



QLectives – Socially Intelligent Systems for Quality
Project no. 231200

Instrument: Large-scale integrating project (IP)
Programme: FP7-ICT

Deliverable D.4.2.3

QScience v3

Submission date: 2012-01-16

Start date of project: 2009-03-01

Duration: 48 months

Organisation name of lead contractor for this deliverable:
University of Fribourg

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document information

1.1 Author(s)

Author	Organisation	E-mail
Matúš Medo	University of Fribourg	matus.medo@unifr.ch

1.2 Other contributors

Name	Organisation	E-mail
Stefano Balietti	ETH Zurich	stefano.balietti@gess.ethz.ch
Kornél Csernai	University of Szeged	csko@inf.u-szeged.hu
Yanbo Zhou	University of Fribourg	nbboob@gmail.com
Christian Schulz	ETH Zurich	cschulz@student.ethz.ch
Christoph Schwirzer	ETH Zurich	cschwirz@student.ethz.ch
Vahe Tshitoyan	ETH Zurich	tvahе@student.ethz.ch
Sasa Tomic	ETH Zurich	satomic@student.ethz.ch
Vahan Hovhannisyan	ETH Zurich	vahanh@student.ethz.ch
Liao Hao	University of Fribourg	jamesliao520@gmail.com

1.3 Document history

Version#	Date	Change
V0.1	9 January, 2012	Starting version, template
V0.2	16 January, 2012	First draft completed
V0.3	20 January, 2012	Corrected version

1.4 Document data

Keywords	QScience, Drupal, Web 2.0
Editor address data	matus.medo@unifr.ch
Delivery date	???, 2012

1.5 Distribution list

Date	Issue	E-mail
	Consortium members	QLECTIVES@list.surrey.ac.uk
	Project officer	Jose.FERNANDEZ-VILLACANAS@ec.europa.eu
	EC archive	INFSO-ICT-231200@ec.europa.eu

QLectives Consortium

This document is part of a research project funded by the ICT Programme of the Commission of the European Communities as grant number ICT-2009-231200.

University of Surrey (Coordinator)

Department of Sociology/Centre
for Research in Social Simulation
Guildford GU2 7XH
Surrey
United Kingdom
Contact person: Prof. Nigel Gilbert
E-mail: n.gilbert@surrey.ac.uk

Technical University of Delft

Department of Software Technology
Delft, 2628 CN
Netherlands
Contact Person: Dr Johan Pouwelse
E-mail: j.a.pouwelse@tudelft.nl

ETH Zurich

Chair of Sociology, in particular
Modelling and Simulation
Zurich, CH-8092
Switzerland
Contact person: Prof. Dirk Helbing
E-mail: dhelbing@ethz.ch

University of Szeged

MTA-SZTE Research Group on
Artificial Intelligence
Szeged 6720, Hungary
Contact person: Dr Mark Jelasity
E-mail: jelasity@inf.u-szeged.hu

University of Fribourg

Department of Physics
Fribourg 1700
Switzerland
Contact person: Prof. Yi-Cheng Zhang
E-mail: yi-cheng.zhang@unifr.ch

University of Warsaw

Faculty of Psychology
Warsaw 00927
Poland
Contact Person: Prof. Andrzej Nowak
E-mail: nowak@fau.edu

Centre National de la Recherche Scientifique, CNRS

Paris 75006,
France
Contact person: Dr. Camille ROTH
E-mail: camille.roth@polytechnique.edu

Institut für Rundfunktechnik GmbH

Munich 80939
Germany
Contact person: Dr. Christoph Dosch
E-mail: dosch@irt.de

QLectives introduction

QLectives is a project bringing together top social modelers, peer-to-peer engineers and physicists to design and deploy next generation self-organising socially intelligent information systems. The project aims to combine three recent trends within information systems:

- **Social networks** - in which people link to others over the Internet to gain value and facilitate collaboration
- **Peer production** - in which people collectively produce informational products and experiences without traditional hierarchies or market incentives
- **Peer-to-Peer systems** - in which software clients running on user machines distribute media and other information without a central server or administrative control

QLectives aims to bring these together to form Quality Collectives, i.e. functional decentralised communities that self-organise and self-maintain for the benefit of the people who comprise them. We aim to generate theory at the social level, design algorithms and deploy prototypes targeted towards two application domains:

- **QMedia** - an interactive peer-to-peer media distribution system (including live streaming), providing fully distributed social filtering and recommendation for quality
- **QScience** - a distributed platform for scientists allowing them to locate or form new communities and quality reviewing mechanisms, which are transparent and promote

The approach of the QLectives project is unique in that it brings together a highly inter-disciplinary team applied to specific real world problems. The project applies a scientific approach to research by formulating theories, applying them to real systems and then performing detailed measurements of system and user behaviour to validate or modify our theories if necessary. The two applications will be based on two existing user communities comprising several thousand people - so-called "Living labs", media sharing community tribler.org; and the scientific collaboration forum EconoPhysics.

Executive summary

During the third year of the QLectives project, the development of QScience has accelerated rapidly. As justified in the previous deliverable D4.2.2, we continue to develop QScience based on the open-source content management system (CMS) Drupal. This solution allows us to use many already prepared modules of this CMS which we describe at the beginning of Chapter 4 and a document linked therein.

In the given period, we were most occupied by work on the fundamental infrastructure required for the proposed evolutionary framework (which was also described in D4.2.2). In particular, to fully enable the evolution of QScience instances, we work intensively on a module of the open-source content management system Drupal (this module is called *Patterns*) which will be of benefit for the whole Drupal community (see Section 3 for details). We have not only ported this module from a previous version of Drupal but we have also added some new features. The Patterns module is available at <http://drupal.org/project/patterns>. Once finished, it will allow to automate the site setup and configuration process which is a very time consuming and repetitive bottleneck. The benefits for QScience will be striking—once the Patterns module is finished, QScience instances can be created and customized in “one click”.

Apart from that, we started work on additional functions that will help make QScience better and easier to use. Among them, interconnection of QScience instances using Drupal-to-Drupal communication is crucial and perhaps the most demanding one. Less technically demanding but still difficult to implement well are various visualisation tools that should help to make QScience useful by being able to present abstract activity,

citation and affiliation data using an attractive and informative interface.

Since the increased number of developers made our internal communication more difficult, in addition to the QScience developers wiki, we now have a tradition of (bi)weekly Skype chats where problems are discussed and new tasks are distributed. We also gathered in person for two weeks in Zürich to better discuss and tackle complicated parts of QScience development. We intend to maintain our regular Skype meetings. Encouraged by the success of our summer coding meeting in Zürich, we plan to hold another one or two coding meetings in the close future.

In the last year of the QLectives project, the development of QScience will enter a heavy testing period. In early spring 2012, we intend to launch QScience for internal testing and evaluation within the QLectives project. Then in summer 2012, QScience should be open to external users and we plan to search for attractive potential users such as learned societies or small research communities. With basic QScience platform finished, attention of developers will shift to implementing the quality and reputation system for QScience that is described in the deliverable D1.2.1.

Contents

1	Introduction	1
2	QScience Vision and Implementation	2
3	Patterns	7
3.1	Pattern files	8
3.2	Components	11
3.3	Automatic Export	13
3.4	Testing	14
4	QScience Functions	18
4.1	QScience Network	18
4.1.1	Rationale	18
4.1.2	Drupal-to-Drupal	19
4.2	QScience Visual Search Toolbox	22
4.3	Skype Integration	26
4.4	Living Science Integration	29
5	Future Development and Dissemination	33
5.1	Time plan	33
5.2	Dissemination plans	34
5.3	Training program and new people joining the project	35

Chapter 1

Introduction

During the third year of the QLectives project, the development of QScience has accelerated rapidly. In addition to work on the fundamental infrastructure required for the proposed evolutionary framework (which was described in the previous deliverable D4.2.2 and is elaborated further on here in Chapter 2), we started work on additional functions that will help make QScience better and easier to use (see Chapter 4). To fully enable the evolution of QScience instances, we work intensively on a module of the open-source content management system Drupal (this module is called *Patterns*) which will be of benefit for the whole Drupal community (see Chapter 3 for details). The benefits for QScience will be striking—once the Patterns module is finished, QScience instances can be created and customized in “one click”.

Chapter 2

QScience Vision and Implementation

“It is interesting to contemplate an entangled bank, clothed with many plants of many kinds, with birds singing on the bushes, with various insects flitting about, and with worms crawling through the damp earth, and to reflect that these elaborately constructed forms, so different from each other, and dependent on each other in so complex a manner, have all been produced by laws acting around us. These laws, taken in the largest sense, being *Growth with Reproduction; Inheritance* which is almost implied by reproduction; *Variability* from the indirect and direct action of the external conditions of life, and from use and disuse; a Ratio of Increase so high as to lead to a Struggle for Life, and as a consequence to *Natural Selection*, entailing Divergence of Character and the Extinction of less-improved forms. [...] There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, *evolved*.”

Darwin, Charles, The Origin of Species, 1859. (emphasis added).

Biology has highlighted the fundamental laws governing the complexity of life more than one hundred and fifty years ago. QScience wants to make use of the very same principles of natural selection and evolution, which have proved to be so successful in sustaining life throughout history, to the development of a general purpose application able to “self-evolve” in a World Wide Web environment. For this, from a methodological point of view, QScience represents a completely novel take in the landscape of software development techniques. Such paradigm could be named *Design by Pattern* for reasons that are explained below.

Evolution in QScience is not about self-writing software, whose realization would rather be a chimera, but is about unleashing the potential of hundred of thousands of developers and users, who everyday create, change, update, revert, rewrite and use some piece of software on the web to accomplish their particular purposes. The rationale for QScience is the idea that, apart from special cases, the bulk of such developing activities is aimed at realizing more or less the same limited number of operations, under small variations. Nowadays, CMS (Content Management Systems) offer the possibility to *adapt* general application frameworks for one’s own needs. On the other hand, such “adaptation” is most of the time rather costly to accomplish, and, most of the time, is not portable. From an economic perspective, it represents *sunken costs*, something done once and impossible to recover. The business of CMS customization has lately become so profitable that there exists a high number of companies offering specialized services in this area¹. QScience aims at transforming this customization activity from a sunken cost into an investment into a public good pool. QScience crystallizes these customization efforts into single portable entities, called *Patterns*, representing the *state* of a software component at given moment in time. From an OOP (Object Oriented Programming) point of view, it is a way to guarantee objects’ persistence. In fact, running a pattern file will reset the state of a software component to what prescribed by the pattern. Running the same pattern twice will not cause any

¹See for example <http://www.joomlayellowpages.com/> and <http://drupal.org/drupal-services>

change, because patterns are idempotent.

It must be acknowledged that Patterns are not really a new idea in the realm of software development. Configuration, or ini, or init files have always been present since the dawn of software programming. What QScience is doing, is bringing them one step forward. QScience is in fact able to automatically generate Pattern files describing the current state of the system, with little or no efforts for the users. Furthermore, these init files, or Patterns, can be shared, rated, and used to generate usage and liking statistics. In a summary, QScience is a modular system with a slim core, whose behavior is entirely determined by which modules are selected, which in turn are all completely and independently configured by Patterns files. Such patterns in turn are supposed to be automatically improved through time by “natural selection” of the best fitting. As long as new modules will be written supporting the Pattern paradigm the QScience ecosystem will be sustained and it will keep growing and diversifying. Very much like life has adapted to very different environmental condition, the QScience software will be able to adapt to serve the different purposes of different communities.

QScience aims at initiating software evolution driven by human selection and creativity. QScience could be considered the natural fusion between a CMS and a social network. However, QScience is more than that. QScience is an ecosystem of evolving communities. In fact, each QScience instance is never isolated. Once activated, a QScience instance starts creating links to other *related* instances. If a friendship link is established, a trusted encrypted connection is formed, and the instances can safely exchange data and information over this channel. Note that such process, although depending on a central server for bootstrapping, is entirely decentralized. QScience instances can navigate the nodes of friendship networks, and discover new interesting content and new communities. For a comprehensive explanation of the mechanism see Section 4.1.

For the implementation of the QScience platform, we chose to rely on the Drupal² framework because of the general and modular nature of Drupal, and also because of the already existing very large community supporting it³. Our development efforts during the past year were therefore directed in three main directions: (i) establishing the design-by-pattern paradigm in Drupal, (ii) creating a decentralized, secure, communication infrastructure between Drupal instances, and (iii) the realization of (at least) one custom QScience instance.

The overall Pattern architecture for Drupal was designed to be scalable and as loosely coupled as possible. Details about the implementation are supplied in Section 3, and the activities in this area could be grouped around the following goals:

- the creation of a Pattern engine for Drupal,
- the creation of a mechanism to automatically export Pattern files from a live Drupal instance,
- the development of a simple API for Drupal module developers,
- the application of the design-by-pattern paradigm the main Drupal modules.

Point (ii) was already briefly introduced above. For what concerning point (iii), QScience, which is a generator of communities, would remain an empty framework, if no community would be added initially. For this reason we implemented a discussion forum for scientific communities, as a first example of working QScience instance. This is basically the port of the Econophysics Forum⁴ in QScience. However, we did not limit ourselves to re-create a one to one copy of the existing Econophysics Forum, but we added a number of features that extend the original version of forum. This could be grouped in the following categories:

- an intuitive search and discovery interface;

²<http://drupal.org>

³For details, see Chapter 2 of the deliverable D.4.2.2

⁴<http://unifr.ch/econophysics/>

- an internal communication system integrated with conventional communication systems, e.g. Skype and email;
- integration with scientific data providers, such as Google Scholar, Mendeley, PlosONE, Arxiv and Springer.

These features are thoroughly described in Section 4.

Finally, besides development activities, during this year we prepared the basis for some of the future extension of QScience of the next year. Based on a survey study conducted by the University of Surrey on a representative sample of the scientific population, we identified a set of existing Drupal configurations (modules, plus customization) which would fit the needs of scientists that emerged during the study. Such Drupal configurations will be ported into QScience under the design-by-pattern paradigm; for details see Section 5.

Our team consists of developers from several partners. It is therefore important that we communicate as often as possible. We have different channels of communication. We use a private Wiki to share and store relevant technical details. We have project pages where we have an issue system. We use Git as our VCS. Having a mailing list is very useful. We also have weekly Skype text meetings. To further improve the speed of development, we had a Summer Coding Meeting which was hosted by ETH Zürich in 8-19 August, 2011. It consisted of a day of discussions followed by several days of work on Patterns design details.

Chapter 3

Patterns

In the last deliverable, we have been using Drupal 6. Over time, Drupal 7 has become stable and is now the target platform. Unfortunately, Patterns was not available for Drupal 7 and there were no plans to port it. We decided to take control and became the 7.x-1.x version maintainers for the Patterns module.

We have not only ported it from 6.x to 7.x, but added some new features and changed a couple of things:

- Restructured the core,
- Cleaner YAML format,
- Optimizations,
- Revised the components that handle the patterns,
- Codemirror2 integration, Libraries support,
- Quick Run interface,
- Various other interface changes,
- Importing from modules,
- SimpleTests.

The Patterns module is available at

<http://drupal.org/project/patterns>

Regarding code versioning, Drupal.org has switched from CVS to Git early 2011. As a result, we decided to switch our code from Bazaar to Git, which went seamlessly and the new system offers great benefits. Drupal.org has its own Git hosting space, however, to be able to contribute, one has to be a maintainer of a project or submit pull requests which can be applied by the maintainers. Since we have a group relatively small developers, we decided to have our code hosted at github.com, a free Git hosting website. We regularly sync to the drupal.org repository.

Our github.com repository is available at

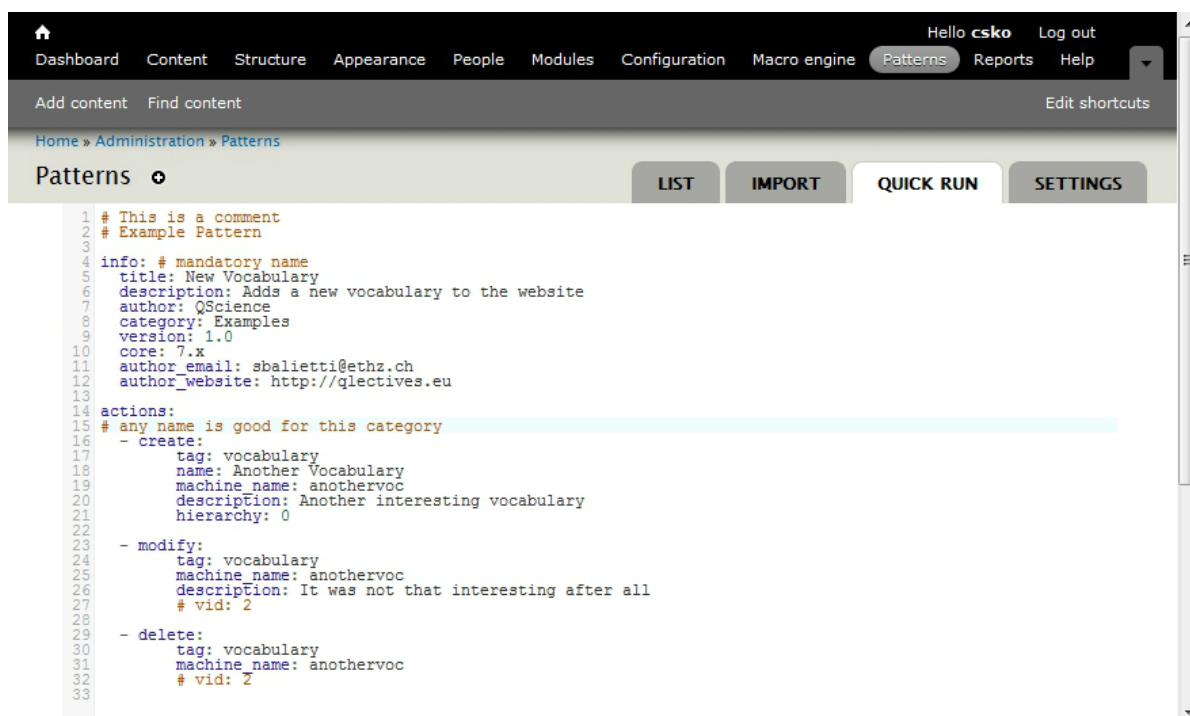
<https://github.com/shakty/Patterns>

Patterns is a relatively big Drupal module. Its core is made up of about 6000 lines and there are 5400 lines for various components, 1200 lines of testing code. We have about 3000 lines of Pattern files.

3.1 Pattern files

Complex Drupal web sites can be created by combining configurations of Modules, Fields, Content Types, Menus, Blocks, Categories, Roles, Permissions, etc... This site setup and configuration process is a very time consuming and repetitive bottleneck.

The Patterns module is built to bypass this bottleneck by managing and automating site configuration. Site configuration is stored in YAML, XML, or PHP files called Patterns. These files have a structure which is easy to read, modify, manage, and share. They are portable, reusable, and extensible. They can be executed manually or as a part of an automated web site deployment. Figure 3.1 presents an example Pattern written in YAML.



The screenshot shows the QLectives web interface. At the top, there is a navigation bar with a home icon, a search bar, and a menu with items: Dashboard, Content, Structure, Appearance, People, Modules, Configuration, Macro engine, Patterns (selected), Reports, and Help. Below the navigation bar, there are buttons for 'Add content' and 'Find content', and a link to 'Edit shortcuts'. The main content area is titled 'Patterns' and has sub-buttons for 'LIST', 'IMPORT', 'QUICK RUN' (selected), and 'SETTINGS'. The main content area displays a pattern file with the following content:

```
1 # This is a comment
2 # Example Pattern
3
4 info: # mandatory name
5   title: New Vocabulary
6   description: Adds a new vocabulary to the website
7   author: QScience
8   category: Examples
9   version: 1.0
10  core: 7.x
11  author_email: sbaliotti@ethz.ch
12  author_website: http://qllectives.eu
13
14 actions:
15 # any name is good for this category
16 - create:
17   tag: vocabulary
18   name: Another Vocabulary
19   machine_name: anothervoc
20   description: Another interesting vocabulary
21   hierarchy: 0
22
23 - modify:
24   tag: vocabulary
25   machine_name: anothervoc
26   description: It was not that interesting after all
27   # vid: 2
28
29 - delete:
30   tag: vocabulary
31   machine_name: anothervoc
32   # vid: 2
33
```

Figure 3.1: An example pattern loaded in the Quick Run page.

A Pattern is a file composed of an info section (generic metadata about the pattern) and a sequence of actions of the type *create*, *modify*, or *delete* grouped in categories (sections). The `tag:` expression tells Patterns which component to use. Every component defines a list of tags it can handle. After choosing the tag, the pattern file has to supply every required data for the particular action. In this example: `name`, `machine_name`, `description`, `hierarchy` are some of the data keys used, and all of them have values assigned. The data is passed to the pattern component as an associative array, most often referred to as `$data`. Some actions have alternative or optional parameter requirements.

Categories can be arbitrarily defined (excluding the reserved words) and are executed sequentially. Two other optional sections `names` are `modules` and `include`. `modules` explicitly defines a list of modules to enable during the execution of the pattern, while `include` loads the actions of another pattern into the current one.

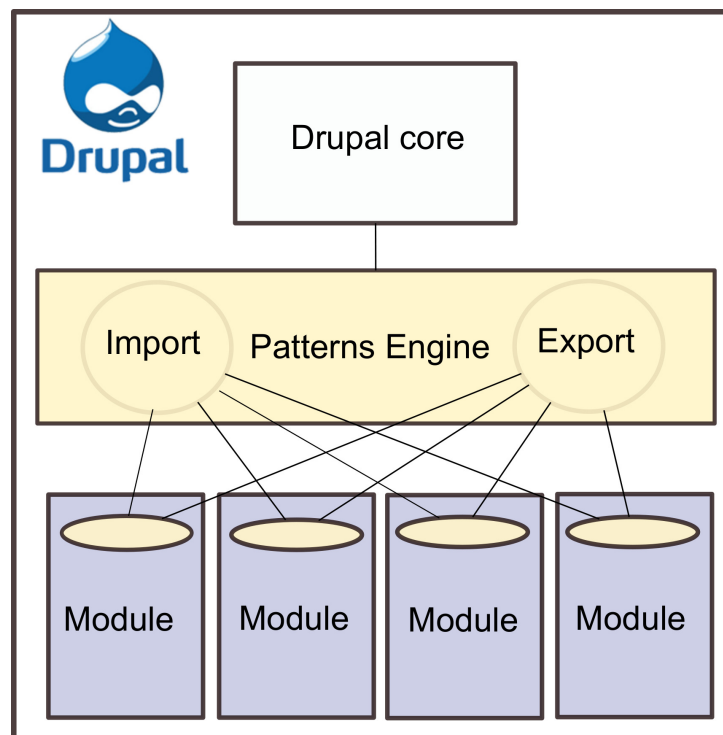


Figure 3.2: The Patterns architecture is designed to be scalable. Each Drupal module implementing a Patterns component (yellow circles) becomes automatically available to the Patterns engine for setting and exporting the state of the module.

3.2 Components

A component must be created to allow a module benefit from patterns. This means that if you want to define and combine features on new or existing Drupal sites as well as automate setup and configuration by patterns, you must have components that support these features. There are some core basic settings that are basic for most of custom sites. Patterns should therefore support most of the Drupal core and the key contributed modules that are commonly used. These are the components that we considered important and implemented:

- **system** – The System module provides basic but extensible functionality for use by other modules and themes. Some integral elements of Drupal are contained in and managed by the System module, including caching, enabling and disabling modules and themes, preparing and displaying the administrative page, and configuring fundamental site settings. A number of key system maintenance operations are also part of the System module.
- **user** – The User module allows users with proper permissions to manage user roles (used to classify users) and permissions associated with those roles.
- **menu** – A menu is a hierarchical collection of links, which can be within or external to the site, generally used for navigation.
- **block** – Blocks are the boxes visible in the sidebar(s) of your Drupal website. It is possible to control whether each block is enabled where it will be placed on the page, and control the visibility of blocks on each page.
- **field** – Custom data fields can be defined for entity types (entities include content items, comments, user accounts, and taxonomy terms). And the field data can be storing, loading, editing, and rendering.
- **taxonomy** – The Taxonomy module allows you to classify the content of your website.

- **node** – The Node module manages the creation, editing, deletion, settings, and display of the main site content. Content items managed by the Node module are typically displayed as pages on your site, and include a title, some meta-data (author, creation time, content type, etc.), and optional fields containing text or other data.
- **pathauto** – Pathauto provides a mechanism for modules to automatically generate aliases for the content they manage.
- **toolbar** – The Toolbar module displays links to top-level administration menu items and links from other modules at the top of the screen.
- **shortcut** – The Shortcut module allows users to create sets of shortcut links to commonly-visited pages of the site.
- **color** – The Color module allows users with the Administer site configuration permission to quickly and easily change the color scheme of themes that have been built to be compatible with it.

In each of these component, all of the related forms are called to realize their features by patterns just as if the submit button had been pressed from a browser window. In this way, patterns can automatically setup all the functions mentioned above and produce a personalized Drupal site according to the administrator's wishes.

Technically, there are a lot of functions that a component should realize. First, it specifies which form should be called for a tag and action in the pattern file. Then, it calls the form and prepares formatted data from the original pattern file for the form. Finally, it validates and submits the form and does additional operations if needed. So each component needs to implement pattern hooks to parse the pattern file and then construct, validate and submit a form which contains the values for specific elements. It's a demanding work to create so many components for a pattern. As an illustration, here is the list of hooks that a component needs to implement:

- `hook_patterns()`: declares that the module is able to handle pattern configurations, and specifies which tags and forms are supported.
- `hook_patterns_prepare($action, $tag, &$data)`: checks the input and adds default values. Eventually, it can raise errors which halt further execution of the pattern.
- `hook_patterns_validate($action, $tag, &$data)`: this hooks is similar to the previous one, however, you are working with the prepared data and you are checking against the database.
- `hook_patterns_callbacks($action, $tag, &$data)`: returns the array of form IDs which will be run sequentially.
- `hook_patterns_build($action, $form_id, &$data, &$action_state)`: gets the form data for the action. This can either be just the form values, or it can be the full `form_state` object.
- `hook_patterns_params($action, $form_id, &$data &$action_state)`: returns extra parameters that the form may require.
- `hook_patterns_cleanup($action, $tag, &$data)`: performs optional additional operations, after the execution of the action has finished.

Here `$action` is one of the actions *create*, *modify*, *delete* and `$tag` is the specific command to execute (vocabulary, field, etc.).

3.3 Automatic Export

The automatic export functions enables the user to automatically create patterns in only three steps.

In the first step (see Figure 3.3(a)), the user selects the module. Or if he wants to extract forms from several modules, he can choose the option to let him show all available forms.

In the second step (see Figure 3.3(b)) the user is shown a list of forms coupled to the tags defined in the component. Only the 'modify' forms are shown. The extraction function now starts calling the hooks which set up the forms. It calls `hook_patterns.get_arguments($action, $form_id, $form_id, &$loop = FALSE)`, if it was defined by the component, to get parameters to give them to the form. Otherwise none are given. Parameters can contain from a single id to almost all the data, only depending how the form itself works and gets the data to edit. Therefore it won't be harder to get this parameters than it is for the module. The parameters let the form decide what data will be shown in the form and therefore what data will be extracted. The function also may return a whole array of return parameters. Then it has to `$loop` to `TRUE`, too. Then the extract function goes through all elements of the array and calls the forms with each parameter. The forms then get submitted. The export function extracts by capturing and at the same time aborting what was sent when submitting.

The third step (see Figure 3.3(c)) consists of showing the pattern to the user for further editing and can be imported with a single click. Complications may arise when more data was extracted than the user and the component were expecting, because the function really captures everything what was sent, including all optional or hidden fields.

3.4 Testing

We create automated regression tests for our pattern framework. The tests can determine success or failure from browser or from the command line and represent an automated part of our pattern development lifecycle. It helps us to ensure that most of the desired functionalities are working as expected and hence improve our code quality.

The test part in pattern module is independent. What we need to test is pattern – a file composed by an info section (generic metadata about the pattern) and a sequence

Choose a module to extract the pattern

- import from all
- block
- color
- field
- menu
- mymodule
- navig_onepage
- node
- path
- shortcut
- system
- taxonomy
- toolbar
- user

Which forms should be inspected?

- block: Create/Modify/Delete Blocks 'modify': block_admin_display_form
- block: Create/Modify/Delete Blocks 'modify': block_admin_configure
- block_extract_all: Create/Modify/Delete Blocks 'modify': block_admin_display_form
- block_extract_all: Create/Modify/Delete Blocks 'modify': block_admin_configure

Select / Unselect all

(a) Choose which module (or if all) should be chosen

(b) Choose the forms which should be inspected

```

236 - modify:
237     tag: block
238     module: system
239     delta: main
240     title:
241     regions:
242         bartik: content
243         seven: content
244     visibility: 0
245     pages:
246     roles:
247         1: 0
248         2: 0
249         3: 0
250     custom: 0
251     types:
252         article: 0
253         page: 0
254     visibility__active_tab:

```

(c) An automatically extracted modify action from the block module.

Figure 3.3: Illustrations of the automatic export feature.

of actions of create, modify and delete. The 'tag:' expression tells the pattern which components to use.

We use two types of tests: Web test and Unit Test. The Web Test is useful to show the clear problem directly on the web and relies on the Drupal testing suite. The Unit Test tests only the function without accessing the database – therefore it runs faster and we can also do traditional unit tests for offline functions this way.

Because pattern modules have a lot of web-oriented functionalities, we have to verify them using tests of both types. The pattern test code works by parsing the pattern file and then constructing a form which contains the values for specific elements. The form is then validated and submitted. Then, it reports *ok* when the result matches what is expected and it reports *error* if there is a mismatch. The test data in the process are then removed.

Our test includes all of components in the pattern module, such as system, user, menu, block, and so forth. Using the test code to test all the functions in each of the component is critical for our code quality. We implemented it as an independent part in the pattern module so that it is now very flexible to add tests for a new component.

```

balistef@bansky:~/sites/dru79/sites/all/modules/patterns$ drush cc all
'all' cache was cleared
balistef@bansky:~/sites/dru79/sites/all/modules/patterns$ drush_test_clean
Removed 26 leftover tables.
Removed 1 temporary directory.
Removed 1 test result.
balistef@bansky:~/sites/dru79/sites/all/modules/patterns$ drush test Patterns
Block component 57 passes, 0 fails, 0 exceptions, and 10 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Color component 24 passes, 0 fails, 0 exceptions, and 4 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Field component with date 97 passes, 0 fails, 0 exceptions, and 20 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Importing files 57 passes, 0 fails, 0 exceptions, and 12 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Menu component 54 passes, 0 fails, 0 exceptions, and 15 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Node component (Content Types) 54 passes, 0 fails, 0 exceptions, and 10 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Pathauto component 86 passes, 0 fails, 0 exceptions, and 20 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
General test 54 passes, 1 fail, 0 exceptions, and 11 debug messages
Test PatternsPrivilegesTestCase->checkPage() failed: There should be no warnings.
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
System component 68 passes, 0 fails, 0 exceptions, and 12 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
Taxonomy component 60 passes, 0 fails, 0 exceptions, and 14 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.
toolbar component 23 passes, 0 fails, 0 exceptions, and 6 debug messages
No leftover tables to remove.
No temporary directories to remove.
Removed 1 test result.

```

[success]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]
[ok]
[status]
[status]
[status]

Figure 3.4: The testing process, as run from the command line.

Chapter 4

QScience Functions

There is a wide range of possible functions that QScience could or should have. Based on our primary use cases (forum for a scientific community and web site of a learned society), we prepared a long list of functions ranging from very basic functions for data and user management to advanced social functionality. For each of these functions we attempted to find a candidate Drupal module that could serve at least as the first step towards implementing the given functionality. The resulting document is too extensive to be included in this deliverable but it is available online on the QLectives wiki.¹ In this chapter, we elaborate only on the most important and challenging functions: communication of QScience instances, visualisation tools, integration with Skype telephony, and integration with the Living Science platform.

4.1 QScience Network

4.1.1 Rationale

In the following, *Drupal-to-Drupal*, a Drupal module allowing peer-to-peer communication among QScience communities will be described. We chose this approach –

¹http://www.qlectives.eu/wiki/images/0/06/QScience_Functionalities16jan2012-drupal_AG.docx

instead of building a peer-to-peer network with the users as nodes – for several reasons. This approach allows easy communication among the QScience websites, e.g. sharing statistics, ratings, data, but also broadcasting news and providing Pattern files to similar communities. The *Drupal-to-Drupal* module fully supports the new – and we believe interesting and valuable – idea of the evolution of QScience instances. Based on the peer-to-peer network of QScience communities, the new quality-trust-reputation model will become a prime candidate of a potentially very computationally demanding step where we will consider the algorithmic complexity and peer-to-peer implementation as of high importance. Compared to this peer-to-peer approach with instances as nodes, the idea of building a peer-to-peer network directly among users seemed less useful. For one thing that would have required the users to install and also regularly run some peer-to-peer software, which we considered rather unhandy, and for another thing this approach lacks most of the advantages our peer-to-peer network for QScience communities provides.

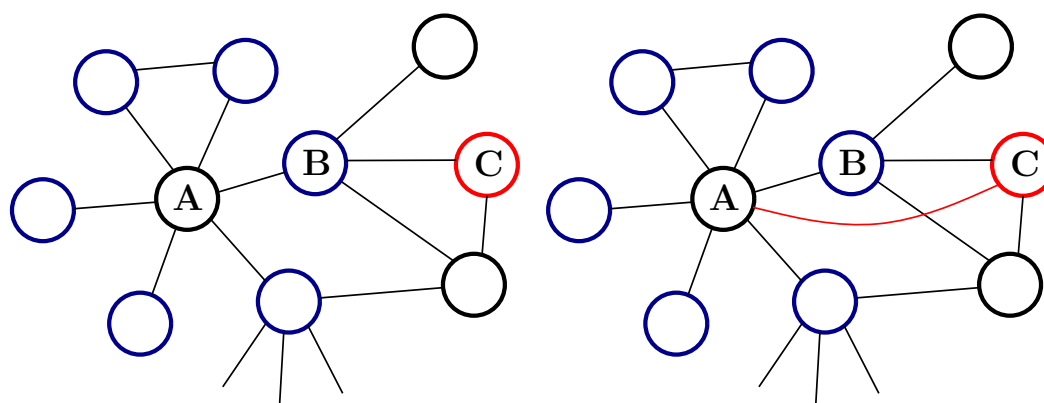
4.1.2 Drupal-to-Drupal

To build a QScience network, Drupal websites hosting QScience communities should be able to communicate with each other. Natively, Drupal supports XML RPC, a standard for remote procedure calls, encoding a call with XML and sending it over HTTP. Communication via XML RPC can be used almost similarly as a PHP-function-call, providing arguments to a remote function and getting back a return value. While being easy to use, neither authenticity nor secrecy can be guaranteed by these remote procedure calls. The *Drupal-to-Drupal* module solves these issues. It allows to build up a QScience Network, a peer-to-peer network of QScience communities where authentic and secret communication can be reached using strong cryptography.

The built network is similar to a friend-to-friend network where users only make directed connections with people they know and also to PGP's web of trust. Building a peer-to-peer network, i.e. a network of communities without central infrastructure,

allows the network of QScience communities to grow without increasing the workload on a single server. Moreover, there is no single point of failure: while some instances are offline, other instances still can communicate with each other. All in all, the peer-to-peer structure allows a dynamically growing network of QScience communities. Several independent networks of communities can grow independently and later eventually unite to a larger network.

The concept of ‘friendship’ between Drupal instances, i.e. QScience communities, is introduced. Friendship is established via public key cryptography using PHP’s OpenSSL module: an instance running the *Drupal-to-Drupal* module holds a public/private key pair. To make friends, two QScience communities sign strings both stating their friendship and including an expiring date using their particular private keys. Once these signed friendship certificates are exchanged, the communities having just made friends can use these certificates to prove this friendship to other communities. To continue friendship new friendship certificates are exchanged automatically and well-timed, otherwise the friendship expires.



(a) Network of communities: *B* is a common friend of *A* and *C*. (b) Network of communities: friendship between *A* and *C* has been established.

Figure 4.1: Illustration of the QScience network.

As an example for establishing friendship, consider a network where communities *A* and *B* are friends as well as *B* and *C*, see Figure 4.1(a). In particular *C* holds *B*’s public key and *A* holds a valid friendship certificate signed by *B* attesting the friendship

between A and B . Now it is easy for A to offer friendship to C . To do so, A signs a certificate for the offered friendship between A and C . Moreover the friendship certificate signed by B for the already established friendship between A and B is sent to C . Now, without communicating with B (B could be even offline), C can verify that A and B are really friends. Now, depending on the trust C has into B 's friends, C certifies the newly established friendship to A . Communities A and C are now friends, see Figure 4.1(b).

To find potentially interesting communities, QScience communities have the possibility to exchange lists of their friends, making it easy to discover new communities. Beside that, friendship can also be established with communities that have no friends in common: it is up to the administrator of the community receiving a request to check whether the other community/the request is confidential.

Best papers

COMMUNITY	AUTHOR	TITLE	RATING	VOTES
Community A	D. Helbing	Managing complexity in socio-economic systems	4.75	18
Community B	J.R.G. Dyer, A. Johansson, D. Helbing, I.D. Couzin and J. Krause	Leadership, consensus decision making and collective behaviour in humans	4.98	28
Community C	D. Helbing, Y. Wenjian and H. Rauhut	Self-organization and emergence in social systems: Modeling the coevolution of social environments and cooperative behavior	4.95	15

Best rated news

COMMUNITY	AUTHOR	TITLE	RATING	VOTES	DATE
Community A	D. Helbing	When We're Cowed by the Crowd	4.9	26	2011-05-28
Community B	S. Balietti	What Causes Deadly 'Crowd-Quakes'?	4.79	24	2011-04-22
Community C	D. Huber	How crowds move: because people want to be free	4.96	28	2011-04-19

User statistics

COMMUNITY	NUMBER OF USERS	AVERAGE POST COUNT	AVERAGE VOTE COUNT
Community A	63	42.7937	15.5238
Community B	42	40.2381	12.9286
Community C	127	39.4173	14.0787

Figure 4.2: Application of the *Drupal-to-Drupal* module: sharing statistics with friends (example using random data).

Beside using cryptography to built-up a peer-to-peer network of QScience communities, cryptography provides the possibility to use authenticated and secure procedure

calls. This allows many possible applications for the *Drupal-to-Drupal* module.

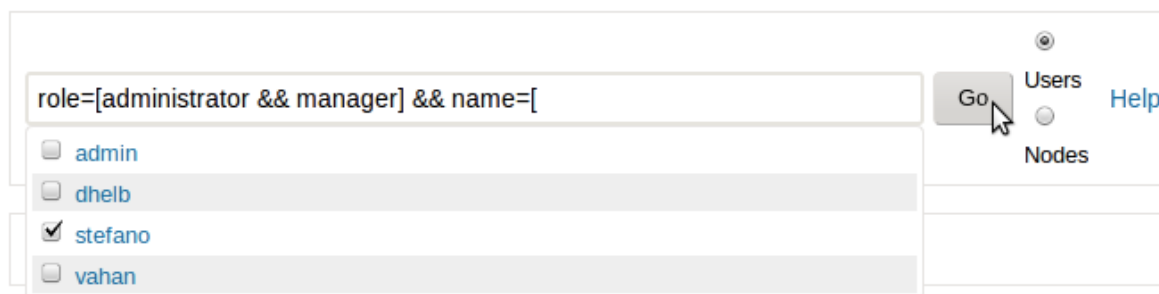
Privileged content can be shared with friends, e.g. a community might want to provide particular statistics to their friend-communities only. More than that, one can group friends, to obtain more fine-grained sharing rules, e.g. a community might want to provide aggregated statistics about its users to all communities that are friends, while more detailed statistics are only shared with particular friends. Other applications include sharing the latest news and publications (possibly with ratings), providing data that another community can use for further processing, and publishing *Pattern* files that might be of use for other communities. Note that, depending on the particular purpose, the shared contents can both be authenticated (by signing it using the sender's private key) as well as encrypted (using the receiver's public key): while for provided *Pattern* files it is important to be authenticated, it might be crucial that confidential data is encrypted before being sent.

4.2 QScience Visual Search Toolbox

QScience Visual Search Toolbox is an easy to use search and visualization tool for users with administrative privileges. It's built to a Drupal module for better integrability and extendibility. The main idea of the module is to have one single textbox, but at the same time many different search and visualization options, usage of which should be very intuitive. New visualization options could be easily added by creating a module which handles the visualization, and afterwards implementing a single function which would link the new visualization to QScience Search Toolbox. The implementation of the idea will be clearer as we guide you through the screenshots.

As you can see on figure 4.3, the user is provided with a single autocompleteion supported textbox in which he starts building the search query. As soon as the user types "field_name=", he is provided with a list of possible values of a "field_name", from which he can choose the ones that he is looking for by ticking the checkboxes.

— REFINE SEARCH



The screenshot shows a query building tool interface. At the top, there is a search bar containing the query: `role=[administrator && manager] && name=[`. To the right of the search bar is a "Go" button. Below the search bar is a list of user names: `admin`, `dhelb`, `stefano` (checked), and `vahan`. To the right of the list are two radio buttons: "Users" (selected) and "Nodes". There is also a "Help" link.

Figure 4.3: Query building tool.

The possible “field_names” are taken from the database and include all possible fields for a given content type (in this case for User). In our case the list displays all possible values of the field “name”. The user can also choose if he wants for example to search for “administrators and managers” or “administrators or managers” by changing the logical operator in the square brackets to “——” or just typing “or”. Thus the search query is built very quickly based on the criteria user provides. After the desired logical query is built, the search can be conducted, and the results are displayed in a separate dialog (see fig. 4.4), allowing users to conduct several searches without having to open a new browser tab, or clearing the results of the previous search. This would allow easy comparison of different search results.

After you get the results in a separate dialog, you can keep searching and refining your results for example by adding specific dates of user registration, names, etc. For the date search a calendar is also provided by the query building tool. You can also choose the fields you want to display in the list of results by just checking/unchecking the checkboxes. If you want to conduct another search, you can just minimize or make smaller the current dialog, and run a new search. After you do it, you will again get a new dialog with the search results according to the query you built (see figure 4.5).

Provided the list of the entries you searched for, you can export the list of the selected results as a csv file by clicking the “Export to CSV” button, you can send an

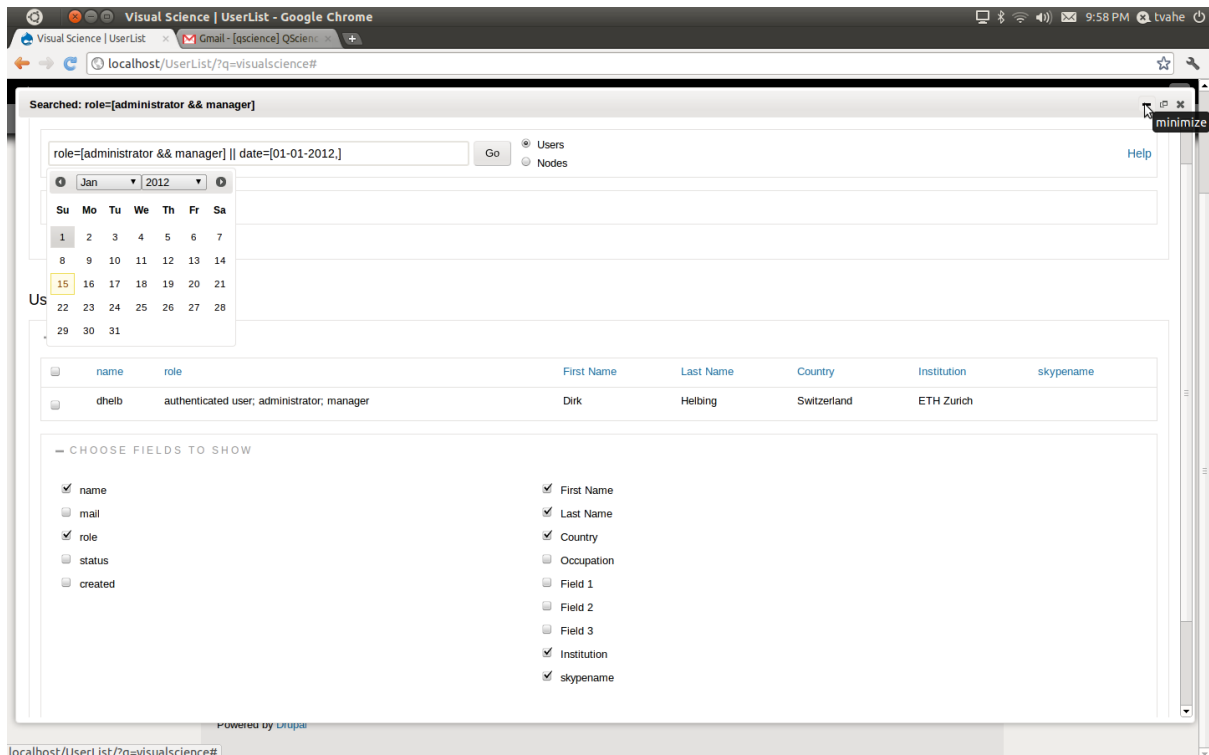


Figure 4.4: Displaying the search results.

email if the entry has a field containing the email address (which is the case if you search for users) by clicking “Enter the message” button (this will open a space where you can enter the message), and you can also contact selected users directly via skype if there is a skypename provided and you are logged in to skype via the webpage. More details about the browser based skype functionalities can be found in section 4.3. Skype functionality is integrated to the Visual Search Toolbox by simply implementing one function, which creates the button, and specifies its behavior. As you can notice from the screenshot on figure 4.5, the user can also save and load earlier conducted searches, thus creating his own custom lists.

The displayed list after the search is just one of the visualization tools that is attached to Visual Search Toolbox by implementing the same function as for the Skype button. In the same manner any other visualization tool (e.g. chart, map) can be attached to the Toolbox by implementing the so called “hook” function. You can see the code snippet in the next screenshot (figure 4.6). On the left side is the part of the code from the

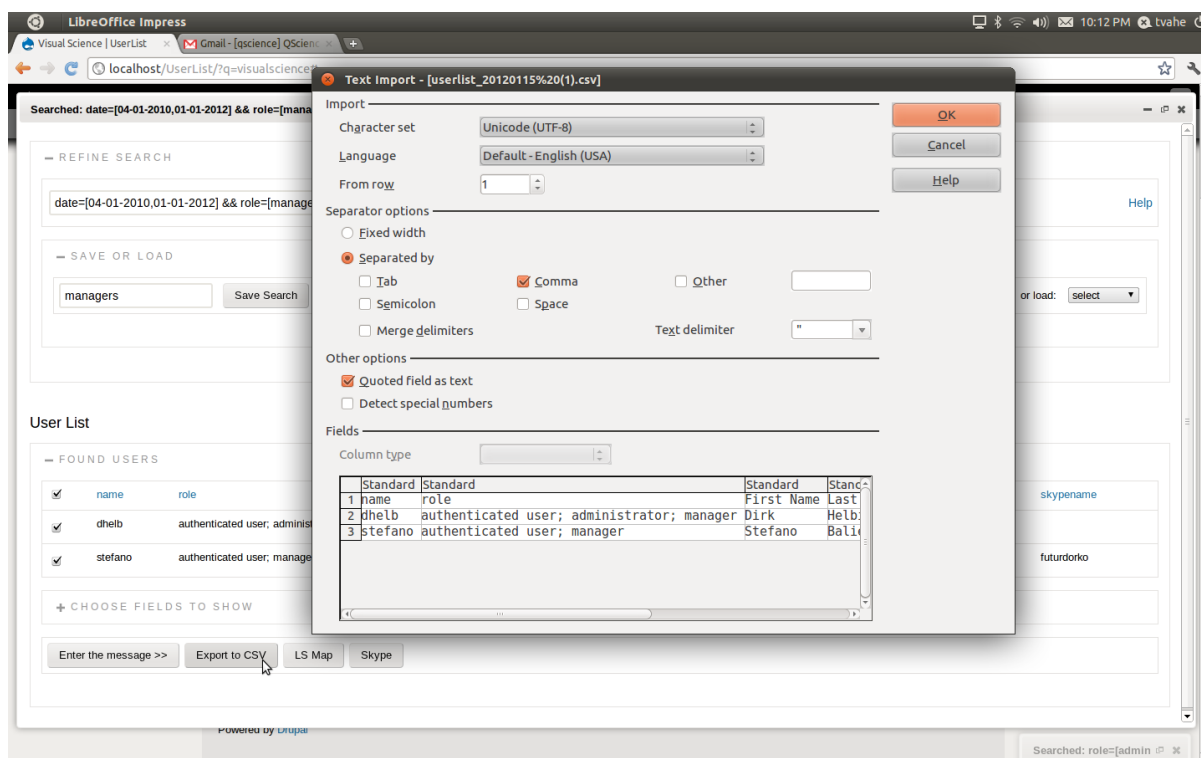


Figure 4.5: Sending an email, exporting to a file and contacting using Skype.

module which implements the visualization. In this case its the user_list module which implements the list visualization. It implements the “hook_science_tab()” function, as described on the right side, where the part of the code from the Visual Search Toolbox Module is presented. In order to invoke the output from the other module, you just have to call the “module_invoke” function with the name of the module, and pass the parameters. There is also a possibility to invoke all the modules which implement the given hook.

In the same manner the Living Science described in section 4.4 was integrated with the Visual Search Toolbox (figure 4.7). After clicking the “LS map” button the request will be sent to the Living Science server, a search will be conducted according to the names of the selected users, and the results will be displayed as shown on the screenshot, in 2 new resizable/closable subsections on the dialog. In one of those the user gets the results as a list, in the other one as locations of the publications on the world map. In this same manner any other visualizations can be attached to the toolbox,

The Visualisation Module (user_list)	Qscience Visual Search Toolbox
<pre> * Implementation of hook_science_tab() * * Returns HTML to show in the User List visualization * @param array \$list The search results to be rendered and shown * @param string \$idSuffix the number of the current search dialog */ function user_list_science_tab(\$list, \$idSuffix) { // Adding jquery ui components used drupal_add_library('system', 'ui.autocomplete'); drupal_add_library('system', 'ui.datepicker'); drupal_add_library('system', 'ui.dialog'); // Adding javascripts drupal_add_js('http://livingscience.ethz.ch/Livingscience/Livingscience.nocache.js', 'external'); drupal_add_js(drupal_get_path('module', 'user_list') . '/user_list.js'); drupal_add_js(drupal_get_path('module', 'visualscience') . '/visualscience.jquery.layout.js'); drupal_add_js(drupal_get_path('module', 'visualscience') . '/visualscience.js'); drupal_add_js(drupal_get_path('module', 'visualscience') . '/visualscience.jquery.dialog-extend.js'); // Adding the css drupal_add_css(drupal_get_path('module', 'user_list') . '/user_list.css'); \$paginated = FALSE; \$header = "User List"; \$empty_msg = "There are currently no users in this category to list."; \$number = 30; \$element = 0; // Creating the output \$output = "<div class='visualscience-user_list' id='visualscience-user_list-' . \$idSuffix . '>"; \$output .= \$idSuffix . "</div>"; if (\$header) { \$output .= "<h3>". \$header . "</h3>"; } \$output .= "<div class='content'>"; \$output .= drupal_render(drupal_get_form('user_list_form_' . \$idSuffix, \$list, \$idSuffix)); if (\$paginated) { \$output .= theme('pager', array(NULL, \$number, \$element)); } \$output .= "</div>"; \$output .= "</div>"; </pre>	<pre> /* * @file * This module is used to search and visualize users/nodes. * The visualization must be done by separate modules, which will be shown as different sections in the d * To do so your module must implement the following: * 1. Implement hook_science_title() - just returns the title of the tab * 2. Implement hook_science_tab(\$list, \$idSuffix) - return the HTMLed visualization of the given list * 3. Keep all the content inside <div id='visualscience_MODULENAME'>...</div> - this is needed to update */ variable_set('visualscience_current_tab', 'user_list'); \$output = "<div class='visualscience-container' id='visualscience-container-' . \$SESSION['searchNumber'] . '>"; \$output .= drupal_render(drupal_get_form('visualscience_search_form_' . \$SESSION['searchNumber'])); \$list = variable_get('visualscience_list'); \$output .= module_invoke('user_list', 'science_tab', \$list, \$SESSION['searchNumber']); \$output .= "</div></div>"; if (\$clear) { variable_del('visualscience_current_tab'); variable_del('visualscience_list'); variable_del('visualscience_search_table'); } return \$output; </pre>

Figure 4.6: Code snippet showing the integration of a module with Visual Search Toolbox.

which can be displayed in new subsections on the dialog.

As an outlook for this module we want to have a very easy to use search tool, which would give the users a possibility to specify their search criteria very easily and intuitively and be able to visualize their results in many different ways and combinations, and be able to choose the one that suits him/her the best. The module will be very easily extendible using the “hook” functions described earlier and demonstrated on figure 4.6, thus providing advanced users with the possibility to create their own visualizations, and change the existing ones to suit them better if there is a need.

4.3 Skype Integration

Skype Integration allows users to have full browser integrated skype functionality, without having to install any additional software. This social feature will keep users longer on the webpage without a need to close it, or to switch to some other program in order to continue working. And as described at the end of this section, there are very attractive possibilities of extending skype functionalities, which would make users even more attached to the website.

The technical part is done using the skypekit python library provided by Skype for

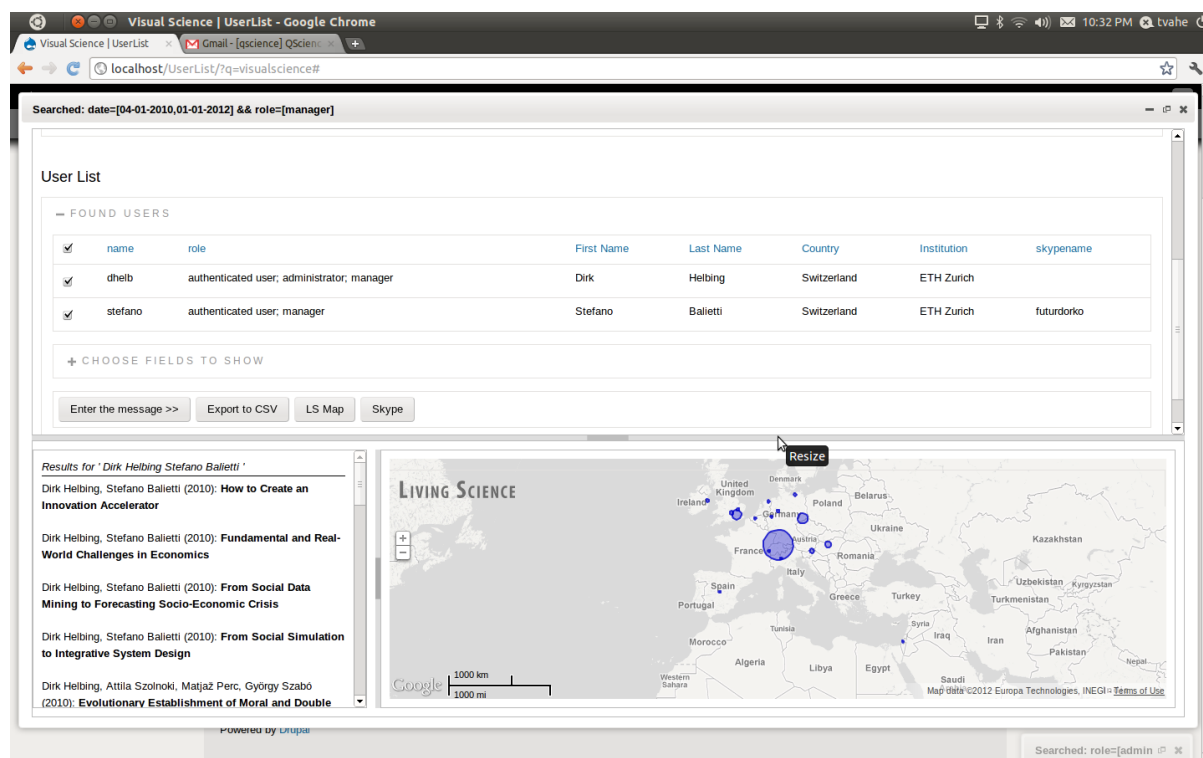


Figure 4.7: The integration of the living science module with Visual Search Toolbox.

the server side handlers, and an advanced javascript dialog based system on the client side. The data communication is done using websockets, which allows us to transmit data much faster than xmlhttp requests, thus improving user experience (fig. 4.8).

One of the many advantages of having browser integrated Skype is that in addition to allowing the users to talk to their skype contacts without leaving the webpage, making online conferences and using all the other features that skype provides, this Drupal module can be integrated with the QScience Visual Search module, which would allow users to easily search for and contact people of same interests, fields of study and profession using the database of the webpage. For more information about the QScience Visual Search module and integration of skype with it see section 4.2.

Also, as mentioned earlier, the big advantage of having browser integrated Skype is the extendibility. Users will be able to send links to pages, files, charts and tables of the website by 1 or 2 clicks. Also, tools like google translate can be integrated, which would allow users to translate their messages on the fly, and this would make the

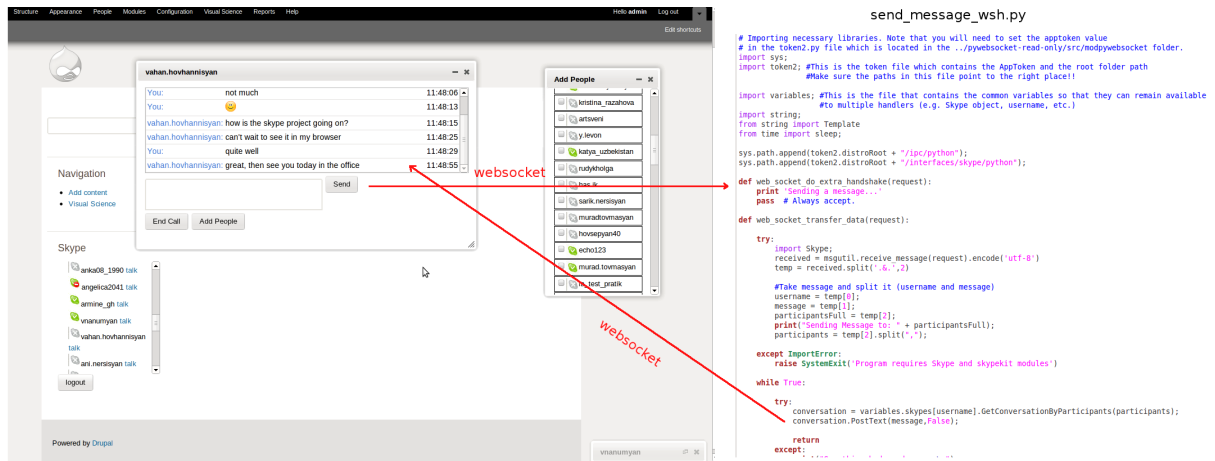


Figure 4.8: Skype integration – a client-server communication example.

communication easier. On figures 4.9 and 4.10 screenshots with short explanations on the integration process and workflow are demonstrated.

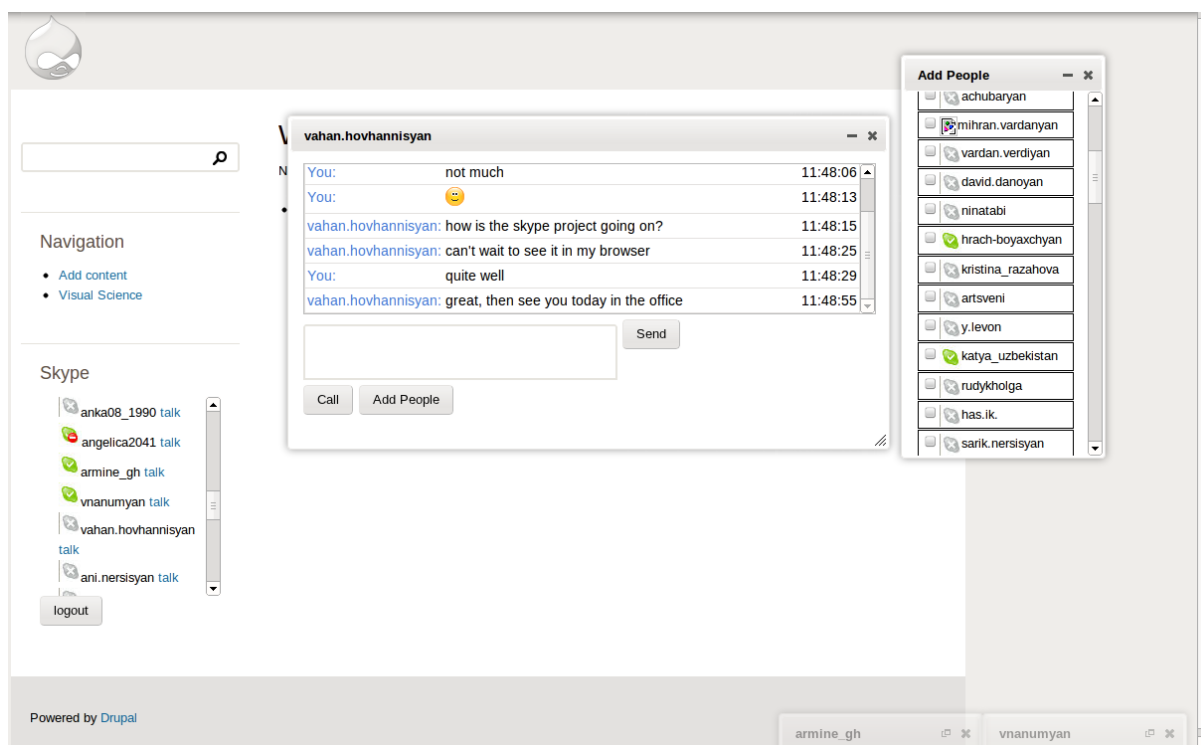


Figure 4.9: Skype integration – after logging in you are able to view the contact list as a Drupal block. By clicking next to the names of contacts you can start a conversation with them. Later on you can add people by clicking the “Add People” button. On the right bottom you can see some minimized conversations.

4.4 Living Science Integration

Living Science is a search engine for publications. Originally developed as a separate web application, available online on livingscience.ethz.ch, it is integrated into QScience through a JavaScript API provided by the Living Science server (see the Living Science screenshots in Figure 4.11).

What it does. Living Science makes use of existing online publication databases together with other freely available information aggregated from the Internet. Instead of visiting several search websites manually, a user is provided with most relevant results in just one place. Additionally, the information aggregation allows for better statistical analysis and is the basis for different means of visualization. Search queries are forwarded to search providers like Google Scholar, Mendeley, Springer, Arxiv, PLoS & Amazon. Additional Search APIs can be added programmatically. Search

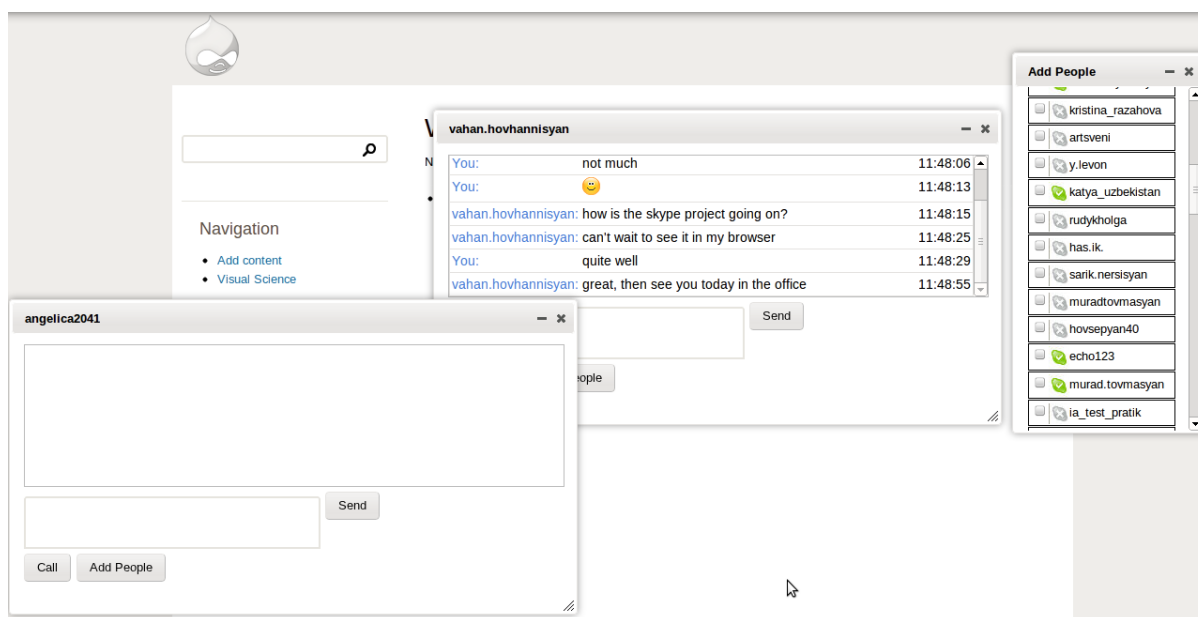


Figure 4.10: Skype integration – you can also have several conversations at the same time, just like the desktop-based Skype.

results can be viewed using several visualization tools. A world map shows cities with publications whose author's affiliation is known to be in that city. A relations graph puts papers with common authors and similar topics close together. A traditional textual list of papers is also provided.

How is affiliation information added? With the exception of PLoS, no Search API has any information about affiliations and locations. Nevertheless, it can possibly be extracted from publication full texts if: 1) they are available and 2) the parsing process can recognize an author's address. Given the very diverse ways of writing university and institution names, a reliable identification method is to use the author's email address domain name. A list of university website domains can be extracted from Wikipedia, University rankings and other online resources. The reliance on full texts currently limits the affiliation matching to Arxiv papers and otherwise freely downloadable PDFs.

Implementation details. The web application is mostly written in Java. The Google Web Toolkit (GWT) is used to compile parts of the Java code to JavaScript for use on client side. From a language point of view it is possible since Java has a much stricter

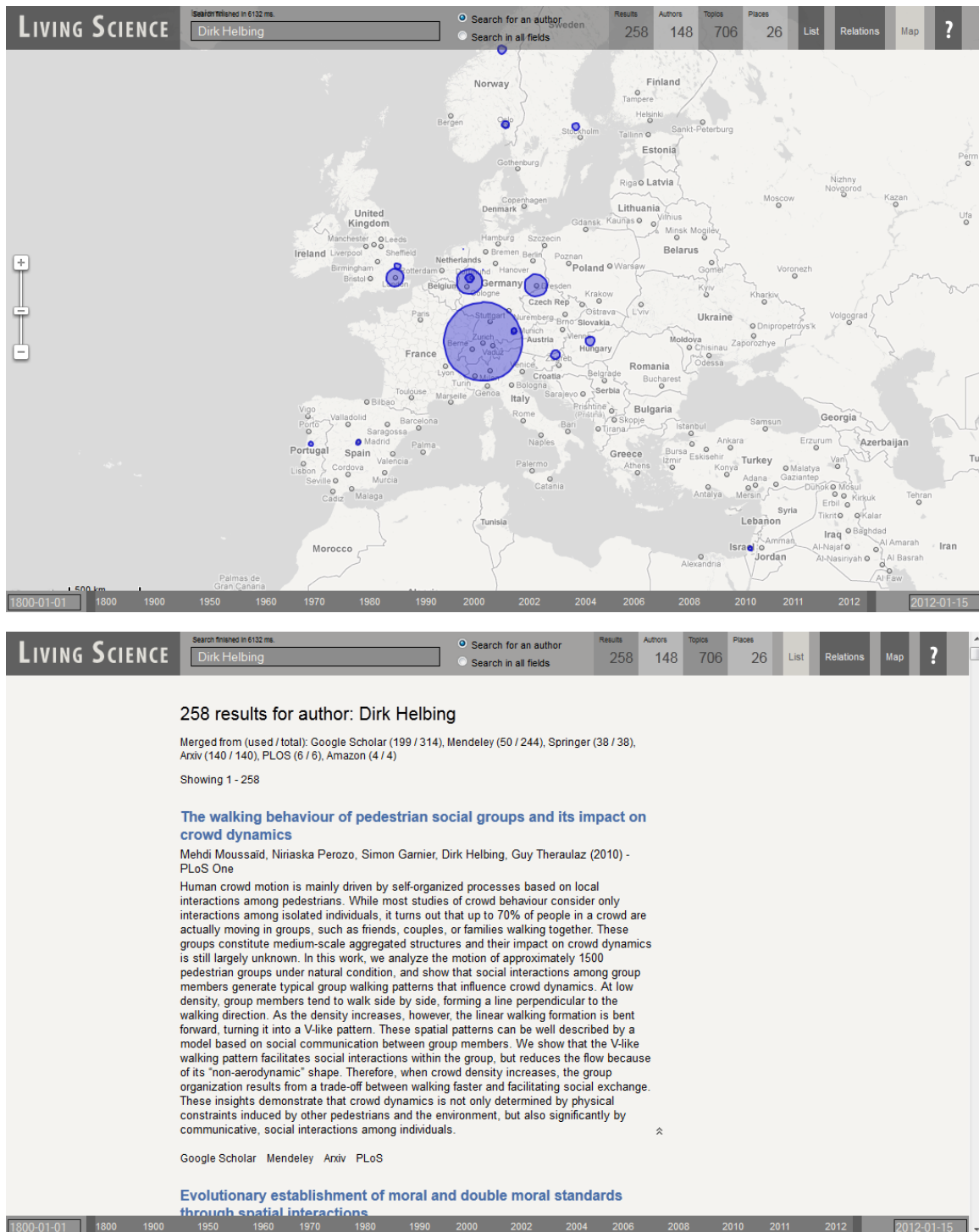


Figure 4.11: Living Science screenshots.

strongly-typed grammar and can be quite straightforwardly mapped to JavaScript. GWT provides a JavaScript mapping for basic Java SE packages. Native JavaScript code can also easily be called from Java side. Communication between client and server is simplified and the development of complex client-side logic calls for Java's mature software engineering tools and practices. One goal of the Living Science application is to put most processing on client side. This leads to less server load and better scalability. Furthermore, less server calls also increases the responsiveness to user input. The Living Science JavaScript API lets an integrating website include the web application as a single JavaScript file and expose certain features to the user, like searching for an author and displaying a map.

Chapter 5

Future Development and Dissemination

Regarding the future development of QScience, we intend to maintain our tradition of regular Skype meetings where problems are discussed and new tasks are distributed. Encouraged by the success of our summer coding meeting in Zürich, we plan to hold another one or two coding meetings in the close future.

5.1 Time plan

In the last year of the QLectives project, the development of QScience will enter a heavy testing period. To be more specific, here is a tentative time plan for the next twelve months:

- February/March 2012: Launching QScience for internal testing and evaluation within the QLectives project,
- Summer 2012: Opening QScience to external users,
- Autumn 2012: Finding attractive external users for QScience,

- Autumn 2012: Implementing a new quality, trust and reputation module for QScience (more details below).

By attractive external users we mean influential scientific groups, research communities, learned societies, or active individuals who would use QScience as their main online outlet. As the first and obvious candidate, there is the Econophysics Forum community which will be probably the first one to use QScience but we would like to contact others to prove that our approach is as modular and customizable as we intend.

As to the above-mentioned new quality, trust and reputation (QTR) module, it will be simply a new Drupal module implementing our new QTR model for science. This model is now being developed in cooperation by the University of Surrey, University of Fribourg, and the Warsaw University. The main goal of the model is to process the data on online activity of scientists (by activity we mean writing a new paper, submitting a new blog post, commenting, and so forth) into quality of scientific objects (papers/blogs/meetings/...), reputation of individuals, and trust relations between individuals (see the deliverable D1.2.1 for details). We are currently in a stage where the basic model is tested by extensive agent-based simulations. In the next stage, we plan to develop various testing scenarios (such as competition of two opposite opinions within a scientific community or a case of a science fraud being suddenly discovered by the community) to further develop our understanding of the model's behavior and possibly also improve the model with respect to robustness to dishonest behavior. After the model evaluation is finished, the model specification will be passed further to QScience developers who will turn it into a Drupal module that will be available to use in any QScience instance.

5.2 Dissemination plans

As described above, we plan to launch a test QScience web site in the next few months. Upon successful testing, QScience will be ready to be offered to groups outside from

the QLectives project, in particular for those that have already shown their interest in the concept. Successful use by third parties is of course the best dissemination activity possible and we believe that we will enter this stage soon.

Once we succeed in porting, developing, and maintaining such a demanding module as Patterns, the Drupal community should recognize this contribution and we hope to attract some Drupal developers to contribute back to QScience itself. In addition, we might participate in Drupal meetings (known as Drupalcons, see the list of past meetings at en.wikipedia.org/wiki/Drupal#DrupalCon_events) which would further increase the visibility of QScience and QLectives as a whole.

5.3 Training program and new people joining the project

QScience project is an attractive project, there are some new people want to join this project. In addition, another advantage of choosing the Drupal platform manifests itself: since Drupal is hugely popular and widely used content-management-system, young project participatns working on QScience gain valuable experience that can help to succeed in the job market.

To accommodate new developers, we set up a training program. Since Drupal is written in PHP and makes use of the PHP Data Objects classes to abstract the physical database from module code, learning the PHP language is the first step for our new developers. However, the project has tight time limits, so in the mean time, new developers get familiar with our development process, such as Weekly skype meeting, online QScience code review, git repository, and others, even before they are capable of contributing actively. Finally, the team leader (Stefano Balietti) assigns easy tasks to newcomers and everyone assists them with their completion so that their start is as fast and frictionless as possible. The developement team of QScience currently has the following new members:

- Christian Schulz (ETH Zurich),

- Christoph Schwirzer (ETH Zurich),
- Pratik Mukerji (ETH Zurich),
- Vahe Tshitoyan (ETH Zurich),
- Sasa Tomic (ETH Zurich),
- Vahan Hovhannisyan (ETH Zurich),
- Liao Hao (University of Fribourg),
- Xiao Rui (University of Fribourg).