Widgets and Spaces: Personal & Contextual Portability and Plasticity with OpenSocial

THÈSE N^{O} 5804 (2013)

PRÉSENTÉE LE 30 AOUT 2013

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
INSTITUT DE GÉNIE ÉLECTRIQUE ET ÉLECTRONIQUE
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Evgeny BOGDANOV

acceptée sur proposition du jury:

Dr A. Karimi, président du jury
Dr D. Gillet, Dr C. Salzmann, directeurs de thèse
Dr E. L.-C. Law, rapporteur
Dr L. Moccozet, rapporteur
Dr C. Ullrich, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2013

A person who never made a mistake
never tried anything new
— Albert Einstein

To my parents...

Acknowledgments

This thesis is only partially my work. It would not be possible without my friends, relatives, colleagues who were helping me (knowingly or unknowingly) during this PhD journey. I am hugely grateful to all of you!

Denis Gillet and **Christophe Salzmann**. Thank you for accepting me into your research family, for giving me the freedom in choosing and shaping the research directions according to my interests, for being my mentors. After our discussions I was always leaving your office inspired and highly motivated. This is indeed something!

I would like to thank my thesis committee members for reading my thesis and being my examinators: **Alireza Karimi**, **Effie Lai-Chong Law**, **Carsten Ullrich**, **Laurent Moccozet**.

My research work was partially funded through the following projects: **ROLE**, **PALETTE**, **SWITCH**, **GO-LAB**. I am thankful to these projects and I did appreciate working in them.

Stéphane Sire, I owe you a huge chunk of my thesis and my personal development. Thank you for your super good ideas, all our discussions and collaboration. It was a pleasure. Particularly, thanks for being my first JavaScript teacher who ignited my interest to this beautiful language!

I was extremely lucky with my colleagues. Everyone is super nice and the ambiance in the group is as one could only dream about. **Sandy Ingram**, **Willson Shibani**, **Adrian Holzer**, **Sten Govaerts**, **Wissam Halimi**, **Jad Naoum** - thank you all for the conversations, ideas exchange, travels, barbecues, tea breaks, movie nights, life advices. **Na Li**, **Laleh Makarem** - you are the best colleagues ever (not only because you shared food with me!) **Andrii Vozniuk** - you joined our group recently but your interest in new things and technologies boosted my performance. I have learned and continue learning a lot of things because of you - thanks! **Anne Remillet**, thank you for being so responsive! **Alain Bock**, thanks for being my training partner.

In addition to my colleagues at EPFL, I would like to thank all other researches with whom I had a chance to collaborate in joint projects and in conferences. Particularily, I want to express my gratitude to **Matthias Palmér**, **Erik Isaksson**, **Fridolin Wild**, **Carsten Ullrich**, and **Freddy Limpens** with whom we co-authored our research papers.

When I started my PhD, our group was part of **Laboratoire d'Automatique** and I have very warm feelings towards all people in this lab. Thank you for being there for me when I needed it, for our breakfasts, social events and the perfect atmosphere in the lab!

Acknowledgments

I would like to say thank you to all my sport partners who shared my passion over the last several years. I am very grateful to the **Schweizerischer Akademischer Skiklub (SAS)** and **Stéphane Chevrier** who found me and brought me into the club. **Fabian Birbaum**, **Pascal Weber**, **Alexander Walpen**, **Mauro Gruber**, **Reto Brunner**, **Dominique Schwab**, **Lada Sycheva** - it was and is a great pleasure to train and compete with you (parties included)!

Andrijana Radivojevic, **Alexandra Paillusson**, **Natasa Pjescic** - you are awesome friends, my best dance, ice skating partners and a lot more!

Zlatko Emedi, **Sean Costello**, **Francis Tschantz**, **David Ingram**, thank you for the time we spent together - I was always enjoying it.

Ira and **Valya**, you were my only friends in Lausanne, when I first came. Thank you for helping me, thank you for our scientific, sport and life discussions. I always enjoy spending time with you! In addition, I would like to thank all my other friends from **Moscow Institute of Physics and Technology (MIPT)**, especially my ski club friends with whom we keep a very close contact.

Niko, thanks for being like me, thinking like me, training like me. My life would be definitely dull if you were not here during the last year. It's a pity we didn't meet long before. You are the best friend, man!

Denise, I can't express in words how joyful and pleasant my life has become in the last two month because of you. I am very thankful to you for the fascinating moments we have spent together and for making my motor spin faster!

Mama, Papa, Zoya, without you I would never make it. Everything I have ever achieved or did is because of you. You always supported all my crazy ideas and my interest in things. Thank you for being my greatest motivators! I am lucky to have you!

Lenush and **Elena Stepanovna**, you are very dear to me and I am glad to have both of you in my life. You were always very supportive and showed a great interest in the things I did and do. It is truly motivating! Thank you!!! **Len**, I came to Lausanne because of you, I started my PhD because of you, and I did a lot of things because of you. Thank you for always giving me freedom and being a person I could rely on in my life. I became a better man because of you and I am deeply grateful to you for everything you did!

Lausanne, 3 July 2013 E. B.

Abstract

Social media platforms are created and exploited for various activities carried out individually or collaboratively and relying on different resources and tools. Social media platforms are inherently contextual; the context being defined as a specific activity carried out for a specific purpose with specific tools and resources, as well as with specific people. These contexts may be spread over different platforms. Thus, users need to collaborate across various platforms, they need to move their environments and data from one platform to another. Every task a person accomplishes has its own specifics. Hence, there is a strong need for users to be able to personalize (shape) their environments to suit their specific needs: by changing a set of tools, adding and removing resources, by adapting the graphical and functional parts of their platforms, and sharing resources with others.

This thesis investigates the challenges of contextualization, portability and personalization within social media platforms through the following research questions. How can we model a user context in a social media platform? How can we enable portability: i.e., to access the same user's environment from different social media platforms and to migrate an environment from one platform to another? How can we enable the easy personalization of user's contexts?

In the first part of the thesis, we formally define the space concept, that materializes the user's context and represents an environment constructed by the user. We propose an OpenSocial space extension that introduces the space concept into OpenSocial specification in the form of Space model and APIs. In addition, we propose a way to build contextual widgets capable of adapting to the user's context.

In the second part of the thesis, we propose the notion of collaborative portable space configuration relying on the space configuration language. We demonstrate how portability of spaces can be achieved with OpenSocial. This includes the classification of various migration methods and scenarios of space portability. In addition, we propose a concept of portable platform interfaces.

In the third part of the thesis, we define plasticity as a measure of a platform ability to be shaped according to users' needs. To address plasticity, we propose the functional skin concept for personalization of graphical and functional interfaces. In addition, we propose cloud aggregation and sharing mechanisms.

Keywords: widget, space, personalization, context, portability, plasticity, opensocial, social media platform, interoperability, migration, collaboration, functional skin

Résumé

Les médias sociaux sont créés et utilisés pour des activités variées menées individuellement ou de manière collaborative et exploitent différents outils et ressources. Les médias sociaux sont intrinsèquement contextuels; le contexte étant défini comme une activité spécifique menée dans un but particulier avec des ressources et des outils choisis, ainsi qu'avec des personnes sélectionnées. Ces contextes peuvent être répartis sur différentes plates-formes que les utilisateurs doivent exploiter conjointement ou entre lesquelles ils doivent échanger leurs données pour collaborer selon leurs besoins spécifiques. Chaque tâche qu'une personne accomplit est unique. Il est donc essentiel pour les utilisateurs de pouvoir personnaliser les espaces qu'ils exploitent dans les médias sociaux en fonction de leurs besoins spécifiques; non seulement en y ajoutant ou supprimant des ressources et en les partageant, mais aussi en modifiant la palette des outils disponibles et en adaptant la visualisation ou les fonctionnalités de l'interface.

Cette thèse explore les challenges liés à la contextualisation, à la portabilité et à la personnalisation des médias réseaux en considérant les questions de recherche suivantes: Comment peut-on modéliser un contexte utilisateur dans les médias sociaux? Comment peut-on favoriser la portabilité: c'est-à-dire comment permettre l'accès à un même espace partagé depuis différents médias sociaux ou comment permettre le transfert d'un espace d'un média social à un autre? Comment peut-on permettre une personnalisation aisée d'un espace partagé?

Dans la première partie de la thèse, le concept d'espace matérialisant un contexte utilisateur et représentant un environnent personnel est formellement défini. Ensuite, une extension du standard OpenSocial est proposée pour permettre la mise en ouvre du concept d'espace au moyen d'une spécification et d'une API (interface de programmation applicative). Finalement, une manière de construire des applications Web contextuelles comme outils utilisateur s'adaptant au contexte est proposée.

Dans la deuxième partie de la thèse, la notion de configuration d'espaces collaboratifs modulables basée sur un langage dédié est introduite. La manière de transférer des espaces partagés grâce à OpenSocial est démontrée. Ceci comprend la classification de différentes méthodes de migration et des scénarii de transfert d'espaces. De plus, le concept d'interface portable indépendante des plates-formes est proposé.

Résumé

Dans la troisième partie de la thèse, la plasticité en tant que mesure de la capacité d'une plate-forme à être personnalisée en fonction des besoins utilisateurs est définie. Ensuite, le concept d'enveloppe de personnalisation graphique et fonctionnelle des interfaces est proposé. Enfin, des solutions simples d'aggrégation de ressources distribuées en nuage et des mécanismes ouverts de partage trans-organisationnels sont développés.

Mots-clés: widget, espace, personnalisation, contexte, portabilité, plasticité, interopérabilité, opensocial, médias sociaux, migration, collaboration, interface fonctionnelle

Contents

List of Figures

List of Tables

Glossary

- **PLE** - Personal Learning Environment
- **VLE** - Virtual Learning Environment
- **LMS** - Learning Management System
- **SRL** - Self-Regulated Learning
- **UI** - User Interface
- **(Web) widget** - a small application that can be installed and executed within a Web page by an end user (the other names - Tool, (Web) App, (W3C) Widget, OpenSocial Gadget)
- **Widget preferences** - the settings used to initialize and deploy the hosted widgets
- **(Widget) bundle** - a set of widgets combined together and used for a particular purpose
- **Space configuration** - an extension of the widget bundle that contains widget preference values for its widgets
- **Space** - an abstract concept that materializes the user's context and aggregates people, resources, tools and other subspaces.
- **(Web) platform** or **Social media platform** - a generic Web site (such as Facebook, LinkedIn, YouTube, BBC News, etc.) supporting interaction among users.
- **(Web) platform interface** - an application that implements the user interface of a Web platform.
- **(Widget) container** - a client-side execution environment that contains one or several widgets and manages their layout and representation within a page, such as navigation between the widgets, widget addition/removal, etc.
- **Widget engine** - the computer program responsible to process the widget code and render a widget on a page. Additionally, it manages the interaction of widgets with the Web platform.
- **(Web) Mashup** - a Web page, or Web application, that uses and combines data, presentation or functionality from two or more sources to create new services.

- **PLE aggregator** - a Web platform or a technology enabling users to create their PLEs
- **Open Mashup Description Language (OMDL)** - mashup and widget bundle specification that includes widgets, layouts, theming, etc.

1 Introduction

This thesis deals with personal and contextual interaction in Web platforms. By Web platform, we mean a platform that enables and supports social interaction among users. Users can exploit such platforms referred to hereafter indistinctly as Web or social media platforms to support various activities related to different contexts. In each context, users can exploit resources (content) and tools required to achieve a specific objective. The user's context is modeled as a space: an abstract concept aggregating people, resources, tools and other subspaces relevant for a specific activity. A space represents a contextual unit in a Web or social media platform. Users can switch between contexts, exchange and share contexts with other people, move a context from one platform to another, personalize the context according to their wishes, etc. This thesis addresses the questions of the portability and the plasticity of contextual spaces. We define plasticity as a measure of a social media platform ability to be customized by users. We define portability as the ability of a social media platform to exchange spaces with other platforms. In this thesis, a tool is represented by a Web widget: a small application created with HTML, CSS and JavaScript that can be installed and executed within a Web page by an end user. Most of the thesis investigations rely on the OpenSocial specification[1] - one of the standards available for Web widgets.

While the investigations carried out in this thesis deal with learning environments, most of the results can be applied to any social media platform exploited for knowledge management or learning purposes. To familiarize the reader with the domain of learning environments, the first chapter introduces Web-based learning platforms: Learning Management Systems (LMSs) and Personal Learning Environments (PLEs), and depicts their places in the existing ecosystem of learning-oriented tools. Widgets are often used to personalize Web-based platforms and represent the key concept of this thesis. We introduce the widget concept, the architecture on which widgets are run, and the related concepts. Then we discuss the state of the art in PLE research and depict an anatomy of a PLE by showing the features, dimensions and principles used to describe PLEs. At the end of the chapter, we summarize the existing challenges and research questions related to the personal and contextual spaces in social media platforms, list our related contributions, and present the used validation approaches.

[1] http://docs.opensocial.org/display/OSD/Specs

1.1 Web-Based Learning Platforms

Learning Management Systems and Virtual Learning Environments (VLEs) have existed for more than a decade and have received a great acceptance within educational institutions and industry where they are used for corporate training. Their main characteristics include:

- Manage users, roles, courses, instructors, facilities, and generate reports
- Course calendar
- Learning path
- Student messaging and notifications
- Assessment and testing handling before and after testing
- Display scores and transcripts
- Grading of coursework and roster processing, including wait listing
- Web-based or blended course delivery

In short, they provide a good basis to support courses, where the teacher is in charge of a course and defines its program, the resources to be used and the evaluation techniques to quantify the students' progress. In LMSs, students act as information consumers: they attend courses, work on provided materials, accomplish assignments, get evaluated and examined.

VLEs and LMSs are very similar, but carry an important difference: VLEs target the education and represent a place where learning occurs through the discussions and collaboration as pedagogical principles dictate. On the other hand, LMSs target the hosting of learning objects and can be merely used for training rather than education. Despite this difference, LMSs and VLEs both target learning and are often very similar, thus we will be referring to both as simply LMSs.

One of the main critics of LMSs comes from the lifelong learning perspective. First, LMSs are not flexible enough to be personalized by learners themselves, they impose a specific learning process and an environment on students (Wilson et al. [2007]); and, second, they are disconnected from the Internet cloud of information (Severance et al. [2008]). Wilson et al. [2007] provide a detailed analysis of VLE limitations and divide the limitations into 6 main categories.

- **Focus on Integration of Tools and Data within a Course Context**. Every course is an isolated unit within an LMS. The tools and needed content are added to the course by a teacher, however there is no connection to other courses in the system or to external knowledge repositories. Since pieces of knowledge and data are often distributed in

space and are highly interconnected, the modular approach limits the explorative nature of learning by cutting out the connections to the sources of information related to the course.

- **Asymmetric Relationships**. In current LMSs, a teacher normally provides knowledge and learners consume it. Lifelong learning expects students to be active, creative and to take control of their learning process. However, because of the asymmetry they acquire a passive role of consumers.

- **Homogeneous Experience of Context**. LMSs provide for everybody within a course context the same content, the same material organization, and the same tools. The problem is that lifelong learning aims to support learners with their personal needs and priorities and to provide individualized experiences for learning.

- **Use of Open E-learning Standards**. SCORM, IMS and other e-learning specifications are adopted by LMSs. However, widespread open standards (RSS, for example) did not find a niche within LMSs. The problem is mainly caused by the closed nature of LMSs that discourages open sharing of content.

- **Access Control and Rights Management**. Another general limitation of LMSs is the access restriction to course participants. Once more, it is against the lifelong learning nature, that attempts to support learning process both at workplace and at home as well as across organizations. Generally, LMS content is not publicly shared and once people leave an organization they loose their access rights and can not even take their accumulated content with them.

- **Organizational Scope**. The interaction with an institutional LMS requires a learner to be somehow affiliated with this organization. This discourages cross-organizational and informal learning that are at the core of lifelong learning.

LMSs isolation from new technologies such as Open Educational Resources, Web 2.0, mobile learning, and their focus on formal learning represents another limitation. According to Forment et al. [2009], this "could create a big gap between teachers and learners, leading to a scenario where students might feel that they can learn better in their own way, using Open Educational Resources, Web 2.0 technologies and other sources of information". Already now, many teachers who are looking for innovative ways for teaching, go beyond LMSs and use technologies and applications not existing in their LMSs as well as innovative teaching approaches. The educational courses are moving to the open Web. One fascinating example of informal learning is the thenewboston[2]. This innovator, while studying programming languages, "began to realize that most of the books seemed to lack excitement. The material was useful, but they were far from entertaining". Thus, he started to record the educational videos helping people to learn programming technologies in an entertaining manner. The

[2] http://thenewboston.org/about.php

popularity exploded. With the tutorials and videos as “a gateway to a higher education, for free” and the quality that “surpass the quality of even the top colleges and universities around the world”, his channel on YouTube contains after 5 years (in January 2013) almost 3000 videos covering all major programming languages (Java, C/CPP, Objective-C, JavaScript, Ruby, Python, PHP, etc.), many development technologies (HTML5, CSS, MySQL, iPhone, Android, Adobe, 3Ds Max, etc.) . It has almost 350 thousand subscribers and more than 100 millions views. The viewers often comment that the quality of the material explanations and the understandability goes well beyond the courses on these topics that they get from their professors. The thenewboston does not stop here: more and more new videos are being introduced (including courses on Physics, Math, Algebra, Biology, etc.) and the team of people producing these videos is growing. As Forment et al. [2009] point out, “learning does not happen in the institution *management* of learning, it happens among students and teachers using whatever technology and resources they find and use in their learning”, and LMSs should support them in doing so.

LMSs lack flexible adaptations to learners’ needs, openness and interconnections with the rest of the world. The recent trend to alleviate the problem is to provide learners with more flexibility and personalization in their learning environments and enable institution-to-institution and institution-to-cloud data flow, which pushes forward the paradigm shift from closed and monolithic LMSs towards open and flexible PLEs. As Van Harmelen [2008] states “there is a shared understanding that the educational approach driving the development of Personal Learning Environments is one of learner empowerment and facilitation of the efforts of self-directed learners, also called autonomous or independent learners”.

Several definitions of a PLE can be found in the literature. One of the early definitions was given by Mark van Harmelen [2006]:

> "**Personal Learning Environments** are systems that help learners take control of and manage their own learning. This includes providing support for learners to
>
> - set their own learning goals
> - manage their learning; managing both content and process
> - communicate with others in the process of learning
>
> and thereby achieve learning goals. A PLE may be composed of one or more subsystems: As such it may be a desktop application, or composed of one or more Web-based services."

PLEs represent effective and efficient learning environments that learners construct and shape during their learning process. This involves finding and aggregating a set of tools that bring together learners and content artefacts in the context of learning activities to support learners in constructing and processing information and knowledge. More specifically, when looking at personally arranged learning environments, that is, an individual’s selection of tightly- and

loosely-coupled tools, close and distant contacts, both created and consumed objects, used for and in main (as well as side) activities, we speak of a **personal learning environment** (Wild et al. [2008]).

A learning environment that learners construct is conceptually different from a platform or a technology that is used during the construction process. Thus, a PLE should not be seen as a category of software or a specific Web platform. To avoid further confusion, we refer to a Web platform where users are able to create their personal learning environments as a **PLE aggregator**.

The fuzziness of PLE definitions infer a huge variety of possible interpretations and implementations: in the early work on PLEs, Van Harmelen [2006] provided the dimensions that characterize the possible PLE aggregators. For example, the dimensions can be heavy-weight/light-weight platform, server/peer-to-peer based, non-collaborative/collaborative, fixed/personalizable, etc. The author also discusses three PLE aggregators and how they map to the suggested dimensions. Since 2006 various systems were created to enable PLE functionality (Gillet et al. [2008], Van Harmelen [2008], Moedritscher et al. [2008], Laga et al. [2009], Yanagida et al. [2009], Reinhardt et al. [2011], Bogdanov et al. [2012a,c]).

PLEs should not be seen as a replacement for LMSs but rather as a complementary technology (Henri et al. [2008], Moccozet et al. [2011], Bogdanov et al. [2012c]). When PLEs target self-regulated and life-long learning, LMSs target the teacher-oriented and course-centric education. People through their educational career progress from pupils and students, that require teachers to guide them through their learning process, into adults capable of managing their own learning. An LMS is a helpful tool to support people at their early educational stage (Forment et al. [2009]). However, they fail to help at the later educational stages where the teacher's role becomes less important, and this is where PLEs are expected to take over. LMSs are "fixing" their limitations by opening themselves to the outside world and adding flexibility. As examples, we can list the recent Massive Open Online Courses (MOOCs) trend that is becoming popular among universities and Learning Tools Interoperability (LTI) standard that allows the integration of external tools into LMSs. PLE research is in its infancy and its main goals are to support people who are mature enough to learn by themselves and to teach people how to learn in a self-regulated manner to enable the early shift from *learn with a teacher* to *learn on your own* paradigms. This thesis looks at the technologies enabling the creation of PLEs and/or the enrichment of LMSs.

1.2 Introduction to Widgets

The software components hosted in a PLE aggregator (sometimes named as Web apps, plugins, portlets, gadgets or widgets) are referred to hereafter simply as widgets or Web widgets. A **Web widget** is a small application that can be installed and executed within a Web page by an end user. Widget code is typically written with HTML, CSS and JavaScript.

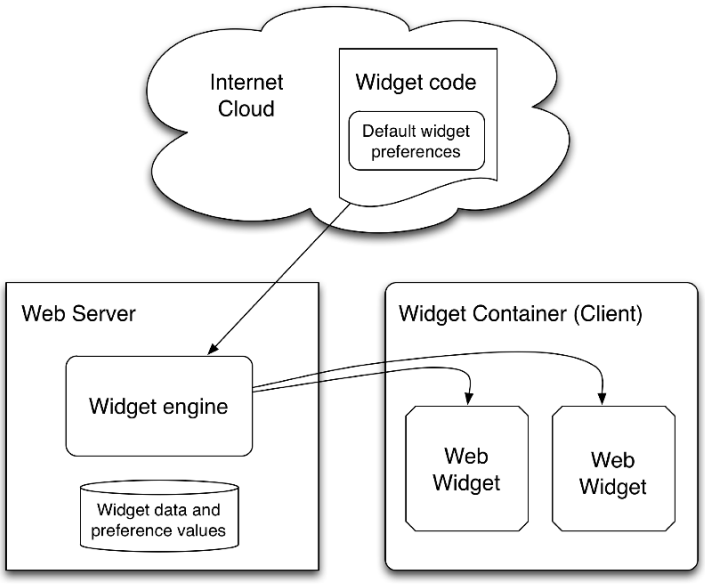


Figure 1.1: Widget architecture

The widget architecture is depicted in Fig. 1.1. A **Web platform** is a generic Web site such as Facebook, LinkedIn, YouTube, BBC News, etc. A **widget container** is a client-side execution environment that contains one or several widgets and manages their layout and representation within a page, such as navigation between the widgets, widget addition/removal, etc. A Web platform might (or might not) contain one or several widget containers. A **widget engine** is responsible for processing the widget code and rendering a widget on a page. In addition, it manages widget interaction with its Web platform. A **platform interface** is an application that implements the client-side interface of a Web platform.

The settings used to initialize and deploy the hosted widgets are referred to as **widget preferences**. For example, the units used to measure a distance can be implemented as a widget preference, where widget users can specify if they want to use the metric system (km) or the imperial system (miles). According to the standards, widget preferences are defined in the widget code and can have default values that the widget author specifies. Once widget code is run by the widget engine, the actual values of the widget preferences that the user sets are propagated into the Web platform.

PLE aggregators may provide a way to organize sets of widgets, which can form a learning context. We refer to such sets of widgets as **widget bundles**. We define a **space configuration** as a configuration file that is an extension to widget bundles (Fig 1.2). Note, that the space configuration concept is different from the space concept as defined in Section 2.2.3. Similarly to widget bundles a space configuration references several widgets, however, in addition, it incorporates the preference values for every widget. If the changes are made to the widget preferences, the space configuration is updated accordingly. The space configuration file

can be stored locally in the Web platform or shared by several Web platforms via a dedicated configuration service.

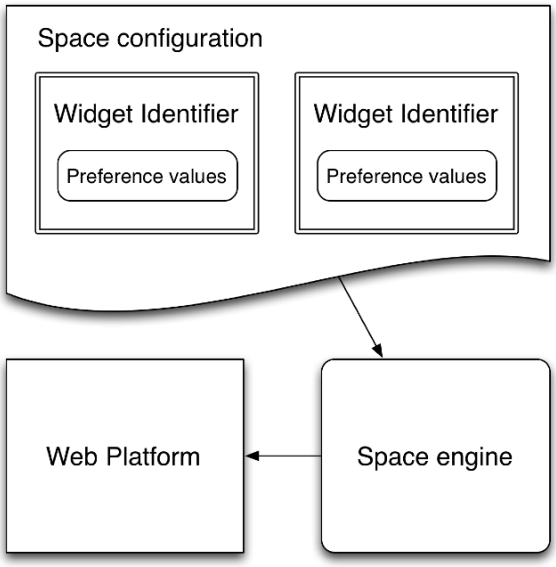


Figure 1.2: Space configuration architecture

Several widget standards exist, for example, OpenSocial widget specification[3] and W3C widgets[4]. W3C widget standard requires a package of all widget code and additional files packed in a special archive with wgt extension. This way widget code can be ported and uploaded to different widget engines. OpenSocial specification works differently: an OpenSocial widget is implemented as an XML file that is located somewhere in the Web and has its own URL. The URL has to be given to a widget engine, that downloads the code, parses it and builds a widget. In both cases, widgets require a widget engine that runs them on users' pages. There are two popular open-source widget engines for each widget standard mentioned above. First, Apache Wookie[5] is the open-source Java implementation of an engine for W3C widgets. OpenSocial widgets are rendered by the Java-based Apache Shindig engine[6].

In addition to being a widget standard, OpenSocial provides a specification for retrieving social data by a widget from its Web platform. It standardizes the structure of the API end-points and the structure of returned data, which enriches widget functionality and possible scenarios of widget usage.

From now on we leave out the prefixes *Web* or *widget* and refer simply to platforms, widgets, containers, engines, etc.

[3] http://docs.opensocial.org/display/OSD/Specs
[4] http://www.w3.org/2008/webapps/wiki/WidgetSpecs
[5] http://incubator.apache.org/wookie
[6] http://shindig.apache.org

1.3 Anatomy of a PLE

1.3.1 PLE Features

Even though there is no fixed definition of what a PLE should be, a common understanding of PLE features is emerging.

Henri et al. [2008] stress the blending of individual and collective learning and claim that PLEs should allow learners to take ownership of their learning and to control their activities.

Dabbagh and Kitsantas [2012] point out the constantly increasing usage of social media tools for coursework by students. The authors introduce a framework for using social media to support Self-Regulated Learning (SRL) in PLEs with three levels: i) **personal information management**, ii) **social interaction and collaboration**, and iii) **information aggregation and management**. Additionally, PLEs allow users to take control of their own learning by giving them a choice of tools and resources to create and manage their learning content for effective and efficient learning (Rubin [2010], McGloughlin and Lee [2010]).

Li et al. [2010] also stress the paradigm shift from the top-down approach where it is "tutors who construct the learning environment and lead the learning process" to "the bottom-up learning paradigm [that] allows students to take responsibility of their own learning experience".

The Northern Ireland Integrated Managed Learning Environment (NIIMLE) project[7] highlights the need for portable courses (Kearney et al. [2005]). Students have to be able to access data from everywhere and to reflect on and update their skills and competences. Even though it is not a direct requirement of PLEs, it shows the direction where LMSs are headed in opening themselves to the outside world and supporting SRL.

Participatory design activities carried out within the European Palette project[8] showed the need to "aim at sustaining collaboration, supporting tacit and explicit knowledge management and enhancing individual and organizational learning in communities of practice (CoPs)" - Rekik et al. [2007], Gillet et al. [2008], El Helou et al. [2008].

SRL is at the center of the Responsive Open Learning Environment (ROLE)[9] EU-funded project that decided to use widgets as a base to construct PLEs for users (Renzel et al. [2010]). In the project, widgets are combined into collections called widget bundles and can be used by learners in a grid layout similar to iGoogle.

Within the Go-Lab European project[10] that started at the end of 2012 widgets are aggregated into inquiry learning spaces to be used by teachers and their pupils for remote labs.

[7] http://www.niimle.ac.uk/home.htm
[8] http://palette.ercim.org
[9] http://www.role-project.eu
[10] http://www.go-lab-project.eu

In the light of their findings regarding current usage of Information and Communications Technology (ICT) by the Geneva University students, Moccozet et al. [2011] established the list of technical requirements and features needed for a smooth integration of a PLE aggregator as a complement to the LMS of the universities that are part of the SWITCH[11] PLE project. These requirements can be summarized as follows. The PLE aggregator to be used among SWITCH partners should:

- allow the aggregation of local and institutional resources in addition to cloud resources and across the different partner universities catalogs;
- foster collaborative work centered on a user (teamwork) rather than on courses;
- allow the system to be easily extended with relevant tools, for instance, in a plug and play fashion (as with widgets or browser plugins);
- offer a solid and versatile ePortfolio solution. ePortfolio is indeed meant to provide a continuum between formal and informal environments that allow students to manage their content and provide a showcase of their proficiencies and learning outcomes while ensuring the interoperability and data mobility from one system to another (e.g., when changing a school or an institution).
- integrate a dashboard-like feature for better dealing with different tools and platforms.

The PLE solution sought by SWITCH PLE partners should, therefore, facilitate seamless interactions between institutional and non-institutional resources and activities from the students' point of view.

We propose to classify the requirements discussed in this section into the following PLE features: (1) **learner empowerment**, (2) **collaboration with others**, (3) **aggregation and management of learning resources**, (4) **aggregation and management of learning tools**, (5) **ubiquitous access to learning resources and tools**, and (6) **reflection and learning process management**.

1.3.2 Six Dimensions to Analyse PLE Aggregators

In Sire, Bogdanov et al. [2010] we propose six PLE dimensions with corresponding implementation features as a framework to analyse the technical requirements to Web platforms that enable users to construct their PLEs. In this thesis, we further extend and improve the proposed six qualitative dimensions used to analyze the main requirements to the PLE aggregator into Aggregation, Communication, Synchronization, Organization, Recommendation, and Configuration dimensions. Every dimension addresses PLE features from Section 1.3.1 and scenarios that can be accomplished in a PLE. The **Aggregation** dimension addresses the

[11] http://www.switch.ch

visual integration of learning tools (features 1 and 4, Section 1.3.1). The **Communication** dimension supports the data exchange between learning tools (features 3-5, Section 1.3.1). The **Synchronization** and the **Organization** dimensions address the collaborative aspect of PLEs (features 1-2, Section 1.3.1). The **Recommendation** dimension measures the support for learning process management (features 1 and 6, Section 1.3.1). Finally, the **Configuration** dimension targets the flexibility in accessing learning materials from different Web platforms (features 1 and 5, Section 1.3.1). We believe that each dimension addresses an important aspect of PLE aggregators.

These 6 PLE dimensions can be used to make decisions on choosing a platform that better serves as a PLE aggregator today as well as to identify the trends and areas where further investigation is needed to pave the way for future PLE aggregators. We do not compare the platforms by the presence or the absence of a specific PLE feature but rather the lower level technical requirements to the Web platform where the feature can be supported. Though the proposed approach can also be used for desktop applications and mobile platforms, we focus here on Web platforms and Web PLE aggregators. More specifically, we focus on Web platforms that have support for hosting widgets.

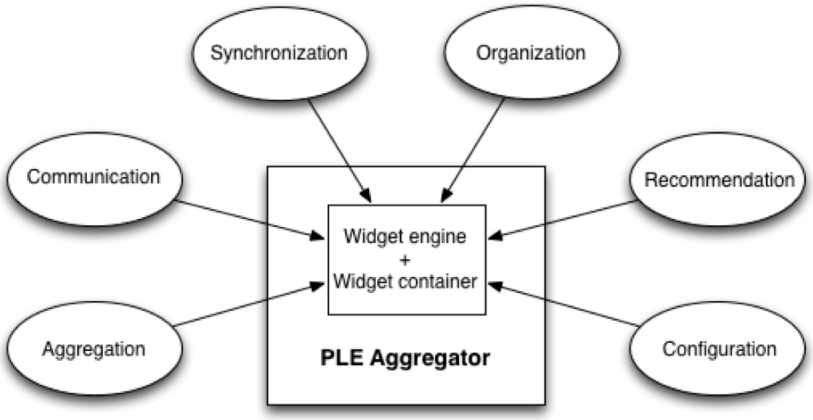


Figure 1.3: Six PLE dimensions

Fig. 1.3 shows these dimensions graphically in what could be an abstract view of a generic PLE aggregator. In this diagram we have made explicit that a PLE construction process is an aggregation of several features, as it allows one to build a learning environment to fit users' needs rather than force them to be satisfied by what is given. For instance, in a PLE aggregator with social integration, a part of the user interface is dedicated to inviting friends and accepting invitations. Similarly, in a personal widget dashboard a part of the user interface is dedicated to browsing widgets and placing new widgets on a grid on the screen.

We should note that these dimensions are not all necessary to build a PLE. For instance, users

that would select a Netvibes[12] personalized homepage as their PLE aggregator would more or less only use the Aggregation dimension. It is even conceivable that somebody uses a PLE aggregator with none of these dimensions. This is the case for users that select a simple blogging tool such as Wordpress[13] to self-reflect on their learning processes by writing text snippets (without plugins and comments, otherwise, there would be some elements of the Aggregation and Organization dimensions).

However, to improve and enrich the user experience, the future PLE aggregators should support several of these dimensions (Section 4.4). The combination of several dimensions will allow learners to design more powerful PLEs, such as for instance collaborative PLEs by including elements of the Aggregation, Synchronization and Organization dimensions. But it will also make PLE aggregators more reliable. For instance, the Communication dimension will make data more portable to avoid data lock in. Moreover, the Configuration dimension will allow users to switch to another PLE aggregator of their choice (even desktop or mobile) with minimal migration issues. Finally, the feature of being able to collaborate across different PLE aggregators will allow users to stay in touch even when their teammates use other platforms.

Table 1.1 summarizes the six dimensions to be used when measuring the PLE support of Web platforms. The definitions of the dimensions are made to both capture as many relevant features as possible as well as to make them independent, implying that Web platforms can support any combination of them. Each dimension is further divided into four features. Each feature represents one of the techniques or approaches used nowadays in various Web platforms that addresses the dimension. The number of features of a dimension that a Web platform supports defines its score in this dimension. It is true that for some dimensions it is possible to find more than four features, however it was decided to take the four most important ones to ease platform comparison. It might be refined to a more flexible scheme if needed. The dimension features were chosen based on the investigation of existing technical standards that address a particular dimension. We provide next a brief overview of the proposed dimensions to set a background for the further discussion. The full analysis of every feature, the detailed description of the dimensions and the comparison of 9 different platforms by assessing which features they support in each dimension are provided in Sire, Bogdanov et al. [2010] and in Section 4.4.

- **Aggregation Dimension**. In PLE scenarios users pick up different tools at different stages of their learning process. This is a constantly changing environment: new tools are added, some tools become obsolete. Many Web platforms allow multiple distinct software components to exist side by side. Examples range from simple pasting of HTML snippets inside blog posts to advanced standard-based widget containers or even Web desktops that come with a full set of tools. Thus, this dimension measures the degree to which the tool aggregation is supported by a platform.

[12] http://www.netvibes.com
[13] http://wordpress.com

Dimension	Features
Aggregation	- **Screen aggregation**: independent software components displayed on the same screen - **Widget standards**: standardizes the runtime API and packaging of a widget so it is not limited to a single platform - **Layout of widgets**: add, remove, group and organize widgets - **Web desktop**: session oriented, comes with a standardized set of widgets that can be opened or closed
Communication	- **Inter-widget communication**: client side messaging between different widgets - **Drag and Drop**: explicit user driven inter-widget communication - **PLE data manager**: independent management of data in a unified manner - **Linked data support**: common properties allowing widgets to understand each others data without preparation
Synchronization	- **Push data updates**: data is pushed to other instances of the same widget in the same context - **Push preference updates**: updates User/Widget preferences or adding new widgets in a context - **Real time data updates**: data is updated continuously, conflict resolution is needed. - **Data and preference history**: version history is kept and can be inspected
Organization	- **List of friends**: manage a list of friends, export and import - **Friends server**: expose friends via protocols such as Facebook connect or OpenSocial - **Access control**: use your friends when specifying access to material or functions - **Independent groups**: allow the formation of groups one can join or be invited to
Recommendation	- **Manual guide**: instructions about the activities that should be performed - **Flow enabled widgets**: Widgets enabled or made visible based on progress - **Scripted inter-widget data flow**: the result of activities in widgets are data that is processed and passed along into initial configurations of the next widget(s) - **Recommendations**: the result of activities in widgets is data that is processed and used to make clever recommendations based on what other users in this situation have liked, as well as user and domain models.
Configuration	- **Feed export and import**: OPML files with feeds in categories - **Generic export and import**: the space configuration language that contains widget preference values - **External configuration**: the space configuration is located separately from the PLE aggregator and updated continuously - **Embedding**: embedding entire PLEs into different PLE aggregators when functionality cannot be brought along in other ways

Table 1.1: The six dimensions and the features for building PLEs

- **Communication Dimension**. Data is one of the most important components of any PLE. For example, learners should be able to represent their learning goals. This representation of learning goals should be understood by both goals tracking and goals management tools (and others). In other words, data should be portable among different PLE aggregators. There are many forms of data portability which have been described in Tolk [2006]. In Web 2.0, data portability is often seen as data created or stored within one application, that can be read and/or copied/moved into another application, and finally interpreted by it. What we see today is mostly simple widgets using established standards such as RSS or ATOM, while more advanced widgets introduce their own choice of data formats and services for content and own vocabularies for preferences. The latter choices are nearly always made based on the perspective of an individual widget and not a wider widget landscape. The results are widgets that have little or no knowledge of other widgets or even of the surrounding context. The data are often specific for a single widget. More recently we have seen the emergence of client-side communication protocols that allow applications integrated at the presentation level, or Aggregation dimension, to exchange data on user's behalf, autonomously or with methods such as Drag and Drop (Dn'D). PLE aggregators could provide good supporting structures that enable widget developers to exploit more cross-widget features as well as to help them make good decisions on data portability.

- **Synchronization Dimension**. The collaborative aspect of PLEs brings a new dimension to learners. To be able to effectively manage content and the learning process a user has to be not only provided with tools to create and update content, but also be aware of changes in the content or the process (Dourish and Bellotti [1992]). This implies that relevant information, such as state changes in a shared object, must be propagated in a timely fashion. The initial architecture of the Internet allowed updates to an object's state to happen only on page reload. Thus, if two people worked on the same document, the first person could see the changes of the second person only after clicking the refresh button in the browser. Initial workarounds such as polling were unsophisticated. Recently, the situation has dramatically improved. With the advent of online chat applications, new protocols were invented which allowed updates to be synchronously propagated following software patterns known as Comet and Reverse Ajax (Crane and McCarthy [2008]) with various specifications and standards. The introduction of WebSockets in the HTML5 makes real-time communication between Web applications a reality.

- **Organization Dimension**. It is proven that collaboration among several people can often improve the learning process (Soller et al. [1999]). Thus, we believe that the presence of the Organization dimension representing the group structure is an advantage of any PLE aggregator. Web 2.0 applications explicitly model the concept of the user and of the user's list of friends, or followers. This information represents the social graph of the user for sharing data and posting notifications. Generally speaking, the social graph information is used by the social container to dynamically define different groups when

the user is interacting with an embedded application, either on the user's own page, most of the time called the user's profile page, or on the pages of other users.

- **Recommendation Dimension**. A PLE aggregator should assist a learner towards the goal achievement by providing guidance either based on explicit rules (set by a mentor or a learner itself) or on the recommendation of useful resources and learning sequences coming from an independent software agent.

- **Configuration Dimension**. In the future we do not expect that there will be a single PLE aggregator that everyone uses. Instead, we find it more realistic that there will be many distinct PLE aggregators that compete with slightly different features, design and interaction paradigms. Thus, it should be easy for users to move between PLE aggregators and also to collaborate across PLE boundaries. To achieve this, it is crucial that a widget or an entire PLE in one PLE aggregator could be easily experienced in another PLE aggregator.

1.3.3 Seven PLE Principles

The 6 PLE dimensions serve to evaluate the PLE compatibility of Web platforms. Jeremic et al. [2011] proposed a new framework to describe PLEs which is highly related to the dimensions we suggested. The authors identified 7 main principles on which PLEs are based. However, whereas our PLE dimensions are focused on technical and implementation aspects of PLE aggregators, the new Principles are more general in nature and besides technical aspects, also cater for pedagogical purposes of PLEs (e.g., interactivity and self-regulation in learning). According to the authors, the 7 Principles are as follows. The **Integration** principle means the possibility to integrate distributed and heterogeneous data sources, tools and services. The **Openness** principle targets open standards for application and device independence, long-term access to content and services, interoperability; open source software for cost-effective customizations to the users' needs and open content for more diverse and constantly evolving and improving educational content. The **Distributed identity management** principle allows users to seamlessly access different tools/services that are part of their PLEs, pull together their profile data from those tools/services, regulate their data usage within their tools/services. The **Context-awareness** principle is the improved efficiency of user's interactions with the environment through capturing and leveraging data about the user's learning context and using these data by search results, proactive recommendations, mediation of communication/collaboration. The **Modularity** principle is the ability to seamlessly "configure" a PLE for any given purpose (i.e., learning goal), by adding new and/or replacing existing content, tools and/or services and support for standardized and light-weight approaches for the development of dynamic (e-learning) mashups. The **Ubiquitous data access** principle is seamless access to and integration of profile data, data about learning activities and learning resources ability to access and use relevant resources regardless of the system/tool/service the user is currently using. The **User centricity** principle imposes the *user at the center* paradigm where learners are responsible for managing their individual knowledge and competences.

These 7 principles address the PLE essentialy on the abstract level. When these principles have to be supported within a PLE aggregator, they have to be broken down into technical requirements for the platform. The six PLE dimensions address these technical requirements. For example, the *Widget standards* and *Layout of widgets* features of the **Aggregation** dimension support the **Modularity** principle by enabling technical support in managing learning tools. The *Embedding* feature of the **Configuration** dimension and *Linked data support* feature of the **Communication** dimension address the **Ubiquitous data access** principle by enabling data format standardization and access to a PLE from different PLE aggregators.

1.4 Challenges, Contributions and Thesis Outline

We have shown in this chapter that six features, six dimensions and seven principles have to be taken into account to enable the construction of an effective PLE. The associated challenges are tackled through 8 contributions summarized below and detailed in the core chapters of this thesis.

1.4.1 Space-related Contributions

The first **challenge** concerns the current usage of PLEs by learners. PLEs are created and exploited for specific learning activities carried individually or collaboratively and relying on different resources and tools. We can hence say that PLEs are contextual; the context being defined as a specific activity carried out for a specific purpose with specific tools and resources, as well as with specific people.

PLEs being contextual, we can ask as the first **research question**: how can we enforce a context throughout a PLE aggregator?

This challenge is tackled by the following **contributions**. We formally define the space concept (**contribution 1**), that materializes the learner's context and represents a PLE that the learner constructs. We propose an OpenSocial Space extension (**contribution 2**) that introduces the space concept into OpenSocial specification in the form of Space model and Space REST and RPC APIs. We propose a way to build contextual widgets (**contribution 3**) capable of adapting to the user's context.

As a **validation**, we implement the space concept in the Graasp platform. The space concept was accepted by the OpenSocial foundation and is recognized within opensource and commercial projects. We also implemented several contextual widgets for Graasp.

These contributions address the Aggregation, Communication and Organization PLE dimensions and the elaborate discussion about the contributions is provided in Chapter 2.

1.4.2 Portability-related Contributions

The second **challenge** is related to the problem of environment portability. People should not be forced to use a single platform: they need to collaborate across "technical" boundaries and platforms, they should be able to move their environments from one platform to another.

Enabling portability is our second **research question**. How can users access the same PLE from different Web platform and how can they migrate a PLE from one platform to another?

As a **contribution**, we propose collaborative portable space configurations relying on the space configuration language (**contribution 4**). We demonstrate how portable spaces are achieved with OpenSocial (**contribution 5**). This includes the classification of migration methods and portable spaces scenarios. In addition, we propose a concept of portable platform interfaces.

We provide three **validation** approaches for these contributions. First, we demonstrate how the interoperability between Moodle and Graasp is achieved with the proposed migration methods. Second, the space sharing is demonstrated on the example of inquiry learning spaces in the Go-Lab project: export of spaces and cross-platform access to them. Third, we show how the interoperability between Graasp and ROLE Widget Store is achieved.

These investigations contribute to the Synchronization and Configuration PLE dimensions and are detailed in Chapter 3.

1.4.3 Plasticity-related Contributions

The third **challenge** is a strong need of users to be able to personalize (shape) their learning environments according to their needs, by changing a set of learning tools, adding and removing the learning resources, by adapting the graphical and functional parts of the Web platform, and sharing learning resources with others.

How can we enable an easy personalization of contexts materialized as PLEs is our third **research question**.

As thesis **contributions**, we define plasticity (**contribution 6**) as a measure of platform ability to be shaped according to users' needs. We propose the functional skin concept (**contribution 7**) used to personalize the graphical representation and functionalities of user interfaces in a Web platform. We propose cloud aggregation and sharing mechanisms (**contribution 8**).

As **validation**, we demonstrate how platform plasticity is achieved in Graasp and how Moodle can be turned into a plastic platform with functional skins. We show how the GraaspIt! tool enables easy cloud aggregation. Finally, we compare five Web platforms with respect to their support of the six PLE dimensions.

These contributions supporting the Aggregation and Communication PLE dimensions are detailed in Chapter 4.

2 Personal & Contextual Space

This chapter focuses on defining and refining the space concept that can represent a PLE on different Web platforms. This concept formalizes the constituents of a PLE from the technical perspective and serves as a PLE unit that can be understood by both learners and Web platforms. As such, it addresses the Aggregation, Organization and Communication PLE dimensions from Table 1.1.

First, we detail the *Space* concept (**contribution 1**), which materializes the learner's context and represents a PLE constructed by a learner. Then, we elaborate on the *OpenSocial Space extension* (**contribution 2**) which standardizes the Space model and Space REST/RPC APIs for Web platforms. Finally, we discuss *Contextual widgets* (**contribution 3**) capable to adapt to the learner's PLE.

As an illustration and validation of the Space concept, we describe the Graasp platform. We first show how the space concept is integrated and used in Graasp. Then, we discuss how users perceived the Space concept. In addition, we present how the Space concept is used within European projects and the OpenSocial Foundation.

2.1 State of the Art

End users of social media platforms enriched with widget containers can compose their own personalized environment. Similarly, end-user mashups can benefit from components created with one of the numerous mashup development environments, which can be Internet-based (Yahoo Pipes, Microsoft Popfly) and desktop-based (JackBe, IBM Mashup Center, Duet). Hence, it is not surprising that widgets and mashups are among the most active areas of development today on the Web. For instance, as of April 2012, the mashups directory of ProgrammableWeb[1] lists as much as 6550 mashups with 3 new ones registered every day. Similarly, the proliferation of social media platforms shows that the social interactions are the main driving force that brings people to the Web.

[1] http://www.programmableweb.com

These two trends are particularly compatible with two crucial outcomes of modern learning systems: to let people construct their own learning environment and to share their learning experiences with others.

2.1.1 Widget Containers and Data Mashups as PLEs

The idea behind Widgets and Mashups is similar to Component-Based Software Engineering (CBSE) and Service Composition, where the process of building a software application or a service is the assembly of prebuilt, reusable and independent blocks called components (Kozaczynski and Booch [1998], Clemente and Hernández [2003], Adamek and Hnetynka [2008]). The main goal behind it is to reduce the implementation efforts and costs through the reuse of the code blocks while improving the system flexibility and reliability. Components are created independently and have clearly defined interfaces and behavior, which makes it fast to rapidly assemble them together to produce a new application.

Daniel et al. [2007] stress that integration of a composite application from components can be accomplished on 3 different layers: data, application and presentation. The presentation level means the user interface of a component being integrated and the application level means the functionality (or business logic) of the component. For data integration, composite applications have their own presentation and application layer, while the data layer is in fact an integration of data independently maintained by the component applications. During the integration process, data is brought together and exposed in a unified view to the composite application. For application integration, a composite application would have its own UI, but its business logic layer is, at least in part, developed by integrating functions exposed by the component applications. Finally, UI integration combines applications by reusing their own user interfaces.

The research in CBSE is focused mainly on the layers of data and application integration, however, Daniel et al. [2007] argue that UI integration does not receive proper attention, though being very important. The authors characterize the main dimensions of UI integration and show that many research results investigated for data and application integration can be applied for UI integration as well.

A Web mashup is a Web page, or Web application, that uses data, application and UI integration based on two or more sources to create new services (Liu et al. [2007], Tuchinda et al. [2008], Hoyer and Fischer [2008], Maraikar and Lazovik [2008]). This is a generic definition of a mashup. In practice, one can see two clusters of mashups: i) data mashups and ii) widget containers. **Data mashup** is a Web application that aggregates data from two or more external online sources and renders the data in a unique way. Thus, it can be seen as a component-based software engineering technique, where reusable sources of data (API end-points as components) are aggregated and repurposed by a mashup to provide a new functionality. In the terms of the classification above, a mashup represents a composite application with data integration. **Widget container** is composed of widgets coming from a different provenance

that the user aggregates to achieve a specific purpose (Laga et al. [2009], Yanagida et al. [2009]). A widget container visually integrates the widgets that have their own UI and, thus, represents a composite application with UI integration. Since one of the main visions for PLEs is the flexible aggregation of tools, services and content by a learner, these two recent trends, mashups and widgets, receive a great interest from PLE researchers and are now converging. PLE aggregators incorporate both trends and often come in a form of a widget-based dashboard or a data mashup that can be further personalized by its owner (Severance et al. [2008], Moedritscher et al. [2008], Wild et al. [2008], Reinhardt et al. [2011], Gonzalez-Tato et al. [2012], Bogdanov et al. [2012b,c]).

Widget containers often focus on providing a customizable personalized environment where various widgets are selected and organized by the user according to some principle. They provide both graphical layout and preferences for individual widgets. A common approach, supported in iGoogle, Netvibes, Pageflakes etc., is to place widgets that are accessed at the same time into a separate tab. This could correspond to a course, a project, an interest, etc. In a similar way, data mashup development environments allow to pick up software components which are connected together with different predefined settings. Some components are without UI, they are used to fetch, aggregate or filter data. The other components provide a UI to display input fields that allow the user to enter data. The output of the mashup can be visualized in a composite graphical view.

A collection of widgets represents a widget bundle. Thus widget containers combine several widgets which are aggregated together graphically, which addresses the PLE Aggregation dimension described in Chapter 1. In data mashups, data is aggregated through some kind of event/data-flow wiring and their main goal is the support for the Communication dimension. However, there is a great potential from a learning perspective in bringing the Aggregation dimension to data mashups and the Communication dimension to widgets. Additionally, both approaches would benefit from addressing the Organization and Synchronization dimension to support collaborative activities as we explain in the following sections.

2.1.2 Context Modeling

As stated before, several widgets or services can be combined to achieve a specific purpose. However, the mere combination of widgets is not enough, since people need to collaborate with others and use different content and learning resources. Thus, there is a need to formalize and model the learning context. By the word *Context* we understand here an activity that the user is currently conducting together with its goals and participating artifacts: people, resources and widgets. Several approaches dealing with this concept started to emerge and take shape in the literature.

The *3A Model* (Gillet et al. [2007b], Rekik et al. [2007], Bogdanov et al. [2008], Helou et al. [2010]) is one of the attempts to model the context concept. There are two well-known theories in the field of Computer Supported Cooperative Work (CSCW): the Activity Theory by Leontyev

[1977] and the Distributed Cognition theory by Hutchins [1995]. The *3A Model* takes its roots in Activity Theory (Nardi [1996], Engeström et al. [1999]), Distributed Cognition (Salomon [1997], Perry [2003], Dror and Harnad [2008]) and Actor Network Theory (Latour [1996], Law and Hassard [1999], Latour [2005]), and proposes a concrete framework for designing a collaborative Web platform. It consists of three main entities (Actor, Activity and Asset) from which the name *3A Model* is derived. The main idea of the *3A Model* can be formulated as follows: "An Actor is exploiting an Asset that is a part of an Activity". An Actor could be a person, a software agent or any other intelligent object such as a remote device. An Asset represents a document or a collection of documents (items), such as a discussion thread, a wiki page or an image album. An Activity is the formalization of a common objective to be achieved by a group of actors. It can be the representation of a tangible entity such as a classroom, or an abstract thing such as a project management environment. An Activity is the context in the *3A Model*.

Both the Activity Theory and the Distributed Cognition Theory help to understand properties and processes of a learning system, however they do not provide concrete design specifications and cannot be directly used to implement a collaboration platform (Halverson [2002]).

Activity Theory represents a descriptive meta-theory or framework rather than a predictive theory, that takes into account an entire activity system (including teams, organizations, etc.) rather than one single individual. It studies the activity as a mediator between the individual subject and the social reality. The unit of analysis in Activity Theory is the concept of object-oriented, collective and culturally mediated human activity, or activity system. This system includes the object (or objective), subject, mediating artifacts (signs and tools), rules, community and division of labor. The objective for the activity is created through the tensions within the elements of the system. Both 3A model and Space model are less general than the Activity Theory but they provide a concrete model for the physical entities involved in the activity rather than the process. As such, these models give a direct guidance on how to represent the entities of a learning system and relations between them. These models can be immediately exploited during the creation of a learning system. In addition, the Space model can be used to analyse and predict what is missing from an existing learning system.

The Distributed Cognition theory tells us that the knowledge and the cognition do not only exist within an individual but are distributed over individual's physical and social environment. The individual places memories, facts, or knowledge on the objects, other people, and tools in the surrounding environment. Distributed Cognition is a useful framework for (re)designing social aspects of cognition by emphasizing the coordination between individuals, artifacts and the environment. The theory sees a system as a set of representations, and describes the information flow between these representations. These representations can exist either in the mental space of the participants or in external representations available in the environment. The Space concept gives us a concrete model and a structure to represent these environments, the distributed islands of individual's knowledge and cognition as well as the relations between them.

Ullrich et al. [2010] define the context as PLE configurations. “From an educator perspective, the ability to share PLE configurations with colleagues inside and outside their own institution, and with students is interesting. PLE configurations can include a set of learning resources and widgets suitable as example for Chinese students to practice an elementary level English course. It can also integrate activities and lists of trusty people having the right competences and ready to collaborate.”

There is a recent trend to allow users to bundle together useful widgets and share these bundles with other people or between Web platforms. The Open Mashup Description Language (OMDL) specification[2] is devoted to solving this issue, while iGoogle[3], Apache Rave[4], ROLE Widget store[5], Graasp and other Web platforms are exploiting the creation and sharing of widget bundles. Another trend is to bring social and content aspects to these widget bundles to model user's contexts.

We describe our contribution into this research area with bundles representing a set of widgets and spaces that provide a model for the user's contexts. We claim that the future of widget containers, mashup platforms, social networks, and PLEs is to support their users in creating widget bundles and spaces, that afterwards can be shared at different coupling levels.

2.2 Space Model

This chapter introduces the space and widget bundle concepts used to support learners in the management of their PLEs (Sire et al. [2009], Bogdanov et al. [2011]) as an extension of the 3A model with widgets. We focus on the description of the model where we elaborate the concepts and walk through the details and solutions. The usage of space and widget bundle concepts for portability and interoperability is elaborated in Chapter 3.

2.2.1 Learning French Scenario

We provide an example of a learning scenario to illustrate the space and widget bundle concepts. It will be used in the subsequent sections. In this scenario, a person *Alice* wants to learn French. To do so, she collects several widgets that she found useful: the *Language Resources* widget displays texts and videos in French, the *Translator* widget helps to translate words and sentences from one language to another, the *Vocabulary Training* widget allows the user to keep a list of unfamiliar words to learn. She adds these three widgets to her PLE. Then she is ready to start studying the French language. Later, *Alice* finds a *Listen to Your Pronunciation* widget helpful to improve her speaking abilities, so she also adds it into her PLE and arranges the widgets on the screen according to her preference (Fig. 2.1).

[2] http://omdl.org
[3] http://www.google.com/ig
[4] http://rave.apache.org/
[5] http://www.role-widgetstore.eu

This set of widgets represents a widget bundle. The bundle contains the four widgets described. Additionally, *Alice* names it "Learning French" and describes it with "A set of widgets helpful to learn French" or indicates how the widgets can be used to achieve the learning goal.

This is a self-directed learning scenario, where the learner herself manages her learning process and finds the needed widgets. The similar scenario can be exploited within the teacher-to-student educational paradigm. A teacher *Bob* could create a similar widget bundle for his students and explain to them how they should use it to learn French. The *Learning French* widget bundle can be also shared by *Alice* with her friends or by *Bob* with his colleagues.

Figure 2.1: *Learning French* bundle created by *Alice*

2.2.2 Widget Bundles

The way people interact with widgets is conceptually different from resources (videos, photos, books, etc). While the resources are content-oriented and represent passive objects; the widgets are functionality-oriented and are active objects in a sense that they can interact with the user and the Web platform, conduct actions on the resources, etc. The aim of a widget

is to provide functionalities that the user needs. Moreover, widgets can often be combined together to help the user to achieve a particular goal (Laga et al. [2009], Reinhardt et al. [2011], Santos et al. [2011]). These combinations of widgets are called widget bundles. In addition, widgets can interoperate with each other within a bundle. In other words, several widgets can complete and extend each other's functionality by providing together some novel functionality.

For example, in the *Learning French* scenario with its four widgets (*Language Resources*, *Translator*, *Vocabulary Training*, *Listen to Your Pronunciation*), *Alice* obtains a conceptually new functionality where she can easily get the translation and practice her pronunciation while watching a movie in French. A widget bundle is a set of widgets that are brought together by a user to achieve a particular goal and can serve as a basis for learning activities. A bundle, if found useful and efficient for this goal, can be shared with other people.

Another widget bundle example is a history bundle that includes *History video*, *Google maps* and *Wikipedia* widgets. Learners are expected to watch the videos. While they are watching a video, the *Google maps* widget displays the location of a historical event being viewed and the *Wikipedia* widget provides textual details about the event.

Learning French and *History* bundles are just templates, each containing a list of widgets, a name and a description of its goal and how to use it. Before being used, bundles have to be instantiated within a particular Web platform, which means a new context has to be created based on the bundle. For example, *Alice* uses the *Learning French* bundle in her widget container. This is the place where the widgets "live". The *Vocabulary Training* widget has a list of words that *Alice* is currently studying. She decides to give the bundle to her friend *Bob*, who takes the bundle and adds it into his own widget container. Afterwards, a new context is created in this container, and *Vocabulary Training* widget contains now *Bob*'s list of words to learn. Thus we have two instantiations of the same widget bundle in two different widget containers.

2.2.3 Space Concept

As mentioned previously, widget bundles in widget containers often consist of only widgets and lack the social aspect (people) and the user's support in organizing content artefacts (resources). The data mashups lack the social aspect and support in organizing tools by the user. The 3A model is flat in a sense that all the constituents are at the same level and there is no hierarchy or grouping concept. Additionally, it mixes together agents and people into Actors, thus complicating its usage in practice.

The **Space** concept is a contribution that unifies and generalizes these different approaches. It represents an extension and reshaping of the 3A model. The **Space** concept introduces a new separate entity, namely, *applications* that represents tools, widgets, applications used by people. In the 3A model, all three entities exist on the same hierarchical level. However, in the space concept, the hierarchy is managed only with spaces and all other entities belong to a

space. Thus, people become members of a space, resources and applications are added into a space. The space serves as a container for other entities and as the context for the associated activities.

> A **Space** is an abstract concept that materializes the user's context and aggregates people, resources, applications and other subspaces (Fig. 2.2). All these artefacts belong to the same activity, a person (or a group of them) is accomplishing towards a common goal. This common goal is the purpose of creating the space. People participate in a space and might have different access rights and roles within the space. They share content resources and applications, used to achieve their goal. A space might have subspaces that help to organize resources and applications in an hierarchical structure. The space encompasses all the activities and actions that people accomplish within the context.

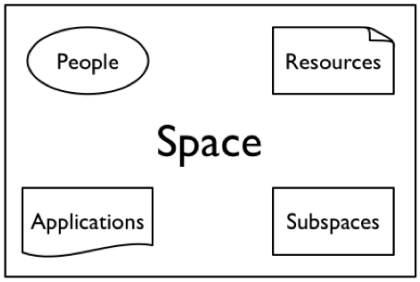


Figure 2.2: Space concept

A space provides a model for a group of people, a university course, a company division, a team project, a forum, a blog, etc. The space model can be found in many social environments in one form or another. For example, the structure of the blogging Web platform can be mapped to the *Space* concept in the following way. Every separate blog represents a separate context with its own goal, namely, the purpose why the blog was created, e.g., "Ski de fond à Lausanne", "Software for Web", etc. Thus, the blogging Web platform represents a collection of spaces - blogs. Every separate blog has its own structure that can be modelled as a space. A blog contains its authors and people that follow the blog or simply read it. All these people are mapped to *people* of the space concept. Blog posts, pictures, and videos represent space *resources*. The author of the blog can add additional widgets to a blog, such as view count widget, total page views widget, etc. These are mapped directly to the *applications* of the space concept. The users' actions (the author added a new post, a user left a comment) represent space *activities*. This is how the mapping from a blogging platform to the space concept can be accomplished. Table 2.1 shows how a similar mapping to a space can be done for several different platforms. We should note that spaces together with their artefacts represent PLEs that learners build to achieve their learning goals.

Web platform	Mapping to space
Google Group	Space → Discussion thread People → Discussion participants Resources → Attachments and URLs Activities → Messages
Blog	Space → Blog People → Followers and author Resources → Posts, pictures, videos Tools → Blog widgets Activities → Posts, comments
Moodle	Space → Course People → Students and teacher Resources → Assignments, solutions, lections Tools → Simulators, wiki, quiz Activities → Comments, questions, forum posts
Dropbox	Space → Shared folder People → People with access rights Resources → Files Activities → Adding and removing files
Youtube	Space → Videos collection People → User itself and viewers Resources → Videos Activities → Views, comments
Google Drive	Space → Folder People → Collaborators Resources → Documents, presentations, drawings Tools → Google Drive apps Activities → Comments, edits, sharing
Facebook	Space → Group People → Group members Resources → Photos, videos, files Tools → not available in Facebook groups Activities → Posts, comments
Graasp	Space → Space People → Owners, contributors, viewers Resources → Files, Google docs, cloud resources Tools → OpenSocial apps Activities → Comments, posts

Table 2.1: Web platforms mapping into space

Another way to look at the spaces is from the social network perspective. People in social media platforms have a list of artefacts surrounding them: a list of friends, colleagues, relatives; a set of documents, videos, photos, files; several tools; and associated activity streams. Thus, in a social network people are at the center and all the entities belong to them (Fig. 2.3). There are several initiatives that provide models for it. For example, both Friend of a Friend (FOAF) and OpenSocial provide a model for people and their relations to other people or entities within a social network.

Figure 2.3: Person in a social model

The important part of a social network is the person's context, that includes groups where the person is a member, projects that the person is involved in, etc. The *Space* concept models these contexts. If we compare the space concept with a model of person we can find many similarities (Fig. 2.4): as in the Person model, a space is at the center and the space artefacts surround it. The space artefacts include people participating in a space; documents, videos and photos shared within the space; shared widgets and activities conducted within the space by its participants. Even though OpenSocial has a Person model, it lacks support for the *Space* concept. To enable the usage of the space concept we submitted a **Space extension to OpenSocial**, which is currently being integrated. The extension introduces a Space model into the specification and, in addition, targets to improve the portability and data interoperability when people migrate between Web platforms or access spaces from different PLE aggregators. Spaces can be seen as a part of a big social network, where every person is linked to both people and spaces that are shared with other people. Spaces in turn can contain some other subspaces (Fig. 2.5).

Figure 2.4: Space in a social model

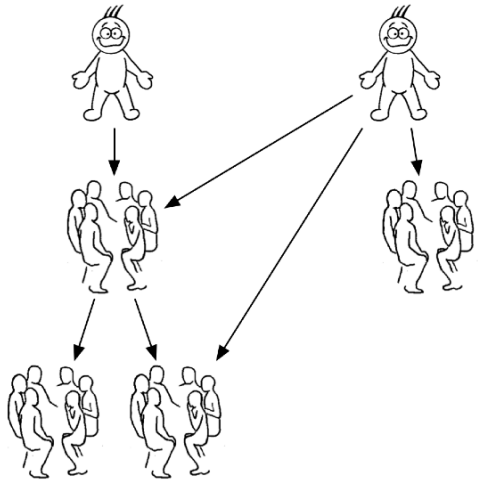


Figure 2.5: Person and space relation in a social model

2.2.4 OpenSocial Spaces

OpenSocial specification was launched by Google in 2007 and now represents an independent standardization organization - OpenSocial Foundation. It provides a standardized model for a social network and standardizes a set of common APIs to retrieve information about a social network from a social media platform. The API standardizes the way information about people, friends, resources, activities is retrieved. A special widget standard (OpenSocial widget) was first proposed and implemented by Google and later spread to other Web platforms. These widgets can be found everywhere ranging from simple blog sites such as Blogger to social media platforms (Myspace, Orkut, Friendster, etc.) and business-oriented networking solutions (LinkedIn, Oracle, XING, etc.). A social media platform that implements support for OpenSocial can ensure that any widget implemented according to the standard will run properly when plugged into a Widget container. In other words, a widget becomes portable and can run in different social media platforms. With the advent of a reference open source implementation for OpenSocial API (Apache Shindig), any social media platform can quickly start hosting OpenSocial widgets.

The OpenSocial specification solves two problems at the same time. First, social media platform developers do not have to create a new naming scheme for their API and, secondly, widgets developed according to the OpenSocial specification become portable and can run on any Web platform that is OpenSocial-compliant. The detailed analysis of the benefits related to portability and interoperability is provided in Chapter 3.

Despite the fact that OpenSocial solves the problem of Web platform extensibility and widgets portability, it has two major limitations. First, it focuses only on people. It provides APIs to retrieve a list of friends for a given person, a list of person's activities, person's albums, a list of person's widgets, etc.

It is known, however, that depending on the person's context, a list of context-specific widgets is needed. The list of people can change from one context to another. The same person might have a completely different role in different contexts. This context could be a workspace, a forum topic, a discussion with friends or planning a trip. Social media platforms often contain the context concept in one form or another. As an example, there might be different discussions and sub-discussions in the same forum, each one with its own topic, own list of participants and own list of resources. Groups or events in Facebook is another example of the context. In many social media platforms the context is a crucial component and if widgets were able to retrieve the user's context (from the hosting platform), it would greatly improve user's experiences (Dey et al. [2001], Wolpers et al. [2007]).

To understand the second limitation, one should distinguish between two types of applications: namely, widgets and Platform Interfaces. While widgets are usually considered to be small applications (weather forecast widget, translator, calendar), a Platform Interface is a relatively complex Web application such as a forum, a text editor or a personal learning environment. These Platform Interfaces can be seen as meta-components since they might

have the capability of managing other applications such as widgets, for example. OpenSocial standard provides good support for widgets, but lacks support for Platform Interfaces.

Since the context concept did not exist in OpenSocial we suggested a contextual space to be introduced and used as a model for this concept (Bogdanov et al. [2011]). Before, with OpenSocial it was impossible to model, for example, a university course with participants and resources. The space extension enables this functionality. In addition, being the representation of a user's context, the space permits the development of Contextual widgets (Section 2.2.5), that take user's context and preferences into account.

We narrow down the definition of the context concept to a space concept as it was introduced before. It should be noted that the support for people, resources and widgets already exists in OpenSocial, however, all these entities are centered around a person and not around the person's space/context as seen in Fig. 2.3 and Fig. 2.4.

Every social concept in OpenSocial specification consists of a model, REST APIs and JavaScript APIs. The **Space extension** to OpenSocial specification describes the space model (Appendix A.1), the corresponding REST APIs (Appendix A.2) and JavaScript APIs (Appendix A.3). The Space model represents a list of fields that any space can contain. The JavaScript APIs represent a wrapping around REST APIs. The Widget concept existed in OpenSocial but there was no model describing a widget, so we had to formally introduce it.

Table 2.2 shows the proposed APIs extension to be able to work with spaces. We should note that in our first proposal we had to introduce the *type* parameter (Bogdanov et al. [2011]). In OpenSocial the *People* service is responsible for retrieving a list of people connected to a user. With the space extension, there is additionally a list of people for every space (for example, a list of space members). In order to handle both scenarios and differentiate between a Person and a Space in the APIs, we suggested to add a field *type* to the API which could be equal to either "@space" or "@person". By doing this, a pair (id,type) would uniquely identify either a space or a person, for which a list of connected people has to be retrieved. However, with the recent trend to use Internationalized Resource Identifiers (IRIs) within OpenSocial, the additional *type* parameter is no longer needed and the parameter *gid* allows uniquely identify a space, a person, or any other OpenSocial object. This is why the *Person* service API contains now only *gid* object that can uniquely identify a Space or a Person. A list of people connected to a person (friends, colleagues) or to a space (members, participants) can be retrieved (Table 2.2, first row).

The other OpenSocial APIs (*Activity Streams*, *Documents*, *Groups*, *AppData*) can be extended in a similar way. For example, *AppData* request returns by default application data for a widget that belongs to a user. With the new extension, a widget can belong to either a person or to a space. Thus, application data can be retrieved for widgets belonging to either people or spaces. The parameter *gid* allows to specify for which item (space or person) application data should be retrieved.

People	**API: /people/{gid}/@all** - All people connected to item (space or person) with id {gid}
Activity Streams	**API: /activitystreams/{gid}/@self** - All activities for item (space or person) with id {gid}
Documents	**API: /documents/{gid}/@self** - All documents for item (space or person) with id {gid}
Groups	**API: /groups/{gid}/@self** - All groups within item (space or person) with id {gid}
AppData	**API: /appdata/{gid}/@self/{appId}** - All data for app {appId} and item (space or person) with id {gid}
Spaces	**API: /spaces/{spaceId}/@self** - Profile record for space {spaceId} **API: /spaces/{gid}/@all** - All spaces for item (space or person) with id {gid}
Apps	**API: /apps/{appId}/@self** - Profile record for app {appId} **API: /apps/{gid}/@all** - All apps for item (space or person) with id {gid}

Table 2.2: OpenSocial space extension

In addition to the extensions of the existing OpenSocial services, we introduced two new services: *Spaces* and *Apps*. The APIs for *Spaces* are similar to those of *People* service. The first API for *Spaces* service (Table 2.2) allows to retrieve detailed information about the space with id equal to *spaceId*. The second API is similar to the extension of *People* API, but instead of returning a list of people, it returns a list of spaces for a given space or a given person. With *People* and *Spaces* APIs, one can traverse the social graph by listing all people and spaces connected to a specific person. Moreover, all subspaces of a space and all people for a space (space members) can be listed.

The first API of the *Apps* service retrieves detailed information about an app with a given identifier (*appId*). The second API allows to get a list of apps for a person or a list of apps that belong to a space. This API is similar to the previously described requests for *People* and *Spaces* services.

For brevity, we do not provide the full details here and they can be found in Appendix A. Additionally, we provided an overview of OpenSocial JavaScript APIs to work with spaces online[6].

2.2.5 Contextual Widgets

Contextual widget is another important contribution of the thesis. It is a widget that can find out the information about its context and adapt its content and representation accordingly. This is made possible with the space concept in the OpenSocial specification, since widgets can retrieve user's context information from a social media platform (the user's current space and people, tools and resources in the space). This possibility can greatly improve widgets personalization and contextualization as well as widgets reuse. By taking context into account, these widgets can better extend the functionality of a hosting platform, because their visual interface, displayed data and functionality can be changed according to the context, in which the user is currently interacting. The same widget might display different people and different resources depending on the user's context.

In the *Learning French* scenario, *Alice* uses the *Language Resources* widget that displays texts and videos in French. One can re-implement this widget as a contextual widget. The new widget can play all the texts and videos it finds in its parent space. *Alice* adds this widget to her *Learning French* space and adds several French resources (videos and texts) to the space. Because the newly created widget is contextual, it can find out the context (space) in which it is located via the OpenSocial Space extension APIs. With its context known, the widget retrieves a list of resources in the *Learning French* space again via OpenSocial Space APIs. When the French videos and texts are obtained by the widget, they are displayed to *Alice* so that she can learn French. *Bob* learns German and has already a *Learn German* space with useful resources. Once he adds the contextual widget *Language Resources* into his space, the widget finds its parent space, retrieves resources (the German texts and videos) from the space and displays them to *Bob* who can start studying. The new widget is Contextual, since it adapts its behavior to the specifics of the user's current context.

2.3 Graasp as a Prototype Implementation of Spaces

The Graasp platform is being developed in the React Group at EPFL to investigate and exploit the potential of social media for collaborative learning and knowledge management purposes. The main objective is to provide users with a stand-alone PLE aggregator to study its acceptance and usefulness for their learning. Another objective is to enable the support of the six PLE dimensions in Graasp based on the space concept. The space concept acceptance is investigated to find out missing parts or concepts needed by learners.

Graasp represents the redesign and improvement of the Web platform called eLogbook (Rekik

[6] http://docs.opensocial.org/display/OSD/Space+Proposal

et al. [2007], Gillet et al. [2007a]) that was previously developed at EPFL as a proof-of-concept implementation for the 3A model. In Graasp, we used modern technologies to simplify the user interface and the interaction with the platform. We reinforced the space as a hierarchical and grouping concept compared to the non-hierarchical structure of the entities of the 3A model. The space became the core feature of Graasp. Last but not least, we introduced applications (as OpenSocial widgets) into the platform.

2.3.1 Graasp Overview

Graasp implements the Space concept and this concept is the core feature of Graasp (Fig. 2.6). The platform contains 4 types of entities: resources, widgets, spaces, and people. It targets the management of people's spaces. Graasp supports users in creating and sharing resources and widgets with other people in the context of a space. More generally, it is a multi-purpose collaborative platform that assists users in external content/widgets aggregation and information organization. In addition, it provides search functionality and recommendations to guide people. The privacy control mechanisms allow users to manage access rights within a space. Widgets are implemented according to the OpenSocial specification, that bring greater flexibility and extensibility to the system. Next we explain in more details the notion of a space, which is central to Graasp.

2.3.2 Space as Graasp Core Feature

Construction of an effective and efficient learning environment involves finding and aggregating a set of tools that bring together people and content artefacts in the context of learning activities to support learners in constructing and processing information and knowledge. These PLE requirements are addressed by Graasp with the space concept as introduced in this chapter, where each space contains applications and resources shared with other people to achieve a particular learning goal. Fig. 2.6 shows a single space and its content as it is seen by users. Graasp helps learners create and manage their own spaces for teamwork and studying. Through spaces it fosters sharing and collaborative work among peers. On the other hand, Graasp makes it easier for students to aggregate and organize local, institutional/company and cloud resources into spaces. The emphasis is made on the simple and efficient accomplishment of these actions. Thus, local resources such as pdf, video, text files can be added into a space by a simple drag-and-drop action from users' desktops. Institutional and external cloud resources such as YouTube videos or SlideShare presentations can be aggregated with GraaspIt! bookmarklet, which is explained in the Section 4.2.4.

Since Graasp provides an open learning environment, there is a clear need for effective privacy control mechanisms protecting against unauthorized access to social data. Instead of adopting a complicated privacy management scheme that is difficult to cope with, the privacy settings are maintained at the space level.

Based on its purpose and its owner's choice, a space can be public, closed, or hidden. Public spaces are globally visible and allow every user to join. Closed and hidden ones are only accessible upon explicit invitations. Hidden spaces are not searchable and they are only visible to space members. Closed and hidden spaces are especially useful when students want to carry out their peer-based projects without being disturbed by others or feeling that they are "observed" by the tutor.

Within a specific space, users are allowed to take different roles: owner, contributor, and viewer. Each role is associated with a set of rights allowing users to perform diverse actions such as moderating the space, adding new resources in the space, commenting, rating, tagging, bookmarking, etc. Assigning different roles in a collaborative space makes users aware of their duties and gives them the opportunity to collaborate by being allowed to perform specific actions.

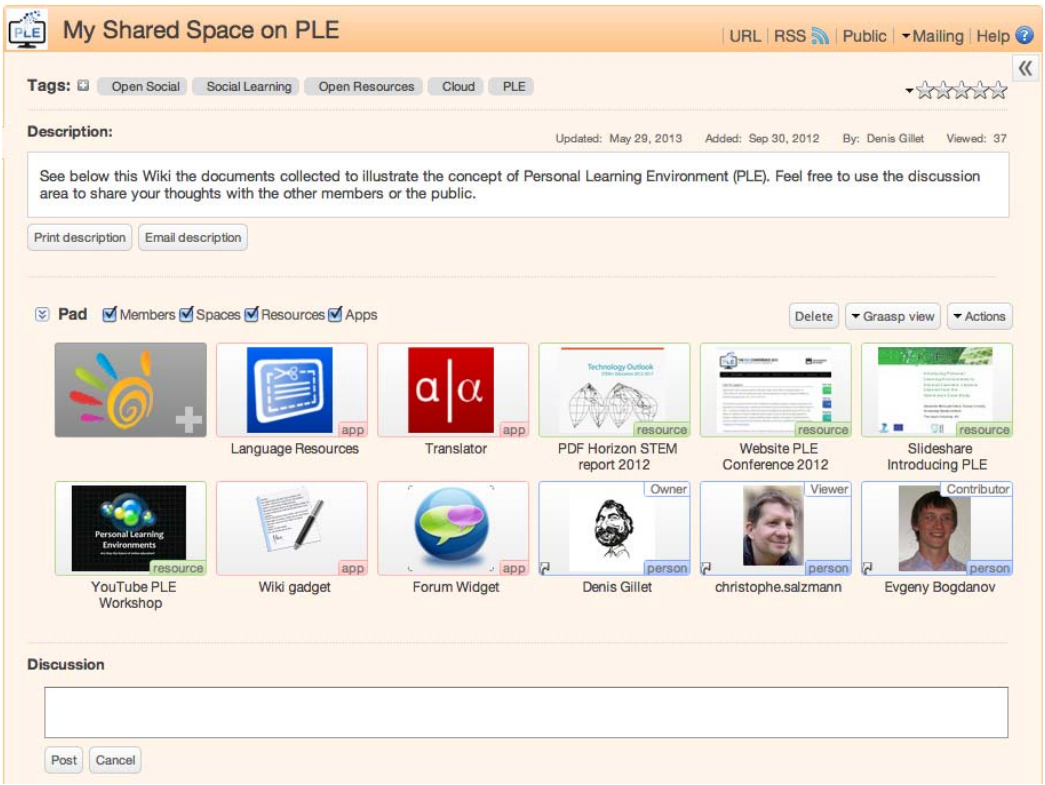


Figure 2.6: A shared contextual space created in Graasp that integrates resources gathered from the Cloud, such as YouTube videos, SlideShare presentations, OpenSocial Widgets, Web pages or PDF documents with previews

2.3.3 Extension via Widgets

Graasp is a highly extensible platform where students or tutors can bring new functionalities in the form of OpenSocial widgets. To render OpenSocial widgets, Graasp uses Apache Shindig. This Apache Shindig is extended with the OpenSocial Space and App APIs to allow widgets in Graasp to save and retrieve information about spaces and resources. Thus, Graasp serves as a widget container, where widgets can run and communicate. Space owners can add into their spaces any tool relevant to their learning activities. This capability reinforces the learning experience because it enables useful learning-oriented tools to be added and launched during the learning process. Through widgets, one can achieve personalization and adaptation of the space behavior to the specifics of learning tasks.

Moreover, different collections of widgets can be associated to different spaces, making the aggregation contextual. Widgets can be bundled together with other resources in spaces and can, thus, be directly linked with relevant content. This feature alone allows teachers to provide handy learning kits that learners can further customize and share. Thanks to this feature, default functionalities of Graasp are made flexible and extensible. For instance, in a project space, students can add a calendar widget configured with a series of milestones and deadlines. They can easily add simulation widgets, educational games or lightweight remote laboratory clients as well as many domain-specific widgets.

2.3.4 Evaluations

Graasp, as a tool that is meant to help students in their collaborative work or personal resources management, has been tested in two different real-world settings. The following sub-sections go into more details for each of these evaluation campaigns.

Tongji University Evaluation

To examine the acceptability of Graasp and the Space concept in terms of supporting collaborative learning, Graasp was used as a collaborative work platform in a project-based course of "Human Computer Interaction" offered at Tongji University in China (Li et al. [2012]). We had 28 undergraduate students involved in the course, and they were divided into 8 teams. Each team was asked to accomplish a group project, and Graasp was introduced to the students at the beginning of the course. Students were entitled to create their project spaces, share resources with each other, play different roles in the project, and work with different learning-oriented apps. A survey was conducted with the students participating in the course, aiming at evaluating Graasp acceptability in sustaining collaborative learning.

Evaluation results show that in general Tongji students who took part in this experiment were satisfied with using Graasp to enhance knowledge management and collaboration while preserving the privacy of their social data. More specifically, 64% of the participants considered Graasp useful as a platform for sharing and organizing resources, 46% of them perceived it as

an adequate place to collaboratively manage their projects, and 46% of them recognized its usefulness as a system aggregating content from various sources. Slightly over a half of the students (52%) confirmed that Graasp improved their motivation for carrying out their teamwork. Regarding the usefulness of Graasp as a collaboration platform, 57% of the students expressed their preference for carrying out teamwork within project spaces. 74% of them thought it was useful to share items with others using spaces. From the perspective of supporting collective knowledge management, 59% of the participants considered it convenient to structure and organize resources using different spaces. When asked whether learning-oriented widgets can enhance their learning experience or not, 63% of the students confirmed that the integration of widgets into their learning process was helpful. As far as the privacy control feature is concerned, 74% of the students were satisfied with having control over the privacy levels of spaces. In addition, students considered Graasp to be convenient in organizing content with subspaces and tags, but these features were less exploited due to the small number of resources created by users in the platform.

Geneva Soft Skills Workshop Evaluation

The second evaluation was held at the occasion of a workshop, entitled Soft Skills, organized by the Geneva University and targeted to new PhD students (Bogdanov et al. [2012a]). The aim of this workshop was to give some advice regarding research work in general, and introduce them with most useful Web 2.0 tools. For the workshop organization, the tutors extensively used Graasp to communicate, to collect, and to organize in different subspaces the material for the workshop. Students who had registered to the workshop (17 in total) have also been invited almost 2 weeks in advance and, thus, got a chance to get accustomed with the tool, before the actual session, by registering and creating their own personal profile. During the workshop, the students, after a short introduction, were handed a small exercise (30 min) to perform using Graasp. In the scenario for this exercise, students were about to write a survey paper and had to collect some relevant material resources, organize them with spaces and/or tags (according to their own convenience), and then share them (by sending invitations) with other people for comments or review.

The qualitative results in Fig. 2.7 show the perceived usefulness of Graasp for aggregating, organizing, and sharing of resources. These results are taken from a quick and short questionnaire answered at the end of the exercise by 13 out of the 17 registered students. One thing to notice is that, beyond the usefulness of Graasp as a whole to aggregate and share resources, the majority of users found the space notion useful for organizing resources. This is a significant result that shows the added value brought by Graasp since the space notion was unknown to most of the students. These results show more generally that the majority of students who answered the questionnaire agree that Graasp is useful for those three purposes and, as such, suited both for personal and collaborative work.

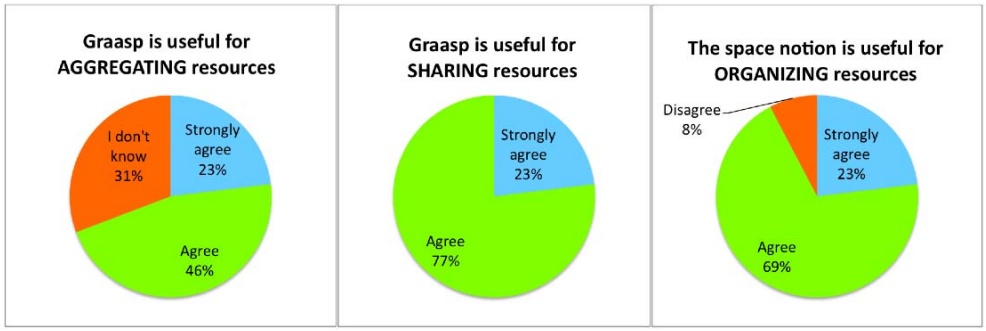


Figure 2.7: Qualitative results for the Geneva experiment concerning Graasp usefulness for aggregating, organizing, and sharing of resources.

2.4 Other Space Usage

In addition to Graasp, the space and widget bundles concepts were found useful in other projects. The main focus of the ROLE project is the usage of widget bundles and spaces to enable self-regulated learning. The widget bundles and spaces are created by people to satisfy their personal needs for learning. Later, these bundles and spaces are shared by people to foster the knowledge exchange and reuse of the existing teaching/learning approaches. The European Go-Lab project uses OpenSocial extended with spaces. In the project, remote and virtual labs are modeled as spaces that contain several widgets representing different functionalities of a lab.

As shown in Table 2.1, the Space concept can model different types of social structures. Indeed, we proposed the space concept to the OpenSocial Foundation and it was positively received by the community. More specifically, the representatives from JIVE, IBM connections, SURFnet, Apache Rave were interested in having spaces for their products as well. The Apache Rave project listed the Space extension as a "highly desired/needed feature"[7]. Thus, we developed a patch to the OpenSocial specification that adds support for spaces and implemented the required services in Apache Shindig which is the open-source implementation of OpenSocial specification. The Space extension was accepted by OpenSocial and is on the roadmap for OpenSocial 3.0[8] while being incubated in the OpenSocial 2.5.

A number of contextual widgets were initially developed for Graasp and Moodle, however, these widgets will work in a wider range of containers with OpenSocial spaces support enabled. Graph widget[9] and Recommender widget[10] are two examples of contextual widgets. Graph widget takes data from documents in a space and shows it on the graph. Recommender widget, when added to a space, shows recommendations for this space: people, widgets, resources,

[7] http://rave.apache.org/
[8] https://opensocial.atlassian.net/wiki/display/OSD/Spec+Changes+-+v3.0
[9] https://github.com/react-epfl/gadget/blob/master/demo/graph/gadget.xml
[10] https://github.com/react-epfl/gadget/blob/master/demo/recommender/gadget.xml

spaces. We also enabled the generation of widget bundles from existing in Graasp spaces and sharing of bundles in the ROLE Widget store, and vice-versa, by creating a space for the user based on existing widget bundles.

2.5 Discussion

The space concept is a generic concept. In this chapter we showed its applicability and usage for the OpenSocial specification. Nevertheless, it can be used for other social modeling mechanisms as well. For example, the Facebook apps already can access information about a Facebook event, group or page. These abstractions are similar to the space concept as described in this chapter, and thus can also enable contextual widgets where a Facebook app can retrieve information about the parent event or page and adapt accordingly. However, because Facebook apps target only Facebook platform, the code for Contextual widgets is not portable and can not be used in other platforms. In contrast, OpenSocial targets the wide variety of platforms, including enterprises, social networks, educational platforms, etc. Another parallel direction is the definition of the space concept within the Semantic Web and Linked Data initiatives. For example, the space ontology[11] was proposed within the ROLE project.

An interesting observation can be done from the Table 2.1 regarding platforms mapping to the space concept: while all these Web platforms have a space concept, resources and people, most of them lack the tools that extend the default functionality provided by the platforms. We believe that this is the area, many Web platforms have a big room for improvement. Imagine, for example, the YouTube platform where a user can add a contextual widget that would take videos from a playlist and run them according to the user wishes. Another example is Google Groups extended with a contextual widget that supports clever filtering of conversations based on the many criteria an expert user would like to have.

2.6 Conclusion

In this Chapter we detailed the three thesis contributions. The **Space** concept represents Personal Learning Environments that learners construct during their learning process. We explained how the abstract Space concept can be brought to the PLE aggregators with the example of the Graasp platform. The second contribution represents an **OpenSocial Space extension** that includes Space model and Space REST and RPC APIs. The Space model and APIs is the standardized way to retrieve and exchange the space information and content between Web platforms implementing the OpenSocial specification. The Space concept together with OpenSocial extension enables the creation of **Contextual widgets**.

The Graasp platform implements the Space concept and the OpenSocial Space extension APIs

[11] http://purl.org/role/specs/terms

and serves as a validation platform for the Space concept. The user evaluations showed that they liked the space concept and they are favorable to the idea of personalizing their learning environments.

The main goal of the Space concept is to model the user's context when the user works in a Web platform. With the introduction of the Space concept to such standards as OpenSocial, the portable and reusable tools (widgets, contextual widgets) become available. The current chapter focused on the general description of the Space concept. The next chapter will present the cross-organizational access to space data and the migration of spaces with their data from one platform to another.

3 Personal & Contextual Portability

The focus of this chapter is the Synchronization and Configuration dimensions based on the Space concept introduced in Chapter 2. Previously, we saw a space as a PLE that the user constructs and exploits within a PLE aggregator. In this chapter, we discuss how a space can be accessed from different Web platforms and how the migration of a space and its associated data between Web platforms can be achieved.

This chapter introduces the notion of *Collaborative Portable Space configuration*, relying on the *Space configuration language*, as the **contribution 4** of the thesis. Another approach to enable portability of spaces is the usage of the OpenSocial Space extension for portability that represents the **contribution 5**. To detail this contribution, we first introduce the concept of space portability and show how the OpenSocial Space extension helps to achieve it. Second, we detail five *Migration methods* that allow a space created in one platform to be used in another platform. Third, we provide five criteria to classify the scenarios of portable space usage. Finally, we introduce the concept of *Portable Platform Interfaces* that enable cross-platform reusable implementations of User Interfaces.

We detail three validation settings to illustrate the described contributions. First, we discuss the Moodle plugin implementing the OpenSocial Space extension and show its interoperability with the Graasp platform on the space level. Second, we show the ubiquitous access to Go-Lab spaces in Graasp, demonstrating spaces export and cross-platform access to spaces. Third, we highlight the interoperability between Graasp and the ROLE Widget Store on the widget bundle level.

3.1 State of the Art

Interoperability has always been an important problem in computer systems and is the focus of many research efforts in both academia and the industry. Turnitsa [2005] suggested the currently used version of Conceptual Interoperability Model with 7 interoperability levels: no interoperability, technical, syntactic, semantic, pragmatic, dynamic and conceptual in-

teroperability. Interoperability is very important for PLEs as well. Teachers and learners in different Web platforms would like to collaborate and cooperate with each other. Students would like to access their resources from the Web platforms they are used to. When changing or leaving a university, people would like to bring along their learning spaces to new platforms. Our solutions to these problems include using OpenSocial, thus next we describe the existing approaches in comparison with OpenSocial.

3.1.1 Interoperability in LMSs

Sharable Content Object Reference Model (SCORM)[1] and IMS Global set of standards[2] are interoperability standards that are very popular in the world of LMSs. The SCORM specification is composed of three parts: Content Packaging, Run-Time and Sequencing.

- The Content Packaging section specifies how content should be packaged and described. It is based primarily on XML.
- The Run-Time section specifies how content should be launched and how it communicates with the LMS. It is based primarily on ECMAScript (JavaScript).
- The Sequencing section specifies how learners can navigate between parts of the course (SCOs). It is defined by a set of rules and attributes written in XML.

IMS represents another set of standards with Question & Test Interoperability and Content Packaging being the most frequently used. Both standards allow to extract a course content as a zip archive from one LMS and add it into another one. Many other standardization efforts exist as well (Section 9, Enoksson et al. [2006]).

Forment et al. [2009] discuss the importance of Interoperability for LMSs. More precisely, the authors raise a problem of opening LMSs to the outside worlds and bringing external tools to LMSs. They investigate the possibility to integrate external third-party tools into LMSs in an interoperable way via the Service Oriented Architecture (SOA) that is "a software engineering approach that provides a separation between the interface of a service, and its underlying implementation. For consumer applications of services, it does not matter how services are implemented, how contents are stored and how they are structured. In the SOA approach consumer applications can interoperate across the widest set of service providers (implementations), and providers can easily be swapped on-the-fly without modification to application code." To enable SOA, they use Open Service Interface Definitions (OSIDs) for authentication and authorization and IMS Learning Tools for Interoperability (IMS LTI) specification to integrate LMSs with external tools. External tools supporting LTI will run in all LMSs that implement the IMS LTI standard and an adaptation of a tool to a specific LMS is not needed anymore.

[1] http://scorm.com/scorm-explained/technical-scorm
[2] http://www.imsglobal.org/specifications.html

For example, the IMS LTI support was added into Moodle 2.2[3] and works in the following way: when a user is logged in within Moodle and tries to open an external tool, Moodle sends signed HTTP POST request to the external tool. The signing is based on a shared secret key and managed by two-legged oAuth protocol. Afterwards an external application can retrieve a user identity, information about the user (first name, last name, address), her role in a course (admin, student, etc.), information about the course (course id, title) and render itself within Moodle.

The OpenSocial specification standardizes the data model and interactions in a social network: people, their friends and connections, their resources, activities, groups, etc. It provides a model to describe the elements of the network and a set of APIs to access the data. Similarly to SOA approach for LMSs, OpenSocial abstracts specific implementations of social networks into Web APIs. If we compare OpenSocial and Moodle Web services described by Conde et al. [2010] or IMS LTI, we can see several similarities: OpenSocial manages authentication and authorization of external tools (implemented as OpenSocial widgets), OSID does the same. The OpenSocial with Space extension enables functionalities similar to LTI: getting user information, getting course information, etc. However, in contrast to the functionalities provided by LTI, that focus solely on LMSs, OpenSocial targets a wider audience with companies and social media platforms and has good support for social networking and interactions, which is limited in the IMS LTI approach. Re-use of tools developed with OpenSocial for other non-LMS related platforms provides another valuable benefit.

The European Project LUISA (Lui [2008]) focused on the usage of Semantic Web Services (SWS) to improve the interoperability for content discovery and search within LMSs.

> "The objective of the overall LUISA work has been to exploit the advantages of a Semantic Web Service Architecture to make richer and more flexible the processes of query and specification of learning needs in the context of Learning Management Systems and Learning Object Repositories. The key innovations of LUISA are:
>
> - Enables the expression of queries in terms of ontologies.
> - Locates the best sources/providers for given queries (learning needs) using Semantic Web Services technology.
> - Suggests tentative compositions based on learning needs.
> - Is able of getting resources considering the user preferences.
> - Enables different query resolution/composition strategies, including educational knowledge."

To enable semantic search and harvesting within LMSs, Rodríguez et al. [2006] investigated in the project how the XML-based representations based on IMS Digital Repositories Interop-

[3] http://moodle.org/mod/forum/discuss.php?d=191745

erability (IMS DRI) can be mapped to semantic ontologies with OWL and what architectural changes are required.

Dietze et al. [2007] discuss the limitations of current approaches to content interoperability within LMSs that are based on data/metadata standards. The resources are usually manually associated to the objectives of a learning process. This association is static and accomplished at design-time, when the process is described, which introduces several limitations of re-usability and dynamic adaptability. They propose a solution based on the usage of Semantic Abstraction Layers and Mappings, that enables the dynamic run-time invocation of Web services according to specific needs and objectives of an actor.

3.1.2 Semantic Web and Linked Data

The goal of Semantic Web is to enable computers to *understand* data and to minimize human intervention when data is processed and *understood*. One of the most popular approaches for semantics is Linked Data. The authors Health and Bizer summarize its key principles in their book “Linked Data: Evolving the Web into a Global Data Space”:

> "So what is the rationale for adopting Linked Data instead of, or in addition to, these well-established publishing techniques [Microformats, CSV data dumps, Web APIs]? In summary, Linked Data provides a more generic, more flexible publishing paradigm which makes it easier for data consumers to discover and integrate data from large numbers of data sources. In particular, Linked Data provides:
>
> - **A unifying data model.** Linked Data relies on RDF[4] as a single, unifying data model. By providing for the globally unique identification of entities and by allowing different schemata to be used in parallel to represent data, the RDF data model has been especially designed for the use case of global data sharing. In contrast, the other methods for publishing data on the Web rely on a wide variety of different data models, and the resulting heterogeneity needs to be bridged in the integration process.
> - **A standardized data access mechanism.** Linked Data commits itself to a specific pattern of using the HTTP protocol. This agreement allows data sources to be accessed using generic data browsers and enables the complete data space to be crawled by search engines. In contrast, Web APIs are accessed using different proprietary interfaces.
> - **Hyperlink-based data discovery.** By using URIs as global identifiers for entities, Linked Data allows hyperlinks to be set between entities in different data sources. These data links connect all Linked Data into a single global

[4] Resource Description Framework

data space and enable Linked Data applications to discover new data sources at run-time. In contrast, Web APIs as well as data dumps in proprietary formats remain isolated data islands.

- **Self-descriptive data.** Linked Data eases the integration of data from different sources by relying on shared vocabularies, making the definitions of these vocabularies retrievable, and by allowing terms from different vocabularies to be connected to each other by vocabulary links.

Compared to the other methods of publishing data on the Web, these properties of the Linked Data architecture make it easier for data consumers to discover, access and integrate data."

3.1.3 OpenSocial and Linked Data

The OpenSocial specification improves the interoperability between platforms. The same goal inspires the Linked Data vision. By comparing two approaches, the pros and cons of both can be found and their applicability in different areas can be evaluated.

Linked Data provides a very powerful and flexible way of describing data in a machine-understandable way. It targets the discovery and integration of data coming from many different sources. It is designed to work on a global (the whole Internet) scale. Due to its design, it has limitations when interoperability is required on a smaller scale: within an organization or between several organizations only. First, it requires a rather steep learning curve (Linked Data and SPARQL) which is a disadvantage comparing to the simple RESTful APIs, which are used by many Web developers[5]. The second limitation is the performance. Since the data might be located in different places in the Web, many HTTP requests have to be issued to get the data. Moreover, the SPARQL implementations require traversal of a graph, which is much slower than retrieving data from a relational database. For the relatively simple tasks of data migration from one organization to another or retrieving data from some organization, the Linked Data approach might be an overkill. The third limitation concerns the dynamic nature of data and privacy. Linked Data promotes the open sharing of data on the Web, which means that organizations create data dumps that can be processed by others. If the data visibility can be easily changed by people (e.g., from private to public) the data dumps should be regenerated immediately, which might not be feasible for a company. The management of authentication to different data-endpoints might be another limitation of Linked Data, when used internally within organizations. The authors in Health and Bizer (Section 6.3) foresee the usage of the crawling pattern for LinkedData applications, rather then the on-the-fly URI dereferencing pattern needed for real-time data exchange.

Some examples of Linked Data usage internally within enterprises exist (BBC example in Health and Bizer, Section 3.2.3), but it is not widely used at the time of this writing. The

[5] http://code.google.com/p/linked-data-api/wiki/API_Rationale

standard-based OpenSocial, though not as flexible as Linked Data, is popular within enterprises and social networks. The main disadvantage is that the social model cannot be easily and arbitrarily extended as in the case of Linked Data. However, OpenSocial provides easy to use REST APIs with JSON-based data representation. The data format and APIs are standardized, which enables interoperability when data is accessed and processed. The OpenSocial does not target the data discovery (as Linked Data) but rather data retrieval and exchange. Since the data is often located in one place (one company), it is very fast to retrieve the data compared to the SPARQL engine.

3.1.4 Open Mashup Definition Language

A recently introduced Open Mashup Definition Language (OMDL) specification provides a standard for widget bundles. It specifies a bundle as a list of widgets and information about their layout. This specification complements OpenSocial spaces and Collaborative Portable Web Spaces (Section 3.2). With the OMDL specification a widget bundle can be extracted from a space existing in some container into a runtime independent XML file and used to instantiate a new space in the same or in a new container. Thus, the OMDL specification provides a portable way to share widget bundles and it is used by Graasp (Section 2.3). The OpenSocial Space and Collaborative Portable Web Spaces are richer than OMDL concepts that target the instantiations of widget bundles inside containers.

3.1.5 Spaces and Contextual Interoperability

In the context of the Communication and Configuration PLE dimensions, we focus on achieving interoperability at the space level. A space can contain several widgets and also the data that was produced by the space users. Thus, it represents a unit of widget composition or mashup instantiation. It becomes natural to consider some mechanisms to keep it independent from the runtime environment through an adequate configuration mechanism. A configuration file independent from the Web platform and containing a list of widgets and their preferences we call a *space configuration*. We distinguish three levels of the space configuration. The first level describes how to combine the components of a space (layout, graphical theme, data-flow, etc.). The second level describes the initial and default values of any preference or property of its components. Finally, the third level describes all the current values that were changed in the second level by the users since they started to interact with the space, and eventually some other user's data which are stored within the space and not in external services. As an important contribution of the thesis, we define three concepts that the space configuration addresses: **portability**, **broadcasting** and **co-editing** (Fig. 3.1).

The *portability* concept addresses scenarios where a new copy of a space configuration or of some configuration part is created. This concept allows users of a space configuration, such as an iGoogle page, to move their space configurations to another platform, such as Netvibes, for example. We can also imagine that some adaptation to the space configuration applies to

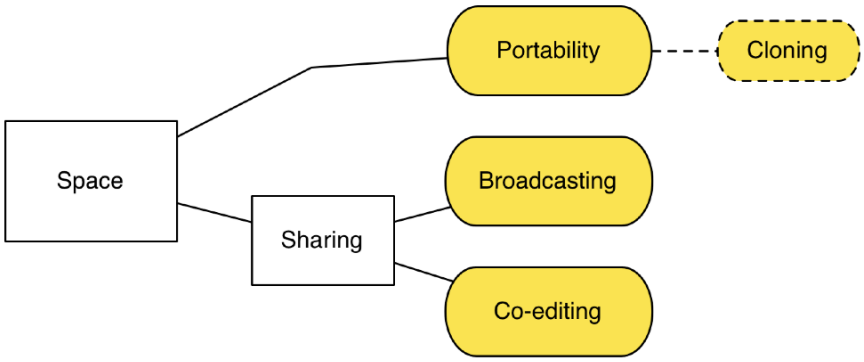


Figure 3.1: Space exchange concepts enabled by decoupling the space configuration from the Web platform.

adapt it to runtime platforms with very different characteristics, such as converting it from a desktop to a mobile platform. The portability concept also addresses the case when a user invites another user to duplicate his own space, starting or not from a snapshot of his personal data. This later usage could be called *cloning*. For example, students or teachers moving from one university to another could carry their sets of tools and associated data with them (as a space configuration) when the migration occurs, since data and tools should belong to a person and not to an exploited Web platform.

In the *sharing* concept, no copies of a space configuration are created, instead the same space configuration is accessed from different Web platforms and/or by different people. Thus, the space configuration is shared by the space owner with other people. We distinguish two types of such sharing: *broadcasting* and *co-editing*. In *broadcasting*, the owner invites other users to join the space and to keep them updated when the space is modified by the owner. During *co-editing*, the owner invites other users and all the changes to the space done by anyone are merged together so that there is only one space configuration. For example, a shared course space created by teachers in their LMSs can be accessed by course students from their own PLE aggregators.

The portability concept can be combined with the broadcasting and co-editing concepts in that different users sharing a same space can be accessing it from different runtime platforms. The co-editing concept can be applied to a single user in the case where this user accesses the same space from multiple platforms. Moreover, since widgets composition and mashups consist of several components and every component can have its own set of properties and user data, sharing can happen at different granularity levels. For instance only some components may be shared, only some properties of some components, or some subsets of user data. Sharing can also happen with different coupling levels depending on the frequency of synchronization between the different instantiation of the space as we will develop in the next sections.

To know to what extent our three concepts based on spaces can be realized today, we compared

two widget platforms, namely iGoogle and Netvibes, and two mashups development environments, namely Yahoo Pipes[6] and Afrous mashup engine[7] (Sire et al. [2009]). We first noted that, to some extent, the Widget platforms already support these concepts at the individual widget level as explained in the Table 3.1.

	Portability	**Broadcasting**	**Co-editing**
iGoogle	Limited	Limited	Limited
Netvibes	Limited	No	Limited

Table 3.1: Current support of the proposed concepts in widgets.

The portability concept is limited on iGoogle as it works only between users accessing the widget from iGoogle. In that case it appears as a "Send my settings for this gadget" checkbox when sharing it with someone. Netvibes also provides an internal form of widget portability since there is a feature to archive a widget and to restore it at any time from/within Netvibes. Netvibes, as well as iGoogle, also permits to embed certain widgets by generating a code snippet which can be cut-and-pasted into another environment, however in that case the preferences and the configuration are hard coded into the code snippet. This type of portability is also limited to users familiar with HTML. The broadcasting and the co-editing concepts are supported in iGoogle, for certain widgets. The concepts are materialized through a "View my content" and a "View and edit my content" options that can be checked when sharing a widget. However, this is limited to sharing on iGoogle. A very limited form of co-editing is also available on Netvibes, as users can access their widgets from the desktop and the mobile version of Netvibes.

The portability concept is supported at the space level on iGoogle and Afrous as they both allow to save and to import their space configuration into a file in different proprietary formats (respectively a gadgetTabML or a JSON file) (Table 3.2). The Yahoo Pipes platform gives the possibility to clone a pipe from a mashup gallery to edit it.

	Portability	**Broadcasting**	**Co-editing**
iGoogle	Yes, proprietary	No	No
Netvibes	No	Yes	No
Yahoo Pipes	Yes, proprietary	Yes	No
Afrous	Yes, proprietary	Yes	No

Table 3.2: Current support of the proposed concepts in space configurations.

The broadcasting concept is supported on Netvibes as users can create one public profile page that their friends can see and which is synchronized with their changes. It is also supported on Yahoo Pipes and on Afrous. On Yahoo Pipes mashups can be embedded into "badges" which

[6] http://pipes.yahoo.com
[7] http://www.afrous.com

can be installed on other platforms through a code snippet that starts a JavaScript runtime library. Similarly an Afrous mashup can be embedded through a JavaScript runtime library, as the full mashup engine is a JavaScript application. For Yahoo Pipes and Afrous, we consider the embedding as broadcasting because only a reference to the mashup is embedded, hence any modification made by the owner will be reflected in the embedded pipe. This applies to the data-flow composition and constant value parts of the configuration, and not to the user's data, as the concept of user's data (or preferences) is not explicit in these mashup platforms.

3.1.6 Proposed Solution

The current solutions have not yet fulfilled all the potentials of spaces, although our selection of 4 platforms shows that there is an effort in that direction. Currently the most advance solution is the gadgetTabML format that Google is using to export and import personal pages on iGoogle, and which is an undocumented internal feature. The OPML file format is more widespread but it only handles feed widgets in tabs, all other widgets and their preferences are lost. The mainstream widget standardization committees, such as W3C, Open Ajax Alliance, and Open Mobile Terminal Platform (OMTP) do not seem to have plans in that direction, and neither do mashup development environment providers. We also could not see such an effort in LMS and PLE areas. The papers (Sire et al. [2009], Bogdanov et al. [2011]) and related work within the ROLE project raised the interest in the space concept. For example, the recent OMDL specification works towards the similar goal.

We suggested two solutions to address the described portability and sharing concepts: i) a space configuration concept that incorporates a space structure and makes it possible to decouple a space from its platform; ii) a solution based on OpenSocial space extension. We will show in Section 3.1.3 that OpenSocial specification is suitable for enabling interoperability for PLEs at the space level.

3.2 Collaborative Portable Space Configurations

This section first details the Space Configuration contribution of the thesis, and, then, describes the second contribution - Space Configuration language.

3.2.1 Space Configuration Elements

We define the following elements for a Space Configuration:

- *List of widget or mashup components with default or user settings:* the list refers to components which are published openly and which can be easily referenced via URIs. Depending on the component format, the settings may be expressed slightly differently. Unless the component settings are exposed as properties, they are not stored in the

space. Components may therefore have separate dependencies on external Web services in the Cloud or real world information for user data storage, but in that case they should rely on other services to share these data.

- *Layout of graphical components:* since not all space containers support the same layouts, the layout should be treated as suggestions.
- *Event and/or data-flow wiring of components:* mashups are created by connecting their components, this is also becoming part of a widgets composition as mechanisms such as inter-widgets communication and drag and drop become available (Isaksson and Palmér [2010], Zuzak et al. [2011]).
- *Participants list:* it reflects the social context in which learning occurs. In the proposed solution, one owner of a PLE space will invite other learners to share an initial space configuration. That initial configuration template corresponding to a scenario (e.g. "get to know each other") will be created by a teacher. The list of participants defines the scope for broadcasting or co-editing when the space is shared.
- *Sharing list:* in order to broadcast or co-edit a space, the sharing list defines precisely the basic units of sharing such as individual properties of a component, a full component, or other properties such as the position of each component in a given graphical layout for instance. We propose as a sharing policy that all the properties not explicitly shared can be changed by each user independently of the others.
- *Refresh rate list:* programming patterns such as Reverse Ajax or Web Sockets allow to develop notification mechanisms in such a way that changes to shared properties can be notified synchronously. The alternative is to refresh them only on page reload. The refresh rate list indicates the preferred mode of update for each shared property.

A configuration can be seen as a file that references several widgets, includes the preferences of the widgets, people's access rights, etc. This configuration is a template that supports the creation of a new space from external resources. However, the configuration should also store all the shared settings which have been updated by the users in order to distribute them to the group. In this regards we currently describe these settings as properties (i.e. key/value pairs), which is common among most widget and mashup platforms. These properties have many different usages in various environments. Some properties serve as user interface preferences, some others as widget state variables, and finally some others as application data storage. The combination of the three orthogonal dimensions: participant lists, sharing rights over a property, and refresh rates can be used in different combinations in the user interface.

3.2.2 Space Configuration Language

As an example we show how to extend the gadgetTabML configuration files so that they can support different broadcasting and co-editing concepts. Our proposition is to use a new XML

element and 3 new XML attributes (Table 3.3), which can be set on the different elements of a host configuration language to decide who can see them, how they are shared and at which refresh rate.

Attribute or Element	**Value**	**Meaning**
<Participants>	user identifiers (e.g. email)	a list of people sharing a space configuration
@participants	user identifiers	a list of people viewing a particular component
@sharing	BROADCASTING (a) EDIT (b) NO (c)	type of sharing, (a) means only the owner can change the value, (b) means everybody can change it, (c) it's a private property space configuration
@refresh	SYNC (i) ASYNC (ii)	delay before updating the shared properties, (i) means synchronously, (ii) means on page reload

Table 3.3: Proposed additions to the gadgetTabML configuration language to support our scenarios.

The example in Listing 3.1 shows the *Learning French* space configuration (Section 2.2.1). Here, Alice is a French teacher and she has shared the *Learning French* space with her students Bob, Charlie and Dave which are declared in the <Participants> element. According to that configuration Alice broadcasts the resourceUrl property of the first LANGUAGE RESOURCES widget (i.e. she imposes the URL with the French text to read). This is declared with the sharing attribute set to "BROADCAST" on the <ModulePrefs> declaration of the property. The updates that Alice will make to the resource URL will be broadcasted asynchronously (i.e. on page reload) because of the refresh attribute set to "ASYNC".

The second widget (TRANSLATOR) is visible by everyone and everyone can define his own settings because there is no sharing attribute defined.

Alice has chosen to display the LISTEN TO YOUR PRONUNCIATION widget only in the view of Bob. This is specified with a participant's attribute on the <Module> that declares the widget. Alice has also chosen to impose the value of the "trainText" <UserPref> property to Bob (attribute sharing set to "BROADCAST") so that she can specify the words and sentences for him to practice.

Finally, all the user preferences of the VOCABULARY TRAINER widget, except the "currentList" (attribute sharing set to "NO"), are shared and can be edited by everyone because its sharing attribute is set to "EDIT", and updates are synchronous.

```
<GadgetTabML version="1.0" xmlns=".../GadgetTabML/2008">
 <Tab title="Learning French" skinUrl="skins/sampler.xml">
   <Layout iGoogle:spec="TWO_COL_LAYOUT_1" />
     <Participants owner="alice">
       alice@kth.se bob@epfl.ch charlie@epfl.ch dave@kth.se
     </Participants>
     <Section>
      <Module type="LANGUAGE RESOURCES">
        <UserPref name="currentTab" value="2"/>
        <ModulePrefs resourceUrl="http://..."
            sharing="BROADCAST" refresh="ASYNC"/>
      </Module>
      <Module type="TRANSLATOR">
        <ModulePrefs url="http://..."/>
        <UserPref name="fromLanguage" value="fr" />
        <UserPref name="toLanguage" value="en" />
      </Module>
      <Module type="LISTEN TO YOUR PRONUNCIATION"
          participants="alice@kth.se bob@epfl.ch">
        <UserPref name="trainText" sharing="BROADCAST"/>
      </Module>
      <Module type="VOCABULARY TRAINER" sharing="EDIT" refresh="SYNC">
        <UserPref name="currentList" value="1" sharing="NO"/>
        <UserPref name="wordLists"
            value="{'1':['france','petite'],'2':['adorer']}"/>
        <UserPref name="progresses" value="[80,70,0,100]"/>
      </Module>
    </Section>
    ...
  </Tab>
</GadgetTabML>
```

Listing 3.1: Extended GadgetTabML file for usage as a space configuration (some namespace declarations and some URLs have been removed for simplification).

This is an example how to map the participants, sharing and refresh rate dimensions onto a host XML-based space configuration language. Of course there are several directions in which the proposition can be refined. For instance, we could imagine a syntax to declare if users can add new widgets/components, if they are shared, and if they can be removed from the space. Similarly, we could define a syntax to define some other configuration elements that could be shared such as the layout. Finally, other extensions are necessary to allow finer grain access control models with sub-groups of users with broadcasting capabilities, and not just use the owner/others dichotomy.

3.2.3 Architecture

Usually widget and mashup platforms consist of a server component, that we call *engine*, and a client-side component, that we call *container*. The engine is mainly responsible for

maintaining the space configurations, usually in an internal database. The widget or mashup components source code may also be installed on the engine server, or it may be available from third party servers. The container is responsible for visualizing the space inside a browser.

We added a *Configuration Server* to this architecture (Fig. 3.2). This component is required to make a space portable, or to share it. To simply clone a space, the space engine needs to copy the current state of its configuration to the configuration server. To start its sharing, the space configuration engine needs to communicate with this configuration server to maintain the configuration up to date as defined by the participants, sharing and refresh configuration settings.

This architecture admits two variants (Fig. 3.2). In the first variant, a single, centralized configuration server has been setup to allow two users to share a configuration from different widget platforms. In the second variant, the configuration data is hosted on different federated configuration servers. The second variant is more flexible in that it allows portability or sharing between platforms that may be connected with different configuration servers, or to have users with different configuration servers in case they need to have an account on it before using it.

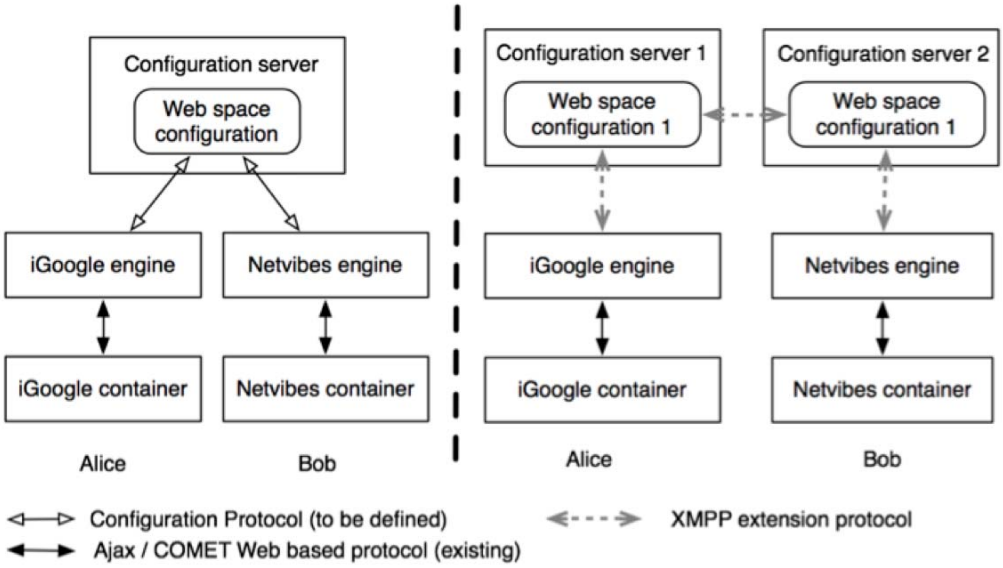


Figure 3.2: Two visions of the Portable Web Spaces architecture.

3.2.4 Example

The following hypothetical example illustrates the usage of portable collaborative spaces. A teacher has a class where students are divided into groups of four. The task of each group is to develop a computer program collaboratively. The program should consist of four main blocks, each solving a particular problem, and a block that combines the four parts to solve the task.

The teacher decides to use the following setting for each group of students: a co-edited widget, that contains the final code, and a widget for every user, where the particular task is specified and student can do some unit testing on their own code. Thus, the resulting space contains these five widgets (Fig. 3.3).

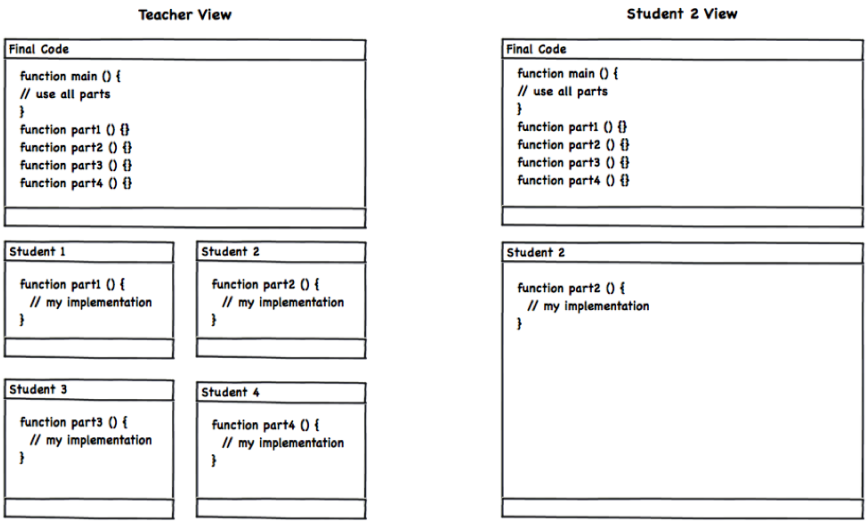


Figure 3.3: Teacher (left) and student (right) views of the collaborative space

The co-edited widget (the most top widget in the picture) is available to all students in a group: its *@participants* attribute (Table 3.3) is set to a list of group students. Access to each of the individual four widgets is limited to one student, by setting the *@participants* attribute of a widget into the corresponding student. Even though the teacher decides to create his space in iGoogle, some of the students prefer to use Netvibes. Since every Web platform builds a space for each user based on the same shared space configuration, the space is available to students in any platform they want. A Web platform builds a space with two widgets for every student: the personal widget to accomplish the individual task and the collaborative widget to integrate the individual solutions together. Every student solves his own task in his personal widget and then incorporates his solution into the co-edited widget. The changes to the co-edited widget are propagated synchronously to all the other students in the group and the teacher. In the end, the teacher gets the final solution to the task in the co-edited widget.

Later the teacher discusses the successful class with his fellow teacher from another university. The latter likes the idea, and in order to try it out he receives the clone of the space configuration to try it in his Pageflakes portal.

3.2.5 Possible Implementation

In this section we propose a possible implementation of a space configuration server. The space configuration server only needs to manage configuration data, thus it can be imple-

mented with any kind of database. To support sharing, it must also propagate data updates. One solution for synchronous updates is an extension of the XMPP protocol: Google Wave Federation Protocol. In that scenario, the configuration server would implement/utilize this protocol and create a new wave for every space configuration. Then, every property of a configuration with a synchronous refresh rate could be saved as a new wavelet into the space wave. The participants list of the space would be copied to the participants list of the wave. If the refresh rate is asynchronous the property would simply be saved into the configuration server database.

The space engine communicates with the configuration server via a predefined protocol, maybe using the Google Wave client-server protocol, not only for synchronous properties but also for asynchronous ones.

The container would communicate with the engine using COMET or Reverse Ajax methods or higher level APIs such as Web Socket API[8].

Then, two options are possible to manage the sharing policy of the different properties. The lightest solution, that would not require a modification of the engine, is to manage the sharing mode at the configuration level. However, this would not allow to support the broadcasting mode, as everyone could actually change the property values locally in her browser and not only the owner. The second option is to manage it at the engine level, so that it prevents users from modifying a shared property if it is broadcasted and they are not owners. In both cases it would also be necessary to modify the container part of the platform, since the user interface should provide specific affordances and feedbacks to the users so that they know what is shared and how. To some extent the widget and graphical components developers should also take this into account. Of course, the widget or component API must also be modified to take into account property changes triggered by remote users, for instance through a callback mechanism.

Regarding the Organization dimension, introduced through the participants list, a potentially interesting solution could be to delegate the management of groups of people to an external service such as a kind of OpenId but for groups, let us call it GroupId. It should go beyond the definition of a simple list of friends, as defined in an OpenSocial container, to support the management of several groups and subgroups a user belongs too. A group managed by a GroupId server could be declared as a participants list in a configuration through a GroupId URL. This way it should be easier to support many collaboration models, for instance to define open/public participation lists, without changing the space configuration language.

The Collaborative Portable Space Configuration approach requires creation and acceptance of a configuration standard, that can be used by several Web platforms. Even though we described the model, provided the details of the configuration language and gave the implementation hints, we do not have a working prototype that exploits this model.

[8] http://www.w3.org/TR/websockets

3.3 Portable Spaces with OpenSocial

This section provides an alternative to space interoperability discussed in Section 3.2. This approach is based on the usage of the OpenSocial Space extension as proposed in Chapter 2. It targets the access of space content via standardized OpenSocial APIs and migration of space data between different Web platforms. The comparison between the two approaches is given in Section 3.9.1.

3.3.1 OpenSocial and Portability

We first discuss the OpenSocial standard from the data interoperability perspective and explain what it means to migrate a widget from one platform to another. OpenSocial provides a standard to structure social data and specifies the APIs to access these data. This standard aims at enhancing data interoperability, code reuse and widgets portability.

When one talks about widget portability, it is important to understand the difference between portable code and interoperable data. Fig. 3.4 shows that widget code runs inside a widget container that conforms to the standard specification. This makes the widget portable among OpenSocial widget containers. In addition, widgets can access data from the container via OpenSocial APIs. These APIs are standardized to be the same in all OpenSocial-compliant containers. The data structure that APIs use is also standardized. A widget is authorized to retrieve and update data in the container via a special security token that the container provides. Any OpenSocial widget is built upon the predefined standard: it "understands" how the APIs look and the structure of the input and output data for these APIs. Widget developers rely on the fact that these APIs and data structure will be the same in different containers, thus implemented widgets will be container-independent.

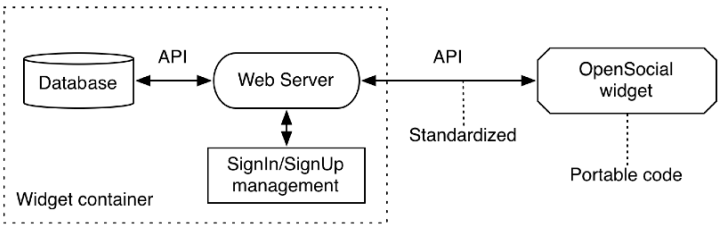


Figure 3.4: OpenSocial widget architecture

It is important to mention that OpenSocial specification is not limited to OpenSocial widgets, but also encompasses platform-to-platform communication, where one social media platform can access data from another one via the OpenSocial APIs, the structure of returned data being understood by everyone. This means that one container can retrieve data from several other social networks by exploiting the same code in each case, which allows to achieve high code reuse.

Widget Portability

The code of a widget is made of CSS, HTML and JavaScript. In addition, widgets may use specific libraries to manage their properties on the page, switch between different views, call HTTP requests to the container and third party services, handle oAuth and signed requests, etc. These common libraries are provided by any OpenSocial-compliant container. As it can be seen in Fig. 3.4, a widget (implemented according to the OpenSocial specification) together with the used APIs can be detached from one container and plugged into another container without any additional code adaptation to the container specifics. Code portability is the capability of a widget to run in different containers.

Data Interoperability

Data interoperability is more complex than code portability (Tolk [2006]). To correctly deal with data coming from/to a container, a widget has to know its structure and API end points, and where the data can be retrieved. To enable data interoperability between different widget containers or between a widget and its widget container, both syntactic and semantic interoperability should be satisfied.

Syntactic interoperability means that two systems are able to exchange data with a given syntax. This is achieved by providing common APIs, that are understood by different containers. Thus, a widget can send/receive data to/from its container, or containers can exchange data with each other. This standard insures that a widget will be able to access data in different widget containers provided they are OpenSocial-compliant. For example, a widget running within one OpenSocial container can call *Person* API and retrieve data from another OpenSocial container, since they both implement the API according to the specification.

Semantic interoperability ensures that data is not only exchanged, but both the widget and its container understand the structure (or semantics) of the data. This is ensured by both the standardization of APIs end-points and the structure of returned data. As example, when a widget is about to retrieve a name and a phone number of a person, it issues a request to the OpenSocial *Person* API end-point. The widget knows in advance the structure and semantics of the returned object (this is provided by the standard), thus it can safely find and process the information, such as the name and the phone number.

Data is interoperable among different OpenSocial containers: it does not matter in which container a widget runs, the data entering the widget has always the structure, that widget understands, and data leaving the widget is understandable by its container. API end-points are also invariant. The code (or actions) a widget should run to retrieve and process the data stay the same independently of the container.

Widget Instance Portability

Widget portability and data interoperability together ensure widget instance portability. A widget instance is a widget code that runs in a container including all the data that belongs to this widget within this container.

To clarify the distinction between a widget and a widget instance, let us consider the following example: a widget with a functionality of showing people' location on Google map. This widget is simply a piece of code that implements this functionality. We extend this widget by allowing it to retrieve user's friends via OpenSocial API and display them on the map. This is still only a widget code but with enabled data interoperability between the widget and its container. The user now adds this widget to iGoogle container and it shows her friends on the map. This process is called a widget instantiation. The widget code plus its data (widget owner, friends of the owner) constitute a widget instance. The user now decides to move from iGoogle container to another one. If the widget can still run and show user friends from iGoogle in the new container, we say that this widget instance is portable.

3.3.2 Portable Spaces

While there is no wide usage for widget instance portability (since widgets are often small applications that do not include much data), portability of spaces and mashups is important due to a richer nature and a vast applicability domain (Severance et al. [2008], Hoyer and Fischer [2008]). A space as a combination of tools (functionality-centric) differs from a data mashup concept which is an application aggregating data from different sources (data-centric), though sometimes these concepts can be very similar.

> **Portability** is the ability to move/access a space existing in one Web platform into/from another Web platform (Fig. 3.5).

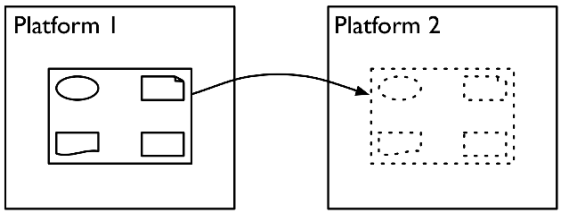


Figure 3.5: Portability concept

As it was explained in Section 2.2.2, a space is often an instantiation of a widget bundle containing a list of widgets and a description how to use them. Before a widget bundle can be used, a space has to be created within a particular Web platform based on this bundle (similarly to a widget instance in Section 3.3.1).

OpenSocial widgets are portable. Thereby, if a bundle is simply a list of OpenSocial widgets, it will be portable as well. The space, on the other hand, contains some data that is saved within its platform. Thus, to achieve space portability, we have to make sure that space data saved in one platform can be either moved to or accessed by another platform. This can be accomplished similarly to the portability of widget instances which will be described in the following sections.

There are many real life scenarios where the portability of spaces is important. People moving between different companies can bring along their personal spaces if the company policy allows it. Employees from different company divisions can access shared spaces from their own platforms. A teacher, moving from one university to another, would like to move his course spaces from one LMS to another. Students from different universities can access the same course from their own platforms. Spaces can be cloned and changed afterwards to satisfy the specific needs of different people. Spaces can be extracted from their platform and behave as stand-alone environments. All the mentioned scenarios improve the personalization and support the user in achieving the accessibility and mobility of data and tools, which is necessary in everyday work, learning, leisure activities.

3.3.3 Migration Methods

The main goal of OpenSocial is stated as follows: “Friends are fun, but they’re only on some Web sites. OpenSocial helps these sites share their social data with the Web. Applications that use the OpenSocial APIs can be embedded within a social network itself, or access a site’s social data from anywhere on the Web.” Thus, OpenSocial is mainly about building portable applications and sharing data. However, one can look at the OpenSocial from the different angle, namely, to explore its support for portability and interoperability, which will enable the migration of spaces between platforms.

The current section shows how space portability can be achieved with OpenSocial. The APIs of OpenSocial with the Space extension help to achieve migration of spaces from one platform to another or accessing space data from different platforms. When we talk about a space there are two parts: data that represent the space and its visual representation (UI). While space data is unique at any point of time, its visual representation might vary in different platforms. Technically speaking a visual representation for a space can be implemented either as a meta-widget or as a part of a platform. By meta-widget we mean a widget code that can render other widgets inside itself. In the first case of space representation as a meta-widget when the migration takes place, both data and space UI can be ported, in the second - only data. We focus here on the first case, the second scenario can be accomplished in a similar manner provided the code responsible for rendering a space within a platform utilises OpenSocial APIs.

Let us consider a space and its meta-widget running in a *Platform1* (http://platform1.com), all data for this space is saved in the *Platform1*. The user decides to move to another *Platform2*

(http://platform2.com). Since both platforms are OpenSocial-compliant, the meta-widget code can be easily ported. However, the user gathered some data in *Platform1*, while interacting with the space. Even though the meta-widget can be successfully moved, the data requires special consideration. There are five possible scenarios (Fig. 3.6). We leave out a scenario where the user does not want to keep the old data, since it is accomplished by simply moving the meta-widget.

Method 1 (Data migration). A user wants to move data completely from *Platform1* to *Platform2*. Being in the *Platform2*, the user will have to login to *Platform1*. Once done, all data can be retrieved from *Platform1*. Since both platforms are OpenSocial-compliant, *Platform2* understands the structure of the data and saves it in its own data storage. Afterwards, the user can forget about *Platform1* since all the data are moved and the meta-widget from now on accesses the data directly from *Platform2* (Fig. 3.6, row 1).

Method 2 (Data access). A user does not care about moving data to *Platform2* thus it is ok if the space widget running in *Platform2* still accesses the data from *Platform1*. The user will have to authenticate to *Platform1* and grant access to this data to *Platform2*. To avoid multiple user authentications, the special authentication token can be saved by using oAuth, for example. Next time the user comes to *Platform2*, the authentication to *Platform1* is done automatically and the meta-widget can work with its data. Thus, while the space widget is physically located in *Platform2*, its data is still accessed from *Platform1* (Fig. 3.6, row 2).

Method 3 (Migration/access hybrid) is a combination of methods 1 and 2, where a part of data can be moved to *Platform2*, while the other part is still accessed from *Platform1*. As an example, the data for the user profile can be located in *Platform1* and accessed from there, while photo albums uploaded by the user can be in *Platform2*. Moreover, the data can be distributed between three or more different platforms. This method is shown in Fig. 3.6, row 3.

Method 4 (Meta-widget embedding). A space runs in *Platform1* as a *meta-widget*, that has full access to data in *Platform1* and renders space UI and space widgets in this platform. This meta-widget can be extracted from *Platform1* in a fashion similar to an iframe and integrated within *Platform2*. Through this *meta-widget*, the space can be brought to *Platform2*.

Method 5 (Meta-widget extraction). In this scenario the space data are extracted from the *Platform1* but are not moved into *Platform2*, instead the data are hard-coded into the meta-widget itself. Thus, the meta-widget can run in different widget platforms and access the data, since all its code already contains the needed functionality and the data.

There is an important implementation detail to consider. Even though the APIs to retrieve data are standardized, the API prefix differs among platforms. Consider this API to retrieve a person object (which id equals 1) from *Platform1* - http://platform1.com/people/1/@self. Normally, in widget code the prefix can be omitted and API would look as /people/1/@self. This means simply: take your host platform as the prefix. However, if a widget is moved from *Platform1* to *Platform2*, but its data are still taken from *Platform1*, some adaptations are required. Namely,

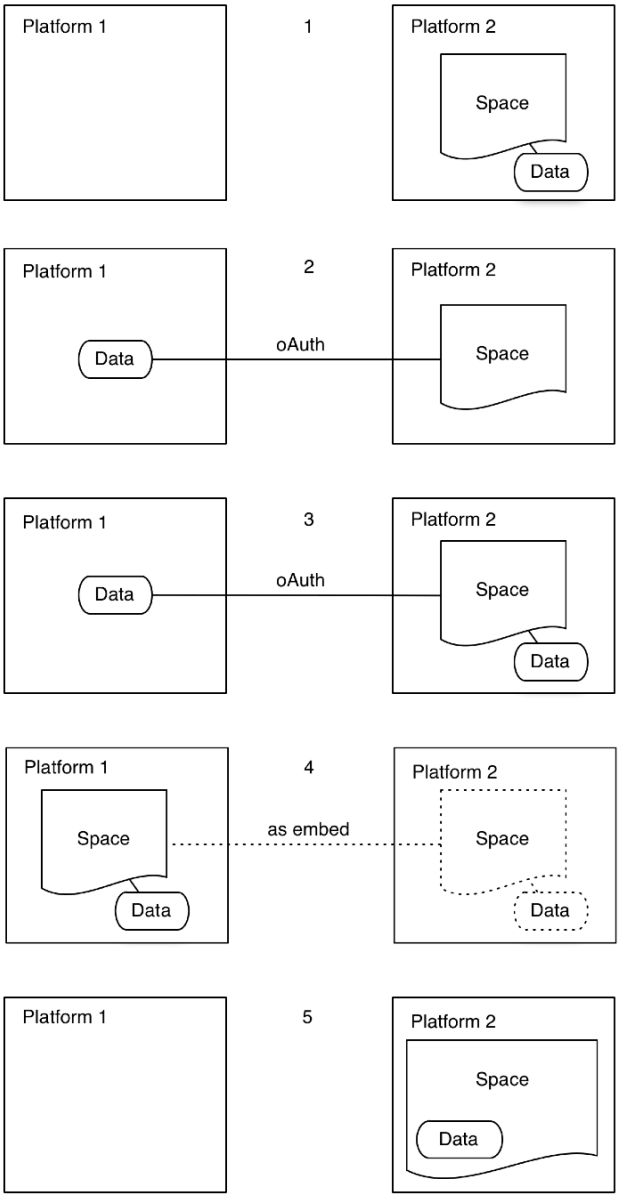


Figure 3.6: Classification of Migration Methods

the prefix should be changed in the widget for different API end-points. This way widget code does not have to be changed, but API requests will be redirected to appropriate platforms. Thus, in Method 3 requests for data to *Platform1* will be prefixed with http://platform1.com and requests for data to *Platform2* will be prefixed with http://platform2.com. This is the only change that has to be done, that can be managed within widget code.

The first three methods discussed in this section additionally apply to the user transition between platforms where the space UI is not implemented as a meta-widget. Let us imagine that *Platform2* uses OpenSocial APIs to retrieve its space data when the UI is being built. With OpenSocial support in both platforms, *Platform2* can access the APIs of *Platform1* and process the data. If, in addition, it implements the functionality (similar to meta-widgets) of handling authentication to *Platform1*, then the user can successfully work with data according to the first three methods presented in Fig. 3.6.

3.3.4 Portable Space Scenarios

OpenSocial in conjunction with the proposed migration methods allows to bring spaces from one platform to another. There are many scenarios of space usage, thus we suggest a classification of space use cases according to criteria defined in Table 3.4.

Criteria names	**Criteria values**			
platform independance	independent		dependent	
data flow direction	aggregator		distributor	
space dissemination	freeze	clone	share	bundle/template
authentication	no	security token	oAuth	secret URL
data access	local	external	embed	built-in

Table 3.4: Classification of Portable Space Scenarios

Platform independence criterion defines a degree of connection between a platform and a space. If a space can be separated from its platform and used as a standalone environment, it is independent. If a space can only "live" inside a Web platform, it is a dependent space. For example, in the Migration method 5 of the previous section the space exists independently of its initial container and, thus, can be seen as an independent space. If it is assigned a URL, it can be shared over the internet with this URL since it is self-contained. On the other hand, a space that extensively exploits user profile, widgets, widget data via OpenSocial APIs can not be easily separated from its platform.

Data flow direction defines a space as an *aggregator* or a *distributor*. In the first case, the space is very similar to a mashup because it aggregates data from different end-points. This way, a person's profile can be taken from *person.com* end-point, a widget list - from *widgets.com*, albums - from *albums.com*, and then this information is displayed within the space. In other words, a space serves as a destination point for information, where data flow enters and

accumulates within the space. *Distributor* spaces behave differently: the data flow is initiated within a space and data is propagated towards external platforms. The information is added into a space by the user and then these data are updated in different end-points on behalf of the user. As an example, consider a space where the user adds several external end-points (LinkedIn, Facebook, Google+). By doing this, the space has access to those external services via oAuth, and as soon as the user updates her postal address or adds photos into an album within the space, the change is propagated to all the end-points. The new postal address and photos will be added to external platforms.

Space dissemination criteria defines how a space is shared with others. When the user decides to migrate a space from one Web platform to another or to allow other people to work with the space, there are four main scenarios that can be accomplished.

- The user can *freeze* the current space, which means all space data is extracted from the platform and attached to a space in some serialized form. Afterwards, the space can not be changed anymore, but it can be easily moved around and rendered outside of the platform (a way to have publicly shared spaces).
- The user can *clone* a space which creates a copy of a space with all the associated data. Afterwards, this new space starts its own separate life. The new space can be changed according to the user's wishes and it can run in both the same and different platforms.
- The user can *share* a space with somebody or access it from another platform. In this case, there is only one instance of a space that is shared among several users or accessed by the same user in several platforms. The space changes are visible to everybody.
- The fourth scenario is similar to the first one, however it is different in the way space is used afterwards. A space that works in some platform is abstracted by a user into a *bundle* (or a *template*). These bundles contain only core space data excluding widget specific data: a list of widgets in the space without their data, the space description, the space name. Afterwards, the bundle can be shared with different people via Widgets repository or as a configuration file.

To start working with a space, users have to instantiate it in their preferred platform. We believe these four scenarios cover major use-cases people come across in the real life.

Authentication criterion classifies space use cases by the way authentication and authorization are accomplished. There are four possibilities: no authentication, security token, authentication (oAuth[9] or similar) and secret URL. A space running in a platform can directly access platform data, no special authentication is required in this case. Space widgets access data from the platform with the aid of a special security token, which neither requires authentication actions from the user. When a space is moved, but still accesses the associated

[9] http://oauth.net/

data from the previous platform, the user normally has to login to that platform. This can be managed by oAuth so that the user does not have to do it every time the space is being open. Another way is to provide a secret URL to access the space. This way the user can share the secret URL with his colleagues and they can access the space without any login required.

Data access is the criterion that classifies the ways a space accesses data when it is migrated from one platform to another. We consider here the four methods described in Section 3.3.3 and depicted in Fig. 3.6: local access (method 1), external access (method 2), embed (method 4) and built-in (method 5).

3.4 Portable Platform Interfaces

The notion of space together with its support in OpenSocial brings new scenarios, that were impossible to implement before. They are called *portable Platform Interfaces*. The Platform Interface is an application that implements the UI of a Web platform. The same portable Platform Interface can be used by different social media platforms to implement (fully or partially) their client-side business logic (Sire et al. [2010]).

With contextual spaces, it is possible to model concepts such as groups of people, events, discussion forums, courses, conferences, etc. As it was shown in Section 2.2.3, the Space concept allows us to represent internal data structures of Web platforms. Thus, a Platform Interface can retrieve the data structures existing in a platform via OpenSocial APIs and build the UI to work with data structures. The usage of standardized APIs to retrieve data allows us to decouple the implementation code of the user interface from the platform. As a consequence, a Platform Interface can be technically implemented as an OpenSocial meta-widget (a widget that can integrate and manage several other widgets) and it solely includes the programming code needed to manage the UI, while all the data is always located in the Web platform.

> **Portable Platform Interface** is a meta-widget that can retrieve the internal data structure of a Web platform as spaces via OpenSocial APIs and represents an implementation of the user interface that can be re-used in other Web platforms.

A Web widget is often a small application that extends the functionality of a Web platform. A Platform Interface is a more general concept representing the functionality of the whole Web platform, it is a Web application that governs the client-side functionality of a whole Web platform. Even though a widget and a Platform Interface are different concepts, each one can be implemented according to the OpenSocial standard (as a portable chunk of code), which ensures portability for Platform Interfaces. Thus, a portable Platform Interface can be seen as a template that can be used by a Web platform, to implement its client-side logic.

Consider the following example. Imagine a platform *Platform1* (that is a groups management Web platform) implemented as follows: the server side code is implemented according to

OpenSocial specification extended with spaces. It has a widget container able to render OpenSocial widgets in the browser (Fig. 3.7). Groups are represented as horizontal tabs. When a tab with a group is active, the list of people in this group is shown in the right area and group widgets are displayed inside the left area side-by-side. The whole user interface is technically implemented as a meta-widget. This meta-widget manages retrieving via OpenSocial APIs a list of groups (as spaces) for a logged in user, tabs switching, retrieving and displaying a list of people and widgets for every tab, etc. The client-side code includes solely the programming logic that builds the interaction interface with the data, however all the data is stored in the *Platform1*. This is the portable Platform Interface, because the widget code can be reused to provide a platform-to-user interface inside another Web platform. If the owner of *Platform1* believes this Platform Interface can be useful for other people, the Portable Interface can be shared in one of the available widget repositories.

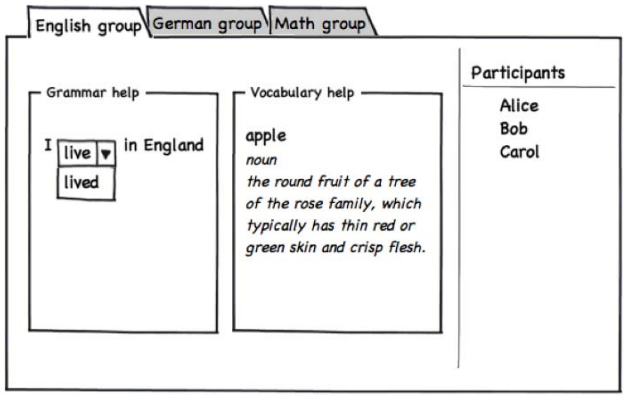


Figure 3.7: Platform Interface at platform *Platform1*

Other people can reuse this Platform Interface according to the following scenario: another developer plans to create a new Web platform *Platform2* that will be showing news to people (Fig. 3.8). The developer would like to organize the news platform in the following manner: tabs correspond to the different news topics (music, economics, sport). For every topic there are several RSS widgets showing related news and a list of people subscribed to this topic. The person goes to a widget repository and finds the URL and the screenshot for the Platform Interface described above. This Platform Interface perfectly suits the developer's requirements. The UI is exactly as expected, however, the data that will be displayed by the widgets and information in the tabs will be specific to the *Platform2*. Assuming the OpenSocial widgets support is already enabled in the *Platform2*, the developer first creates topics (as spaces) in the database of the *Platform2* and then assigns widgets and people to every topic. Once done, the found Platform Interface is added to the *Platform2*. This Platform Interface retrieves the data from the *Platform2* via OpenSocial APIs and builds the UI for these data (Fig. 3.8).

In this scenario the UI of a Platform Interface includes management of tabs, displaying people related to a tab from a social media platform, and rendering widgets existing within a tab. This functionality is rather complex (comparing to a simple widget) and represents a Platform

Interface. Since this Platform Interface is implemented as an OpenSocial widget, it is portable between different platforms. Even though the difference in the interface is transparent for an end-user, for Web developers this is a step forward in simplifying the development process. We should note that even without spaces widgets are reusable and portable, however without spaces it is difficult to achieve portable Platform Interfaces.

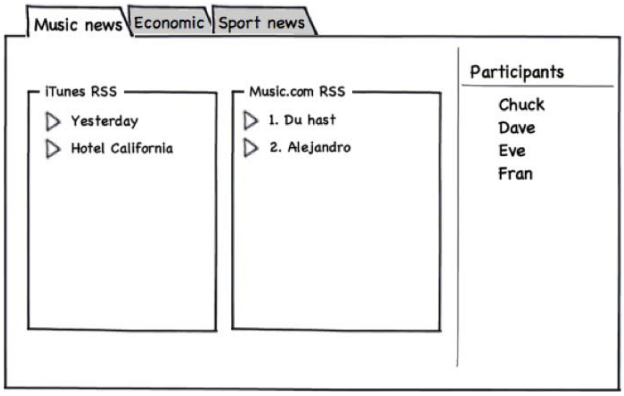


Figure 3.8: Platform Interface at platform *Platform2*

3.5 Technical Validation

The main goal of this chapter is to propose a solution to work with spaces and widgets in a cross-platform way. It is possible to divide all solutions into two main groups: space *sharing* and space *migration*. Scenarios in the *sharing* group are characterized by uniqueness of space data. In other words, when a space is accessed from different containers, no copies of its data are made. The data continues to be stored in one place and all Web platforms working with the space access the same data. For the *migration* group, the space data are moved from the initial location into a new one. Thus, a clone of a space is made. The implementations proposed fall into one of the two categories. These implementations use the migration methods from Section 3.3.3.

Graasp implements several mechanisms to share and exchange widget bundles and spaces in order to address the Communication and Configuration PLE dimensions. It supports the cross-platform interaction of people with spaces and enables sharing of created contexts with peers.

We describe three use cases validating the ideas presented in the current chapter: the interoperability of Graasp with Moodle platform, ubiquitous access to spaces created in Graasp within the Go-Lab project and the export and import of spaces between Graasp and ROLE Widget Store.

3.6 Validation 1: Graasp and Moodle Interoperability

We discuss how the main PLE components can be brought into LMS via OpenSocial widgets and the Space extension. We adopted OpenSocial specification for our investigation and our main goal was to bring the benefits of PLEs to students and teachers but, at the same time, decrease as much as possible the amount of new functionalities they will have to learn or deal with - a requirement requested by several teachers. With widgets, students and teachers become flexible in personalizing their environments and can bring much more functionality into LMSs than is available there by default. For this task, we implemented plugins for a popular LMS - Moodle.

Our main research goals were to find out if PLE flexibility improves interactions and learning within a course for both teachers and students and whether people find useful the space concept and the possibility to build their learning environments themselves. ROLE project is producing educational widgets based on OpenSocial and several teachers showed interest to use these widgets within their courses in Moodle, thus we decided to enable this functionality.

We introduce in more details the Moodle platform, describe its limitations and show how our solution tackles these limitations. We detail how the plugin was used in university courses and how students perceived it. Finally, we detail interoperability on the space level between Graasp and Moodle.

3.6.1 Moodle Plugin Description

Moodle is a popular LMS to manage courses that is the de-facto standard among many Educational Institutions. It is a plugin based PHP application that can be extended via additional modules. These modules have to be installed on a Moodle server by a system administrator. The Moodle view, as shown to students and teachers, consists of a main center area and a narrow right column with blocks (Fig. 3.9). The center area contains main course resources, such as a wiki page, a forum, a lesson, a quiz, etc. The right block contains some helper plugins that a teacher can add to every center page, e.g., a calendar, upcoming events, latest news, a recent activity, etc. These are to extend and enrich the functionality of the main center page.

The Moodle flexibility and adaptability is achieved via visual themes and server side plugins, thus an intervention of system administrators is required every time a change should be done at the Moodle plugin level. Teachers and students are not involved in the process of the customization. Teachers, for example, can not add or remove plugins on their own. On the other hand, widgets are client-side applications that can be added by the user to a system by skipping server side installation, which makes them easy to add.

The OpenSocial plugin for Moodle exists in two variations. The first one adds a new module to Moodle, which is similar to the standard module pages[10]. Once it is installed to Moodle, a

[10] https://github.com/vohtaski/shindig-moodle-mod

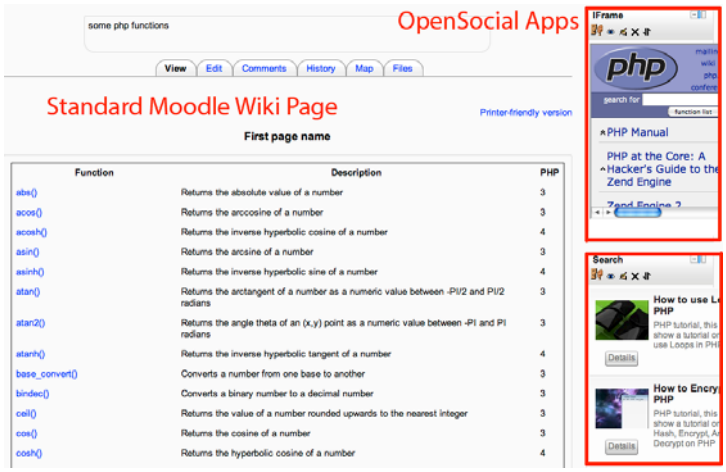


Figure 3.9: Widgets as blocks on the right

teacher can add a "Widget space" to the course, assign widgets to it, and choose whether a 1, 2 or 3 column view should be used to display widgets (Fig. 3.10). The resulting outcome (as displayed to students) is the page with widgets displayed as a grid, where students can work with several widgets simultaneously (Fig. 3.11).

The second part of the plugin adds a new block to Moodle[11]. A teacher can add widgets to the right column for already existing in Moodle wiki pages, lessons, forums, etc. (Fig. 3.9)

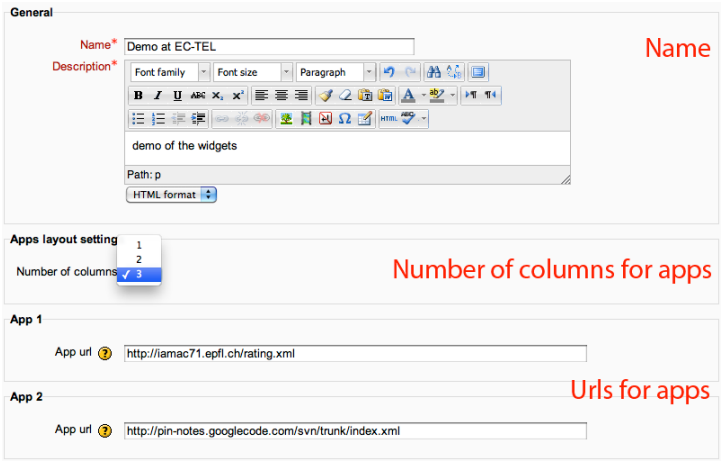


Figure 3.10: A teacher creates a space with widgets for a course

One of the main benefits of this plugin is the large pool of available widgets that can be used by teachers. Once the OpenSocial plugin is installed in Moodle, a teacher can achieve the needed functionality without bothering system administrators with server-side plugins installation. The plugin enhances the flexibility in choosing the resources and tools according to the course

[11] https://github.com/vohtaski/shindig-moodle-block

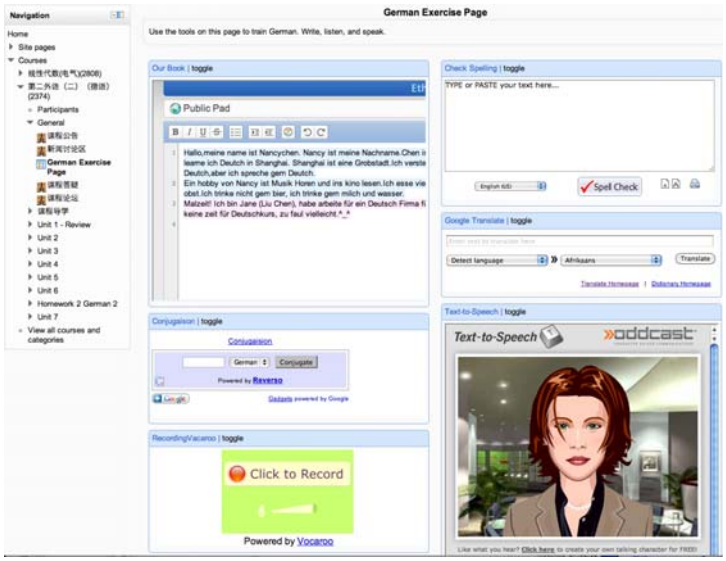


Figure 3.11: Widgets as displayed within Moodle

specifics: teachers can easily add and remove widgets needed for a course, develop their own ones, etc. Widgets can be found in the existing widget repositories (iGoogle directory, ROLE Widget Store, etc.). Teachers can re-use existing educational resources and learning objects from external Web sites. Depending on the desired integration level, teachers can either use an iFrame widget that simply integrates a Web site URL or develop their own widgets that provide a deeper integration.

From the implementation perspective, the plugin consists of two parts. The first part is a Widget engine (Apache Shindig) that renders OpenSocial widgets on a Moodle page. The second part is a PHP module that is responsible for the configuration of the page containing the widgets, adding and removing them to/from the page and gluing Moodle with Shindig engine. OpenSocial API provides the standardized way to retrieve and exchange information between the Moodle installation and the other social networks, which improves data portability and interoperability. Widgets can query Moodle for data via Shindig engine: they can retrieve the currently logged in user, the current course, its participants as well as save and get arbitrary data. The privacy and security are managed via Shindig engine and it is under the full control of the administrators. However, a widget installed within a course runs on behalf of a teacher who added it and can do the same actions that teachers can normally do in their courses. Thus, it is a teacher who is responsible for checking the trustfulness of a widget, before adding it into Moodle. The ability to retrieve a course information and its participants is achieved via OpenSocial space extension that allows widgets to adapt to the specific context of the course. For example, a wiki widget (Fig. 3.11, top left) can save data for a course and restrict access to only people engaged in this course. The same wiki widget will behave differently being added to another course: it will have a different wiki history and a different list of participants.

3.6.2 Usage at the Online College of Shanghai Jiao Tong University

The Moodle plugin has been used in the courses at the distance university of Shanghai Jiao Tong University (SJTU School of Continuing Education, SOCE). SOCE students are adult learners who study for an associate or bachelor degree (Ullrich et al. [2010]). The college implements blended learning, i.e., students can come to classrooms in person to attend lectures or watch the lectures live through the Web. All lectures are recorded and available for subsequent non-live view. Teaching and learning follows a traditional pattern which is very teacher-centric, with most students watching the lectures rather passively. Within the ROLE project, we investigated how to use existing technologies and tools to provide a larger amount of opportunities for interaction and creation (Bogdanov et al. [2012c]). For instance, tools like Voice Recorders and Text-to-Speech allow foreign language students to practice their pronunciation by recording themselves and comparing their speech to the "original" one. Other tools, such as collaborative text editors enable students to work on joint texts in an easier manner than by using forums. A large percentage of the widgets used in SOCE are existing Web pages that train very specific domain knowledge, such as the usage of German articles and French verbs, or visualize data structures such as linked lists.

The Moodle plugin has been used at SOCE since August 2011 to add ROLE technology via widgets to a number of courses in the domain of foreign languages as well as computer science. The courses are "Business English", "English Newspaper Reading", "Data Structures", "German" and "French". Figure 3.11 contains screenshots of the Moodle page as used in the course. In the lectures "Business English" and "English Newspaper Reading" the teacher used widgets in several ways. Firstly, and in a similar manner to the other lectures, by converting existing resources into widgets suitable for use in training students for various aspects of business English, such as writing CVs, business emails, etc. Secondly, in order to make the course more realistic, the teachers used a role play scenario in which students set up fictitious companies and products. Students were then instructed to create a Web page for their company and a slide set presenting their products. The resources were uploaded in Moodle and students used a rating widget to assess resources authored by their peers.

In the German and the Computer Science courses, the teachers used the Moodle plugin in order to provide additional exercises during the semester and also to offer training opportunities. But ostensibly the PLE aggregator was utilised by the students mostly for exam preparation. During the courses the Moodle plugin was referred to as a PLE and we will use this naming further on. The teachers converted existing Web resources consisting of exercises training various aspects of German grammar and visualizations of data structure algorithms to aid the students understand and enhance their learning opportunities. Several widgets and embedded tools were extended with the functionality helpful for the overall learning process. For instance, the users' interaction with the tools was captured and used in a visualization which allowed the teacher and the students to see how often they interacted with the resources and to compare their activities to those of their peers.

3.6.3 Plugin Evaluation

The current section describes the results of a questionnaire that was conducted with students who were using the described plugin in the courses at SJTU university. Our main goals were to find out whether students see a value in using PLE components brought via the OpenSocial widgets and whether they look for more flexibility, i.e., they can manage widgets on their own.

20 students responded to the questionnaire (a half from the French course and another half from the German course). In general, students perceived the PLE to be useful for their learning tasks (Fig. 3.12). They found it to be helpful to learn in an independent manner, to accomplish work more effectively and they would like to use it in the future.

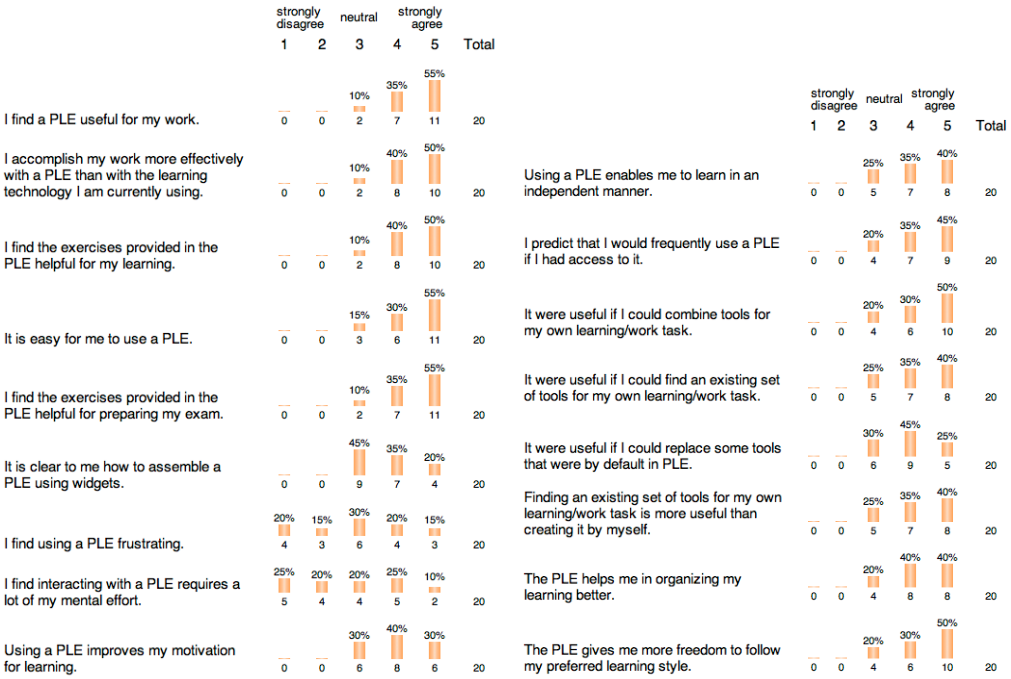


Figure 3.12: Questionnaire results

The second issue we evaluated was whether students were looking for more personalization and flexibility. The questionnaire showed that students feel comfortable in organizing their own environment for learning by customizing the bundles of widgets offered by teachers, assembling their own sets of widgets for their learning tasks and to search for existing sets of tools (Fig. 3.12). Figure 3.13 shows how students rated the tools that were offered by teachers in the Moodle plugin. Even though the majority of widgets were useful for students, a number of them (Listen to your pronunciation, Record yourself, Spell check, Activity visualization) were not highly appreciated by several students. This indirectly confirms the fact that students were looking for more flexibility: they would prefer not to have some widgets at all or probably replace them with other alternatives. Thus, the functionality to remove/hide some widgets on the page or to replace them would be appreciated by some students.

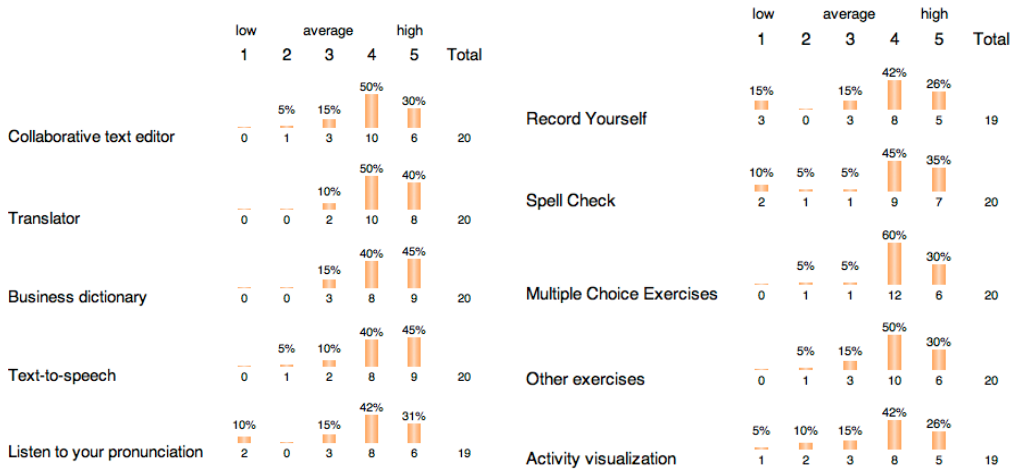


Figure 3.13: Tools as rated by students on a scale from 1 (not useful) to 5 (very useful)

3.6.4 Space Sharing via OpenSocial APIs

The following implementation shows *Data migration* and *Data access* methods from Section 3.3.3. As a source *Platform1* we use *Graasp* platform. The destination *Platform2* represents our local installation of LMS *Moodle*. The *Moodle* installation is extended with a plugin that allows to add and render OpenSocial widgets and spaces. Both platforms use Apache Shindig as a backend to manage widgets. The Apache Shindig version of *Graasp* platform is extended with *Space* APIs. Both platforms implement OpenSocial *Person* API and the user has an account in both of them. In this scenario the user accesses her personal space existing in *Graasp* within *Moodle*.

The space in *Platform1* is built as a meta-widget to display the information about the user from the platform (name and description) as well as a list of her spaces (a thumbnail picture and a name for each). Since *Graasp* implements both the standard OpenSocial *Person* API and the API for the *Space* extension, the *Graasp* space receives the information from its platform and displays it, which can be seen on the left in Fig. 3.14. Next, the user decides to move the space meta-widget from *Graasp* platform to *Moodle*. All that has to be done is to take a widget URL from *Graasp* and to create a new widget for this URL within *Moodle* (Fig. 3.14). The widget runs correctly since both platforms are OpenSocial-compliant.

Being added into *Moodle*, the widget discovers that *Person* API is supported. Thus, the widget automatically displays the user name and the description taken from the *Moodle* platform via OpenSocial *Person* API (the left part of Fig. 3.15). The widget also finds out that the *Space* API is not supported in *Moodle*. Thus, additional actions are required from the user to enable access to the *Graasp* Space API. In the current scenario, the user is prompted by the widget to approve access to Graasp spaces (the left part of Fig. 3.15). In more complex scenarios, the widget could also ask the user from which platform the spaces should be taken from. Thus,

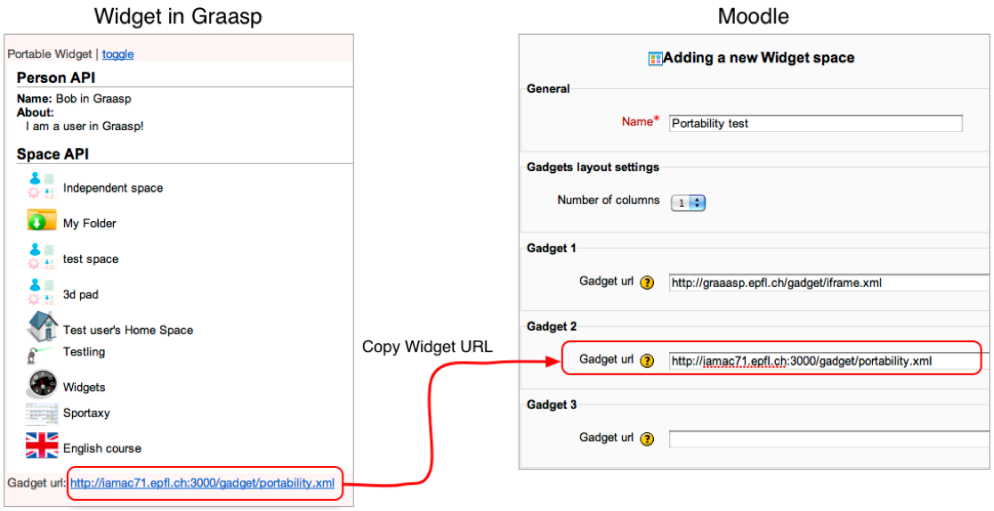


Figure 3.14: Space as a meta-widget in Graasp and its addition to Moodle

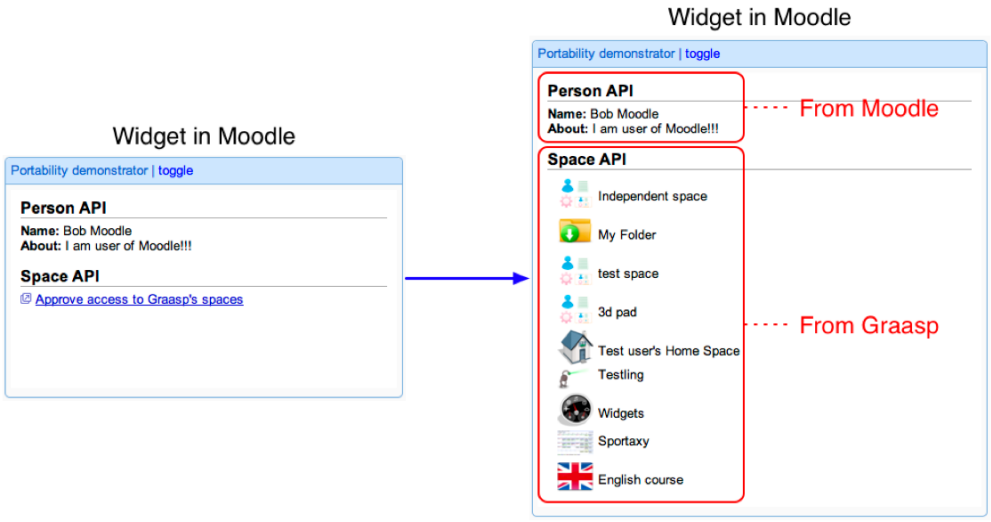


Figure 3.15: Space running in Moodle

the user has to provide credentials, so that the widget can login on her behalf into *Graasp* and retrieve spaces information. Once the widget is successfully authenticated, in addition to user's profile from *Moodle*, it displays user's spaces from *Graasp* (the right part of Fig. 3.15).

The next time the user opens the widget she will have to login into *Graasp* again. To solve this usability problem, we use oAuth protocol, so that *Graasp* behaves as an oAuth provider. On the other hand, widgets in *Moodle* behave as oAuth consumers (support for this is provided by Apache Shindig). Once the user provides her credentials, the special oAuth token is saved by the Apache Shindig installation at *Moodle* and the widget will be able to automatically authenticate into *Graasp* during the next visit. In case there is a single sign-on (SSO) mechanism enabled between two platforms, the user can even skip the first login into *Graasp* required by oAuth.

In the described implementation scenario, the user is able to move the space meta-widget from *Graasp* to *Moodle* and still access the associated data (a list of spaces) used in *Graasp*. This scenario corresponds to the *Migration/access hybrid* method in Section 3.3.3, where some data is taken from the new platform and some is still accessed from the previous one. Moreover, the widget could retrieve *Person* data from *Graasp* in the same way it is done for spaces. This way, the *Person* object from *Graasp* can update the fields of the *Person* object from *Moodle* in case the user wants to merge or migrate the profile information.

3.6.5 Scenarios for Space Migration

In the sharing scenario, data does not leave its initial Web platform. However, in the migration scenario, data is physically moved from one platform to another. The scenario is the following. A teacher decides to migrate his Moodle course space into Graasp. The course contains several widgets and resources. The teacher wants to have a space in Graasp with the same course name, description, widgets and resources as in Moodle. There are several ways to achieve it with OpenSocial.

The first approach is based on using OpenSocial APIs (Method 1 - Data migration, Section 3.3.3). However, in order to simplify the development of migration scenarios, an additional tool is needed. This tool enabling data migration should allow the user to

- specify from which Web platform data is to be taken from,
- login to the external platform,
- specify which APIs are to be used for data extraction (Person, Space, etc.),
- choose which fields are to be taken during the migration process.

This tool can be implemented as a special library for Web platforms or as a *Migration widget*. In addition, it requires the management of multiple oAuth end-points within an OpenSocial

widget (which is not possible at the moment in Apache Shindig). The user should be able to dynamically change the oAuth end-points or the widget specification should support the definition of multiple oAuth end-points in the Modules section of a widget. Also, a possibility to change a domain address for OpenSocial API calls would be a helpful step towards enabling the migration. This can be accomplished as a feature for Shindig that changes the domain names of OpenSocial osapi requests. For example, it could work as follows.

```
osapi.setDomain("http://graasp.epfl.ch")
osapi.setDomain("http://moodle.epfl.ch")
```

The second approach helps to achieve migration via JSON serialization, where a file with serialized data is moved from one place to another instead of accessing OpenSocial APIs. Since OpenSocial data is served in a JSON format, it can be serialized as a JSON file. This file can be later passed to another platform, that can extract the needed data. Since OpenSocial APIs already understand OpenSocial JSON format, it would be beneficial to reuse them for this task. A new tool should be able to parse JSON file and find the blocks that can be directly passed to the corresponding OpenSocial API. Afterwards, these APIs can automatically save data to the new platform.

3.7 Validation 2: Ubiquitous Access to Go-Lab Spaces

Graasp is used within the European Go-Lab project as a platform to manage remote and virtual labs[12]. The goal of the Go-Lab project is to support pupils and teachers in schools. In the project scenarios, pupils are expected to conduct remote laboratory experiments by following the inquiry-based learning paradigm. For every remote laboratory a teacher creates a space in Graasp. Widgets are assembled into the space to provide the laboratory functionality and help pupils to accomplish the assignments. Resources required for the lab are added as well. Once the construction of the lab as a Graasp space is completed, the teacher can share it with her pupils. One of the main project requirements is to avoid for legal reasons (EU teen privacy) the creation of accounts for pupils and having them sign in to the labs. Thus, the space *portability* and space *sharing* become instrumental for these scenarios, as we will discuss next.

3.7.1 Space Sharing via Embedded Meta-Widget

To support teachers in sharing remote labs with their pupils, we enabled in Graasp the sharing approach where a created space can be brought to another container in a way similar to an embed HTML tag or open in a browser as a URL (Section 3.3.3, method 4 - Meta-widget embedding). From the user perspective it looks the following way: the teacher signs in to Graasp and opens a space with widgets. Graasp has a special view where all widget instances are shown on the page at the same time and the user can change their order and width

[12] http://www.go-lab-project.eu/prototypes

(Fig. 3.16). Once the spatial arrangement is completed, there is a possibility to export the space (Fig. 3.17). After the space is exported, the teacher is provided with a *Private URL* or an *Embed code* that could be used in a browser separately from Graasp. The resulting appearance and functionality of this exported space is similar to the space in Graasp, however, additional space management features of Graasp are not displayed (Fig. 3.18). The space is not physically migrated from Graasp but is accessed from another platform, thus if the teacher changes the order or the size of the widgets in Graasp space, it affects the exported space and vice-versa.

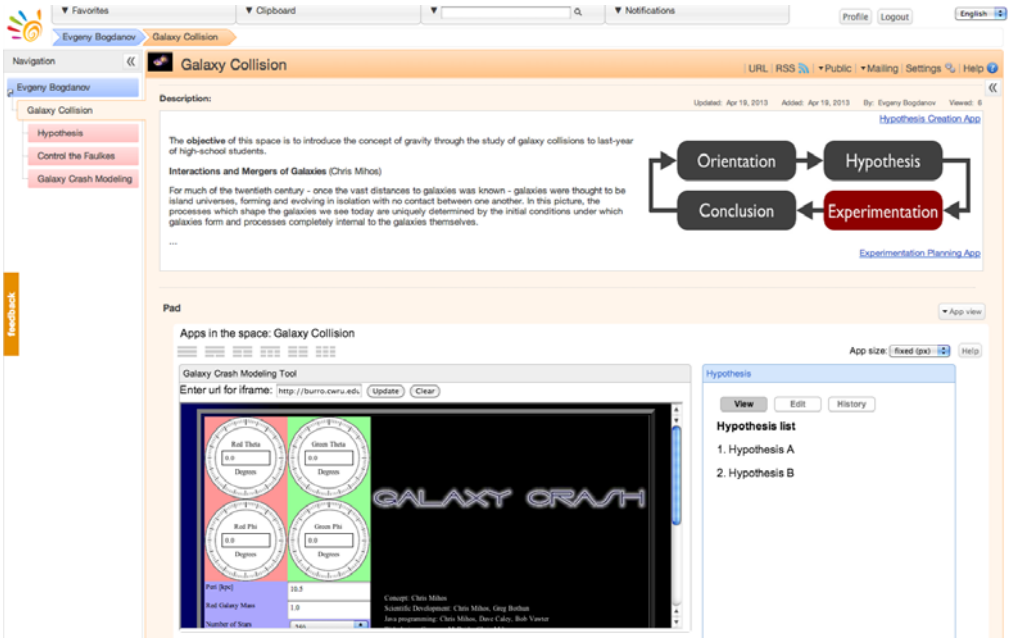


Figure 3.16: Space with widgets in Graasp

In the Go-Lab scenario the teacher gives the extracted private URL for the space to the pupils. They can open it directly in the browser and can work with the lab without additional logins or account creation (Fig. 3.18). The data they produce are saved in the corresponding Graasp space. If we compare the default Graasp view as it is seen by teachers in Fig. 3.16 and the extracted private URL view that is given to pupils (Fig. 3.18), we can see that the interface for pupils is lighter and does not have extra features that are not needed to conduct a lab experiment.

The URL the teacher receives has a special hash value that maps to the user identity. Thus, the user should not publicly share this URL, because everyone who can access the URL, has the same access rights in the platform as the user herself. The exported space represents an OpenSocial meta-widget. Apache Shindig serving as a widget container in Graasp uses the security token that contains the encrypted information about logged in user id, space id and an expiration timeout after which the token will be invalidated. When the export takes place, the mapping between hash value and user id and space id is dynamically created in Graasp database. Whenever the user accesses the private URL, a new security token is generated

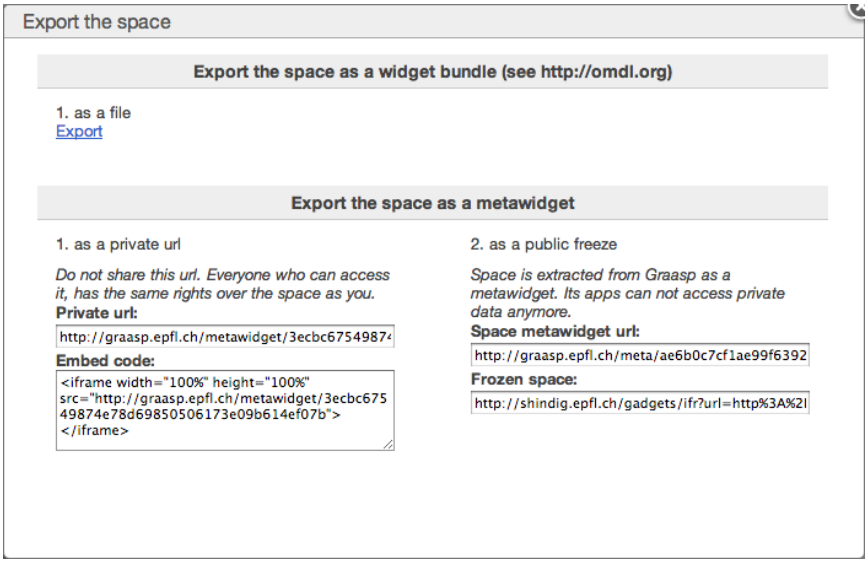


Figure 3.17: Space export interface in Graasp

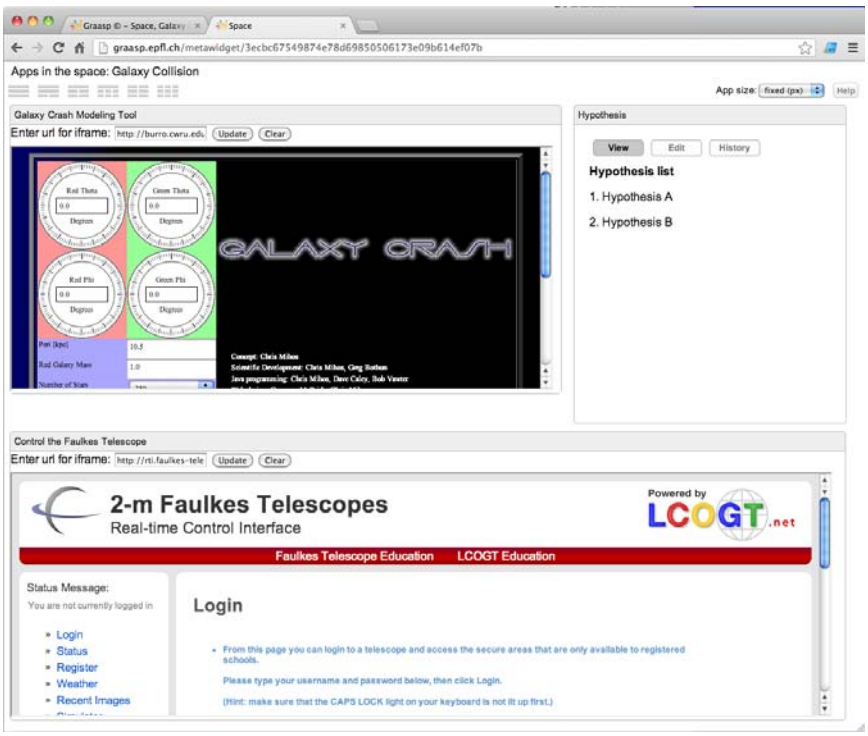


Figure 3.18: Space exported as a private URL and opened in the browser

and the space meta-widget is rendered on the page. In the scenario described, the *private* URL serves as an access point to a space for the user, so that she does not have to login every time the space is accessed. For the Go-Lab scenarios it means that from the Graasp point of view, the actions accomplished by different pupils are considered as done by one person (the teacher). This specificity is addressed in two other approaches to manage user authentication when a space is exported from Graasp, that are alternative to the *private URL* mechanism. This is the future research goals that are briefly summarized in the next paragraph.

The first approach is to provide a login interface within the extracted meta-widget, where the user can authenticate into Graasp and the widget will identify the user working with it. Though it is useful in many scenarios, it is a limitation for the Go-Lab project where pupils should not create accounts. The second approach is the *shared URL*, where the login information is not saved on behalf of a particular user creating a URL but is shared among all people accessing the space. In the *private URL* scenario, all actions done by pupils are seen by the platform as the actions of the teacher. In the *shared URL* scenario, all actions are seen as done by an anonymous user and a more granular scheme can be used to differentiate between the pupils. This is the typical Go-Lab scenario where the teacher shares a lab space with the pupils. The teacher extracts the space as a *shared URL* and gives the URL to the pupils. They can work with the space meta-widget without login. However, the widget should save data on behalf of each pupil and differentiate them. For example, pupils might use their names, email addresses or pictures to uniquely identify themselves within this extracted meta-widget. Afterwards, the user identifier can be saved in cookies to be reused for the subsequent sessions. Further investigations are ongoing to find the best approaches to manage anonymous users and the adaptations needed on both Graasp and Apache Shindig levels.

3.7.2 Migration with Built-In Data

We saw previously the scenario where the teacher shared the created space with students. In case the teacher uses widgets that do not use any OpenSocial APIs, the space can be extracted from the container as a space freeze (Section 3.3.3, method 5 - Meta-widget extraction). The next section shows its implementation in Graasp.

When viewing a particular space, the user can extract it from Graasp as a public freeze (the right part in Fig. 3.17). This means that a new meta-widget is automatically generated and all the space data (its name, its list of widgets, their order, their width, etc.) are built into the widget code. This meta-widget can be run in an interoperable way in other Web platforms with OpenSocial since all the data it needs is already accessible within the widget itself and the widget does not require any interaction with its platform. The user is provided with a meta-widget URL that can be used to render a space within other OpenSocial platforms. For example, a new widget can be created in Moodle that uses the frozen URL extracted from Graasp. It is also possible to simply render the space as a standalone URL using the Apache Shindig installation in Graasp.

There are two main limitations of this approach. The first limitation is that the data can not be updated anymore by the meta-widget since it is built-in. The second limitation is that widgets that need to save data into their container will loose access to the platform once the freeze is done. However, the proposed approach can be generalized to the case of widgets using OpenSocial APIs. For example, either a data freeze should be done not only on the space level but also on the widget level or there should be found a way to redirect OpenSocial API calls within a widget to a local frozen json database instead of the widget container which might require the extension to OpenSocial or an additional implementation within Apache Shindig.

3.8 Validation 3: Graasp Interoperability with Widget Store

A space represents a personal environment built by a user for a particular goal. However, once a space is created and exploited by one user, it might as well be useful for the other people. Thus, a space helps to share and exchange experiences and best practices among people. If we look at the *Learning French* scenario from Section 2.2.1, we can see that the similar scenario is useful for many people. Thus, if *Alice* already has a *Learning French* space in Graasp, it would be useful to be able to extract it from Graasp in an interoperable fashion. This way other people can create their personal learning environments (spaces in Graasp or other PLE aggregators) based on the learning French based on *Alice*'s experiences. The OMDL specification is a convenient way to exchange widget bundles between people. In order to enable sharing of widget bundles, the Web platform should provide support for extracting a widget bundle from an existing space and to create a new space based on an existing widget bundle.

3.8.1 Widget Bundle Export

Every space in Graasp can be extracted and exported as an OMDL file (see dialog in Fig. 3.17, top). When an extraction is accomplished the user is provided with an OMDL file, that contains the space name, thumbnail URL, its description and lists all space widgets. The user has an option to download the OMDL bundle as an XML file or get a URL where the file can be retrieved. The OMDL bundle can be imported into another Web platform (for example, Apache Rave platform[13]) or shared on the ROLE Widget store (widget repository) for further reuse by different people in different Web platforms supporting widgets.

3.8.2 Widget Bundle Import

Additionally to the export of OMDL bundles, the Graasp user can import OMDL files to create new spaces based on bundles or to add widget bundle content to the existing spaces. The OMDL file can be either uploaded manually (as an XML file or as a URL) or the user can browse through the bundles available in the Widget store.

[13] http://rave.apache.org/

Whenever the user decides to create a new space, he can search for a bundle in the Widget store. Figure 3.19 depicts the interface shown to users. The user can browse the existing bundles or search by name. By clicking on a bundle thumbnail (on the right), the user can view the detailed description of the bundle and see a bigger screenshot (on the left). Once the needed bundle is selected, a new corresponding space is created in Graasp. By default, the space receives the name, the description and thumbnail URL of the bundle. The widgets listed in the bundle are created and assigned to the space. However, the user can select which widgets from the bundle he wants to add and which he does not (Fig. 3.20). The name and the description of the new space can be adapted as well.

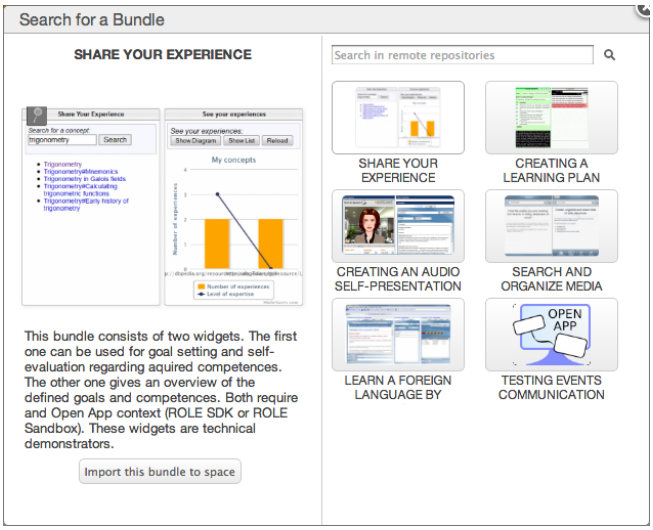


Figure 3.19: The Graasp bundle search interface

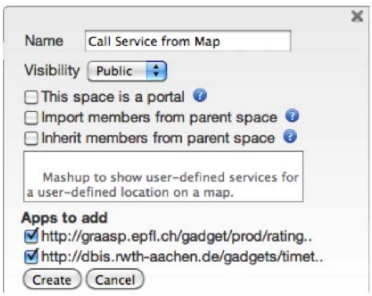


Figure 3.20: Create space from bundle popup in Graasp

If the user already has a space, the same steps can be used to add an OMDL widget bundle to this space. In this case, instead of new space creation, widgets will be added from the bundle to the existing space and user can change the space to the name and description taken from the bundle.

From an implementation perspective, Graasp has a service that daily queries the SPARQL end-point of the Widget Store to build a local copy of widgets and bundles. Afterwards, when

the user searches for widgets, Graasp queries this local service to get a list of widget bundles according to the parameters the user specifies. The Graasp service can be scaled to support other widget and widget bundle repositories such as iGoogle, etc.

3.9 Discussion

3.9.1 Space Configurations vs OpenSocial Spaces

We detailed two approaches to enable sharing and migration of spaces between Web platforms. Why do we need two solutions and what is the difference between them? It turns out, they target different problems.

The Collaborative Portable Space Configuration approach can be seen as an extension to the widget bundles standard. Widget bundles represent sets of widgets combined together for a specific objective and the OMDL specification serves to make these bundles portable. Later, a bundle can be instantiated into a space within a widget container, which means it will be used by several people and the widgets can save some widget preferences. In addition to widget bundles, the Collaborative Portable Space Configuration approach incorporates the social part (a list of users inside a space with corresponding access rights) and some data (widget preferences) as well as specifies how updates in the configuration have to be propagated to the users. The main idea is to separate a configuration for each space from its Web platform. We mentioned the two main scenarios where it could be used. First, several Web platforms can implement shared access to the same space configuration file and every container synchronizes space changes via this configuration file. Thus, this configuration file is a shared synchronization point allowing users to collaborate over the same space being in different Web platforms. Second, since the configuration file is decoupled from its Web platform, it can be cloned and used as a configuration file for a new space in another Web platform. Once the configuration file is cloned, there are two spaces that live separately. They have identical configuration files but later the configurations will start to deviate from each other.

On the other hand, the OpenSocial spaces approach has another target: enabling APIs to access space information and content from a Web platform plus standardizing a model (or data structure) of a space and its content. This enables one Web platform to access spaces from other platforms, supports space migration from one container to another and sharing of the whole user environment as a meta-widget between different Web platforms.

The space configurations can be seen as a centralized approach with the space configuration at the center, where user environments are built within a Web platform based on this central configuration. In contrast, OpenSocial space has a distributed nature, where different parts of a space can be accessed separately via their APIs and the user's environment can be a mashup of data coming from different Web platforms.

3.9.2 Portable Space Benefits

We presented a novel way of using OpenSocial to enable migration of spaces with their associated data from one Web platform to another. Moreover, we showed how a space created in one container can be accessed from another container both on data and user interface levels. The OpenSocial standard is in active development and the standardization process by itself introduces some difficulties that have to be tackled. Namely, different containers can implement different versions of the standard or some containers can extend the standard specification which makes their widgets container-dependent.

Next, we summarize the benefits that OpenSocial spaces bring us. Migration approaches introduced in the previous sections permit a user to carry data and tools along when moving from one container to another. In addition, we illustrate how our approach helps to solve the problems rising from the fact that containers implement different versions of the OpenSocial standard or extend it with container-specific APIs.

Portable Tools and Data

Data interoperability and space portability are indeed needed, since users are not bounded to a particular widget container but can use spaces in different Web platforms. As example, students attend a course at some university and exploit OpenSocial widgets. Later, after graduation, they could still be interested in the data they collected during the study. They will not use university facilities anymore, but depending on the university policy the students could either port all the data with them via OpenSocial or continue having access to it. They might want to continue working with the same OpenSocial widgets used in the university or start to use a new one. With OpenSocial, the new widgets should automatically understand the old data coming from the university. As it was presented before, this scenario is not limited to OpenSocial widgets but applies to any OpenSocial-compliant container, where, for example, data can be directly moved from one university to another university (or social media platform).

Support for New APIs

New versions of OpenSocial specification are regularly created, with new APIs being introduced and some other APIs becoming deprecated. Web platforms might implement different versions of the OpenSocial specification. Thus, some APIs might be available in some platforms and unavailable in the other ones. A widget developer might target the newer version of OpenSocial specification, however it is still possible to bring the new functionality to the platforms only supporting older versions of the specification. Consider example where the *Platform1* (Fig. 3.21) does not support new OpenSocial Document API, while the *Platform2* supports it. Normally, a widget developed for *Platform2* will not able to work with documents, when being run in *Platform1* since this platform does not support them. However, by using

the techniques described in Section 3.3.3, the widget can redirect all Document API requests to *Platform2* that supports them. In the case the user has an account in *Platform2* which will require authentication or *Platform2* provides a service of saving documents, the widget will work properly in the *Platform1*, even though this platform does not support the new API. In this case, the *Platform2* becomes an external hosting for documents with respect to *Platform1*.

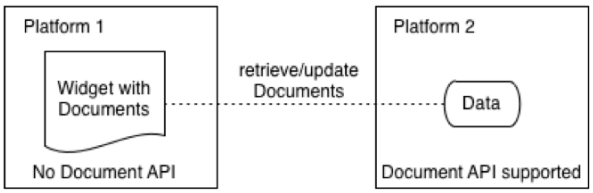


Figure 3.21: Example: New API Support

Support for OpenSocial Extensions

Domain-specific extensions to OpenSocial can be handled in a manner similar to the support for new APIs. Lately, some OpenSocial containers started to define their own APIs by extending the OpenSocial specification - SciVerse, SocialSite, etc. These APIs often reflect the domain specifics of the Web platform. Therefore, SciVerse extended OpenSocial APIs with requests to retrieve published papers information. These additional APIs are not yet part of the standard OpenSocial specification, thus they are not supported in other OpenSocial-compliant containers. This makes widgets that use these extensions container-dependent. However, the extended functionality could be brought to other OpenSocial containers via the techniques described in Section 3.3.3. This way widgets will work properly even in the containers that do not support the extended APIs. An OpenSocial widget that takes advantage of the extensions can retrieve standard OpenSocial information from its hosting container, and call the extended APIs of another container for the information about publications or spaces.

Simplicity

Another important benefit of using OpenSocial to achieve portability and interoperability is its simplicity. OpenSocial provides REST APIs to describe resources. The alternative SOAP and WSDL Web services standards are often overcomplicated and have low adoption outside of enterprises (Vinoski [2007], Maraikar and Lazovik [2008]), while REST is a scalable and easy to use approach that is widely used nowadays. The data returned by OpenSocial requests is by default in the simple and extensively used JSON format.

Limitations

The portability approaches for space sharing and migration with OpenSocial have the following limitations. First, a Web platform should comply with OpenSocial specification by providing

an implementation of the APIs. Normally, any Web platform can do it via open-source Apache Shindig project. Our proposals rely heavily on the OpenSocial space extension. To make the proposed solutions work in different Web platforms, the Space extension should be supported. At the moment, the space concept is not a part of OpenSocial specification, thus a Web platform has to integrate the space extension patches to their Apache Shindig in order to use the proposed solutions. Starting from the OpenSocial version 3.0, the proposed solutions will be natively supported in Apache Shindig.

3.10 Conclusion

Thanks to the contributions of this chapter, it is now possible to design a space in one platform and exploit it in another platform. The *Collaborative Portable Space configuration* concept allows to decouple a space from its Web platform as a *space configuration* file and be used in a collaborative manner by several people located in different Web platforms. The OpenSocial Space extension allows to move a space and associated data from one platform to another (portability scenario) or to access a space created in one platform from another platform (sharing scenario). Various techniques enabling portability and sharing are classified as *Migration methods*. In addition, we provided five criteria to classify the scenarios of portable space usage. The concept of *Portable Platform Interfaces* helps to decouple the code implementing User Interface functionality from its Web platform. This decoupled implementation of the User Interface can be re-used in another Web platform.

Chapter 2 established the Space concept and this chapter revealed the different space portability and sharing scenarios. The presented contributions address several PLE requirements. One of the main requirements to a PLE aggregator is the plasticity which means users ability to freely shape their environments according to their needs. The next chapter shows how the PLE plasticity can be achieved.

4 Personal & Contextual Plasticity

The ultimate goal of PLEs is to fulfil users' needs rather than force them to be satisfied with what a Web platform is offering. In a PLE, users should be able to shape their learning contexts as they want, by adding and changing learning tools, by adding and removing learning resources, by adapting the graphical and functional parts of Web platforms, and by sharing learning resources with others.

This chapter discusses how learners can control the User Interface (UI) and the content of their PLEs, which is the way to personalize PLEs according to their needs. We coined the term **plasticity** to describe these configuration mechanisms (**contribution 6**). We define **plasticity** as a measure of a Web platform ability to let users customize the UI and aggregate (or share) Cloud content.

A Functional skin (**contribution 7**) is a crucial concept towards achieving platform plasticity. It is an extension mechanism introduced to customize the graphical and functional parts of the user interface within a platform and can be seen as a client-side plugin. This concept is detailed in Section 4.1.2.

The **contribution 8** concerns aggregating and sharing resources between a Web platform and the Cloud. First, to simplify the process of aggregating resources from the Cloud into a Web platform, we propose a plugin-based aggregation architecture allowing users to bring with one click resources from the Cloud into their platforms. Second, we propose the mechanisms for users to share and migrate their space from their platforms into other locations in the Cloud.

The combination of several features (space concept, functional skin, aggregation architecture and space sharing techniques) increases the plasticity of a Web platform, as demonstrated with Graasp in Section 4.2. Finally, the six PLE dimensions described in Chapter 1 can be seen as a quantitative measure of platform plasticity. Section 4.4 provides a comparison of 5 platforms with respect to the PLE dimensions.

4.1 Space Personalization

4.1.1 Plasticity Definition

The Plasticity concept shows how easily users can customize a Web platform. The users aggregate external resources from the cloud into their Web platforms and share resources with others in the cloud. On the other hand, the users need to change the interfacing with the information. If they are working with a space in their Web platform, they need to be able to customize the visual or graphical interfaces. Besides, the users need to change the provided functionality features. For example, they could be interested to display and work with only some parts of the space content or to apply actions to a space that are not supported by the platform.

> **Plasticity** is a measure of a Web platform ability to let users personalize the user interface and aggregate Cloud content (Fig. 4.1).

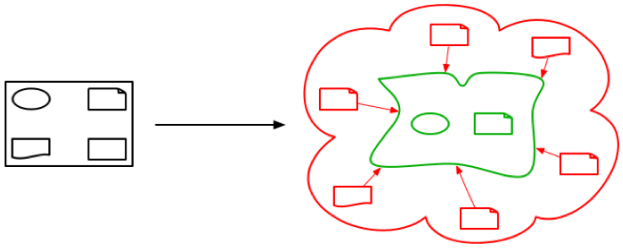


Figure 4.1: Plasticity concept

4.1.2 Functional Skin

Functional skins allow users to easily personalize their interaction with spaces. They are implemented as meta-widgets and can be seen as client-side interaction plugins for spaces. Users can easily add functional skins to every space according to their needs and switch between different skins.

> A **Functional Skin** is a client-side interaction plugin for a space implemented as a meta-widget, that can retrieve space meta-data and space content via OpenSocial APIs and provide users with visual and functional features alternative to its Web platform (Fig. 4.2).

Functional skins (introduced in Bogdanov et al. [2011]) improve end-user experiences and personalization. While the space data and the data structure do not change, the visual representations and actions that user is allowed to perform with these data (functionality) might

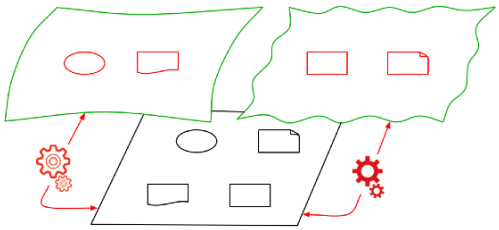


Figure 4.2: Functional Skin concept

differ from one Functional Skin to another. Thus, Functional Skins represent different interaction schemes. Being in a space, users can switch from one Functional Skin to another to change the way the interaction with the space is performed and, thus, satisfy their particular needs.

The functional skins are indeed skins, since, for example, a space can have several different functional skins that would display the learning space and enable interaction in different ways. We add the word *functional*, because in addition to the visual appearance change (that normal skins provide), the functional skin alters the available functionalities that users are provided with. This technique allows users to have *functional skins* for their spaces. It provides both a way to change the visual representation of data and the actions to be performed with these data. One of the main benefits of this approach is that a user can easily change the provided default functionalities to work with data. This enables code reuse since the same functional skin can be used by different people for their own spaces. People themselves can find and add new functional skins to handle space data. Functional skins can be used in different Web platforms, by providing users with the flexibility in choosing a Web platform, in which they prefer to work.

The purpose of functional skins is to change the interaction interfaces within spaces. A space might have meta-data (name, description, thumbnail image) and includes content such as members, resources and widgets. A Web platform that implements spaces already has an interface where users can create, manage and work with the space meta-data and content. The functional skin, implemented as a widget, can be assigned by the user to a particular space. Every space might have one or more functional skins associated with it. The choice is given to the user to select one of the available skins. When the user selects the desired skin, the programming code implementing the functional skin is rendered by the widget engine used by the platform. During rendering process, the space identifier is passed to the skin. Since the skin knows the identifier of its containing space, it uses OpenSocial APIs to retrieve the space meta-data (name, description, thumbnail) and also the content of a space (members, resources, widgets). When the needed data is received, the functional skin builds the visual representation for the retrieved data using the related JavaScript and CSS libraries. Additionally, the skin code might allow the user to change the space meta-data and content by calling OpenSocial APIs to update the space. Besides, since the skin is a meta-widget, it can save extra data as widget preferences. Through this approach, more space data can be

saved within the skin compared to what would be possible in standard Web platforms. In addition, extra functionalities can be enabled. For example, the skin can provide users with options to specify the order and the size of the widgets in a space or set and display only the most important resources in a space. This extra information is persisted into the Web platform storage as key-value pairs.

As an example, we consider a scenario where a knowledgeable person in Computer Science decides to create a space to support learners in mastering Common Algorithms of Computer Science. For this goal, the mentor creates a space called *Learn Computer Science*. Then learners (Alice, Bob, Chuck) and some subspaces (*Introduction*, *Basic Algorithms*, *Complex Algorithms*) are added to the space. Then the mentor structures information and populates every subspace with widgets and resources helpful to master Computer Science. The resulting view for the space *Basic Algorithms* is depicted in Fig. 4.3.

During the studying process some of the visual parts are not needed for learners as they might be irrelevant and cause distraction. Thus, the mentor wants to have a special view for learners that provides only the needed functionalities for learning: two widgets displayed side-by-side and a list of resources as a column on the right (Fig. 4.4).

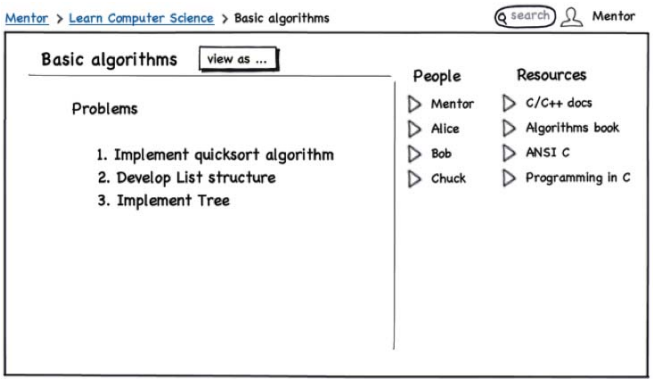


Figure 4.3: Default functional skin for *Basic Algorithms* space

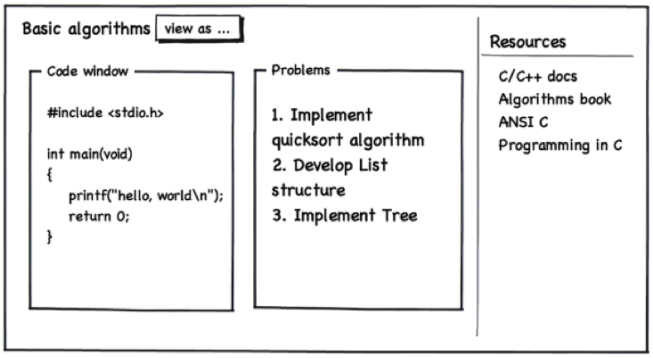


Figure 4.4: Learning-focused functional skin proposed by the mentor

With functional skins, both the mentor and the learners are provided with a way to change the visual representation of the data structure and available functionality through meta-widgets. The mentor can select a publicly available repository and find a functional skin that provides a functionality similar to Fig. 4.4 (this functional skin could be developed either by the mentor or by some other developers). The URL for this skin is retrieved and added into the list of URLs for *view as ...* button in the *Basic algorithms* space. Since this functional skin supports the OpenSocial Space extension, it processes information about the space, widgets and resources inside and displays information as illustrated in Fig. 4.4. The mentor is only required to find the URL for this widget and no additional implementation is required on the mentor's side to add this widget as a functional skin for the space. Thus, the mentor can easily share this functional skin with other people. In the case of a different Web platform (i.e., iGoogle) that supports OpenSocial Space extension, this functional skin can be used as well.

In addition, learners can add and change functional skins themselves. For example, a learner wants to have an additional area showing a list of people in the bottom-right area under *Resources* (Fig. 4.4). The learner can either search for another functional skin in widget repositories or develop a new one. Once the widget is ready to be used, its URL can be added to the space and the learner can switch to this new functional skin by choosing with *view as ...* button.

4.1.3 Functional Skin in Graasp

Graasp implements the Functional Skin concept (Section 4.1.2) as an extension mechanism introduced to customize the graphical and functional interface to work with spaces. The functional skin feature can be seen as a client-side plugin. It is an XML file with some JavaScript code that follows OpenSocial specification. It can be created by any user and added in a space at runtime without the intervention of platform developers.

Once a space is created in Graasp, the core part of the interface (the Pad - Fig. 4.5) enabling authorized users to interact with the aggregated resources can be further personalized with functional skins. In addition to the standard Graasp view used to populate spaces and to visualize their full content and members (Fig. 4.5), Graasp offers two built-in functional skins that can be chosen using a popup menu: the Resource view (Fig. 4.6) and the App view (Fig. 4.7). The Resource view displays a list of all resources that exist in a space and provides links for individual or full download. In addition, previews of resources can be displayed. The App view displays all widget instances from a space as a visual mashup. In this view, widgets can be resized and their order can be modified through drag and drop. The layout and the order of widgets are persistent among page reloads.

Functional skin in Graasp is a crucial element towards achieving platform plasticity. The possibilities to personalize the space are unlimited. Through functional skins the users can use spaces for their own professional/personal tasks. A person interested in music can use a functional skin to play MP3 audio files from a space and its sub-spaces. A project manager

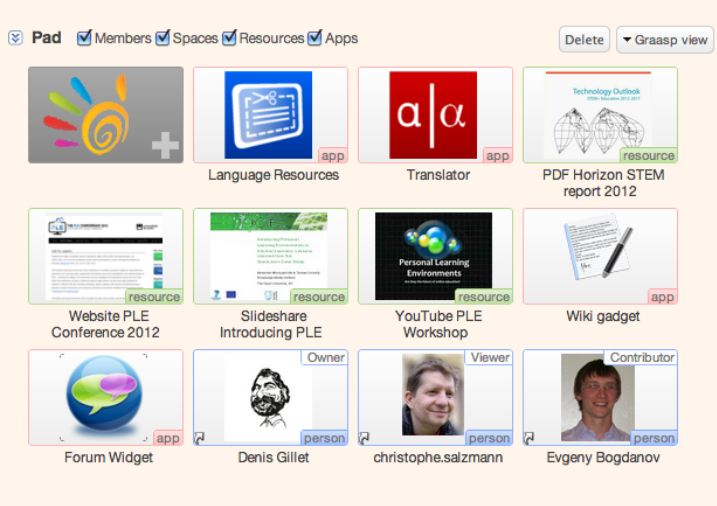


Figure 4.5: Default Graasp interface

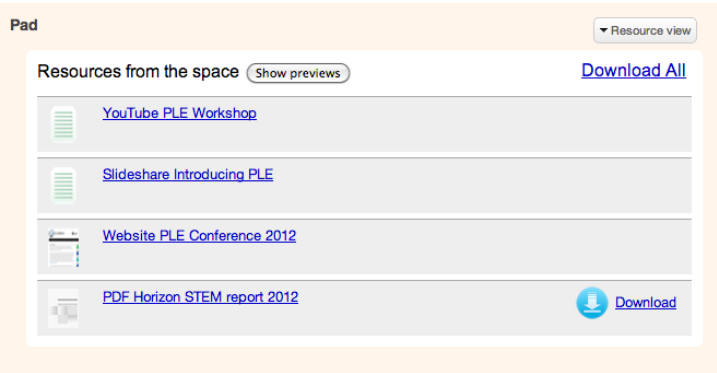


Figure 4.6: Functional skin - Resource view

can see when different documents were added to a space in a functional skin that shows space resources in a calendar according to their creation date. Skins can be easily shared among users, so even people with no programming skills can find useful skins. As another example, Fig. 4.8 demonstrates a functional skin showing the space apps and resources grouped by their creators.

To support meta-widgets and functional skins in Graasp, we had to implement several adaptations to the default Apache Shindig implementation. First, since the functional skins are built to change the interaction with the space, we use the OpenSocial Space extension with the corresponding APIs. The functional skin should be aware of its context (or a space where it is used). This is accomplished with the OpenSocial Space extension. Second, we enabled a meta-widget to be a widget container, meaning that it can render widgets inside itself, which is not possible by default within Shindig. In other words, a functional skin can get a list of widgets from a space and then display them inside itself. Finally, we had to add a security token field

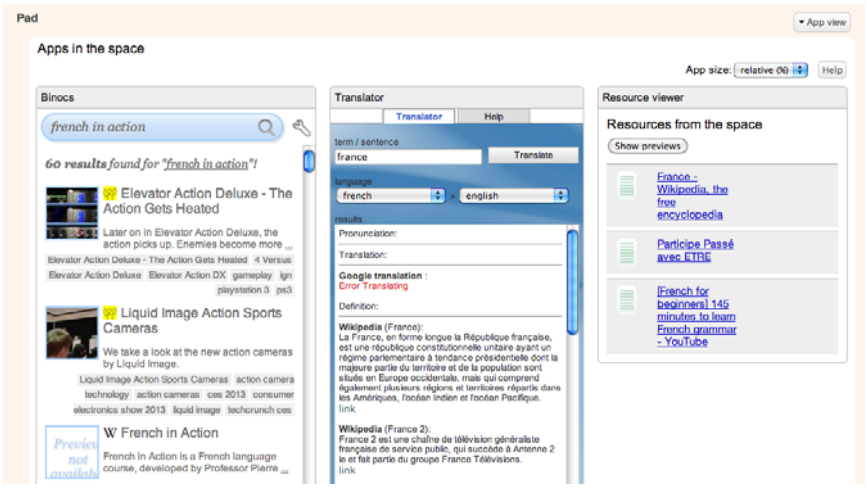


Figure 4.7: Functional skin - App view

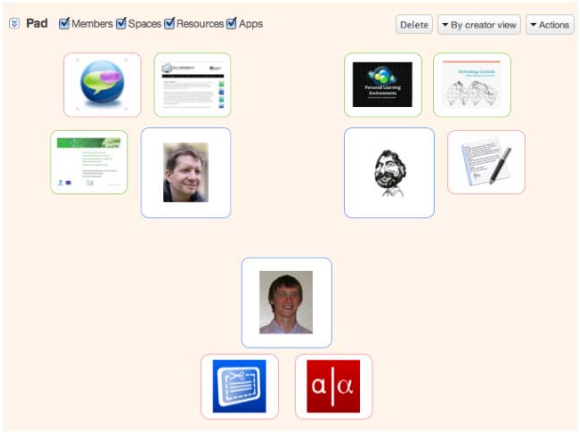


Figure 4.8: Functional skin - Group by creator view

in the App API (which is a part of the Space extension). Thus, when a list of widgets is received by a meta-widget, it can render them successfully, since the security token is a necessary requirement for widgets to function. For functional skins to work properly, these changes have to be integrated into Apache Shindig in different Web platforms using OpenSocial.

4.1.4 Functional Skin for Moodle

In Section 3.6, we described the OpenSocial plugin for Moodle. The plugin opens up Moodle to the world of OpenSocial widgets and enables flexibility for teachers. It allows teachers to control their teaching processes, which is the first step towards shrinking the gap between LMS and PLE worlds. This sections shows the benefits of adding functional skin concept into Moodle.

Currently, there is no support for functional skins in the Moodle plugin, however the introduction of this feature would increase the Moodle plasticity. In addition to all the benefits of functional skins described in the previous section, the Moodle plugin has two main limitations to which the functional skin concept is a direct solution.

The first problem concerns widget management. At the moment adding a new widget with Moodle plugin is a cumbersome process. Moreover, widgets can not be removed from the space. This feature requires additional implementation efforts. The problem deteriorates Moodle plasticity, since users can not easily aggregate and remove widgets in their spaces. A widget management functional skin providing support for adding and removing widgets in a space is a reusable solution for this problem.

The second problem is the management of widget layout in Moodle. Again, it becomes a plasticity problem because users are not flexible enough in the widget arrangement in their spaces. For the OpenSocial Moodle plugin, we had to develop a special interface that showed widgets in a grid of 1, 2 or 3 columns (Fig. 3.11). This involved a relatively high effort of implementing PHP part to retrieve widgets for a space and then JavaScript and HTML parts to render them on the page. Still, at the moment the Moodle plugin does not allow a teacher to change the widget order or resize a widget. If we are to implement it, it will require high implementation costs. However, with the support for functional skins in Moodle, we could easily reuse the *App view* functional skin developed for Graasp and shown in Fig. 4.7 that already enables the required features.

4.2 Aggregation and Sharing in Cloud

4.2.1 Graasp as a Plastic Platform

Graasp is meant to serve as a plastic Personal Learning Environment. It represents a collaborative Web-based platform that combines flexibility of a social media container with learning applications. In this regard, it particularly suits the self-directed learning paradigm whereby learners progress in a given discipline through personal and collaborative projects lead with a greater degree of autonomy (Li et al. [2010]). To give an example, a typical PLE could be composed of a mail client (e.g., Thunderbird), an account in a professional social networking platform (e.g., LinkedIn), an account at bookmarking platform (e.g., delicious.com), and an access to a university course material catalogue. While not aimed at replacing these tools or other Web 2.0 tools, Graasp provides in one platform the same functionalities. It can be used for communicating with peers or collecting and organizing resources coming either from the Web or from any third party repository including LMSs. Moreover, it allows learners to deal with different contexts by supporting the space notion, which is meant to help users aggregate and organize their online activities, resources, contacts, etc.

Many widget-based platforms provide support for aggregating tools and arranging them on user's pages. However, a mere tool aggregation is not enough for a plastic PLE. Graasp

provides a set of features to achieve the plasticity. The space concept allows learners to create different contexts for each learning goal/task they have. Within a space, they can aggregate the needed widgets and learning resources from the Cloud. The aggregation process is simplified with the Cloud aggregation architecture introduced in Section 4.2.3, that allows learners to easily aggregate widgets and resources into a space. The interaction with a space and the visual representation of its content can be changed with Functional Skins. The created learning spaces can be shared by the user with other people and accessed from the other Web platforms. All these features together enable users to shape their learning spaces as they wish. In the next section, we go through all the Graasp features enabling plasticity and detail their implementation and usage.

4.2.2 Space as an Aggregation Unit

Graasp has at its core the Space concept, that serves as an aggregation unit enabling plasticity. Users can aggregate resources and widgets into Graasp and distribute them to different spaces depending on the goals a space tries to achieve. Within a space, users can organize resources and widgets at their wishes. Sub-spaces within a space provide an hierarchical structure for the information organization.

Resources within Graasp are documents (PDF or image files, for example) or URLs and Web page screenshots for Cloud resources (YouTube videos, SlideShare presentations, Wikipedia articles, etc.). Support for adding Google Docs into a space is also available. Thus, Graasp enables users to add to their spaces all kinds of resources that they might find useful, both hard drive and cloud-based resources.

In addition to resources, Graasp allows users to aggregate widgets into their spaces. Currently, widgets are represented by OpenSocial widgets, though other standards (W3C widgets) can be added in a similar way. Existing OpenSocial widgets can be added manually to Graasp. Since every OpenSocial widget has a URL, this URL can be specified when a new widget is added in a space.

In addition, Graasp helps users to add widgets from several existing repositories in a faster and more convenient way. The ROLE Widget store and iGoogle repository provide browsing capabilities to search for widgets. The Widget store has already a pool of learning-oriented widgets and the number of available widgets is constantly growing. The iGoogle repository lists around 300 thousand widgets for different purposes. Graasp provides two simple ways to aggregate widgets from these repositories. First, when the user is browsing through widgets in the store or in iGoogle, a widget of interest can be added into Graasp through a click on the *GraaspIt!* bookmarklet. Second way to add widgets from the store is by exploiting the search mechanism within Graasp. When a new widget is to be added to Graasp, the user can open a window where it is possible to search for widgets available in the widget store (Fig. 4.9). Widgets can be added similarly to Widget bundles search interface described in Section 3.8.2.

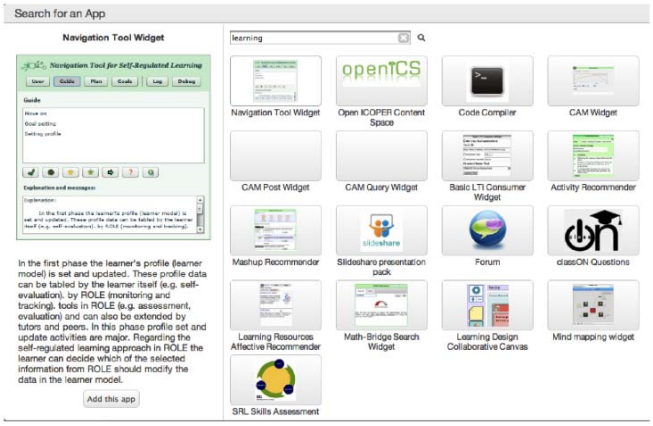


Figure 4.9: The Graasp widget search interface

4.2.3 Cloud Aggregation Architecture

To simplify resource aggregation from the cloud (Fig. 4.10), we propose a plugin-based Cloud aggregation architecture, where users can add resources from different Web sites into their spaces in a few clicks. The cloud aggregation architecture consists of two parts: a Web browser bookmarklet and a plugin manager (Fig. 4.11). When the user visits a Web page and clicks on the bookmarklet, the JavaScript code is injected into the page and executed. This code sends the page URL to the plugin manager. The plugin manager finds a plugin that corresponds to the URL and sends the plugin code back to the browser. Once the plugin code is loaded in the browser, it scrapes the page data (such as a name, a description, an embed tag, or some other information available) and sends this information to the Web platform. Then, the Web platform processes the information and creates either a new resource or a new widget based on the received information. The next section demonstrates the proposed architecture implemented as the GraaspIt! bookmarklet in Graasp.

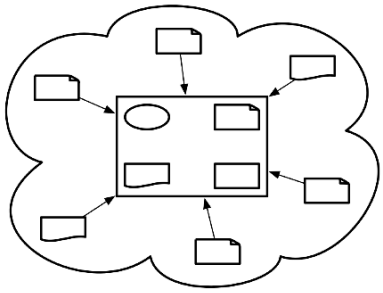


Figure 4.10: Cloud Aggregation concept

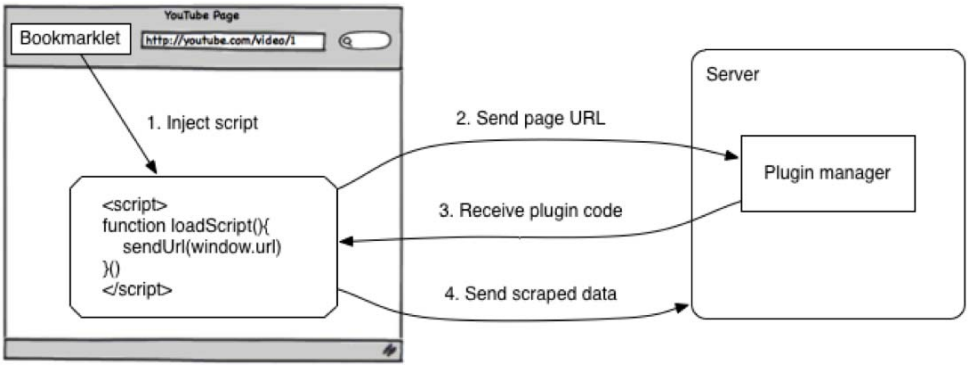


Figure 4.11: Cloud aggregation architecture

4.2.4 GraaspIt! for Cloud Aggregation

In Graasp, the Cloud aggregation architecture is represented by *GraaspIt!* a bookmarklet and the corresponding plugin manager. *GraaspIt!* is a bookmarklet that can be activated from the browser bookmark bar at any time when surfing the Web. Once a user is logged into Graasp, there is an option to add the bookmarklet to the browser bookmarks bar. Later, when the user visits a page of interest, the *GraaspIt!* button can be clicked and the user is provided with a dialog to add the item into Graasp (Fig. 4.12). After editing the name (if needed) and clicking on the "Add to Graasp" button, the resource will be immediately added into the personal Graasp clipboard. Then, it can be moved into the appropriate Graasp space with a simple drag-and-drop. *GraaspIt!* is an important tool that makes the Graasp platform plastic. With *GraaspIt!* users can easily populate their spaces with resources and widgets existing in the Cloud.

GraaspIt! provides a unified way of aggregating information from various Web sources (Fig. 4.13). *GraaspIt!* scrapes the information from the page the user opens in the browser and sends it to the Graasp backend. The *GraaspIt!* code is executed on the page when the user clicks on the bookmarklet. User identification is done via security token built into the bookmarklet. The bookmarklet sends a page URL to the plugin manager of Graasp as in the Cloud aggregation architecture. In Graasp, the plugin manager can be further divided into three parts.

First, to extract the useful information from the page, the plugin manager tries to use the embed.ly library[1], which recognizes more than one hundred Web sites and enables the embedding of their content. If the action is successful and the corresponding embed.ly plugin found, then a new resource is created in Graasp. This resource is given the name, the description and the embed code obtained from embed.ly service.

Second, if the first part of the plugin manager did not work (for example, the page is not supported by embed.ly), *GraaspIt!* tries to retrieve a customized plugin from Graasp for the

[1] http://embed.ly

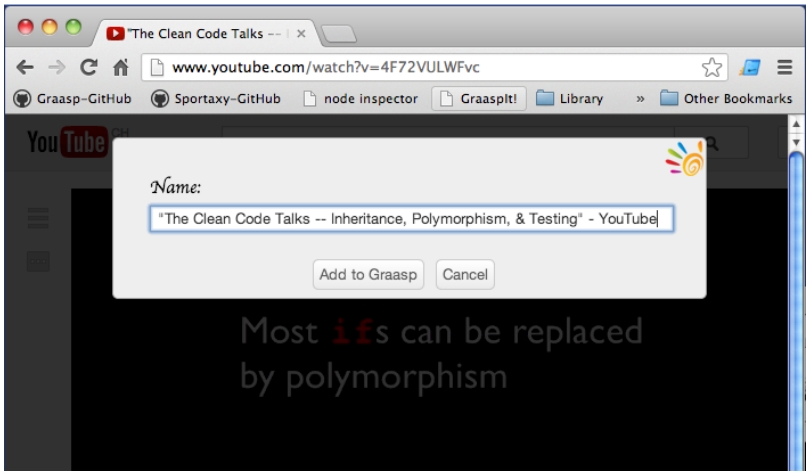


Figure 4.12: The GraaspIt! dialog

Figure 4.13: Content aggregation from various platforms thanks to the Cloud aggregation architecture

selected Web site. Each JavaScript plugin is mapped to a Web site URL. Based on the mapping scheme, *GraaspIt!* selects which plugin should be used for the URL open in the browser. Afterwards, the JavaScript file with the relevant code is loaded and executed. For example, if a widget store plugin is mapped to the URL role-widgetstore.eu/tool, then whenever the user clicks *GraaspIt!* on this URL, the widget store plugin will be loaded and executed. The plugin code extracts the name, the description and the embed code for the open URL according to the plugin logic and, then, *GraaspIt!* sends this information back to Graasp which creates a new resource or a new widget based on the extracted data.

Third, if neither embed.ly works nor a customized plugin exists for the Web site, the third part of plugin manager creates a screenshot of the Web page and saves it as a resource in Graasp, providing in such a way a combined bookmarking and archiving feature.

The Cloud aggregation architecture implemented in Graasp enables open content providers to add support for their own repositories or platforms via customized plugins. The *GraaspIt!* plugin manager is open-source and can be found on Github[2]. An example plugin for the ROLE Widget store is detailed in Appendix A.5.

4.2.5 Easy Space Sharing and Migration

Being able to easily share and migrate spaces from one platform to another is an important part of plasticity and Graasp enables it in several ways. As Section 3.8 detail, a space can be shared as an OMDL bundle. Both the existing space can be exported as an OMDL bundle and the user can import the existing OMDL bundles into Graasp to create new spaces based on them.

Section 3.7 represents two other approaches. First, a Graasp space can be extracted for use by other people within other platforms as a private or secret URL via the meta-widget approach. This way users can either access the shared URL in the browser or the space can be embedded into another Web platform as an iFrame. The second approach is the export of a widget bundle from a Graasp space as a public meta-widget URL. This means a new widget XML file is generated and this new meta-widget has the required space data built-in.

We should note an important plasticity aspect for space sharing that relates to the functional skin concept. At the moment, when a space is shared in Graasp via private or shared URL, the final interface displayed to users opening the space URL is fixed. It means, when a person shares a space, he does not have the possibility to change the resulting interaction interface to work with the shared space. However, to improve the plasticity the user should have enough flexibility in this case and it is one of the requirements for the Go-Lab project. Fortunately, it can be accomplished in a way similar to the functional skin concept. When the user is performing space extraction as a private URL, he might specify which functional skin has to be utilized for the final interface of the shared space. This information can be used during the

[2] https://github.com/react-epfl/graaspit

process of building the private URL and, thus, the resulting space interface will be personalized by the user according to his needs.

4.3 Related Work

4.3.1 Moodle and W3C widgets

The Apache Wookie plugin[3] for Moodle allows to bring W3C widgets into Moodle. Similarly to OpenSocial plugin, widgets can be added inside Moodle. The Moodle user interface, as it is shown to students and teachers, consists of a main center area and a narrow right column with blocks (Fig. 3.9). The W3C widgets can only be added into the blocks area which is a big limitation because widgets serve only as an addition for existing in Moodle wiki pages, lessons, etc. Nonetheless, it is a parallel step in improving Moodle as a PLE aggregator. We believe that a future plugin, that would integrate both OpenSocial and Wookie plugins, will be the best solution targeting both OpenSocial and W3C widgets.

4.3.2 ROLE SDK Moodle Plugin

The OpenSocial plugin for Moodle presented in Section 3.6 takes several steps forward in turning Moodle into a PLE aggregator. Here, we discuss an alternative approach that was created in ROLE project, that brings an actual PLE aggregator into Moodle. This alternative plugin is described in Bogdanov et al. [2012c]. A ROLE SDK platform is running side-by-side with Moodle and the new plugin allows to embed the ROLE SDK into Moodle. The new plugin does not integrate with Moodle database. Instead, the plugin retrieves a token that it passes on to the ROLE SDK, allowing it to access Moodle Web services (Conde et al. [2010]) on the authenticated user's behalf. In this way, the SDK is able to access information such as course metadata and participants, and make that information available to widgets. The ROLE plugin augments this with support for widgets to store data within the context of a course, similar to the App Data and Media Items of OpenSocial, and other functionality that would not be available from Moodle alone, such as real-time communication. The interface provided to widgets is generic rather than Moodle-specific, and it is possible to implement similar integration that offers the same widget interface also for other LMSs.

This alternative plugin has improved personal aspects. It integrates the ROLE Widget store where people can search for widgets and add them. Thus, students can manage widgets alongside those chosen by the teacher, which is difficult to enable in the OpenSocial plugin because Moodle requires additional access rights, which students normally do not have. Students can also change the preferences of any widget, configuring it to their needs, overriding the teacher's preferences. Furthermore, widgets can be rearranged and resized. A personal dashboard is added to the bottom of the Moodle page and contains widgets that are chosen by the student without the teacher's involvement. The dashboard is available independently of

[3] https://moodle.org/mod/data/view.php?rid=3319

the course, which means that students can add widgets that they can access in every course. As the ROLE SDK is actually running side-by-side with Moodle, it is possible to access it directly bypassing Moodle.

However, there are several problems with this solution. First is the UI complexity of the Moodle page layout due to the embedding of the whole ROLE SDK interface inside. Second is the limited richness of APIs for widgets compared to the OpenSocial APIs available in the OpenSocial plugin. Third is the usage of APIs specific to ROLE SDK that are not OpenSocial compatible which makes difficult the reuse of existing OpenSocial widgets.

4.3.3 Liferay as a PLE

Ullrich et al. [2010] investigated the usage of PLEs in a similar setting (a French course at SOCE), but with significant differences. First, the PLE aggregator was implemented in an external system (Liferay[4]) which was not integrated into the school LMS, thus introducing an additional layer of complexity due to the different user interface and additional log in.

4.3.4 Learning Tools Interoperability

Another option to bring external tools to Moodle is the Learning Tools Interoperability (LTI) specification described in Section 3.1.1. The main benefits of this approach compared to OpenSocial are the fact that the database to Web Services mapping (via LTI) is already done by many LMSs supporting LTI (for example, Moodle[5]) and it is relatively easy to add support for LTI to new LMSs. On the other hand, to add support for OpenSocial, a similar mapping (database to OpenSocial APIs) has to be implemented for new LMSs (the OpenSocial plugin already does it for Moodle), which in general requires some time. Once LTI is implemented within an LMS, new external tools can be integrated into it. However, there are several drawbacks of LTI approach. First, with this approach only one tool can be integrated at a time. Another limitation is security signing keys that have to be shared between an LMS and external tools when they are being added. It adds an overhead that is comparable to the OpenSocial plugin installation process. It should be mentioned also, that even though OpenSocial might require more work to integrate, it offers a rich APIs set for external tools to interact with an LMS. The amount of information tools can retrieve with LTI is very limited. LTI specification targets exclusively LMSs, while OpenSocial represents an open standard with wider audience. We again see the problem (mentioned by Wilson et al. [2007]) of closed nature of LMSs where open standards do not get enough attention. Depending on how much flexibility on the APIs level an external tool needs and what requirements on the plugin installation process are, a tool developer would choose either OpenSocial plugin or LTI approach.

[4] http://liferay.com
[5] http://moodle.org/mod/forum/discuss.php?d=191745

4.3.5 Monolithic versus Modular Approaches

This section provides an example of migration from monolithic to the modular approach at Ecole Polytechnique Fédérale de Lausanne (EPFL) within the Automatic Control course (Bogdanov et al. [2012b]).

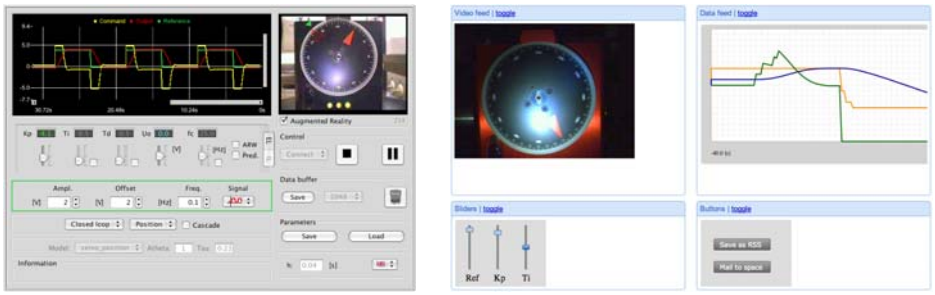
Flexible education is a recent approach used by institutions to transfer knowledge from universities to students. In Gillet et al. [2005] the authors stated that both a flexible access to experimentation resources and availability of collaboration facilities are required. The flexible access to experimentation is provided to students with the help of virtual and/or remote laboratories. In addition, Web collaboration facilities are provided to support students while conducting an experiment. For example, the resources produced by students, such as simulation results or measurements, can be saved in a shared space to be accessed by others. These two major requirements shape the solutions for remote experimentation laboratories.

Many of the existing solutions are developed as complex monolithic stand-alone frameworks that handle both the remote experimentation aspect and collaborative work flow. However, according to Salzmann and Gillet [2008], Salzmann et al. [2013], this monolithic structure is difficult to adapt to varying user requirements, evolving curriculum and new technologies. In Salzmann and Gillet [2007], the authors explain that the high development overhead and the difficulties associated with the integration of the remote experimentation within existing LMSs has refrained the spread and the acceptance of common monolithic solutions.

Despite the limitations of existing solutions, the need and justifications for virtual and remote laboratories are still present. Salzmann et al. [2013] proposed a new smart device+widget paradigm to divide the current monolithic solution into smaller universal components (Web widgets) that users can re-aggregate dynamically to form a personal environment. Similarly, some intelligence and the flexibility are added to a remote experimentation server to provide more autonomous actions and to support a wider range of clients and protocols. The space concept finds its application within flexible education due to its inherent modularity.

At EPFL, the existing Remote Experimentation Device (RED) for the Automatic Control course laboratory was implemented as a Java applet (Fig. 4.14a). Due to the limitations of this approach (mainly, evolution and maintenance) it was decided to completely re-engineer the remote lab solution as a bundle of light-weight OpenSocial widgets and smart devices (Salzmann et al. [2013]). The widgets can be combined together to provide the same functionality that existed in the Java applet solution. The resulting implementation can be seen in Fig. 4.14b. From the comparison of two pictures, the main functionalities are still the same, however, the modular approach has many valuable advantages discussed in the next paragraph.

Turning a monolithic application into a modular one with OpenSocial widgets has many advantages. First, it brings more flexibility to teachers and students, since they are able to assemble modules on their own or replace some modules at their will. They can even extend the proposed set of modules with other modules such as a chat tool for communication or



(a) as java applet (b) as widget bundle

Figure 4.14: RED device as a set of widgets

a collaborative formula editing tool. Additionally, maintenance and development costs are greatly reduced, since universities do not have to maintain a standalone application but rather relatively simple modules. In addition, already existing modules can be reused. Second, the widgets use pure Web technologies which makes them portable between various Web browsers (mobile devices included). Users do not have to install additional plugins such as Flash or Java. The usage of widget standards allows to bring the remote experimentation widget bundle into other LMSs that students may already use (e.g., Moodle or Sakai[6]). Furthermore, widgets can save and retrieve data to/from their Web platform. This feature helps users to directly store data through widgets into LMS (or another platform) where they run experiments with avoidance of additional authentication steps.

4.4 Platform Analysis with PLE Dimensions

We illustrate our comparison of several platforms with respect to the PLE dimensions shown in Table 1.1. We have investigated the support for each of the four features per dimension and simplified the findings to whether it is supported or not. The platforms we analysed are Moodle, Moodle with the Wookie engine, Moodle with OpenSocial plugin[7], ROLE SDK and Graasp. Table 4.1 presents the PLE features availability in different platforms and Fig. 4.15 depicts the spider diagrams with the results.

First, we clarify now how different dimensions were interpreted during the analysis. Because of the design decisions, the layout of widgets in Graasp and OpenSocial Moodle plugin is achieved through a functional skin where apps view is implemented as a meta-widget. Both platforms rely on this functional skin which enabled the features of Aggregation dimension for them. Inter-widget communication is enabled via OpenApp library. Even though it is supported only on the widget level and not on the level of a Web platform itself, we set this PLE feature as available, because it can be accomplished by users. Linked Data support feature was interpreted not as a support for Linked Data but rather the availability of common properties

[6] http://sakaiproject.org

[7] we considered a version with support for functional skins and the migration widget

Dimension	Features	Moodle	Graasp	ROLE SDK	Wookie Moodle	OpenSocial Moodle
Aggregation	Screen aggregation	x	x	x	x	x
	Widget standards		x	x	x	x
	Layout of widgets		x	x		x
	Web desktop					
Communication	Inter-widget communication		x	x	x	x
	Drag and Drop		x	x	x	x
	PLE data manager		x	x		x
	Linked data support		x	x		x
Synchronization	Push data updates			x	x	
	Push preference updates					
	Real time data updates					
	Data and preference history					
Organization	List of friends		x			
	Friends server		x			x
	Access control	x	x	x	x	x
	Independent groups					
Recommendation	Manual guide	x	x	x	x	x
	Flow enabled widgets					
	Scripted inter-widget data flow					
	Recommendations		x	x		
Configuration	Feed export and import					
	Generic export and import		x	x		
	External configuration					
	Embedding		x	x		

Table 4.1: Platform analysis by the PLE features

allowing widgets to understand each other's data without preparation. OpenSocial enables similar functionality for OpenSocial widgets. For the Synchronization dimension, even though widgets can implement it, there is no standardized way to enable it with OpenSocial or native implementation in Graasp or Moodle. For the Organization dimension we do not consider only user friends, but also people with whom a space can be shared.

From Fig. 4.15 one can see how different PLE dimensions are supported by the platforms. Moodle by itself has weak support for PLE dimensions, however, with plugins for adding widgets, many PLE features can be supported. Note that even though all the platforms have a solid technical foundation for building PLE, they have very limited support for helping a user with the learning process which is the main responsibility of the Recommendation dimension. Synchronization dimension is another area that deserves particular attention. This is to enable the real-time collaboration between people during their learning activities, but it is very poor at the moment.

Graasp has good support in all but poor in Recommendation and Synchronization dimensions. To improve its applicability as a PLE, support for real-time updates and the mechanisms to manage a learning process have to be added. The best solution is to enable these functionalities through functional skins. This way the solutions can be re-used within the OpenSocial Moodle plugin or another Web platforms. The lack of Runtime support in OpenSocial Moodle

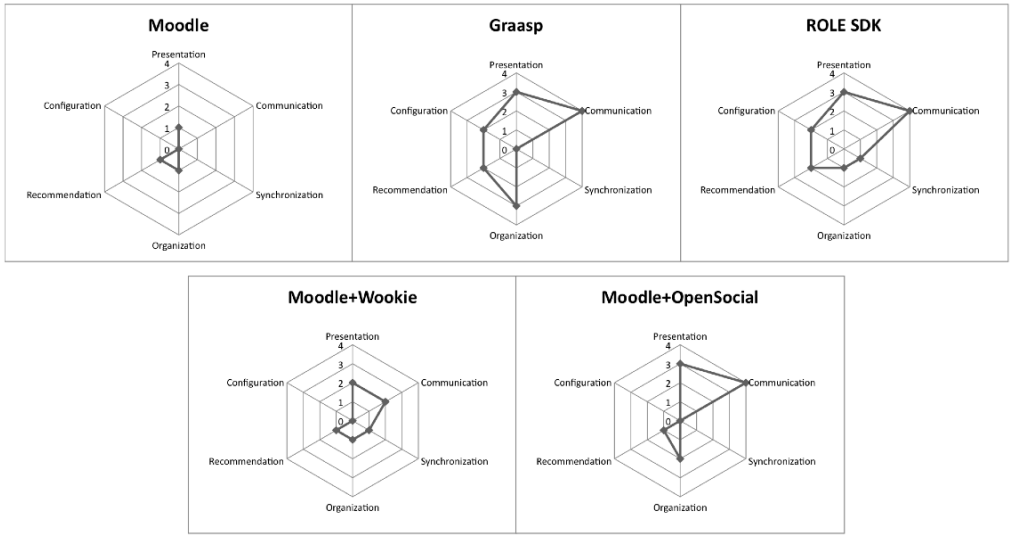


Figure 4.15: Platform comparison by six PLE dimensions

plugin can be solved in ways similar to Graasp, via a space extraction as a meta-widget with secret and private URLs. However, this can not be done on the functional skin level and requires an implementation on the plugin level.

The conducted comparative analysis of the platforms by the PLE dimensions support revealed their shortcoming when they are to be used as PLE aggregators. The analysis not only showed that there are some problems but constructively showed where the problems lie and paved the way to the possible solutions. The analysis results coincide with the platform limitations we intuitively defined. This indirectly confirms the usefulness of PLE dimensions benchmarking and that it should be applied to find out the problems with other platforms that target to support PLEs.

4.5 Conclusion

This chapter is focused on achieving personal and contextual plasticity for PLEs. We introduced the *Functional Skin* concept and detailed the concept of *Cloud aggregation architecture* and its implementation in Graasp: namely, *GraaspIt!*.

We detailed how Graasp enables plasticity for its users i) through the integration of the Space concept to represent a PLE built by the user, ii) GraaspIt! to easily aggregate resources from the Cloud, iii) the Functional Skin support to personalize the interaction with spaces, and iv) by sharing created spaces in the Cloud. We showed how the Moodle plugin introduced in Chapter 2 can benefit from the Functional Skin concepts towards achieving plasticity. Finally, we provided a comparative analysis of five Web platforms for their support of the PLE dimensions.

5 Conclusion

PLE is a relatively new concept that draws more and more attention. Innovative research is required to understand what the PLE place in learning and education domains is. The main goal of this thesis was to tackle the challenges of enabling PLE construction as personal and contextual spaces with a high degree of portability and plasticity. This chapter summarizes the thesis contributions, discusses the proposed solutions and the possible future directions for each solution.

5.1 Space-related Contributions

We introduced the **Space concept** to model user contexts (**contribution 1**). This concept is crucially important for PLEs in particular and for Personal Environments in general, where users or learners assemble their environments (contexts) for different purposes. We showed that this concept is inherent to many Web platforms. By analysing the OpenSocial specification that models and standardizes the social networking functionalities of Web platforms, we found its limitation in modeling people's contexts. As a solution, we proposed an extension to OpenSocial (**contribution 2**) that introduces spaces into the specification that overtakes the limitation and provides greater flexibility for users. The **OpenSocial Space extension** permits the creation of **Contextual widgets** that adapt to their context (**contribution 3**).

Despite the generality of the space concept, its applications (Contextual widgets, Functional Skins, Portable Platform Interfaces, etc.) have the dependency on OpenSocial specification and assume that the targeted Web platform exploits widgets. Therefore, to benefit from the proposed solutions, a Web platform has to implement the OpenSocial standard (or similar) and to support client-side plugins (widgets) that can access space data through APIs.

5.2 Portability-related Contributions

5.2.1 Space Sharing

Sharing environments (or spaces) with different users is an important feature of a PLE aggregator. Moreover, it has to be supported not only within one single Web platform but also across different Web platforms. In this thesis, we explored two different approaches to achieve this goal: **Collaborative Portable Space configurations** (**contribution 4**) and **Portable OpenSocial Spaces** (**contribution 5**).

Collaborative Portable Web Space approach proposes space configuration elements that can be shared by several Web platforms. In this approach, there exist one configuration of a space, and different containers are responsible to synchronize the state of all space instances when configuration changes occur. We introduced a **Space configuration language** to enable this approach and drafted the possible implementation for it.

The **OpenSocial space meta-widget** approach also targets a problem of space sharing but from a different perspective. A space existing in one container is brought to another container via a special meta-widget. Thus, even though all space data is physically located in the first container, it can be accessed within the second one. This approach is general and does not require special preparation from the second container. It can even be used on a single HTML page. The space meta-widget contains the secret URL, through which users access it. From the widget point of view, all people accessing it with the secret URL represent the same person (secret URL creator). Thus, further investigations are required to differentiate between people accessing the same secret URL and enable the flexible management of people's identities within such a meta-widget. Another approach to space sharing is to incorporate all space data inside the meta-widget. This way a space meta-widget is self-contained and can be rendered on any OpenSocial-compliant Web platform. We provided an initial proof-of-concept solution validating the concept, however more research is needed in this area to build a solution that can be used in practice by real users.

5.2.2 Space Migration

Often it is not enough for people to solely access a space existing in some container, instead they want to migrate it into their preferred Web platform. To support this scenario, we designed a solution that involves a **Migration widget** and the usage of OpenSocial APIs. The migration widget, when added to the user's Web platform, can retrieve information from another platform via corresponding OpenSocial APIs and save it into the user's platform again via OpenSocial APIs. By using the OpenSocial specification we can avoid the problem of data format conversion. One limitation concerns the richness and flexibility of OpenSocial: only those models existing in OpenSocial and implemented by both containers can be migrated.

The presented solution includes the migration widget. However, it is not a necessary require-

ment for a Web platform to enable migration of OpenSocial spaces. If two Web platforms support OpenSocial APIs, then the migration can be achieved with the aid of a dedicated library added to the destination Web platform.

5.3 Plasticity-related Contributions

People are looking for ways to personalize (shape) their learning environments according to their needs. We defined **plasticity** as a measure of a Web platform ability to let users customize the UI and aggregate (or share) Cloud content (**contribution 6**).

The **Meta-widget concept** in combination with the OpenSocial space extension enables **Functional Skins** (different graphical and functional interfaces for the same space data) and **Portable Platform Interfaces** (the same graphical and functional interfaces for different data structures), which represents **contribution 7** that improves plasticity of Web platforms. The **Cloud aggregation architecture** that permits users to easily aggregate external resources from the Cloud into their Web platforms is **contribution 8** that increases plasticity of Web platforms.

Overall, the implementations in Graasp demonstrate how plasticity can be achieved in a Web platform through the space concept, space personalization via functional skins, migration and sharing of spaces, and the Cloud aggregation architecture.

5.4 PLE Dimensions

We refined the **six PLE dimensions** serving as an analysis grid to estimate the applicability of a Web platform to be used as a PLE aggregator. We defined six dimensions: **Aggregation**, **Communication**, **Synchronization**, **Organization**, **Recommendation**, and **Configuration**, and selected four features for each dimension. The number of features of a dimension that a Web platform supports defines its score in this dimension. Thus, our analysis grid shows how good the overall PLE support of a Web platform is and helps to find out the strong and weak points of a Web platform in each dimension. Moreover, the analysis grid allows to compare Web platforms by their PLE support.

For some dimensions, it is possible to find more than four features or some features are more important than the others during the comparison, however we decided to take the four most important ones to ease the comparison of platforms. This is one of the limitations of the analysis grid, that should be further refined into a more flexible scheme.

To show the feasibility and evaluate the acceptance of the ideas presented in this thesis, we implemented them in two Web platforms. The Moodle plugin brings OpenSocial widgets and spaces into Moodle LMS and provides more flexibility for teachers in organizing their courses. Graasp is a Web platform with spaces and OpenSocial widgets at its core that is a standalone PLE aggregator. The evaluations with learners and teachers showed that they are pleased with

the space concept in both platforms.

The analysis of Moodle with OpenSocial plugin and Graasp with the six dimension analysis grid discovered their weak points, that have to be addressed in the future work: namely, both platforms are limited in the Synchronization and Recommendation dimensions. These limitations can be improved by introducing the corresponding features of these PLE dimensions, for example: real time data updates and flow enabled widgets (Table 1.1). Some ideas implemented in Graasp can be applied to Moodle OpenSocial plugin to improve its support in the Configuration dimension, for example: space meta-widget sharing.

To conclude, we should specifically stress, that even though this thesis targets Personal Learning Environments, most of the ideas discussed can be used for any Personal Environment, be it research, work, sport, training or other domain.

5.5 Future Work

Future work aims at improving the OpenSocial Space extension. In addition, actions towards a wider adoption of the Space concept within companies and research organizations is ongoing. The European Go-Lab project uses the space concept and the Graasp platform to manage inquiry learning spaces for remote and virtual labs. This project serves as a test bed for the space concept and the OpenSocial Space extension. Moreover, the concept of functional skins is used within the project to enable various graphical and functional representations of spaces that are specific to the project requirements. The scenarios of space portability are being implemented and explored to enable the sharing and portability of inquiry learning spaces. For example, the scenario of sharing a space by a teacher with her students is currently investigated.

A An appendix

We provide here our extensions to OpenSocial that introduce the Space concept into the specification. Please note, that due to the size of the specification, we do not provide the full specification but only our extensions, however we give the links to the corresponding full specification in each appendix section.

A.1 Appendix A: Space Extension Models

The Space Extension introduces two new models (Space and App) into the OpenSocial specification[1] and changes some other models to adapt them to spaces.

A.1.1 Group

OpenSocial Groups are used to tag or categorize people for a specific Context and their relationships or roles (space members, person's friends, space owners, etc.). Each group has a display name, an identifier which is unique within the groups in that Context, and a URI link. A group may be a private, invitation-only, public or a personal group used to organize friends. A group also serves as an access control role within Context (e.g., owner, contributors, viewers, moderators, etc.)

A.1.2 Context

Each context returned MUST include the "id", "service" and "object" non-empty fields

Context object represents either a Space or a Person. An OpenSocial app can be opened either on a page of a Person or on a page of a Space. Context object for this app represents exactly this information. Imagine you're checking out a coworker's profile on Orkut. In this case, Context object will contain information about a Person (your coworker). In case you're checking out

[1] http://iamac71.epfl.ch/os_space/Social-Data.xml

an English Course space, the Context object will contain information about this space (English Course).

```
Context = "{"
              "id      : " Resource-Id ","
              "service : " people | spaces ","
              "object  : " Person | Space
          "}"
```

A.1.3 Space

Each space returned MUST include the "id", "parentId", and "displayName" fields with non-empty values, but all other fields are optional, and it is recognized that not all Service Providers are able to provide data for all the supported fields. The field list below is broad so that there is a standard field name available among Service Providers that do support any of these fields.

```
Space = "{"
           "id          : " Resource-Id ","
           "displayName : " string ","
           [ #Space-Field ]
        "}"
```

Valid definitions for Space-Field are listed in the table below.

Field Name	Field Type	Description
addresses	Plural-Field <Address>	A physical mailing address for this Space.
appData	Plural-Field <AppData>	A collection of AppData keys and values.
parentId	Resource-Id	**Required**. ParentId defines where this space belongs to (a person or a space).
description	string	The small description of this Space. It might contain Space's motto or goal or target audience
displayName	string	**Required**. The name of this Space, suitable for display to end-users. Each Space returned MUST include a non-empty displayName value. The value provided SHOULD be the primary textual label by which this Space is normally displayed by the Service Provider when presenting it to end-users.
emails	Plural-Field <string>	E-mail address for this Space. The value SHOULD be canonicalized by the Service Provider, e.g. shindig.group@plaxo.com instead of shindig.group@PLAXO.COM.
hasApp	Boolean	Indicating whether the space has application installed.

id	Resource-Id	**Required**. Unique identifier for the Space.
logos	Plural-Field <string>	URL of logo images of this space. The value SHOULD be a canonicalized URL, and MUST point to an actual image file (e.g. a GIF, JPEG, or PNG image file) rather than to a Web page containing an image. Service Providers MAY return the same image at different sizes, though it is recognized that no standard for describing images of various sizes currently exists. Note that this field SHOULD NOT be used to send down arbitrary photos added to this space, but specifically image associated with this space.
ims	Plural-Field <string>	Instant messaging address for this Space. No official canonicalization rules exist for all instant messaging addresses, but Service Providers SHOULD remove all whitespace and convert the address to lowercase, if this is appropriate for the service this IM address is used for. Instead of the standard Canonical Values for type, this field defines the following Canonical Values to represent currently popular IM services: aim, gtalk, icq, xmpp,msn, skype, qq, and yahoo.
location	string	
name	string	The textual identifier for a space. For sites that ask for a unique name of a space (e.g. opensocialgroup or partuza).
phoneNumbers	Plural-Field <string>	Phone number for this Space. No canonical value is assumed here. In addition to the standard Canonical Values for type, this field also defines the additional Canonical Values mobile, fax, and pager.
profileUrl	string	Space's URL, specified as a string, where this space can be found. This URL must be fully qualified. Relative URLs will not work in gadgets.
rating	integer	Average rating of the space on a scale of 0-10.
status	string	Space's status, headline or shoutout.
tags	Plural-Field <string>	Tags associated with this space.
thumbnailUrl	string	Space's image thumbnail URL, specified as a string. This URL must be fully qualified. Relative URLs will not work in gadgets.

updated	Date	The most recent date the details of this Space were updated (i.e. the modified date of this entry). The value MUST be a valid Date. If this Space has never been modified since its initial creation, the value MUST be the same as the value of published. Note the updatedSince Query Parameter can be used to select only spaces whose updated value is equal to or more recent than a given Date. This enables Consumers to repeatedly access a space's data and only request newly added or updated contacts since the last access time.
urls	Plural-Field <string>	URL of a Web page relating to this Space. The value SHOULD be canonicalized by the Service Provider, e.g. http://englishtandem.com/about/ instead of ENGLISHTANDEM.COM/about/. In addition to the standard Canonical Values for type, this field also defines the additional Canonical Values blog and profile.
utcOffset	Date-UTC-Offset	The offset from UTC of this Space's current time zone, as of the time this response was returned. The value MUST conform to the Date-UTC-Offset. Note that this value MAY change over time due to daylight saving time, and is thus meant to signify only the current value of the space's timezone offset.

A minimal Space example (application/json representation):

```
{
  "id"          : "http://example.org/space/1"
, "displayName" : "English course"
, "description" : "English learning course at EPFL"
, "parentId"    : "http://example.org/people/666"
}
```

A.1.4 App

Each app returned MUST include the "id", "name", "parentId" and "appUrl" fields with non-empty values, but all other fields are optional, and it is recognized that not all Service Providers are able to provide data for all the supported fields. The field list below is broad so that there is a standard field name available among Service Providers that do support any of these fields.

```
App = "{"
```

```
    "id  : " Resource-Id ","
    "url : " string ","
    [ #App-Field ]
"}"
```

Valid definitions for App-Field are listed in the table below.

Field Name	Field Type	Description
appData	Plural-Field <AppData>	A collection of AppData keys and values.
appType	string	Type of App (for example: OpenSocial App, W3C widget, etc.)
appUrl	string	**Required**. Url where the app source code can be retrieved
author	string	Name of app's author
authorEmail	string	Email address of app's author
parentId	Resource-Id	Required. ParentId defines where this app belongs to (a person or a space).
description	string	The small description of this App. What this app does, how it can be used, etc.
displayName	string	**Required**. The name of this App, suitable for display to end-users. Each App returned MUST include a non-empty displayName value. The value provided SHOULD be the primary textual label by which this App is normally displayed by the Service Provider when presenting it to end-users.
height	number	The height of this app
id	Resource-Id	**Required**. Unique identifier for the App.
ims	Plural-Field <string>	Instant messaging address for this App. No official canonicalization rules exist for all instant messaging addresses, but Service Providers SHOULD remove all whitespace and convert the address to lowercase, if this is appropriate for the service this IM address is used for. Instead of the standard Canonical Values for type, this field defines the following Canonical Values to represent currently popular IM services: aim, gtalk, icq, xmpp,msn, skype, qq, and yahoo.
name	string	The name of this App, suitable for display to end-users. Each App returned MUST include a non-empty displayName value. The value provided SHOULD be the primary textual label by which this App is normally displayed by the Service Provider when presenting it to end-users.
profileUrl	string	Url of this app instance, specified as a string, where this app can be found (as shared gadget instance in iGoogle for example). This URL must be fully qualified. Relative URLs will not work in gadgets.

rating	integer	Average rating of the app on a scale of 0-10.
tags	Plural-Field <string>	Tags associated with this app instance.
srcTags	Plural-Field <string>	Labels for this app provided by its author, e.g. "science" or "math". These values SHOULD be case-insensitive, and there SHOULD NOT be multiple tags provided for a given app that differ only in case. Note that this field consists only of a string value.
thumbnailUrl	string	App's thumbnail URL, specified as a string. This URL must be fully qualified. Relative URLs will not work in gadgets.
srcThumbnailUrl	string	Thumbnail url for this app that is provided by app author. This URL must be fully qualified. Relative URLs will not work in gadgets.
srcScreenshotUrl	string	App's screenshot URL, specified as a string. This URL must be fully qualified. Relative URLs will not work in gadgets.
updated	Date	The most recent date the details of this App were updated (i.e. the modified date of this entry). The value MUST be a valid Date. If this App has never been modified since its initial creation, the value MUST be the same as the value of published. Note the updatedSince Query Parameter can be used to select only apps whose updated value is equal to or more recent than a given Date. This enables Consumers to repeatedly access an app's data and only request newly added or updated contacts since the last access time.
utcOffset	Date-UTC-Offset	The offset from UTC of this App's current time zone, as of the time this response was returned. The value MUST conform to the Date-UTC-Offset. Note that this value MAY change over time due to daylight saving time, and is thus meant to signify only the current value of the app's timezone offset.
version	string	The App's version

A minimal App example (application/json representation):

```
{
  "id"          : "http://example.org/apps/18"
, "displayName" : "Chat app"
, "description" : "This app allows you to communicate with other people."
, "appUrl"      : "http://example.org/app.xml"
, "parentId"    : "http://example.org/spaces/1"
}
```

A.1.5 GroupId

Use of predefined groups for spaces

GroupId	Definition
@members	All people who explicitely joined the space.
@owners	Space members with unlimited rights
@contributors	People who contribute content to the space (but less rights than owners)
@creator	The creator of a space

A.2 Appendix B: Space Extension REST APIs

The Space Extension introduces several new REST APIs (Space and App) into OpenSocial specification[2] and changes some other APIs to adapt them for work with spaces.

A.2.1 People

Get a list of People

Containers MUST support retrieving information about multiple people in a single request. It returns a list of people with explicitly given ids or related to a specific Resource-Id and Group (e.g. space members, user friends). Requests and responses for retrieving a list of people use the following values:

```
REST-URI-Fragment = "/people/" Resource-Id [ "/" Group-Id ]
```

Get People Request Parameters A request for a Collection of Person objects MUST support the Standard-Request-Parameters, the Collection-Request-Parameters, and the following additional parameters:

Name	Type	Description
userId	Resource-Id or Array<Local-Id>	IRI of object for which people are retrieved (Space, Person) or an array of Local-Ids of people to retrieve.
groupId	Group-Id	The Group Id of the specific group of users related to Resource. Defaults to "@self", which MUST return all the Person object(s) for the Resource.
fields	Array<String>	An array of Person field names. For standard values, see the Person object.

[2] http://iamac71.epfl.ch/os_space/Social-API-Server.xml

networkDistance	number	Containers MAY support the network distance parameter, which modifies group-relative requests (@friends, etc.) to include the transitive closure of all friends up to the specified distance away.

Examples Here's an example of a REST request to retrieve the list of a space members, and the associated response.

HTTP Request

```
GET /people/api.example.org%2Fspaces%2F8/@members?fields=name,gender HTTP
  /1.1
HOST api.example.org
Authorization: hh5s93j4hdidpola
```

HTTP Response

```
HTTP/1.x 207 Multi-Status
Content-Type: application/json
{
   "id"      : "spaceMembers"
,  "result" : {
     "startIndex"   : 1
   , "itemsPerPage" : 2
   , "totalResults" : 100
   , "entry"        : [
       {
         "id"     : "api.example.org/people/8"
       , "name"   : { "unstructured"  : "Jane Doe" }
       , "gender" : "female"
       }
     , {
         "id"     : "api.example.org/people/68"
       , "name"   : { "unstructured" : "John Smith" }
       , "gender" : "female"
       }
     ]
   }
}
```

Create a Person/Relationship

Containers MAY support request to create a new relationship between user from one side and user or space from another side. This is a generalization of many use cases including invitation, contact creation, space membership request. Containers MAY require a dual opt-in process before the friend or membership record appears in the collection, and in this case SHOULD return a 202 Accepted response, indicating that the request is 'in flight' and may or may not

be ultimately successful. Note, that not all implementations support creation/deletion of a Person account through the API for security reasons.

Create Person/Relationship Request Parameters A request to create a relationship MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
userId	User-Id	User ID of the person initiating the relationship request. Defaults to "@me", which MUST return the currently logged in user.
groupId	Group-Id	The Group Id specifying the type of relationship. Defaults to "@friends" for people and "@members" for spaces. For example, "@contributors" would add a person to a list of contributors in a space.
context	Context	The target of the relationship. Can be a space for membership request or a person for contact creation request.

A.2.2 Spaces

Containers MUST support the Spaces Service. Individual operations are required or optional as indicated in the sections that follow. Containers MUST use the following values to define the Spaces Service:

```
XRDS-Type    = "http://ns.opensocial.org/2008/opensocial/spaces"
Service-Name = "spaces"
```

Get

The Get method supports several different types of queries, from requests for a space or group of spaces to information about what profile fields are available.

Get a Space

Containers MUST support retrieving information about a space. Requests and responses for retrieving a space use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/spaces/" Local-Id
REST-Query-Parameters  = ENCODE-REST-PARAMETERS(GetSpace-Request-Parameters)
REST-Request-Payload   = null
```

```
RPC-Method              = "spaces.get"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(GetSpace-Request-Parameters)

Return-Object           = Space
```

Get Space Request Parameters A request for a Space object MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
resourceId	Local-Id	Local id of a space to retrieve. Note that the resourceId parameter is not applicable in the REST protocol, as the space is identified in the REST-URI-FRAGMENT.
fields	Array<String>	An array of Space field names. For standard values, see the Space object.
escapeType	Escape-Type	Specifies the type of escaping to use on any AppData values included in the response. Defaults to "htmlEscape".

Examples Here's an example of a REST request to retrieve a space and the associated response.

HTTP Request

```
GET /spaces/666?fields=name,parentId,description HTTP/1.1
HOST api.example.org
Authorization: hh5s93j4hdidpola
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id"          : "api.example.org/spaces/666"
, "name"        : "English course"
, "parentId"    : "api.example.org/people/john.doe"
, "description" : "English learning course at EPFL"
}
```

Get a list of Spaces

Containers MAY support request to create a new space for a person or a new sub-space for an already existing space. Requests and responses for creating a relationship use the following values:

```
REST-HTTP-Method       = "POST"
```

```
REST-URI-Fragment      = "/spaces"
REST-Query-Parameters  = null
REST-Request-Payload   = Space

RPC-Method             = "spaces.create"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(CreateSpace-Request-Parameters)

Return-Object          = Space
```

Get Spaces Request Parameters A request for a Collection of Space objects MUST support the Standard-Request-Parameters, the Collection-Request-Parameters, and the following additional parameters:

Name	Type	Description
resourceId	Resource-Id or Array<Local-Id>	IRI of an object for which spaces are to be retrieved (Space, Person, ...) or an array of Local-Ids of spaces to retrieve.
fields	Array<String>	An array of Space field names. For standard values, see the Space object.

Examples Here's an example of an RPC request to retrieve the list of spaces for a person, and the associated response.

HTTP Request

```
POST /rpc HTTP/1.1
Host: api.example.org
Authorization: hh5s93j4hdidpola
Content-Type: application/json
{
  "method" : "spaces.get"
, "id"     : "mySpaces"
, "params" : {
    "resourceId" : "api.example.org/people/john.doe"
  , "fields"     : "name,description"
  }
}
```

HTTP Response

```
HTTP/1.x 207 Multi-Status
Content-Type: application/json
{
  "id"     : "mySpaces"
, "result" : {
    "startIndex"   : 1
  , "itemsPerPage" : 2
```

```
  , "totalResults" : 100
  , "entry"        : [
      {
        "id"          : "api.example.org/spaces/1"
      , "name"        : "English course"
      , "description" : "English learning course at EPFL"
      }
    , {
        "id"          : "api.example.org/spaces/2"
      , "name"        : "Movies watching"
      , "description" : "A space with my friends for organizing our
        movies watching evenings"
      }
    ]
  }
}
```

Retrieve a list of supported Space fields

Containers MAY support REST requests for supported space fields. Requests and responses for retrieving supported space fields use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/spaces/@supportedFields"
REST-Query-Parameters  = null
REST-Request-Payload   = null
Return-Object          = Array<String>
```

Create a Space

Containers MAY support request to create a new space for a person or a new sub-space for an already existing space. Requests and responses for creating a relationship use the following values:

```
REST-HTTP-Method       = "POST"
REST-URI-Fragment      = "/spaces"
REST-Query-Parameters  = null
REST-Request-Payload   = Space

RPC-Method             = "spaces.create"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(CreateSpace-Request-Parameters)

Return-Object          = Space
```

Create Space Request Parameters A request to create a Space MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
space	Space	An object to initialize fields of the space. Note that name and parentId fields are required, where parentId is an IRI of an object for which a space is to be created

Examples Here is an example of a request to create a space for a person using the RPC protocol:

```
POST /rpc HTTP/1.1
Host: api.example.org
Authorization: <auth token>
Content-Type: application/json
{
  "method" : "spaces.create"
, "id"     : "createSpace"
, "params"  : {
    "space" : {
      "name"     : "English course"
    , "parentId" : "api.example.org/people/666"
    }
  }
}
```

If successful, the associated response contains IRI id of the newly created space:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id"     : "createSpace"
, "result" : {
    "id"       : "api.example.org/spaces/1"
  , "name"     : "English course"
  , "parentId" : "api.example.org/people/666"
  }
}
```

Update a Space

Containers MAY support updating the properties of a Space object. If the request is successful, the container MUST return the updated Space object. Requests and responses for updating the fields of a Space use the following values:

```
REST-HTTP-Method       = "PUT"
REST-URI-Fragment      = "/spaces/" Local-Id
```

```
REST-Query-Parameters  = null
REST-Request-Payload   = Space

RPC-Method             = "spaces.update"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(UpdateSpace-Request-Parameters)

Return-Object          = Space
```

Update Space Request Parameters A request to update a space MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
resourceId	Local-Id	Local-Id of the space to be updated.
space	Space	A Space object containing the updated fields.

Delete a Space

Containers MAY support requests to remove a Space. Requests and responses to remove a Space use the following values:

```
REST-HTTP-Method       = "DELETE"
REST-URI-Fragment      = "/spaces/" Local-Id
REST-Query-Parameters  = null
REST-Request-Payload   = null

RPC-Method             = "spaces.delete"
RPC-Request-Parameters = "{"
                             <"> "resourceId" <"> ":" <"> Local-Id <">
                         "}"

Return-Object          = null
```

A.2.3 Apps

Containers MUST support the Apps Service. Individual operations are required or optional as indicated in the sections that follow. Containers MUST use the following values to define the Apps Service:

```
XRDS-Type    = "http://ns.opensocial.org/2008/opensocial/apps"
Service-Name = "apps"
```

Get

The Get method supports several different types of queries, from requests for an app or group of apps to information about what profile fields are available.

Get an App

Containers MUST support retrieving information about an application. Requests and responses for retrieving an application use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/apps/" Local-Id
REST-Query-Parameters  = ENCODE-REST-PARAMETERS(GetApp-Request-Parameters)
REST-Request-Payload   = null

RPC-Method             = "apps.get"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(GetApp-Request-Parameters)

Return-Object          = App
```

Get App Request Parameters A request for an App object MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
resourceId	Local-Id or @current	Local id of an app to retrieve or @current to retrive the currently open app. Note that the resourceId parameter is not applicable in the REST protocol, as the app is identified in the REST-URI-FRAGMENT.
fields	Array<String>	An array of App field names. For standard values, see the App object.
escapeType	Escape-Type	Specifies the type of escaping to use on any AppData values included in the response. Defaults to "htmlEscape".

Examples Here's an example of a REST request to retrieve an app and the associated response.

HTTP Request

```
1 GET /apps/5?fields=name,appUrl HTTP/1.1
2 HOST api.example.org
3 Authorization: hh5s93j4hdidpola
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id"     : "api.example.org/apps/5"
, "name"   : "Chat app"
, "appUrl" : "http://appstore.org/app.xml"
}
```

Get a list of Apps

Containers MUST support retrieving information about multiple apps in a single request. Requests and responses for retrieving a list of apps use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/apps/" Resource-Id
REST-Query-Parameters  = ENCODE-REST-PARAMETERS(GetApps-Request-Parameters)
REST-Request-Payload   = null

RPC-Method             = "apps.get"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(GetApps-Request-Parameters)

Return-Object          = Collection<App>
```

Get Apps Request Parameters A request for a Collection of App objects MUST support the Standard-Request-Parameters, the Collection-Request-Parameters, and the following additional parameters:

Name	Type	Description
resourceId	Resource-Id or Array<Local-Id>	IRI of an object for which apps are to be retrieved (Space, Person, ...) or an array of Local-Ids of apps to retrieve.
fields	Array<String>	An array of App field names. For standard values, see the App object.

Examples Here's an example of a REST request to retrieve the list of all apps for a person Jane Doe, and the associated response.

HTTP Request

```
GET /apps/api.example.org%2Fpeople%2Fjane.doe?fields=name,appUrl HTTP/1.1
HOST api.example.org
Authorization: hh5s93j4hdidpola
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "startIndex"   : 1
, "itemsPerPage" : 2
, "totalResults" : 100
, "entry"        : [
    {
      "id"      : "api.example.org/apps/1"
    , "name"    : "Chat app"
    , "appUrl" : "http://appstore.org/chat.xml"
    }
  , {
      "id"      : "api.example.org/apps/2"
    , "name"    : "Search app"
    , "appUrl" : "http://appstore.org/search.xml"
    }
  ]
}
```

Retrieve a list of supported App fields

Containers MAY support REST requests for supported app fields. Requests and responses for retrieving supported application fields use the following values:

```
REST-HTTP-Method       = "GET"
REST-URI-Fragment      = "/apps/@supportedFields"
REST-Query-Parameters  = null
REST-Request-Payload   = null
Return-Object          = Array<String>
```

Create an App

Containers MAY support request to create a new app for a person or for a space. Requests and responses for creating a relationship use the following values:

```
REST-HTTP-Method       = "POST"
REST-URI-Fragment      = "/apps"
REST-Query-Parameters  = null
REST-Request-Payload   = App

RPC-Method             = "apps.create"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(CreateApp-Request-Parameters)
```

```
Return-Object          = App
```

Create App Request Parameters A request to create an App MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
app	App	An object to initialize fields of the app. Note that name and parentId fields are required, where parentId is an IRI of an object for which an app is to be created

Examples Here is an example of a request to create an app for a person.

HTTP Request

```
POST /apps HTTP/1.1
HOST api.example.org
Authorization: hh5s93j4hdidpola
Content Type: application/json
{
  "parentId" : "api.example.org/people/john.doe"
, "name"     : "Search app"
, "appUrl"   : "http://appstore.org/chat.xml"
}
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "result" : {
    "id"       : "api.example.org/apps/2"
  , "name"     : "Search app"
  , "parentId" : "api.example.org/people/john.doe"
  , "appUrl"   : "http://appstore.org/chat.xml"
  }
}
```

Update an App

Containers MAY support updating the properties of an App object. If the request is successful, the container MUST return the updated App object. Requests and responses for updating the fields of an App use the following values:

```
REST-HTTP-Method       = "PUT"
REST-URI-Fragment      = "/apps/" Local-Id
```

```
REST-Query-Parameters  = null
REST-Request-Payload   = App

RPC-Method             = "apps.update"
RPC-Request-Parameters = ENCODE-RPC-PARAMETERS(UpdateApp-Request-Parameters)

Return-Object          = App
```

Update App Request Parameters A request to update an app MUST support the Standard-Request-Parameters and the following additional parameters:

Name	Type	Description
resourceId	Local-Id	Local-Id of the app to be updated.
app	App	An App object containing the updated fields.

Delete an App

Containers MAY support requests to remove an App. Requests and responses to remove an App use the following values:

```
REST-HTTP-Method       = "DELETE"
REST-URI-Fragment      = "/apps/" Local-Id
REST-Query-Parameters  = null
REST-Request-Payload   = null

RPC-Method             = "apps.delete"
RPC-Request-Parameters = "{"
                             <"> "resourceId" <"> ":" <"> Local-Id <">
                         "}"

Return-Object          = null
```

A.2.4 AppData

All OpenSocial REST APIs have to be unified to support the Space concept. We show how it can be done on the example of AppData REST API, which mainly includes the introduction of Resource-Id into the API. The other OpenSocial APIs (Groups, ActivityStreams, Albums, etc.) can be extended in a similar way.

Resource-Id

The Resource-Id is an Internationalized Resource Identifier (IRI) that globally and uniquely identifies an Object within a container (Space, Person, Activity, etc.).

```
Resource-Id  = Domain-Name "/" Service-Name "/" Local-Id
Service-Name = OpenSocial service as defined in the Social API Server
```

Example: mysocial.com/people/1234

Get AppData

Requests and responses to retrieve AppData use the following values:

```
REST-URI-Fragment = "/appdata/" Resource-Id "/" Group-Id [ "/" App-Id ]
```

A request to retrieve AppData MUST support the Standard-Request-Parameters, the Collection-Request-Parameters, and the following additional parameters:

Name	Type	Description
resourceId	Resource-Id or Array<Resource-Id>	IRI of object or objects whose AppData is to be retrieved.

Update AppData

Requests and responses to update AppData use the following values:

```
REST-URI-Fragment = "/appdata/" Resource-Id "/" "@self" [ "/" App-Id ]
```

A request to update AppData MUST support the Standard-Request-Parameters, and the following additional parameters:

Name	Type	Description
resourceId	Resource-Id	IRI of the object whose AppData is to be retrieved.

Delete AppData

Requests and responses to delete AppData use the following values:

```
REST-URI-Fragment = "/appdata/" Resource-Id "/" "@self" [ "/" App-Id ]
```

A request to delete AppData MUST support the Standard-Request-Parameters, and the following additional parameters:

Name	Type	Description
resourceId	Resource-Id	IRI of the object to which AppData belongs.

A.3 Appendix C: Space Extension RPC APIs

The Space Extension introduces several new RPC APIs (Space and App) into OpenSocial specification[3] and changes some other APIs to adapt them for work with spaces.

A.3.1 osapi

getContext

Signature	<static> osapi.getContext(callback)
Description	Gets the current context for this app. An app should be able to know whether it belongs to a space or to a person and their respective ids. This request returns info about the page where app is open (Space or Person).
Parameters	Callback function "callback" that receives the Context object as a response.

Examples

The osapi.getContext request when the app is open on the space with id 5:

```
1 osapi.getContext(function (context) {
2   if (!context.error) {
3     context.id       // "api.example.org/spaces/5"
4     context.service  // "spaces"
5     context.object   // A Space object
6   }
7 })
```

A.3.2 osapi.spaces

create

Signature	<static> osapi.Request osapi.spaces.create(params)
Description	Builds a request to create a space via the Spaces service.
Parameters	This method takes a single parameter, which is a JavaScript object representing the parameters defined by the Spaces service's create method.

[3] http://iamac71.epfl.ch/os_space/Social-Gadget.xml

Returns	A osapi.Request to create a space via the Spaces service. Executing this request MUST create a space and return the newly created space.

get

Signature	<static> osapi.Request osapi.spaces.get(params)
Description	Builds a request to get a space or spaces via the Spaces service.
Parameters	This method takes one parameter, which is a JavaScript object representing the parameters defined by the Spaces service's get method.
Returns	A osapi.Request to get a space or spaces via the Spaces service. Executing this request MUST return a Space or a Collection of Spaces.

update

Signature	<static> osapi.Request osapi.spaces.update(params)
Description	Builds a request to update a space via the Spaces service.
Parameters	This method takes a single parameter, which is a JavaScript object representing the parameters defined by the Spaces service's update method.
Returns	A osapi.Request to update a space via the Spaces service. Executing this request MUST update a space, but does not return any information.

delete

Signature	<static> osapi.Request osapi.spaces.delete(params)
Description	Builds a request to delete a space via the Spaces service.
Parameters	This method takes a single parameter, which is a JavaScript object representing the parameters defined by the Spaces service's delete method.
Returns	A osapi.Request to delete a space via the Spaces service. Executing this request MUST delete a space, but does not return any information.

Examples

A simple example to request a list of spaces that belong to the john.doe person:

```
1 osapi.spaces.get({ resourceId: "api.example.org/people/john.doe" })
2   .execute(function (list) {
3     list[0].displayName
4   })
```

A.3.3 osapi.apps

create

Signature	<static> osapi.Request osapi.apps.create(params)
Description	Builds a request to create a app via the Apps service.
Parameters	This method takes a single parameter, which is a JavaScript object representing the parameters defined by the Apps service's create method.
Returns	A osapi.Request to create a app via the Apps service. Executing this request MUST create a app and return the newly created app.

get

Signature	<static> osapi.Request osapi.apps.get(params)
Description	Builds a request to get a app or apps via the Apps service.
Parameters	This method takes one parameter, which is a JavaScript object representing the parameters defined by the Apps service's get method.
Returns	A osapi.Request to get an app or apps via the Apps service. Executing this request MUST return a App or a Collection of Apps.

getCurrent

Signature	<static> osapi.Request osapi.apps.getCurrent()
Description	A convenience over osapi.apps.get() that builds a request to retrieve the information about the currently running app, as specified in the security token, from the App service.
Parameters	None
Returns	A osapi.Request to get an app from the Apps service. Executing this request MUST return an App.

getParent

Signature	<static> osapi.Request osapi.apps.getParent()
Description	A convenience API that builds a request to retrieve the parent of the currently running app, as specified in the security token.
Parameters	None
Returns	A osapi.Request to get an app from the Apps service. Executing this request MUST return an App.

update

Signature	<static> osapi.Request osapi.apps.update(params)
Description	Builds a request to update a app via the Apps service.
Parameters	This method takes a single parameter, which is a JavaScript object representing the parameters defined by the Apps service's update method.
Returns	A osapi.Request to update a app via the Apps service. Executing this request MUST update a app, but does not return any information.

delete

Signature	<static> osapi.Request osapi.apps.delete(params)
Description	Builds a request to delete a app via the Apps service.
Parameters	This method takes a single parameter, which is a JavaScript object representing the parameters defined by the Apps service's delete method.
Returns	A osapi.Request to delete a app via the Apps service. Executing this request MUST delete a app, but does not return any information.

Examples

A simple example to request a list of apps that belong to the space with id 5:

```
1 osapi.apps.get({ resourceId: "api.example.org/spaces/5" })
2   .execute(function (list) {
3     list[0].displayName
4   })
```

An example to request the current app:

```
1 osapi.apps.getCurrent().execute(function (app) {
2   app.displayName
3   app.parentId    // an IRI of its parent
4 })
```

A.4 Appendix D: Space Extension Shindig Implementation

Due to the big size of the implementation code (around 5000 lines of code) we do not provide the details here. The patch for Apache Shindig that implements OpenSocial Space extension can be found in the section *Shindig patch with spaces* in the following URL:

`http://docs.opensocial.org/display/OSD/Space+Proposal`

A.5 Appendix E: GraaspIt! Plugin Example

Section 4.2.4 introduced the GraaspIt! concept. This Appendix provides an example of GraaspIt! plugin for the ROLE Widget Store.

Content providers interested to create a GraaspIt! plugin for their Web platforms can use the existing *pluginmock.js* file as a basis for the development of their own plugins. This file can be found on Github[4]. As an example, let us consider the plugin that adds to Graasp in one-click OpenSocial widgets available in the ROLE Widget store (Listing A.1). The plugin file is called *widgetstore.js* and the matching URL for the service is *role-widgetstore.eu/tool*. Thus, when the user runs plugin on the pages matching this URL, the plugin *widgetstore.js* is called. As the code snippet in Listing A.1 shows, the plugin implements three scraping functions *findName*, *findEmbedCode* and *findThumbnail*, that return respectively the name, the URL and the thumbnail URL of the OpenSocial widget.

```
/**
 * WidgetStore (widgetstore.js)
 *
 * matches URLs: "role-widgetstore.eu/tool"
 *
 */

...

/**
 * Returns the name of the OpenSocial widget
 */
findName: function () {
  return $("#content-content .secondColumn .title")
},

/**
 * Returns the thumbnail URL of the OpenSocial widget
 */
findThumbnail: function () {
  return $(".thumbnail-area img.imagecache-thumbnail").attr("src")
},

/**
 * Returns the OpenSocial widget URL
 */
findEmbedCode: function () {
  return $("#edit-source-2-wrapper input").val()
}
...
```

Listing A.1: The GraaspIt! plugin for Role Widget store

[4] https://github.com/react-epfl/graaspit

Bibliography

LUISA. Learning Content Management System Using Innovative Semantic Web Services Architecture. European FP6-IST Project, 2008. http://cordis.europa.eu/projects/rcn/79379_en.html.

J. Adamek and P. Hnetynka. Perspectives in Component-Based Software Engineering. In *Proceedings of the 2008 International Workshop on Software Engineering in East and South Europe*, SEESE '08, pages 35–42. ACM, 2008.

G. Adomavicius and A. Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. In *IEEE Transactions on Knowledge and Data Engineering*, volume 17, pages 734–749, Piscataway, NJ, USA, June 2005. IEEE Educational Activities Department.

M. Blattner, E. Glinert, J. Jorge, and G. Ormsby. Metawidgets: Towards a Theory of Multimodal Interface Design. In *Computer Software and Applications Conference, 1992. COMPSAC '92. Proceedings., Sixteenth Annual International*, pages 115–120, sep. 1992.

E. Bogdanov, C. Salzmann, S. El Helou, and D. Gillet. Social Software Modeling and Mashup based on Actors, Activities and Assets. In *3th European Conference on Technology Enhanced Learning (EC-TEL) - Workshop on Mash-Up Personal Learning Environments (MUPPLE-08)*, pages 42–47, 2008.

E. Bogdanov, S. El Helou, D. Gillet, C. Salzmann, and S. Sire. Graaasp: a Web 2.0 Research Platform for Contextual Recommendation with Aggregated Data. In *Proceedings of the 28th of the International Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '10)*, pages 3523–3528, 2010.

E. Bogdanov, C. Salzmann, and D. Gillet. Contextual Spaces with Functional Skins as OpenSocial Extension. In *4th International Conference on Advances in Computer-Human Interactions*, pages 158–163, 2011.

E. Bogdanov, F. Limpens, N. Li, S. El Helou, C. Salzmann, and D. Gillet. A Social Media Platform in Higher Education. In *Global Engineering Education Conference (EDUCON)*, pages 1–8. IEEE, 2012a.

E. Bogdanov, C. Salzmann, and D. Gillet. Widget-Based Approach for Remote Control Labs. In *9th IFAC Symposium on Advances in Control Education*, pages 189–193, 2012b.

E. Bogdanov, C. Ullrich, E. Isaksson, M. Palmer, and D. Gillet. From LMS to PLE: A Step Forward through OpenSocial Apps in Moodle. In *Advances in Web-Based Learning - ICWL 2012*, volume 7558 of *Lecture Notes in Computer Science*, pages 69–78. Springer Berlin Heidelberg, 2012c.

E. Bogdanov, C. Ullrich, E. Isaksson, M. Palmer, and D. Gillet. Towards PLEs through Widget Spaces in Moodle. In *Computer Science and Information Systems Journal (ComSIS) - submitted*, 2013.

B. Charlier, F. Henri, D. Peraya, and D. Gillet. From Personal Environment to Personal Learning Environment. In *Fifth European Conference on Technology Enhanced Learning (EC-TEL10), Workshop on Mash-Up Personal Learning Environments (MUPPLE10)*, Barcelona, Spain, 2010.

P. Clemente and J. Hernández. Aspect Component Based Software Engineering. In *The Second AOSD Workshop on Aspects Components and Patterns for Infrastructure Software ACP4IS*, pages 39–42, 2003.

M. A. Conde, D. A. G. Aguilar, A. Pozo de Dios, and F. J. G. Penalvo. Moodle 2.0 Web Services Layer and Its New Application Contexts. In *Technology Enhanced Learning. Quality of Teaching and Educational Reform*, Communications in Computer and Information Science, pages 110–116. Springer Berlin Heidelberg, 2010.

D. Crane and P. McCarthy. Comet and Reverse Ajax: The Next Generation Ajax 2.0. Book. In *Apress. ISBN 1590599985*, 2008.

L. Crane, A. Awe, P. Benachour, and P. Coulton. Context Aware Electronic Updates for Virtual Learning Environments. In *International Conference on Advanced Learning Technologies ICALT*, pages 173–175, 2012.

N. Dabbagh and A. Kitsantas. Personal Learning Environments, Social Media, and Self-Regulated Learning: A Natural Formula for Connecting Formal and Informal Learning. In *The Internet and Higher Education*, volume 15, pages 3–8, 2012.

I. Dahn. Meta-Widgets and Space Tools – Comparison of Two Integration Concepts in Technology Enhanced Learning. In *Unpublished paper*.

F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul. Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. In *IEEE Internet Computing*, volume 11, pages 59–66. IEEE Computer Society, 2007.

A. K. Dey, G. D. Abowd, and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. In *Human-Computer Interaction*, volume 16, pages 97–166, Hillsdale, NJ, USA, 2001. L. Erlbaum Associates Inc.

S. Dietze, A. Gughotta, and J. Domingue. Context-aware Process Support through Automatic Selection and Invocation of Semantic Web Services. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, pages 199–206, 2007.

P. Dourish and V. Bellotti. Awareness and Coordination in Shared Workspaces. In *Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work CSCW 92*, volume 3, pages 107–114. ACM Press, 1992.

S. Downes. Feature: e-learning 2.0, 2005. URL http://elearnmag.org/subpage.cfm?section=articles&article=29-1.

I. Dror and S. Harnad. Offloading Cognition onto Cognitive Technology. In *Cognition Distributed: How Cognitive Technology Extends our Minds*, pages 1–23, 2008.

A. Eck and L. Soh. Intelligent User Interfaces with Adaptive Knowledge Assistants. In *Technical Report TR-UNL-CSE-2009-2011*, pages 1–19. Department of Computer Science and Engineering, University of Nebraska – Lincoln, 2009.

S. El Helou, M. Tzagarakis, D. Gillet, N. Karacapilidis, and C. Yu Man. Participatory Design for Awareness Features: Enhancing Interaction in Communities of Practice. In *Proceedings of the 6th International Conference on Networked Learning*, pages 523–530, 2008.

Y. Engeström, R. Miettinen, and R.-L. Punamäki. Perspectives on Activity Theory. Book. Cambridge University Press, 1999.

F. Enoksson, M. Palmer, A. Naeve, S. Arroyo, D. Fuschi, and T. Pariente. LUISA. Learning Content Management System Using Innovative Semantic Web Services Architecture. In *Deliverable 3.1*, 2006. http://www.luisa-project.eu/dvd/docs/Public_deliverables/M6/WP3_D3.1_StateOfTheArt_v1.0_final.pdf.

J. R. Erenkrantz, M. Gorlick, G. Suryanarayana, and R. N. Taylor. From Representations to Computations: the Evolution of Web Architectures. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 255–264, New York, NY, USA, 2007. ACM.

M. A. Forment, M. J. C. n. Guerrero, M. A. C. González, F. J. G. Peñalvo, and C. Severance. Interoperability for LMS: The Missing Piece to Become the Common Place for Elearning Innovation. In *Visioning and Engineering the Knowledge Society*, volume 5736 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 2009.

K. Fruhmann, A. Nussbaumer, and D. Albert. A Psycho-Pedagogical Framework for Self-Regulated Learning in a Responsive Open Learning Environment. In *International Conference eLearning Baltics Science (eLBa Science 2010)*, pages 1–2, 2010.

D. Gillet and E. Bogdanov. Personal Learning Environments and Embedded Contextual Spaces as Aggregator of Cloud Resources. In *Proceedings of the 1st International Workshop on Cloud Education Environments (WCLOUD)*, pages 38–40, 2012.

D. Gillet and G. Fakas. eMersion: A New Paradigm for Web-Based Training in Mechanical Engineering Education. In *International Conference on Engineering Education*, volume 8, pages 10–14, 2001.

D. Gillet, A. V. N. Nguyen Ngoc, and Y. Rekik. Collaborative Web-Based Experimentation in Flexible Engineering Education. In *IEEE Transactions on Education*, volume 48, pages 696–704, 2005.

D. Gillet, S. E. Helou, Y. Rekik, and C. Salzmann. Context-Sensitive Awareness Services for Communities of Practice. In *12th International Conference on Human-Computer Interaction, HCI 2007, Beijing*, volume 0000 of *LNCS*, pages 22–27. ACM Press, 2007a.

D. Gillet, C. M. Yu, and S. El Helou. Turning Web 2.0 Social Software into Collaborative Learning and Knowledge Management Solutions. In *ERCIM News*, pages 35–36, 2007b.

D. Gillet, S. E. Helou, C. M. Yu, and C. Salzmann. Turning Web 2.0 Social Software into Versatile Collaborative Learning Solutions. In *First International Conference on Advances in Computer Human Interaction*, pages 170–176. IEEE, 2008.

J. Gonzalez-Tato, M. Llamas-Nistal, M. Caeiro-Rodriguez, and J. Alvarez-Osuna. Towards a Collection of Gadgets for an iGoogle e-Learning Platform. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–9, 2012.

C. A. Halverson. Activity Theory and Distributed Cognition: Or What Does CSCW Need to DO with Theories? In *Computer Supported Cooperative Work*, volume 11, pages 243–267. Springer, 2002.

T. Health and C. Bizer. Linked Data: Evolving the Web into a Global Data Space. Book. URL http://linkeddatabook.com/editions/1.0/.

S. E. Helou, N. Li, and D. Gillet. The 3A Interaction Model: Towards Bridging the Gap between Formal and Informal Learning. In *Third International Conference on Advances in Computer-Human Interactions. ACHI '10*, pages 179–184, 2010.

F. Henri, B. Charlier, and F. Limpens. Understanding PLE as an Essential Component of the Learning Process. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*, pages 3766–3770, Vienna, Austria, June 2008. AACE.

V. Hoyer and M. Fischer. Market Overview of Enterprise Mashup Tools. In *Service-Oriented Computing – ICSOC 2008*, volume 5364 of *Lecture Notes in Computer Science*, pages 708–721. Springer Berlin Heidelberg, 2008.

E. Hutchins. How a Cockpit Remembers Its Speeds. In *Cognitive Science*, volume 19, pages 265–288. Elsevier, 1995.

E. Isaksson and M. Palmér. Usability and Inter-widget Communication in PLEs. In *Fifth European Conference on Technology Enhanced Learning (EC-TEL10), The 3rd Workshop on Mash-Up Personal Learning Environments (MUPPLE10)*, 2010.

Z. Jeremic, J. Jovanovic, and D. Gasevic. Personal Learning Environments on the Social Semantic Web. In *Semantic Web Journal*, volume 4, pages 23–51, 2011.

D. H. Jonassen and L. Rohrer-Murphy. Activity Theory as a Framework for Designing Constructivist Learning Environments. In *Educational Technology Research & Development*, volume 47, pages 61–79. Springer, 1999.

D. Kearney, D. O'Hare, G. McClure, M. McKee, S. Higgins, and T. Wilshart. Northern Ireland Integrated Managed Learning Environment (NIIMLE). In *Final Report of the NIIMLE project*, 2005. http://www.elearning.ac.uk/mle/learner_recs/niimle/NIimle.pdf.

W. Kozaczynski and G. Booch. Component-Based Software Engineering. In *IEEE Software*, volume 15, pages 34–36. Institute of Electrical and Electronics Engineers, Inc, 445 Hoes Ln, Piscataway, NJ, 08854-1331, USA„ 1998.

S. Krug. Don't Make Me Think! Book. In *A Common Sense Approach to Web Usability*, 2000.

M. Kurze. Personalization in Multimodal Interfaces. In *Proceedings of the 2007 Workshop on Tagging Mining and Retrieval of Human Related Activity Information TMR 07*, pages 23–26. ACM Press, 2007.

N. Laga, E. Bertin, and N. Crespi. Building a User Friendly Service Dashboard: Automatic and Non-intrusive Chaining between Widgets. In *Proceedings of the 2009 Congress on Services*, pages 484–491. IEEE, 2009.

B. Latour. On Actor-Network Theory: a few Clarifications. In *Soziale welt*, pages 369–381. JSTOR, 1996.

B. Latour. Reassembling the Social – an Introduction to Actor-Network-Theory. page 316. Oxford University Press, 2005.

J. Law and J. Hassard. Actor Network Theory and After. Book. Blackwell Publishers, 1999.

A. Leontyev. Activity and Consciousness. Book. In *Philosophy in the USSR, Problems of Dialectical Materialism*. Progress Publishers, 1977.

N. Li, C. Ullrich, S. El Helou, and D. Gillet. Using Social Software for Teamwork and Collaborative Project Management in Higher Education. In *Lecture Notes in Computer Science (LNCS)*, volume 6483 of *Lecture Notes in Computer Science*, pages 161–170. Springer, 2010.

N. Li, S. Ingram-El Helou, and D. Gillet. Using Social Media for Collaborative Learning in Higher Education: A Case Study. In *Proceedings of the 5th International Conference on Advances in Computer-Human Interactions*, pages 285–290, 2012.

O. Liber. Colloquia – a Conversation Manager. In *Campus-Wide Information Systems*, volume 17, pages 56–62. Emerald Group Publishing Limited, 2000.

F. Limpens and D. Gillet. A Competence Bartering Platform for Learners. In *Advances in Web-Based Learning-ICWL 2011*, pages 148–153. Springer, 2011.

F. Limpens and D. Gillet. Towards Agile Competence Development. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 667–672. IEEE, 2012.

X. Liu, Y. Hui, W. Sun, and H. Liang. Towards Service Composition Based on Mashup. In *IEEE Congress on Services*, pages 332–339. IEEE Computer Society, 2007.

R. Lopes, H. Akkan, W. Claycomb, and D. Shin. An OpenSocial Extension for Enabling User-controlled Persona in Online Social Networks. In *4th International Workshop on Trusted Collaboration (TrustCol 09)*, pages 1–5. IEEE, 2009.

Z. Maraikar and A. Lazovik. Reforming mashups. In *The 3rd European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2008)*, 2008.

C. McGloughlin and M. J. W. Lee. Personalised and Self Regulated Learning in the Web 2.0 Era: International Exemplars of Innovative Pedagogy Using Social Software. In *Australasian Journal of Educational Technology*, volume 26, pages 28–43. ME Sharpe, 2010.

N. Mikhaylov and D. Chernov. From Virtual Lab to Virtual Development Lab. In *9th IFAC Symposium on Advances in Control Education*, pages 177–182, 2012.

L. Moccozet, O. Benkacem, B. Ndiaye, V. Ahmeti, P. Roth, and P.-Y. Burgi. An Exploratory Study for the Implementation of a Techno-pedagogical Personal Learning Environment. In *Proceedings of the PLE Conference*, 2011.

F. Moedritscher, G. Neumann, V. M. Garcia-Barrios, and F. Wild. A Web Application Mashup Approach for eLearning. In *Proceedings of the OpenACS and LRN Conference*, pages 105–110, 2008.

B. A. Nardi. Context and Consciousness: Activity Theory and Human-Computer Interaction. Book. The MIT Press, 1996.

H. S. Nwana. Software Agents: An Overview. In *Knowledge Engineering Review*, volume 11, pages 205–244, 1996.

M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl. PolyLens: A Recommender System for Groups of Users. In *Proceedings of the European Conference on Computer Supported Cooperative Work*, pages 199–218. Kluwer Academic Publishers, 2001.

M. Palmer, S. Sire, E. Bogdanov, D. Gillet, and F. Wild. Mapping Web Personal Learning Environments. In *4th European Conference on Technology Enhanced Learning (EC-TEL) - 2nd Workshop on Mash-Up Personal Learning Environments (MUPPLE-09)*, pages 31–46, 2009.

M. Perry. Distributed Cognition. In *HCI, Models, Theories, and Frameworks*, pages 193–224. London: Morgan Kaufman, 2003.

S. Pietschmann, V. Tietz, J. Reimann, C. Liebing, M. Pohle, and K. Meissner. A Metamodel for Context-aware Component-based Mashup Applications. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services*, pages 413–420. ACM, 2010.

M. Prensky. Digital Natives, Digital Immigrants Part 1. In *On the Horizon*, volume 9, pages 1–6. MCB UP Ltd, 2001a.

M. Prensky. Do They Really Think Differently? In *On the Horizon*, volume 9, pages 1–9. Colorado State Library, Library Research Service, 2001b.

W. Reinhardt, C. Mletzko, H. Drachsler, and P. Sloep. AWESOME: A Widget-based Dashboard for Awareness-support in Research Networks. In *Proceedings of The PLE Conference*, pages 1–15, 2011.

Y. Rekik, D. Gillet, S. El Helou, and C. Salzmann. The eLogBook Framework: Sustaining Interaction, Collaboration, and Learning in Laboratory-Oriented CoPs. In *International Journal of Web-based Learning and Teaching Technologies*, volume 2, pages 61–76. IGI Publishing, 2007.

D. Renzel, C. Hoebelt, D. Dahrendorf, M. Friedrich, F. Moedritscher, K. Verbert, S. Govaerts, M. Palmer, and E. Bogdanov. Collaborative Development of a PLE for Language Learning. In *International Journal of Emerging Technologies in Learning (iJET)*, volume 5, pages 31–40, 2010.

E. Rodríguez, M.-a. Sicilia, and S. Arroyo. Bridging the Semantic Gap in Standards-based Learning Object Repositories. In *Proceedings of the Workshop on Learning Object Repositories as Digital Libraries: Current challenges*, pages 478–483, 2006.

N. Rubin. Creating a User-centric Learning Environment with Campus Pack Personal Learning Spaces. In *PLS Webinar, Learning Objects Community*, 2010. URL http://community.learningobjects.com/Users/Nancy.Rubin/Creating_a_User-Centric_Learning.

M. Sabbouh, J. Higginson, S. Semy, and D. Gagne. Web Mashup Scripting Language. In *Proceedings of the 16th international conference on World Wide Web WWW 07*, pages 1305–1306. ACM Press, 2007.

G. Salomon. Distributed Cognitions: Psychological and Educational Considerations. Book. Cambridge University Press, 1997.

C. Salzmann and D. Gillet. Challenges in Remote Laboratory Sustainability. In *Proceedings of the International Conference on Engineering Education (ICEE)*, 2007.

C. Salzmann and D. Gillet. From Online Experiments to Smart Devices. In *International Journal of Online Engineering (iJOE), REV 2008 (5th Remote Engineering and Virtual Instrumentation)*, volume 4, pages 50–54, 2008.

C. Salzmann, D. Gillet, P. Scott, and K. Quick. Remote Lab: Online Support and Awareness Analysis. In *IFAC World Congress*, pages 8135–8140. The International Federation of Automatic Control, 2008.

C. Salzmann, D. Gillet, F. Esquembre, H. Vargas, J. Sánchez, and D. Sebastián. Web 2.0 Open Remote and Virtual Laboratories in Engineering Education. In *Collaborative Learning 2.0: Open Educational Resources*, pages 369–390. IGI Global, 2013.

J. L. Santos, K. Verbert, S. Govaerts, and E. Duval. Visualizing PLE Usage. In *Proceedings of EFEPLE11 1st Workshop on Exploring the Fitness and Evolvability of Personal Learning Environments*, pages 34–38. CEUR workshop proceedings, 2011.

C. Severance, J. Hardin, and A. Whyte. The Coming Functionality Mash-up in Personal Learning Environments. In *Interactive Learning Environments*, volume 16, pages 47–62. Routledge, 2008.

S. Sire and C. Vanoirbeek. Small Data in the large with Oppidum. In *XML London (submitted), from June-15th to June-16th, London, UK*, 2013.

S. Sire, E. Bogdanov, M. Palmer, and D. Gillet. Towards Collaborative Portable Web Spaces. In *4th European Conference on Technology Enhanced Learning (EC-TEL) - 2nd Workshop on Mash-Up Personal Learning Environments (MUPPLE-09)*, pages 59–71, 2009.

S. Sire, E. Bogdanov, D. Gillet, F. Wild, and M. Palmer. Introducing Qualitative Dimensions to Analyze the Usefulness of Web 2.0 Platforms as PLEs. In *International Journal of Technology Enhanced Learning (IJTEL)*, pages 40–60. Inderscience, 2010.

A. Soller, F. Linton, B. Goodman, and A. Lesgold. Toward Intelligent Analysis and Support of Collaborative Learning Interaction. In *Proceedings of the Ninth International Conference on Artificial Intelligence in Education*, pages 75–82. IOS Press, 1999.

A. Soylu, F. Mödritscher, F. Wild, P. De Causmaecker, and P. Desmet. Mashups by Orchestration and Widget-based Personal Environments: Key Challenges, Solution Strategies, and an Application. In *Electronic Library and Information Systems*, volume 46, pages 383–428. Emerald Group Publishing Limited, 2012.

A. Tolk. What Comes After the Semantic Web - PADS Implications for the Dynamic Web. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, page 55. IEEE Computer Society, 2006.

R. Tuchinda, P. Szekely, and C. A. Knoblock. Building Mashups by Example. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI 08)*, pages 139–148. ACM Press, 2008.

C. Turnitsa. Extending the Levels of Conceptual Interoperability Model. In *Proceedings of the IEEE Summer Computer Simulation Conference*, 2005.

C. Ullrich, R. Shen, and D. Gillet. Not Yet Ready for Everyone: An Experience Report about a Personal Learning Environment for Language Learning. In *Advances in Web-Based Learning–ICWL 2010*, pages 269–278. Springer, 2010.

M. Van Harmelen. Personal Learning Environments. In *Sixth IEEE International Conference on Advanced Learning Technologies ICALT06*, volume 16, pages 815–816. IEEE, 2006.

M. Van Harmelen. Design Trajectories: Four Experiments in PLE Implementation. In *Interactive Learning Environments*, volume 16, pages 35–46. Routledge, 2008.

S. Vinoski. REST Eye for the SOA Guy. In *IEEE Internet Computing*, volume 11, pages 82–84. IEEE Computer Society, 2007.

F. Wild, F. Mödritscher, and S. Sigurdarson. Designing for Change : Mash-Up Personal Learning Environments. In *eLearning Papers*, volume 9, pages 1–15, 2008.

F. Wild, T. Ullmann, P. Scott, T. Rebedea, and B. Hoisl. Applicability of the Technology Acceptance Model for Widget-based Personal Learning Environments. In *Proceedings of the 1st Workshop on Exploring Fitness and Evolvability of Personal Learning Environments*, pages 39–48, 2011.

S. Wilson. Design Challenges for User-interface Mashups: User Control and Usability in Inter-widget Communications. http://scottbw.wordpress.com/2012/03/07/design-challenges-for-user-interface-mashups-user-control-and-usability-in-inter-widget-communications. Blog paper. 2012.

S. Wilson, O. Liber, M. Johnson, P. Beauvoir, P. Sharples, and C. Milligan. Personal Learning Environments: Challenging the Dominant Design of Educational Systems. In *Journal of eLearning and Knowledge Society*, volume 2, pages 173–182. Citeseer, 2007.

M. Wolpers, J. Najjar, K. Verbert, and E. Duval. Tracking Actual Usage: the Attention Metadata Approach. In *International Journal Educational Technology and Society 11*, pages 1176–3647. Press, 2007.

T. Yanagida, H. Nonaka, and M. Kurihara. Personalizing Graphical User Interfaces on Flexible Widget Layout. In *Science And Technology*, pages 255–264. ACM Press, 2009.

L. Zhou, S. El Helou, L. Moccozet, L. Opprecht, O. Benkacem, C. Salzmann, and D. Gillet. A Federated Recommender System for Online Learning Environments. In *Advances in Web-Based Learning–ICWL 2012*, pages 89–98. Springer, 2012.

I. Zuzak, M. Ivankovic, and I. Budiselic. A Classification Framework for Web Browser Cross-Context Communication. In *CoRR*, 2011. URL http://arxiv.org/pdf/1108.4770v1.pdf.

Evgeny BOGDANOV

Chemin de la Forêt, 5A
Ecublens, 1024, Switzerland

E-mail: evgeny.bogdanov@gmail.com
Phone: +41 (0) 79 4545 930

http://people.epfl.ch/evgeny.bogdanov

Key Strengths

PhD at EPFL
Experienced Web developer
Self motivated and competitive
Fluent in English, French, German, and Russian

EXPERIENCES

Research European Projects

2013 – 2014 – *GO-LAB* - EU project
In all EU projects, I participate in research investigations, implement Web-based prototypes and test them with people. We write deliverables for the projects collaboratively with other partners.

2009 – 2013 – *Responsive Open Learning Environments (ROLE)* - EU project

2008 – 2008 – *PALETTE* - EU project

Development Projects

2009 – Now – *Graasp* - Social media platform for education
Lead developer: I did major improvements of platform usability and performance, I investigate new technologies and bring them into the platform, I conduct user studies and participate in promoting the platform.

2008 – Now – *Sportaxy* - Web platform for athletes, coaches and sport organizations
Co-founder: I came up with the idea and built the Web platform from the scratch, I participate in company management and promoting

2010 – Now – I contribute to open-source projects: *OpenSocial* and *Apache Shindig*

EDUCATION

2008 – 13 – Ph.D. in Computer Science - Widgets and Spaces: Personal & Contextual Portability and Plasticity with OpenSocial
Ecole Polytechnique Fédérale de Lausanne (EPFL)

2005 – 07 – M.Sc., Applied Mathematics and Physics (major in System Programming)
Moscow Institute of Physics and Technology (State University)

EDUCATION (continued)

2001 – 05 – B.Sc., Applied Mathematics and Physics (major in System Programming)
Moscow Institute of Physics and Technology (State University)

COMPETENCIES

Computer Skills

Strengths
- Web development project management
- User Interface designing and building
- Client-Server programming unification
- Asynchronous programming
- Test-driven development
- Performance optimization

General skills
- Development frameworks: *Node.JS, Ruby on Rails*
- Server administration: *Linux, MacOS, NGINX, Apache, HAproxy*
- Tools: *GIT, SVN, LaTeX, VIM, Shell scripts, MySQL, etc.*
- Libs: *JQuery, Backbone.JS, Express.JS, Prototype, Jade, Stylus*

Programming
- Expert: *JavaScript, HTML5, CSS3, Ruby*
- Good knowledge: *Java, C/C++, Bash, PHP*
- General knowledge: *Go, Python, Objective-C*

Language Skills

English – Fluent
French – Fluent
German – Advanced
Swiss German – Beginner
Russian – Native

INTERESTS

Programming
- Learn new technologies and programming techniques
- Social coding at github: *vohtaski*
- OpenSource contributions

Sports
- SwissLoppet Cup winner, 2012
- 37th at international Engadin ski marathon
- 8th at Rollerski World Cup, 2012
- International Czech university champion in nordic skiing, 2011

INTERESTS (continued)

– Several times Swiss University champion in cross-country ski
– MIPT champion in boxing 72-77kg, 2004
– Cross country ski, cycling, figure skating, beach volley, dancing

Blogs

– *Software for Web*
– *Ski de fond a Lausanne*
– *Sportaxy blog*

PUBLICATIONS

(1) E. Bogdanov, S. El Helou, D. Gillet, C. Salzmann, and S. Sire. Graaasp: a Web 2.0 Research Platform for Contextual Recommendation with Aggregated Data. In *Proceedings of the 28th of the International Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '10)*, pages 3523–3528, 2010.

(2) E. Bogdanov, F. Limpens, N. Li, S. El Helou, C. Salzmann, and D. Gillet. A Social Media Platform in Higher Education. In *Global Engineering Education Conference (EDUCON)*, pages 1–8. IEEE, 2012.

(3) E. Bogdanov, C. Salzmann, S. El Helou, and D. Gillet. Social Software Modeling and Mashup based on Actors, Activities and Assets. In *3th European Conference on Technology Enhanced Learning (EC-TEL) - Workshop on Mash-Up Personal Learning Environments (MUPPLE-08)*, pages 42–47, 2008.

(4) E. Bogdanov, C. Salzmann, and D. Gillet. Contextual Spaces with Functional Skins as OpenSocial Extension. In *4th International Conference on Advances in Computer-Human Interactions*, pages 158–163, 2011.

(5) E. Bogdanov, C. Salzmann, and D. Gillet. Widget-Based Approach for Remote Control Labs. In *9th IFAC Symposium on Advances in Control Education*, pages 189–193, 2012.

(6) E. Bogdanov, C. Ullrich, E. Isaksson, M. Palmer, and D. Gillet. From LMS to PLE: A Step Forward through OpenSocial Apps in Moodle. In *Advances in Web-Based Learning - ICWL 2012*, volume 7558 of *Lecture Notes in Computer Science*, pages 69–78. Springer Berlin Heidelberg, 2012.

(7) E. Bogdanov, C. Ullrich, E. Isaksson, M. Palmer, and D. Gillet. Towards PLEs through Widget Spaces in Moodle. In *Computer Science and Information Systems Journal (ComSIS) - submitted*, 2013.

(8) D. Gillet and E. Bogdanov. Personal Learning Environments and Embedded Contextual Spaces as Aggregator of Cloud Resources. In *Proceedings of the 1st International Workshop on Cloud Education Environments (WCLOUD)*, pages 38–40, 2012.

(9) M. Palmer, S. Sire, E. Bogdanov, D. Gillet, and F. Wild. Mapping Web Personal Learning Environments. In *4th European Conference on Technology Enhanced Learning (EC-TEL) - 2nd Workshop on Mash-Up Personal Learning Environments (MUPPLE-09)*, pages 31–46, 2009.

(10) D. Renzel, C. Hoebelt, D. Dahrendorf, M. Friedrich, F. Moedritscher, K. Verbert, S. Govaerts, M. Palmer, and E. Bogdanov. Collaborative Development of a PLE for Language Learning. In *International Journal of Emerging Technologies in Learning (iJET)*, volume 5, pages 31–40, 2010.

(11) S. Sire, E. Bogdanov, D. Gillet, F. Wild, and M. Palmer. Introducing Qualitative Dimensions to Analyze the Usefulness of Web 2.0 Platforms as PLEs. In *International Journal of Technology Enhanced Learning (IJTEL)*, pages 40–60. Inderscience, 2010.

(12) S. Sire, E. Bogdanov, M. Palmer, and D. Gillet. Towards Collaborative Portable Web Spaces. In *4th European Conference on Technology Enhanced Learning (EC-TEL) - 2nd Workshop on Mash-Up Personal Learning Environments (MUPPLE-09)*, pages 59–71, 2009.