developers to keep their applications in their servers, and provide a lightweight mechanism to replace certain parameters. Basically, the application developer only needs to provide the default application configuration using a special attribute in a script HTML tag, as well as a description of this configuration in other script with a different special attribute.

When adapting the application, the AppComposer checks the configuration description and generates a user interface with a selection of these options (see Figure 11). This configuration is limited to the extensibility of the description mechanism, which supports strings, boolean options and lists of strings.

When publishing the application, every time somebody attempts to load the application, the AppComposer downloads the original application and replaces the default configuration script by an automatically generated one which includes the options selected by the teacher. This way, while the URL points to the App Composer, the changes in the original application are always reflected.

2) Pure plug-ins: Application developers can create their own native applications in Python. This, however, requires a deployment in a server. This can be done in the main AppComposer server by contacting their administrators and contributing code, or by deploying the AppComposer in a different server. Independent deployments are discouraged because they have a potential negative impact in terms of usability for teachers, since they will need to know where these external AppComposers are.

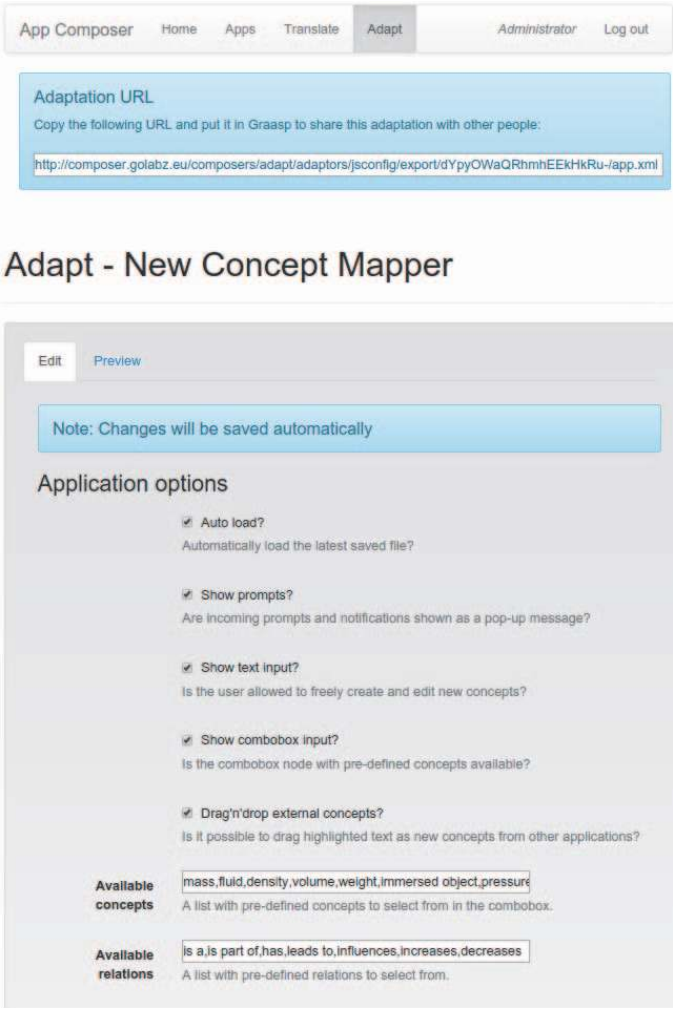

Fig. 11. Adapting a pure web application. The controls are generated automatically from the configuration retrieved from the original application.

The main advantage of this plug-in systems is that there is no restriction in the software development. Developers can keep secrets (such as keys to external services), store additional information per user, interact directly with the database, optimize and cache certain parts, etc.

3) Future plug-ins: Further plug-in types will be added to increase the extensibility of the system. One is currently under development, and it is focused on making it easy to customize remote laboratories. It relies on *gateway4labs* [18], a software system that integrates existing remote laboratories in OpenSocial. The remote laboratory provides an HTML file that provides a customization panel (e.g., Figure 12), where the remote laboratory developer creates the whole customization in an HTML iframe. This panel provides the desired settings to the AppComposer. Finally, the AppComposer generates a link to the final gateway4labs system including a link to the desired configuration (hosted in the AppComposer), which is provided to the laboratory upon reservation request. This way, teachers can adapt the laboratory (using a panel provided by

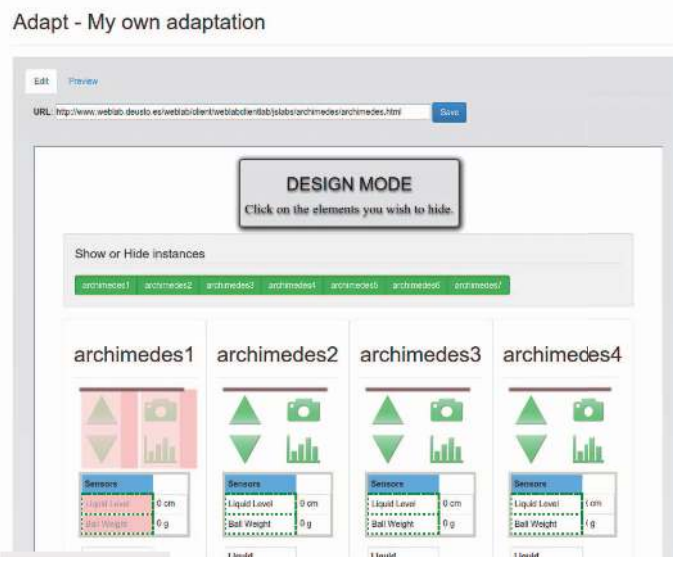

Fig. 12. Adapting a remote laboratory. Red parts will not be displayed.

the laboratory developer), and use the customized version in the Go-Lab ILS platform.

Other types of plug-in will also be included to make the development process easier. For example, providing a tool that enables application developers to provide a customized version of the adaptation panel is also desirable, so external developers can provide a fine-grained configuration manager of their application.

VII. Intellectual Property rights

As described, teachers can use the AppComposer to customize an application (either through translation or adaptation). In principle, this can potentially lead to intellectual property rights issues. However, they are limited. In the case of the Translator, no input is actually required from the original developer. The actual modifications to the application are limited to adding references to new language bundles, and the application needs to support internationalization by itself. However, there could still be issues if the original developer didn't want the application translated. In the case of the Adaptor, the application needs to be made explicitly compatible by the original developer. Thus, there should be no issues.

VIII. Conclusions & Future work

The Go-Lab platform and the AppComposer are currently undergoing a pilot evaluation, which makes the software available and actively used by real teachers and students in schools around Europe. In the future, the usability and usefulness evaluation results that are gathered through these pilots will be used to improve the platform and its user experience. With regard to usability evaluation, an expert evaluation is also planned, where usability experts analyse the user interface using heuristic rules and common UI patterns and practices. Additional new features are also planned. Particularly, significant efforts will be dedicated to further integrate the AppComposer with the rest of the components in the Go-Lab ecosystem. This will likely involve:

- Providing a consistent workflow through the tools (mainly through Graasp, the AppComposer and the Lab Repository)
- Streamlining the user experience by providing a consistent look and feel
- Extending the integration between the Lab Repository and the AppComposer so that Apps can be found in the Repository and translated or adapted with the click of a button straight from the Lab Repository; and enabling the AppComposer to know whether an application in the repository meets the requirements for translation or adaptation.
- Making the authentication interaction between Graasp and the AppComposer more transparent for the user, which will be possible through *OAuth2*.

Acknowledgment

This research was partially funded by the European Union in the context of the Go-Lab project (Grant Agreement no. 317601) under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7). This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of its content.

References

[1] T. De Jong, M. C. Linn, and Z. C. Zacharia, "Physical and virtual laboratories in science and engineering education," *Science*, vol. 340, no. 6130, pp. 305–308, 2013.

[2] T. de Jong, S. Sotiriou, and D. Gillet, "Innovations in stem education: the go-lab federation of online labs," *Smart Learning Environments*, vol. 1, no. 1, pp. 1–16, 2014.

[3] D. Gillet, T. de Jong, S. Sotirou, and C. Salzmann, "Personalised learning spaces and federated online labs for STEM education at school," in *2013 IEEE Global Engineering Education Conference (EDUCON)*, Mar. 2013, pp. 769–773.

[4] S. Govaerts, Y. Cao, A. Vozniuk, A. Holzer, D. G. Zutin, E. S. C. Ruiz, L. Bollen, S. Manske, N. Faltin, C. Salzmann *et al.*, "Towards an online lab portal for inquiry-based stem learning at school," in *Advances in Web-Based Learning–ICWL 2013*. Springer, 2013, pp. 244–253.

[5] M. J. Rodriguez Triana, S. Govaerts, W. Halimi, A. C. Holzer, C. Salzmann, A. Vozniuk, T. de Jong, S. Sotirou, and D. Gillet, "Rich open educational resources for personal and inquiry learning. agile creation, sharing and reuse in educational social media platforms," in *International Conference on Web & Open Access to Learning*, no. EPFL-CONF-203595, 2014.

[6] Y. Beldarrain, "Distance education trends: Integrating new technologies to foster student interaction and collaboration," *Distance Education*, vol. 27, no. 2, pp. 139–153, Aug. 2006.

[7] U. Hoppe, M. Milrad, C.-K. Looi, P. Dillenbourg, M. Scardamalia, N. Balacheff, E. Soloway, C. Norris, R. D. Pea, J. Cherniavsky, C. Patton, T. Brown, M. Sharples, K. Kinshuk, S. Hsi, J. Roschelle, and T.-W. Chan, "One-to-one technology-enhanced learning: an opportunity for global research collaboration," *Research and Practice in Technology Enhanced Learning*, vol. 1(1), pp. 3–29, 2006.

[8] K. Kumaran and V. Nair, "Future trends in e-learning," in *Distance Learning and Education (ICDLE), 2010 4th International Conference on*. IEEE, 2010, pp. 170–173.

[9] L. P. Rieber, "Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games," *Educational technology research and development*, vol. 44, no. 2, pp. 43–58, 1996.

[10] L. Serafimov, "Html5 support in mobile learning tools," in *Conference proceedings of" eLearning and Software for Education"*, no. 02, 2012, pp. 283–286.

[11] M. W. Allen, *Michael Allen's Online Learning Library: Successful e-Learning Interface: Making Learning Technology Polite, Effective, and Fun*. Wiley. com, 2011, vol. 3.

[12] H. Leemkuil, T. Jong, and S. Ootes, "Review of educational use of games and simulations." 2000.

[13] S. Auer, M. Weidl, J. Lehmann, A. J. Zaveri, and K.-S. Choi, "I18n of semantic web applications," in *The Semantic Web–ISWC 2010*. Springer, 2010, pp. 1–16.

[14] L. Rodriguez-Gil, P. Orduña, J. García-Zubia, I. Angulo, and D. López-de Ipiña, "Graphic technologies for virtual, remote and hybrid laboratories: Weblab-fpga hybrid lab," *2014 10th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pp. 163–166, 2014.

[15] M. Latorre, A. Robles-Gómez, L. Rodríguez-Gil, P. Orduña, E. Sancristobal, A. C. Caminero, L. Tobarra, I. Lequerica, S. Ros, R. Hernández, M. Castro, D. Lopez-de Ipiña, and J. Garcia-Zubia, "A review of webapp authoring tools for e-learning," in *Accepted to Global Engineering Education Conference (EDUCON), 2014 IEEE*. IEEE, 2014.

[16] L. Rodriguez-Gil, M. Latorre, P. Orduna, A. Robles-Gómez, E. Sancristobal, S. Govaerts, D. Gillet, I. Lequerica, A. C. Caminero, R. Hernandez *et al.*, "Opensocial application builder and customizer for school teachers," in *Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on*. IEEE, 2014, pp. 31–33.

[17] I. Gustavsson, J. Zackrisson, L. Håkansson, I. Claesson, and T. Lagö, "The visir project–an open source software initiative for distributed online laboratories," in *Proceedings of the REV 2007 conference, Porto, Portugal*, 2007.

[18] P. Orduña, S. Botero Uribe, N. Hock Isaza, E. Sancristobal, M. Emaldi, A. Pesquera Martin, K. DeLong, P. Bailey, D. López-de Ipiña, M. Castro, and J. García-Zubia, "Generic integration of remote laboratories in learning and content management systems through federation protocols," in *2013 IEEE Frontiers in Education Conference*, Oklahoma City, OK, USA, Oct. 2013, pp. 1372–1378.