Problems and implications for web-based wizardry

# Introduction

The Wizard-of-Oz (WOz) technique is a method used to simulate the intelligence of a system. The simulation is conducted by replacing a system’s functionality with a human experimenter (a “wizard”) who interprets the user’s actions and mimics the functionality, with or without the user’s knowledge. These simulations can be carried out to try, discuss, demonstrate and evaluate ideas, systems, or partly developed prototypes. J.F. Kelley (1983) coined the “OZ paradigm”, reporting the development process of a natural language computer application called CAL (Calendar Access Language), where a human replaced the language processing components. The “OZ paradigm” refers to the man behind a screen using technology to personate a wizard in the novel “The Wonderful Wizard of Oz” (Baum 1900). Gould, Conti and Hovanyecz (1983) reported on a similar experimental technique as used by Kelley. Gould et al. replaced the automatic speech recognition components in a listening typewriter system with a human typist who wrote what the participants in the study dictated. The typist’s writing was depicted on the user’s computer screen. Similar experiment has been carried out earlier such as the evaluation of a self-service Airline Ticket vendor where functionality was performed by a human operator instead of the system itself (Erdmann & Neal 1971).

As argued by Pettersson and Siponen (2002) amongst others, the popularity of the WOz technique in studying language technology and natural language interfaces can be explained by the nature of such systems and technology: “Automatic interpretation of text or speech is difficult and the Wizard-of-Oz technique thus gives systems developers a chance to test systems before it is even possible to make them.” (Pettersson & Siponen 2002, p.293) However, the Wizard-of-Oz technique has shown to be eligible for other application areas as well[[1]](#footnote-1). “Since the system looks real to the test user [that is, test participant]. One could use Wizard-of-Oz mock-ups to test design ideas when there are reasons to believe that simple tests by sketches and slides […] will not provide the right responses.” (Molin & Pettersson 2003, p.77) By using the Wizard-of-Oz technique the user is deceived into believing that he/she is interacting with an automated system, why his/her responses will be more accurate than responses to interacting with for example a paper prototype.

There are several systems supporting WOz experiments. One such system was developed at Karlstad University during the early 2000’s. The system, called Ozlab, enabled prototyping, evaluating and testing of graphical multimedia interfaces, without needing any previous programming. Though, due to Ozlab’s dependency on what is now an out-dated version of Macromedia Director, the system is being redeveloped as a web based system.

## Research question

The basic question that this report tries to answer is simply this:

What problems and implications follow a web-based WOz system?

This question has many aspects, however. In spite of the fact that systems environments constantly changes, there have been several WOz system presented as generic WOz tools the last decade. Will they stand up to measure? Further, web-based solutions, to what extent do they manage to free themselves from system-dependencies?

Furthermore, because there now is a first release of the web-based Ozlab system it is time to evaluate this version.

Finally, using a web-based tool opens up to possibilities to see the tool as websites rather than actual program entities. Thus, what are the implications of URLs on a web-based WOz system?

## Literature review

In order to find implementations of the Wizard-of-Oz technique in research and Ozlab-related solutions, a literature review was conducted. The starting point in the search for publications was to find Ozlab related systems and solutions. As I initially wanted to find solutions that was not based on obsolete technology, the search begun for publications from the year 2000 and forward. In order to provide a fuller picture of both methodological and technical advancements, publications from earlier years was included as well. The aim was to provide at least one example from each found research area where the WOz technique has been used. The search resulted in 52 publications. In search for WOz implementations, 10 representative publications from earlier years were included.

Publications were found by searching a few major databases, reviewing references in found publications and by advice from my supervisor. The three tables in Appendix 1 summarize the search strategy.

During the literature review a number of limitations to the Wizard-of-Oz technique were found.

* **Reliability**: if conducting structured tests where it is important that the sessions can be compared and results be quantified, one must make sure that the responses given by the wizard(s) is consistent. It should be noted that consistency should be handled differently depending on the experimental set-up (for instance if several wizards are used). Otherwise the results could be regarded as less reliable.
* **Effectiveness, efficiency and reuse of prototypes and results**: if the underlying simulation system is built each time an experiment will be conducted, the efficiency of WOz experiments could be decreased. However, using a generic WOz tool would take care of this issue. When it comes to reuse of the prototypes and results, one should note that WOz is a rapid prototyping technique, or rather a throw-away prototyping technique. Thus, WOz is used to find the best possible idea or design, not to produce source code.
* **Ethical considerations**: it is common to hide the wizard and the fact that a human is actually composing the system’s responses to the test subject when conducting WOz experiments, i.e. one is deceiving the test subject into believing that he/she is interaction with a fully computerized system. Such experimenters must carefully deal with the situation, making sure that the method is not misused or that test subject is not put in a compromising situation.
* **Delays and time lag**: certain systems are not appropriate to simulate by using the WOz technique, such as action games, due to that a human cannot meet the demands on response time. Time lags and delays can be caused by the WOz system or the experimental set-up which of course should be kept in mind. However, delays and time lag does not need to be regarded as a big issue, especially not if one compares the WOz technique with e.g. paper prototyping.
* **Cognitive load – The Wizard’s tasks**: The wizard(s) undertake a large amount of stress during conducted experiments, depending on what the wizard is supposed to simulate and how the used WOz system supports the wizard. Some praise using a multi-wizard setup to resolve this problem.

The found limitations affect one another, more or less. Therefore it is not possible to entirely divide the limitations. For example, the reliability of WOz incorporated studies can be affected by any of the other presented limitations such as time delays or the wizard’s cognitive load.

Finally it must be noted that for some kinds of WOz studies some of the limitations are not really limitations, especially not in explorative prototyping, i.e. where the wizard tries responses not conceived in advance, or in demonstrations.

These limitations have more to do with how and if the technique is used, than with how a WOz system should be developed. Since the presumption of the present report is that WOz is used the above classification of limitations will not be elaborated on; instead system-specific problems will be highlighted.

## Structure of the report

In order to explain what a web-based solution for a Wizard-of-Oz setup means, this reports starts by explaining how the new version of Karlstad University’s Ozlab system works. This is done in section 2. The section then ends with evaluating the systems-dependencies of generic WOz tools.

Section 3 tackles the questions of how it is to actually work with the first release (summer 2013) of the web-based Ozlab.

Web-based technology entails not only using web browsers but also web addresses. A few interesting observations are made in section 4 on the implications of URLs on web-based WOz systems.

Conclusions are found in section 5.

# The Ozlab system

Ozlab is a WOz supporting system developed at Karlstad University. Ozlab may be used as tool for designing, testing, evaluating, experimenting and discussing graphical interfaces and interaction design, before effort is put into development in any programming language. The functions of the redeveloped web-based Ozlab system originates from an earlier Ozlab system based on Macromedia Director.

## Director-based Ozlab: System overview

In Ozlab no automatically functioning prototypes exists. The prototypes, or as called in Ozlab terms: the *interaction shells*, are manually controlled by a wizard. Pettersson (2002) argues that Ozlab “[…] supports explorative experiments in interactivity design by letting experimenters manipulate directly the output on the user’s screen. The focus is specifically on simple graphical human-computer interaction.” (Pettersson 2002, p.144) By using the Director-based Ozlab system the outcome is not program code. Instead, the user of Ozlab can design and test a concept with the intended end-users, before any programming is conducted. Doing so, Molin and Pettersson (2003) argue that Ozlab “can aid the process of formulating the requirements specification for multimedia systems” (p.78). Multimedia systems in this case refer to systems that “are characterized by the important role the system’s extrovert parts have. […] Such systems are, to a large extent, defined through their user interfaces.” (Molin & Pettersson 2003, p.70) The authors furthermore argue that “most information systems nowadays have their ‘multimedia’ parts” (p. 70).

The earlier Ozlab system was based on Macromedia Director 8.5 (or MX). To prepare and run a Wizard-of-Oz test several entities was used: Ozlab Testrunner, Ozlab Setup, Ozlab FileUpdater and a template file (.dir) with pre-programmed Ozlab-specific functions. (Siponen, Pettersson & Alsbjer 2002)

To build and design a prototype the template file would be opened in Macromedia Director. To design the interface of the prototype, the designer would add graphics, text, videos and pre-recorded sound to the library (called “Cast”) in the template file. As the Ozlab prototypes are designed in Macromedia Director the built-in tools e.g. drawing and writing could also be used to create objects. To make the prototype come alive, that is to function on another level than just communicating the interface via plain pictures, pre-programmed Ozlab-specific functions, called *behaviours*, could be used to add certain functionality to the dummy objects. Such behaviour could be e.g.: “objectMoveableByTP” which would allow the test participant (TP) to drag and drop objects; or “textFieldEditableByTP” which would allow TP to write text in input fields. By using the timeline in Macromedia Director (called “Score”) the designer could create different state changes, pages or as called in Ozlab: *scenes,* in the prototype.

To run an interaction design test in Ozlab Testrunner, the prototype file(s) needed to be copied from the wizard’s computer to the test subject’s computer. Further, the communication between the computers, handled by Macromedia Multiuser Server 3.0, needed to be established. These settings were configured in Ozlab Setup. Ozlab FileUpdater, using the settings from Ozlab Setup, was used to copy the file(s) from the wizard’s to the test subject’s computer, and after a redesign of an interaction shell only the changed files was updated to quicken time-to-test if changes were made while a test subject were waiting. When fully configured, Ozlab Testrunner was started on each computer, allowing an interaction design test or demonstration to start. (Siponen, Pettersson & Alsbjer 2002)

During the test or demonstration the wizard’s and the user’s interface was mirrored. In order to control and simulate the “system’s” responses the wizard had wizard-specific controls such as navigating to different scenes, opening new interaction shells, hiding/showing objects, pausing the test, etc. The test participant’s mouse cursor was duplicated as an enlarged cursor in the wizard’s interface, letting the wizard easily follow what objects the user interacted with (and therefore allowing the wizard to produce appropriate responses). User input was collected in a log which could be consulted during or after a test session.

### Director-based Ozlab: Usage and application

The Director-based Ozlab system was used during courses given at Karlstad University, and in several research projects. For example: Molin (2004) used Ozlab to design and evaluate a touch screen interface for a hip surgery robot, collaborating with the prospective user groups and designers; Pettersson (2002) report on the pilot study of Ozlab with inexperienced multimedia designers as wizards; Nilsson (2005) conducted user tests on a prototype of pedagogical software for children, using Ozlab; in collaboration with Swedish Civil Contingencies Agency several iterations of user tests were conducted on different aspects of a software, reported by Nilsson (2006) and Kilbrink (2008); Pettersson and Nilsson (2011, p.500) used Ozlab when assessing “code quality when it was either programmed based on mock-upped and user-tested designs, initially made from perceived needs by real users, or programmed only according to perceived needs by real users”; and Lindström and Nilsson (2009) used Ozlab as a usability testing tool in the PrimeLife project, and Pettersson and other in the initial year of the PRIME project (c.f. for example PRIME project, Privacy and Identity Management for Europe, a 6FP EU project; usability work reported in deliverable series D6.1a-d- www.prime-project.eu). For further examples of Ozlab usage, refer to Pettersson (2003) and the webpages about Ozlab[[2]](#footnote-2).

### Plans for redevelopment of the Ozlab system

After Macromedia was acquired by Adobe in 2005[[3]](#footnote-3), the Ozlab system risked to suffer from being based upon an outdated program. During a university course given at Karlstad University and in their bachelor thesis, Lamberg and Brundin (2011) researched potential solutions for redevelopment, by the following criteria:

* **[Support] Optimal workflow**: Creating graphical objects, adding functionality, Ozlab testing, (separately) editing graphics and functionality (p.33).
* **Software independence**
* **[Support] Naïve users**: “Users who have no prior knowledge of Ozlab, usability or user testing” (p.3)
* **Long-term sustainability**: “[…] maintenance and well structured source-code” (p.27)
* **Simplicity**: “[…] when handling the system there should be fewer steps to take when going from an idea to a finished prototype. […] Another thing that needs to be improved […] is that the user in the current system is often forced to make detours in the system to solve some really trivial problems in the workflow.” (p.29)
* **Functionality**: “a number of components that should be built into Ozlab, such as a simple way to make a drop list. Several users have stated that a certain amount of drag drop functionality to help create basic components would be useful.” (p.30)
* **Reusability**: of prototypes and images created in Ozlab (pp. 30-31).
* **Unconventional interfaces**: Ozlab should support testing of ideas and prototypes that are unconventional, i.e. “[…] new technologies such as interactive table interfaces or even mobile phones that could be twist- and bendable in the future.” (p.30))

Lamberg and Brundin (2011) suggested several bases for a redeveloped Ozlab: Ozlab could be based on Adobe’s Photoshop, an HTML5 editor, or XML. Though, the authors found that building Ozlab based on Photoshop or an HTML5 editor would make the system dependent on software and certain file types, as well being less accessible for naïve users. The authors argued that Ozlab should be based on XML, since this would best support the workflow as well as the previously listed criteria (2011, p. 42).

The suggestion of developing the new Ozlab system as an XML solution was not fully rejected. As shown in a report by Lamberg (2011), the XML solution would in fact not be a pure XML solution. Javascript and other programming languages would, actually, be needed. Thus, for the ongoing re-development another mark-up language was chosen as the main one: HTML5 combined with Javascript.

## Web-based Ozlab version 1 - design and implementation

The first version of the web-based Ozlab system (in the following sections referred to as the web-based Ozlab system, or the web-based Ozlab system version 1) uses the following techniques and frameworks: MVC 4.0[[4]](#footnote-4), IIS 8[[5]](#footnote-5) with WebSockets[[6]](#footnote-6), JQuery[[7]](#footnote-7) and Sencha Ext JS 4.2[[8]](#footnote-8). Ozlab can be accessed via any web browser but runs best in Google Chrome.

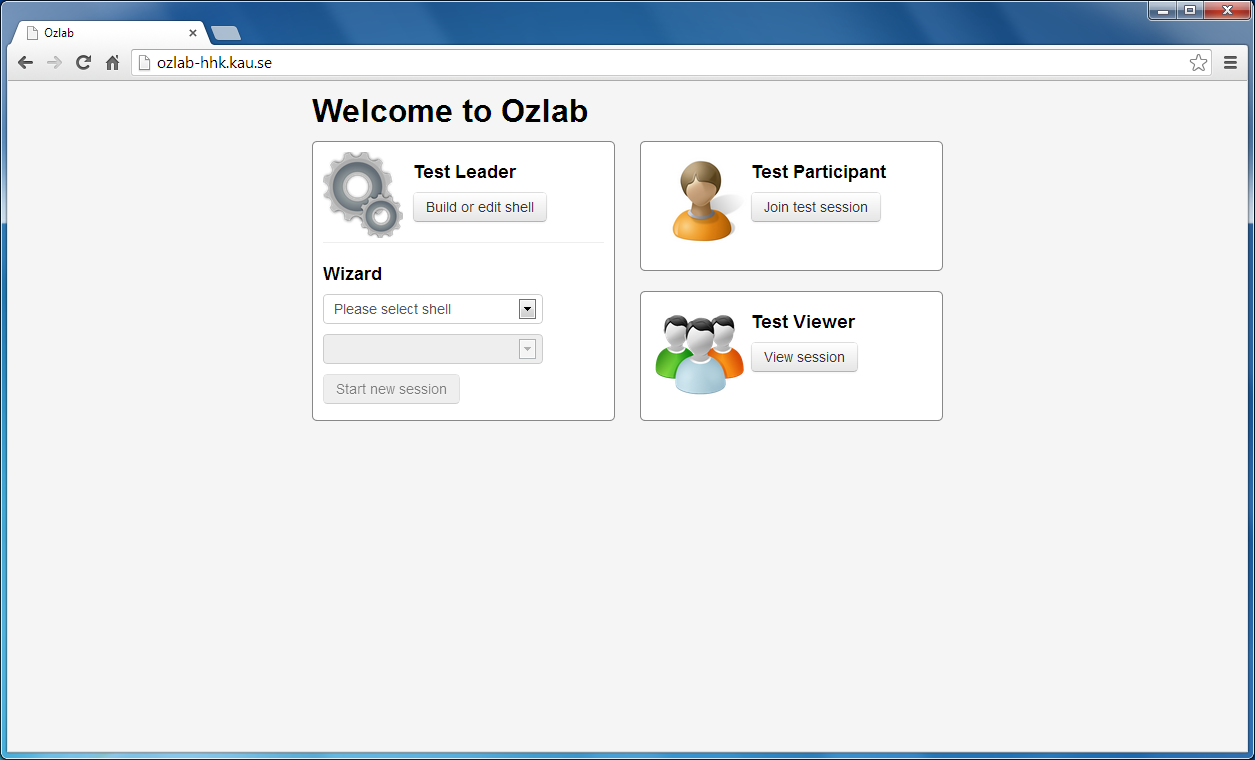


Figure 1. The landing page of Ozlab (version 1).

Ozlab consists of three entities: Shell Builder, Test Runner and Test Viewer. When accessing Ozlab in a web browser the user ends up at the landing page (see figure 1) where the Ozlab user can choose between four roles: As a Test Leader the user can “Build or edit shell” (which will start the Shell Builder); Start a test as a Wizard by choosing a shell and scene in that shell (this will open the wizard’s view of the Test Runner); join a test session as a Test Participant (this will start the user’s view of the Test Runner); or view a test session by starting the Test Viewer.

The distinction between the role Test Leader and Wizard is made since when building and editing an interaction shell in the Shell Builder, no “wizardry” is going on: the user cannot see the interaction shell or any changes the Test Leader makes in the shell.

Shell Builder, Test Runner and Test Viewer can be run on the same computer but in different web browser windows, allowing the designer to easily preview the interaction shell. The current implementation allows one wizard, one test participant and several Test Viewers to be connected to the same session.

The test leader and wizard are advised to run Ozlab on a large screen to best support the workflow.

### Shell Builder – designer’s/test leader’s workspace

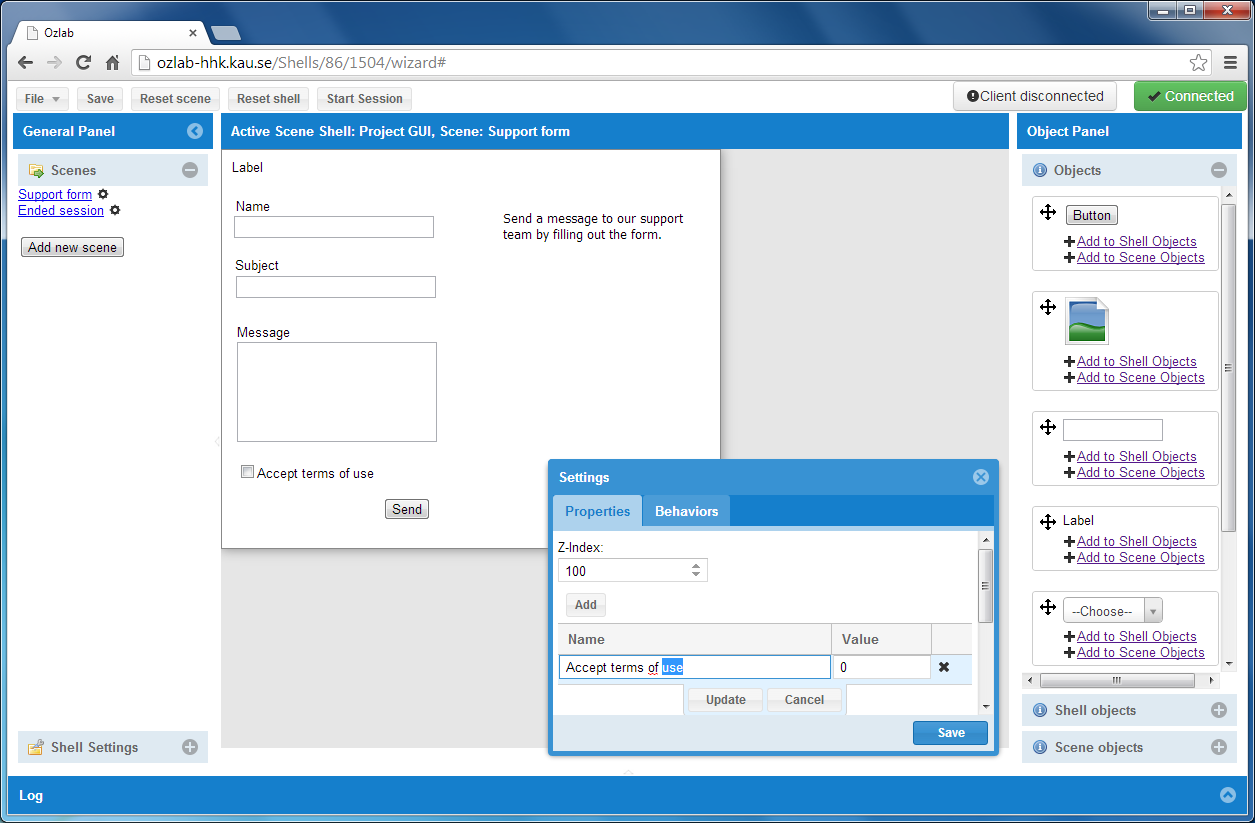


Figure 2. The Shell Builder interface as seen by the Test Leader (version 1).

An interaction shell can be designed and edited in the Shell Builder (see figure 2). To the left a General Panel is placed where Scenes are listed and general Shell Settings available. In the middle of the interface the chosen scene is displayed. The size of the scene can be altered under Shell Settings. In figure 2 the scene area (which is what the user will be able to see during test) is 500 by 400 pixels (displayed as a white sheet). The designer can add generic objects (seen in Object Panel to the right) to the interaction shell by dragging and dropping them from the panel to the scene area. In the current implementation, 7 generic objects are available: Button, Image, Input field, Label, Dropdown menu, Radio button and Checkbox.

Complex or commonly used objects can be added to either Shell objects (available throughout the whole interaction shell) or Scene objects (available at the current scene only) for re-use.

Ozlab provides the wizard with a set of Behaviors that can be added to objects. In version 1 of the web-based Ozlab system 7 Behaviors exist: *GoToScene* provides an automatic link to another scene in the same shell; *MakeObjectSnap* will center a draggable object over a snap point; *ObjectDraggableForTL* and *ObjectDraggableForTP* allows Test Leader and/or Test Participant to drag the object which has the behavior*; OpenLink* will if clicked open a link to an external website in the browser window; *SendAudio* allows pre-recorded audio to play; and *Vibrate* will make an Android device vibrate.

### Test Runner

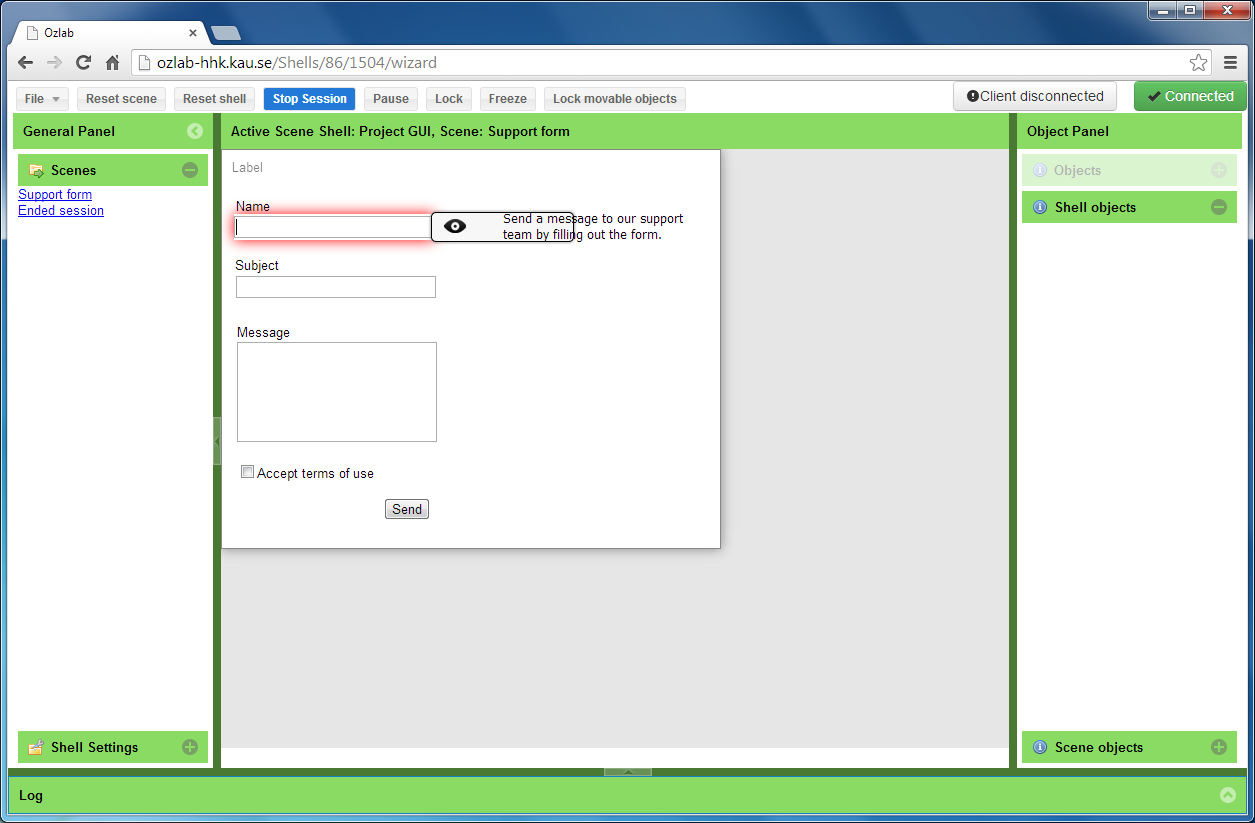


Figure 3. The Test Runner interface as seen by the wizard, in version 1 of the web-based Ozlab system.

The Test Runner is where tests are conducted in the Ozlab system. Setting the Test Leader side of Ozlab in a test state can be made directly from the Shell Builder, by clicking the button “Start Session”. Starting a test can also be done directly from the landing page by choosing the Wizard role. The wizard’s interface has the same overall look in Shell Builder and Test Runner, why a green color is used to differentiate between states (see figure 3). The wizard can follow the user’s actions in real-time as they appear in the scene area (in the middle), where also the user’s cursor is duplicated in an enlarged version, followed by a fading trail. User actions that can be followed are for example clicks, text inputs, dropping after dragging and right/left click.

Objects that have been added as Shell or Scene objects can be dragged and dropped to the scene. The wizard cannot change object’s settings during runtime, but all objects can be hidden for the user (by clicking the eye icon as displayed in the middle of the scene area). The wizard can choose from a set of functions at the top of the interface. By clicking *File* an interaction shell can be opened; *Reset scene* and *Reset shell* will reset all objects to their original state in either the active scene or the entire shell; *Stop Session* will terminate the test session and take the wizard back to the *Shell Builder*. The remaining four buttons allows the wizard to: show a black waiting screen for the test participant (*Pause*); show the test participant’s cursor as a wait cursor (*Lock*); make it look like the test participant’s cursor is entirely frozen (*Freeze*); and lock all movable objects for the test participant (*Lock movable objects*). Pause and Freeze allows the wizard to make changes in the scene or navigate to a different scene, without displaying the changes for the test participant. The test participant cannot continue to interact with the interaction shell during these states.

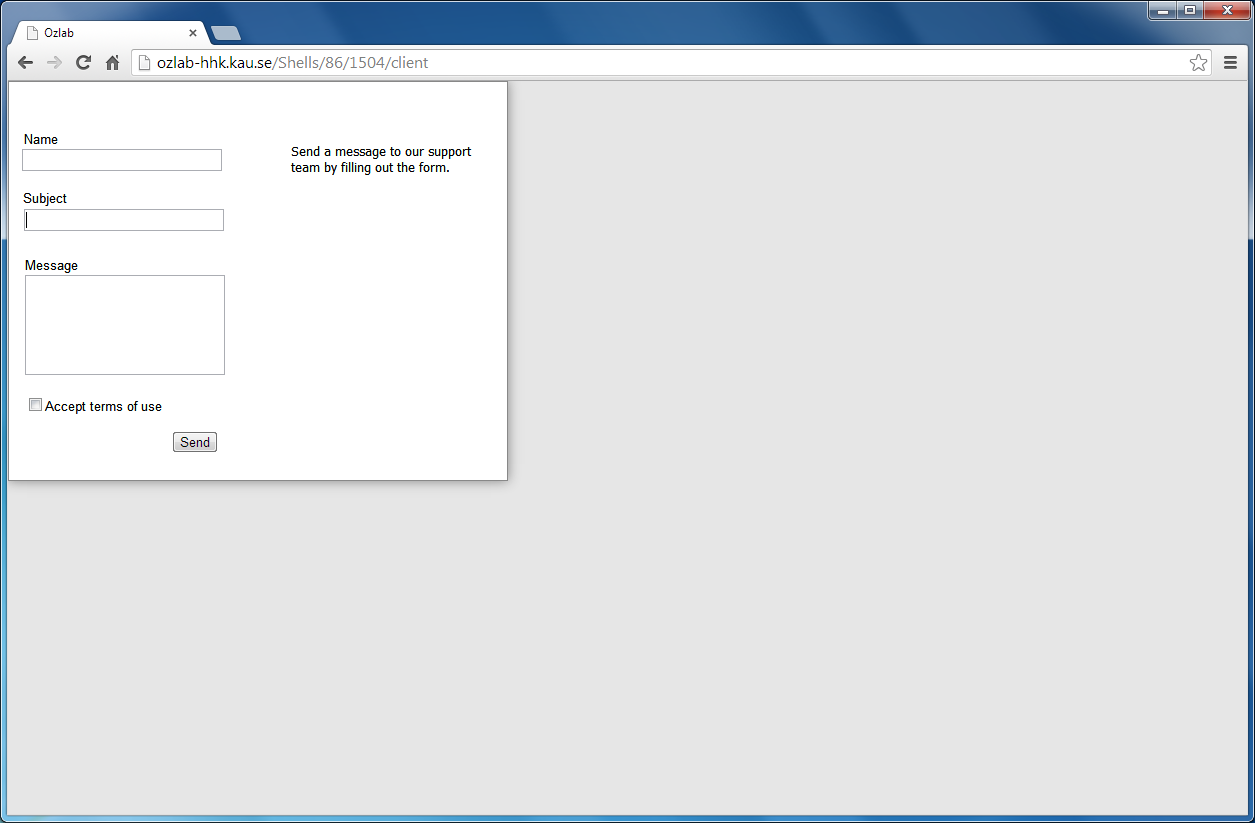


Figure 4. The Test Runner interface as seen by the test participant (without the browser controls hidden).

When running the Test Runner as a test participant, the interaction shell is displayed without the wizard’s controls (see figure 4). An interaction shell as shown in the figure where the scene area is smaller than the browser window, will display a grey area outside the scene area. Objects added by the wizard in the grey area will not appear in the test participant’s interface. The browser can be run in full-screen mode if the test leader wants to hide the chrome for the user, i.e. hide every trace of the window being a web browser’s window.

### Test Viewer

Test Viewer works like a video monitor as it shows running sessions without letting the persons looking engage in the on-going interaction. It displays the interaction shell and participant’s enlarged cursor without any wizard controls.

## Systems-dependencies in generic WOz tools

Unlike paper prototyping, a shared issue amongst the generic WOz tools is that they are vulnerable in a longevity perspective. The intention with this subsection is to show how different aspects of this perspective affect a tool.

**Tools based on a program**

A WOz tool that is based on a specific program is open to vulnerability if the program is either updated or gets outdated. The updates of the program can end in a mismatch between the WOz specific functionality and what input/output the underlying program accepts. If the underlying program gets outdated, it may become impossible to run, install, etc., as the program is not supported by the distributor.

For the Director-based Ozlab system, also described in section 2.1, this became an issue as Macromedia Director was acquired by Adobe and not further supported by Macromedia.

**Tools depending on a specific programming language**

Vulnerability can be an issue for WOz tools that depend on a specific programming language. One example is that of the Director-based Ozlab system which was dependent on not only Director, but also LINGO which is the programming language by which functionality is programmed in Macromedia Director. The programming language would decrease the chances of further developments, as the programming language is not commonly used.

This issue seems related to all generic WOz tools presented in Appendix 2.

**Tools adapted for tests on a specific platform** (as Android devices)

Some of the tools are adapted for running tests and prototypes on a specific platform. That is, the WOz tool makes use of specific possibilities granted by the platform or is adapted to specific limitations of the platform. Naturally such adaptation is needed if one wants to conduct tests on specific platforms. For example, if conducting tests on handheld devices which make use of touch input, the tool needs to be adapted to interpreting such user inputs.

Adapting the WOz tool to a specific platform, however, makes it vulnerable in the sense of durability. If the platform is redesigned, e.g. in what inputs it allows or what output it enables, the generic tool needs to be adapted to the redesign.

In tools such as WozARd by Alce et al. 2013, and the Android tool by Linnell et al. (2012) this vulnerability could be an issue, even though it is hard to determine from just the articles.

### Generic web-based WOz tools

Schlögl, Doherty, Karamanis and Luz (2010, p.113) argue “[…] by having a fully web-based implementation of a WOZ framework we would be able to offer new possibilities when it comes to running WOZ based user studies. That is, in theory it would not matter anymore whether a wizard is hidden next door or actually works from a different country since the framework providing the interfaces for the different parties and collecting the data would live online.”

As argued by Schlögl et al. a web based Wizard-of-Oz system enables remote experiments to be conducted. If the system furthermore is accessed and run in a web browser, the set-up difficulties and dependency to a certain platform is decreased:

“Existing WOZ and DM [Dialogue Management] tools mostly require a certain platform dependent configuration of the host system in order to run smoothly. Also they typically need an installation routine and a very specific experiment setup (i.e. several computers acting as clients and servers, multiple screens, cameras, microphones, etc.)” (Schlögl, Doherty, Karamanis and Luz 2010, p.113)

In order to conduct tests with the Director-based Ozlab system, especially if using “Mini-Ozlab”, one had to go through a cumbersome environment set-up. Several steps needed to be slavishly followed to be able to run tests (see section 2.1). In order to allow the communication and writing/reading of files between the two computers used in the tests, firewalls and sometimes sharing settings had to be modified.

The Director-based Ozlab was furthermore platform and software dependent, as it had to be run on computers with Windows installed and needed Macromedia Director 8.5 or MX as well as MultiUser Server (as described in 2.1).

The web-based Ozlab system is less platform-dependent than the Director-based Ozlab. Supposing the web-based Ozlab system is already set up and running on an IIS 8 web server, the wizard only needs a computer with the web browser Google Chrome installed to access and use Ozlab.

The web-based Ozlab system is intended for experiments on graphical interaction, just as the Direct-based Ozlab system was. Even though the web-based Ozlab does not support integration of language technology components as in e.g. WebWOZ (Schlögle et al. 2010), or is specifically developed to support design and tests on speech user interfaces as in e.g. SUEDE (Klemmer et al. 2000), experiments on speech interfaces in Ozlab can be conducted by using a microphone and/or voice disguiser (see e.g. the very first experiment with the first version of Ozlab; Pettersson, 2002). However, this would be a setup outside of the actual WOz system setup.

One could argue Ozlab prototypes to be runnable on every device which can run a modern web browser. However, this statement is not entirely true. The web-based Ozlab system is software dependant when it comes to which browser that renders the system properly, namely Google Chrome. In addition to Android devices, Google Chrome is available for iOS devices. Though in the case of iOS devices, the web browser engine[[9]](#footnote-9) WebKit must be used on due to Apple’s rules.

“The iOS version is different to that, because Apple's iOS rules dictate it has to use the WebKit engine used for Apple's Safari browser.”[[10]](#footnote-10)

Chrome and Safari used to run the same web browser engine, but since April 2013 Chrome instead uses Blink[[11]](#footnote-11), while Safari continues to use WebKit[[12]](#footnote-12). This means that even though the look and feel of a Google Chrome browser is the same on iOS devices as on Android devices, the, interpretation, rendering and display of mark-up language and formatting information are executed differently.

In addition to the Google Chrome dependency, the framework used for the interface of the web-based Ozlab system is based on Sencha Ext JS. To run the web-based Ozlab system Windows Server 2012, namely IIS 8 with web sockets, is furthermore needed.

These software dependencies can make the web-based Ozlab vulnerable to the listed issues in this section. For an elaboration on limitations of the web-based Ozlab system, see section 3.5.

# Being wizard in the web-based Ozlab

Ideally WOz would’ve been used when developing the web-based Ozlab system, as requirements of multimedia systems are hard to define (Pettersson & Molin 2003). Though, that would demand a wizard simulating a wizard interface for another wizard, who in his or her turn simulates a prototype for a user. Instead of this cumbersome setup, the researchers met developers a few times during development to accept or reject implemented functions, or negotiate function specification. Finally, when the developers found the specified system complete, the system was first tested for bugs and missing features by the researchers, and secondly the wizard interface of Ozlab was evaluated during an undergraduate course at Karlstad University.

## Incentive

Prior to the university course evaluations the researchers were focusing on if the delivered system met the requirements. During these initial tests bugs and missing features was found. Though, as later was shown, not all errors, interaction complications and bugs were identified. One could argue that this might have something to do with the fact that the researchers were also involved in the development process, which could have made them blind to some of the errors identified later. Further, the evaluation made within the course showed that some students adapted their behaviour and interaction patterns to the system, avoiding the bugs and errors. The researchers might have adapted to the errors in the initial tests as well. Furthermore, throughout the evaluation during the course the Ozlab system was extensively used by up to four groups at the same time, a procedure not conducted by the researchers. This could have brought more system errors to the surface.

Note that even if each experimenter gets his own web address for his interaction shell, i.e. each group uses their own web site to design their mock-up; the web server’s work-load is increased as four groups are connecting to the server simultaneously.

Letting the students use the unpolished Ozlab system was met with some frustration. But, by observing the students, who were both naïve designers and novice Ozlab users, use the system the bugs and interaction problems became clearer. The students’ willingness to adapt to an incomplete system could be explained by the fact that they took a course where the task was to create and conduct tests on an interaction shell. Some of the students acted with huge patience, while others who did not adapt to the system’s faults ended up frustrated. By letting the students use the unfinished system, many real problems were explicated in a few weeks’ time. The researchers’ and the students’ adaptation show that the new system would not have been scrutinized thoroughly had only a single test group been used.

## Course case procedure

The course aims to introduce information systems as a discipline but focus on human-computer interaction, through practicing how prototyping can be used as a tool to communicate and develop design ideas. During the course, the students used Ozlab to design and evaluate a Connect Four[[13]](#footnote-13) game from a set of (“naturally” vague) requirements:

"Connect Four is a computer game where the player can play against the computer according to the usual rules. The colours of the game pieces should be green and red to match the company's logo. Once the player has selected a colour the game starts, either the player or the computer begins. If someone gets four in a row the computer notifies who won. There should be a help function. But probably the player shouldn’t be able to get help during an ongoing game? There must also be a maximum time for consideration. And if the player will be able to choose from a number of times for consideration, the player must do this before the party starts." (Translated from Swedish):

The requirements are intentionally ambiguous, inviting the students to motivate their design choices. The students were divided into 12 groups of 3-4 people each. All groups were given an introduction to the Ozlab system and then had the opportunity to develop their interaction shells in the usability lab while given supervision.

After two weeks of designing their interaction shells the groups conducted one pilot test each. After a day or two these pilot tests were followed by “real” tests with three test participants for each group, resulting in a total of 36 test sessions. The sessions were conducted in the usability lab at Karlstad University that consists of three rooms: a control room, a test room and a reception room. In the control room the wizard could observe the test participant through a mirrored glass wall and hear what is being said in the test room thanks to a microphone placed in the test room. The wizard could also see the participant’s screen via a duplicated screen.

## Lessons learned from the first set of web-based Ozlab wizards

The students used one or a combination of the following tools and techniques when producing the images used in their interaction shells: Photoshop, MS Paint, Adobe Flash, paper sketches (which were scanned), digital photos and/or images fetched from the web.

Most groups let the same person act wizard during their test sessions. In order to analyse how the wizards interacted with Ozlab each person acting wizard were asked to make a screen recording.

### Wizard’s interaction patterns

Acting wizard can mean to undertake a large amount of cognitive load and stress, as revealed in the literature review conducted prior this report (see section 1.2 Literature review). In the case of the students the stress of acting wizard were especially present for those who did not practice the wizard role (enough) prior conducting the tests. (That is, prior to pilot test and “real” test; such practicing could of course also be labelled “pilot testing” but the Ozlab team, with long experience of WOz, tend to reserve the word “test” including “pilot test” to cases where more or less “real” users participates, that is, when participants are not from the design group).

The screen recordings (see summary in Table 1) show that some groups had not practiced the possible navigation paths through the interaction shell or discussed how the wizard should act and respond to potential ways a participant could interact with the prototype. These observations support the endorsement for increasing the wizard’s knowledge of the simulated system and available information prior to conducting the simulations, as recommended by Dahlbäck, Jönsson and Ahrenberg (1993).

However, when analysing the screen recordings from the test sessions, it became clear that the time needed for the students to learn how to act as a wizard (that is for example where to focus attention, in what way the participants might interact with the prototype and what responses one must be ready to give as a wizard) was short. Often the role of being a wizard was refined already during the first session. For example, in some interaction shells the test participant could not move to new “scenes” without the wizard simulating a scene change, why for a hasty change of scenes the wizard should focus the mouse cursor over the list of scenes to the left of the interface. Though, during the first test session several wizards held the mouse cursor in the scene area, close to where the test participant interacted with the prototyped interface. Later during the same session, or in following sessions, the wizards seem to have learnt that if lingering the cursor over “wrong” areas, one must move the cursor to the scene list, locate the link to the scene corresponding the test participant’s interaction and then click the scene to display the scene change. Of course, the amount of time for doing this is not extensive, but could be argued to be more stressful than to just click the link to go to the expected scene.

Even if the time for learning how to act as a wizard was shown to be short in the case of the students, learning as the experiment goes on may result in inconsistency of how the wizard simulate the responses of the system. Inconsistency, as pointed out by for example Dahlbäck et al. (1993) (and discussed above), might affect the reliability of the results.

During one test session the user was presented with the page which indicates that the wizard has not started the test session yet (see figure 5). The test participant then commented “Who is the wizard then? Is it [name of person in the test group]?” Perhaps the test participant did not realize what wizard in this case meant, but even so, revealing the test method is sometimes unwanted, why the waiting-to-start page should not contain any unwanted methodology clues.

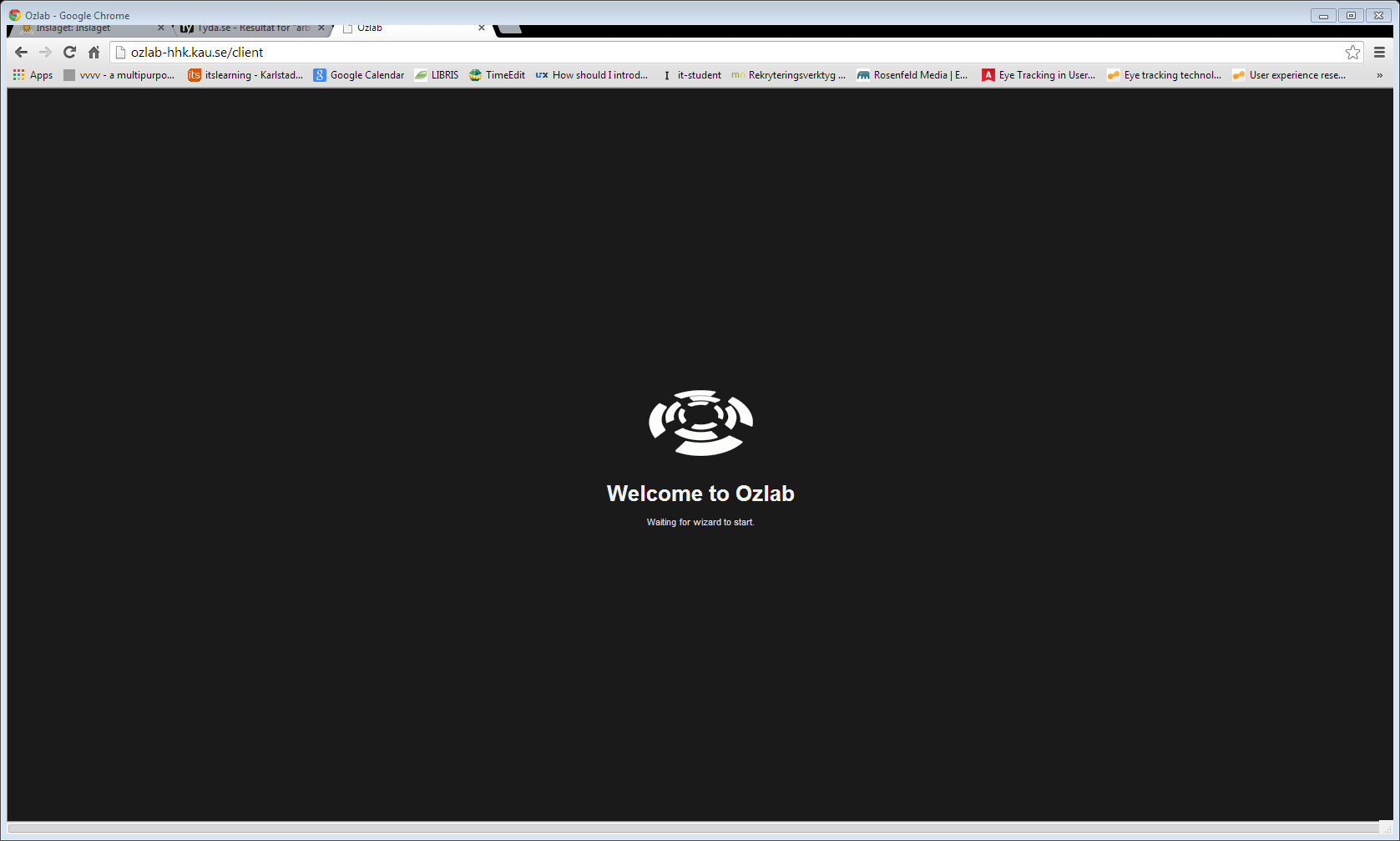


Figure 5. The page displayed at TP’s computer before the wizard has initiated the test session, saying “Welcome to Ozlab Waiting for wizard to start”

In Table 1 a summary of the observations from the screen recordings of the wizard’s interface are shown. Observations which are seen as Ozlab specific rather than WOz specific are marked in italics. The recordings of group G were hard to analyse, as the Test Viewer was started on the test participant’s computer instead of the user view of Test Runner.

Table 1. Wizards’ interaction patterns.

|  |  |  |
| --- | --- | --- |
| **#** | **The Wizard… /The human shell builder…**  **[**TP = test participant] | **Wizards from group X (number of groups)** |
| 1 | Holds the mouse cursor in “wrong places”. | B, A, D, I, K (5 groups) |
| 1b | Learns to hold the mouse cursor over scenes/objects that TP probably will interact with. | B, A, D, I (4 groups) |
| 2 | Includes automatic links between scenes in the interaction shell (links which TP can use to change scenes). | A, C, E, F (4 groups) |
| 3 | Navigates to unexpected scenes (in relation to what the TP interacted with). | B (1 group) |
| 4 | *Uses the object itself to drag and drop, instead of the “move icon” on the tooltip.* | B, A, C, D, E, H, I, G, K, F, J, L (12 groups) |
| 5 | *Uses the “move icon” to drag and drop objects.* | I (once), K, J (from object panel) (3 groups) |
| 6 | Included “Quit” in the students’ shells as an available option for TP, but no scene indicating that the game is terminated is included in the interaction shell. | B, A, K, J (4 groups) |
| 7 | *Forgets to reset scene(s).* Resetting a scene in this case was important when the game was supposed to re-start. Without the reset, all the pieces on the board were still showing from the previous game. | B, D (between shell switching), E, G, K, F, J (7 groups) |
| 8 | *Clicks the object itself to show/hide, instead of the eye-icon in the tooltip.* (TL could believe that he/she must click the object to show the tooltip, or TL could believe that hiding / showing objects is possible by clicking the object itself) | D, C (2 groups) |
| 9 | *Starts Test Viewer instead of “Join as participant -> Test Runner” at TP’s computer.* | G (1 group) |

As argued in section 1.2, inexperienced users of the Wizard-of-Oz technique might not benefit from using a WOz system or implementing the technique all by themselves. As shown by #1-3,6,7 in Table 1, this assumption seems to be true. #4-5 and #8-9 shows that some adjustments to the wizard controls and the Ozlab landing page (see section 4.1) should be made.

## Identified bugs and interaction problems in the first version of the web-based Ozlab system version 1

When observing the students use the web-based Ozlab system to design their Connect four prototypes during the university course, a lot of bugs and interaction problems could be identified. The most occurring problem was a reappearing connectivity issue (“Trying to reconnect”) which appeared in both Shell Builder (most common) and Test Runner (less common). Some groups learnt to avoid it (and some didn’t experience it). It was later found to depend on a programming bug.

Even if several interaction problems were known before the course started, it was a valuable exercise for the students to report the problems they experienced and a valuable source of information on where certain implantation short-cuts “hurt” as the rather extensive lists below shows. The web solution made some function specifications from the old Ozlab less unambiguous and some were too expensive to implement. Therefore feedback of perceived usefulness and lack thereof was important for targeting revisions of the web-based Ozlab.

**Identified interaction problems relating to the Shell Builder:**

* Cannot move several objects at the same time.
* Dialogue “Are you sure…” appears every time an object is being deleted.
* Cannot delete several objects.
* Cannot add behaviours to several objects at once (already on the scene area).
* Behaviours need to be added in specific order.
* The object “Label” is being used for embedding iframes (illogical).
* No possibility to “undo” actions.
* Dragged and dropped at the scene area directly from the generic Objects cannot be reused, why the designer must add common objects directly to Shell or Scene objects if re-use is expected.
* No support for keyboard commands for actions, e.g. save/undo/copy/paste/cut.
* Videos (movie files) cannot be uploaded as an object, directly from the computer.
* Radio buttons/check boxes cannot be shown as a horizontal list (only vertical).
* No possibilities to add “invisible” (at least for TP) fields for interaction (for simulating scene links or “clickable” objects)
* Cannot “lock” objects (to avoid unwanted deletions or moves).
* Cannot change the order of the scenes in the Scene list.

**Identified interaction problems relating to the Test Runner:**

* Objects are not “snappable” when dragged from Object panel and directly dropped over a snap point on the scene.
* Cannot group several objects (for simultaneous moving, hiding/showing).
* Tooltip for hiding/moving objects is shown not quick enough and overlapping objects make the tooltip hard/impossible to use.
* Cannot add new objects during runtime.
* No change of TP/TL’s mouse pointer over simulated links (works only for links added in the Label object).
* TL/TP cannot follow/see objects being dragged until they’ve been dropped.
* Drag and drop doesn’t work on handheld devices.
* Large/high resolution images loads slowly.
* HTML objects (i.e. input fields, drop down menus) are duplicated which for example makes it hard to follow what the user is writing in a text field, as the input is not shown until the test subject leaves the input field.

## Limitations in the architecture of the web-based Ozlab system

### No multi-wizard setup

Several wizards/users cannot be connected to the same Ozlab site at the same time, why a multi-wizard setup is not supported through a single shell. Though, several sites have been created which allows several users to simultaneously connect to Ozlab’s Shell Builder and Test Runner. The wizard and test subjects can alter the URL to connect to the different sites.

### Cannot add content during runtime

The first version of the web-based Ozlab system does not allow the wizard to add new content to the interaction shell during run-time. This is a problem for experiments where such functionality is needed, for example when conducting explorative tests. Adding content during run-time is one idea for further development of the web-based Ozlab system. Until such functionality is added to Ozlab, a possible work-around has been found, see section 4. It should be noted that the work-around might not be applicable to all experiments.

### Connectivity and security

As the new Ozlab system is accessed through web browsers over internet connections, the connectivity could be an issue when using Ozlab. If conducting experiments in environments where the connectivity is less reliable (or non-existent), one solution could be that of running the server locally. The TP device would then connect to the wizard’s computer through some local connection. This solution should though be further tested, as well as how “bad” internet connection the Ozlab system can handle.

With a web-based Ozlab system security issues follows as well. Not only from the point of needing to protect the system and the network on which the system is running. Using firewalls also limit the access possibilities for “friendly” accesses. This could be an issue if access is granted through specific ports, and the same ports are blocked in the network from which the wizard is connecting.

### Browser panels and browsing

Without running Google Chrome in full-screen mode, it seems like hiding the typical browser panels and controls are hard to accomplish. Hiding the typical browser window properties is essential when running mock-ups in a web-browser, especially if conducting tests on a prototype of a system which is not intended to be viewed in a browser window. Though, hiding the browser controls is important when running a test of a mock-upped web site or web application as well. Otherwise the test participant can navigate to other than the experiment web page or close the browser window, which obviously will put the wizard out of control over the experiment.

### Intended lack of responsiveness

The implementation does support experiments conducted on different devices as the interaction shell runs in a web browser. However, the interaction shell does not automatically recognize the size of the screen on the device, on which the interaction shell is running. This means that interaction shells built in Ozlab is not responsive and does not adapt to the device on which the interaction shell is running. Instead, if the content of the interaction shell is too big for the device’s screen scrollbars shows up in the browser. This means that when designing and building the interaction shell, the designer must take the screen size into consideration. But, if the system does account for and adapts to different puppet device sizes automatically, the wizard would lose control over how the design is displayed. Such automatically adaptation would in fact make tests of several design solutions hard, and defeat the whole purpose of conducting WOz tests with Ozlab.

### No automatic scrolling on the wizard’s side

Graphical interfaces might be continuing below “the fold” (as Steve Krug (2006) calls the “hidden” content which the user only gets to by scrolling or scanning down the page), i.e. the content might be higher than the device screen’s height. When the interaction shell is larger than the screen of the device, and the user is scrolling in any direction, the wizard must manually scroll the wizard interface to be able to see what the user is viewing and interacting with. On a computer setup this is easy, since the wizard can see the user’s pointer at all times. On a handheld device which implements pen or touch input, following the user’s interactions and scroll through the interaction shell is not as easily accomplished for the wizard. For obvious reasons, on such devices no mouse pointer is shown for the wizard when the user is not interacting with the interaction shell as the user scrolls by using his/her finger or a pen.

So this is an idea for further development of the Ozlab system: the TP device signals back to the Ozlab system what part of the scene is visible every millisecond to the TP.

## Evaluation of workflow and related matters

If evaluating the web-based Ozlab system according to criteria by Lamberg and Brundin (2011) which was listed in 2.1.2, it is clear that several of the criteria are met, whilst some are not:

**[Support] Optimal workflow**: The web-based Ozlab do support the “optimal” workflow as defined by Lamberg and Brundin (2011). However, the authors’ optimal workflow does not include the possibility for the wizard to add content during runtime, which should be included (but is still not implemented in the first version of the Ozlab system).

**[Support] Naïve users**: The naïve users in the course case (section 3) could create prototypes and run tests by using Ozlab. Much of the support needed was due to bugs and connectivity issues described in section 3.4. Without these issues the naïve users could have used the Ozlab system to a large extent by themselves, as they created their prototypes and ran their test sessions rather independently. However, it should be noted that naïve users might not benefit from using the system without any kind of support, as the Wizard-of-Oz technique is not appropriate for all kinds of system simulations. Furthermore, the limitations to the WOz technique listed in 1.2 might not be clear to naïve users, but should be regarded when preparing a WOz experiment.

**Simplicity**: As described in section 2, the web-based Ozlab is simplified when it comes to workflow as well as set-up. However, if the server needs to be set up, then it will be much more complicated.

**Functionality**: As shown in section 2.2 the first version of the web-based Ozlab system has a number of built-in components and does allow the wizard to drag-and-drop objects when creating an interaction shell.

**Reusability**: it is possible to reuse interaction shells in Ozlab by importing shells into other shells. Images developed for the interaction shell can of course be used elsewhere, but would then miss added objects such as dropdown menus or input fields. The system intentionally do not support any export of programming code, as Ozlab is intended to be used as a means of throw-away prototyping.

Some issues with the web-based Ozlab’s **Software independence** and **Long-term sustainability** have been recognized in section 2.3. Any system which is used in a context which is currently and continuingly changing is though vulnerable when it comes to these criteria.

However, it should be noted that even though the long-term sustainability is increased for the web-based Ozlab, but the documentation should be further improved to ease future maintenance.

When it comes to the criteria to support **Unconventional interfaces**, which is related to the discussion in section2.3, it is hard to say whether the Ozlab system fulfils this criteria as one cannot foresee the future. Though, the web-based Ozlab system should at least be able to support unconventional interfaces better than the Director-based Ozlab system, as mock-ups can be run on any device with Google Chrome installed.

# Implications of URLs on a web-based WOz system

During the course case, technical tests and a pilot study in field a few observations on how URLs can affect the Ozlab system, and how URLs can be used to circumvent the issue of not being able to add content during runtime was made.

## Re-design of the Ozlab landing page

The landing page of Ozlab version 1 is not adapted to if a designer, wizard, test participant or test viewer is accessing the Ozlab system. During the course case (see section 3) it was found that the students had some problems understanding how to open the Shell Builder and how to start a test.

Two students proposed a re-design of the web-based Ozlab landing page during their internship. Their idea was that the URL would determine which view of Ozlab the user was presented with. If accessing the Ozlab system with the default URL, a landing page adapted to test participants and test viewers would be shown (see figure 5). If “/wizard” or “/tl” (as in Test Leader) was added to the end of the default URL, a landing page adapted to test leaders and wizards would be shown (see figure 6). Two separate landing pages would decrease the potential risk of giving away the test method (i.e. Wizard of Oz) to the test participant and perhaps more importantly, simplify the graphical interface for test participants as well as test leaders. However, with the web-based Ozlab system version 1, the test leader must already remember a specific URL to which site the experiment is supposed to be conducted (more about this limitation, see the following section). Adding another specific suffix to that URL could complicate the use of the Ozlab system.

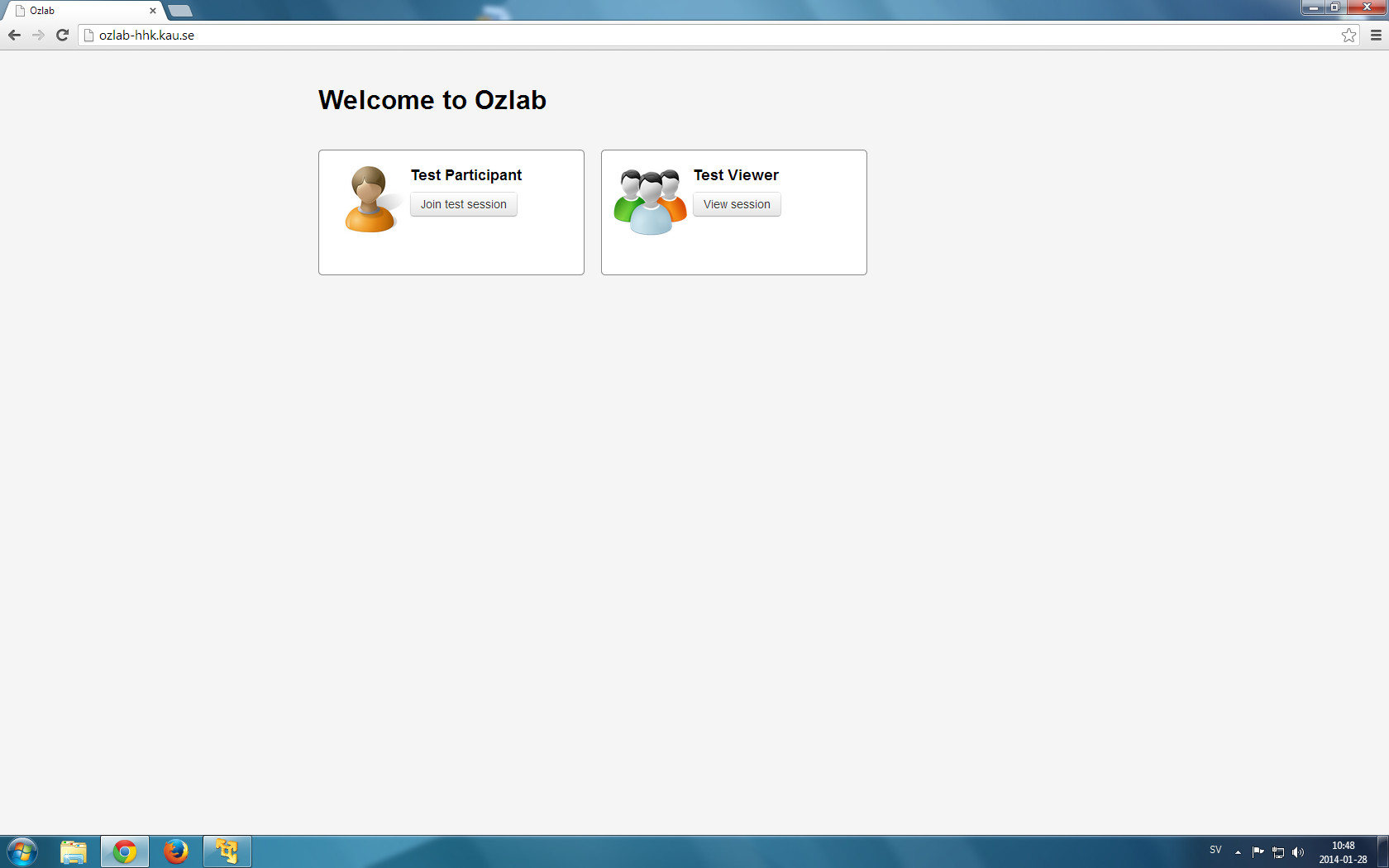


Figure 5. The proposed default landing page.

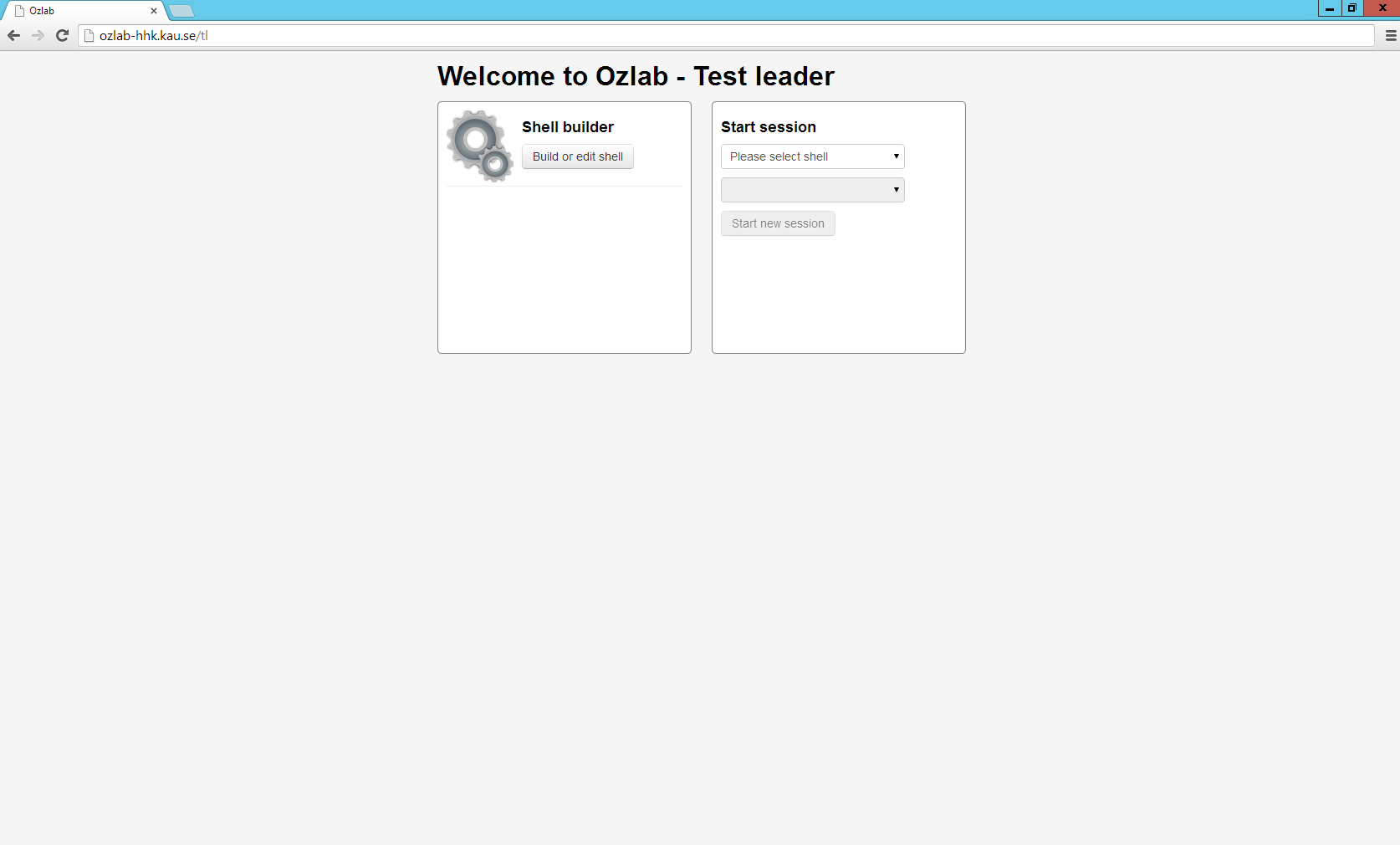


Figure 6. Proposed landing page for the Test Leader, by using a modified URL.

However, presenting different landing pages to different roles was argued less important, as test participants seldom join an ongoing test session by themselves. Instead test participants are most often presented with the wait-screen (as shown in figure 5 in section 3.3) when beginning an experiment. Furthermore, in order to avoid complicating the use of the Ozlab system by specific URLs, another re-design of the landing page was proposed and implemented (see figure 7). The landing page in figure 7 is divided in two columns depending on what the Ozlab user wants to do: (1) conducting a test (column Ozlab Test Runner) or (2) build an interaction shell (column Ozlab Shell Builder). The landing page is responsive, meaning it adapts to what kind of device the Ozlab user is accessing the page by. As it was argued that one seldom would build or edit an interaction shell on a mobile device, this column was placed to the right. Therefore, the right column will appear after the left column when accessing the landing page by using a mobile device.

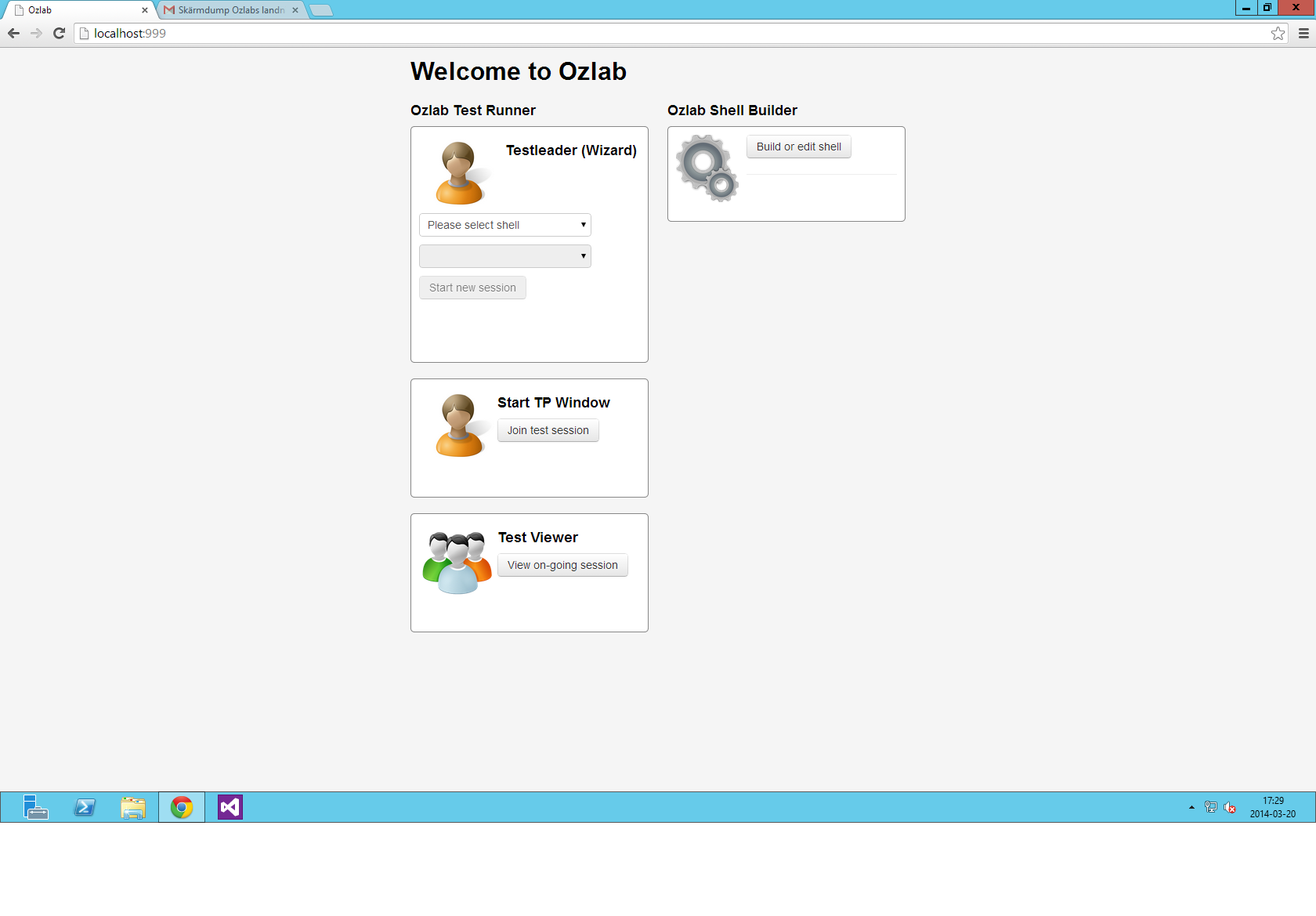


Figure 7. The implemented landing page of the web-based Ozlab system.

## Considerations of URLs in multi-shell sessions

Three solutions to adding content during run-time have been considered in this study.

The background to the “run-time addition of content” requirement is the following thoughts. In order to let visitors to an event, a cultural heritage or a city, ask questions and gain information through the Ozlab system objects such as texts and pictures must be possible to add during runtime. Though, new objects such as pictures cannot be added in active sessions in the first version of the web-based Ozlab system. Furthermore, in the first version of the web-based Ozlab, there is no possibility to write (paste text) in an object that isn’t also writable for the user at the same time.

There are three possible solutions to this problem:

**(1)** the possibility to add content during runtime is implemented; **(2)** a workaround with two shells is used; or **(3)** a new object, namely text box, and a *behaviour* is added to the Ozlab system which lets only the wizard add text to the text box during runtime. The third solution, however, does not allow for graphical wizard output why either the first or the second solution is preferable in the early iterations of the experiments.

It should be noted that if Ozlab is to be used as an information provider in remote settings, the screen size of the users’ devices will in many cases be unknown for the shell builder/designer as building the interaction shell (as described in section 4.5.4 and 4.5.5). This could be an issue if the user will use his/her own device to access Ozlab.

Option (2) above needs some elaboration. By using two interaction shells (shell A and shell B) on two separate Ozlab sites (site A and site B), one or two wizards can interact with the same test participant, as well as add content during run-time. This is done by letting the participant pose his/her question on site A. The wizard then prepares the answer on site B which session is not yet running. When the session is started at site B, the wizard displays a link at site A to site B –for the test participant, site B is opened in the same web browser window as site A. To clarify, site A and site B have distinct URLs.

During a first pilot run, or rather a technical test, the workaround just presented was thought to make the wizard lose control over at which page within shell A the user will return to when clicking an A-link in Shell B. Though, in discussion with one of the interns, a solution was found. Instead of linking shell A and shell B together with shell-specific URLs, the link was modified to a site-specific level. That is, by using http://[ozlab-address]/TP instead of http://[ozlab-addresss]/[shell-specific identifier]/TP the user will return to the site A, and the Ozlab system will automatically find what shell is running and what scene the wizard is displaying.

Without removing the shell-specific identifiers in the URL, the first version of the web-based Ozlab system would automatically open the first scene in the interaction shell, when using the second kind of link like in the example in the previous paragraph. Depending on how the shell was built and designed, this means that if a visitor to an event asked a question in Spanish, and probably would like to return to a Spanish page, the default question page (scene 1, perhaps in English) might be displayed. This would have happened even if the wizard was currently on the second scene in site A, the Spanish question page. Without removing the shell-specific identifiers in the URL, the wizard would not notice if the visitor asks another question as the wizard is currently on the “wrong” scene.

# Conclusions

This report covers problems and implications that follow a web-based WOz system. As mentioned initially, in spite of the fact that systems environments constantly changes, there have been several WOz system presented as generic WOz tools the last decade. The question is, do they stand up to measure? The literature review in section 2.3 showed indicated longevity problems while only one system, WebWOZ, besides Ozlab was elaborated on web features. It is unclear to what extent its developers managed to free themselves from system-dependencies, while Ozlab depends on Sencha Ext JS and Microsoft IIS server, and Google Chrome as clients.

One goal with the re-development of the web-based Ozlab system was to move away from the dependencies of software. It was shown that it is less dependent on software than the Director-based Ozlab system. However, as shown in this report, maintenance will always be an issue even if software independence is reached as the system exists in context that is always changing.

When evaluating the first release of the web-based Ozlab system as reported in section 3, several errors and some limitations was found, which should be taken care of before calling Ozlab a working system. Further, section 3.5 and section 4 demonstrated that accessing the system through a web-browser showed some advantages and limitations due to the use of the web as a system platform. The possibility to access the system is increased, but connectivity and security must be thought of in a different way.

# References

Alce, G., Hermodsson, K. & Wallergård, M. (2013). WozARD: A Wizard of Oz Tool for Mobile AR. *MobileHCI 2013*, Munich, Germany, August 27-30, pp.600-605.

Ardito, C., Buono, P., Costabile, M. F., Lanzilotti, R. & Piccinno, A. (2009). A tool for Wizard of Oz studies of multimodal mobile systems. *HSI 2009*, Catania, Italy, May 21-23, pp. 344-347.

Benzmüller, C., Horacek, H., Kruijff-Korbayová, I., Lesourd, H., Schiller, M. & Wolska, M. (2007). DiaWOz-II – A Tool for Wizard-of-Oz Experiments in Mathematics\*. In *KI 2006: Advances in Artificial Intelligence*. Springer Berlin Heidelberg, pp. 159-173.

Caelen, J. & Millian, E. (2002). MultiCom, a Platform for the Design and the Evaluation of Interactive Systems. Application to Residential Gatewats and Home Services. In *Les Cahiers du numérique*, 3 (4), p. 149-171.

Coutaz, J., Nigay,. L. & Salber, D. (1995). Multimodality from the User and System Perspectives. In ERCIM’95 Workshop on Multimedia Multimodal User Interfaces, Crete, Greece.

Coutaz, J., Salber, D., Carraux, E. & Portolan, N. (1996). NEIMO, a Multiworkstation Usability Lab for Observing and Analyzing Multimodal Interaction. *CHI’96*, April 13-18, p. 402-403.

Dahlbäck, N., Jönsson, A. & Ahrenberg, L. (1993). Wizard of Oz Studies – why and how. *Knowledge-Based Systems*, 6 (4), Butterworth-Heinemann Ltd, p. 258-266.

Davis, R. C., Saponas, T. S., Shilman, M. & Landay, J. (2007). SketchWizard: Wizard of Oz Prototyping of Pen-Based User Interfaces. *UIST’07*, Rhode Island, USA, October 7-10, p. 119-128.

Dow, S., Lee, J., Oezbek, C., MacIntyre, B., Bolter, J.D. & Gandy, M. (2005). Wizard of Oz Interfaces for Mixed Reality Applications. *CHI 2005*, Portland, Oregon, USA, April 2-7, pp. 1339-1342.

Erdmann, R. L. & Neal, A. S. (1971). Laboratory vs. Field Experimentation in Human Factors–An Evaluation of an Experimental Self-Service Airline Ticket Vendor. *Human Factors*, 13 (6), pp. 521-531.

Gould, J. D., Conti, J., & Hovanyecz, T. (1983). Composing letters with a simulated listening typewriter. *Communications of the ACM*, 26(4), pp. 295-308.

Grill, T., Polacek, O. & Tscheligi, M. (2012). ConWIZ: A tool supporting contextual Wizard of Oz simulation. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, Ulm, Germany, December 4-6, pp. 21-28.

Karlstad University. (2013-09-03) *Ozlab – A GUI testing station at Karlstad University*. Available: <http://www.kau.se/en/ozlab>. [2014-03-21]

Kelley, J.F. (1983). An empirical methodology for writing User-Friendly Natural Language computer applications. In *CHI’83 Proceedings*, December, pp. 193-196.

Kilbrink, N. (2008) *Användningstester Plattformen*. Arbetsrapport. Karlstads Universitet.

Klemmer, S. R., Sinha, A. K., Chen, J., Landay, J. A., Aboobaker, N. & Wang, A. (2000). SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pp. 1-10.

Krug, S. (2006) *Don’t Make Me Think! A Common Sense Approach to Web Usability, Second Edition.* Berkeley: New Riders.

Lamberg, C. (2011) HTML5 for Ozlab. Student report on the course ISGC03 Future web standards and mobile multimedia. Available: http://www.kau.se/sites/default/files/Dokument/subpage/2011/12/pdf\_html5\_for\_ozlab\_by\_camilla\_pdf\_10208\_pdf\_43106.pdf [2014-02-24]

Lamberg, C. & Brundin, A. (2011). *Evaluating the future development options for Ozlab*. Bachelor thesis Information Systems. Karlstad: Karlstad University.

Li, A.X. & Bonner, J.V.H. (2013). Using wizard-of-oz method to build multipurpose platform for domestic ambient media research and applications. *Multimedia Tools and Applications*, pp. 1-16 (on-line pre-publication 2013-03-13).

Li, Y., Hong., J. I. & Landay, J. A. (2004). Topiary: A Tool for Prototyping Location-Enhanced Applications. *UIST’04*, New Mexico, USA, October 24-27, pp. 217-226.

Lindström, M. & Nilsson, J. (2009). *Usability test report. Third pilot test of “Trust Evaluation”. PrimeLife-project*.

Linnell, N., Bareiss, R. & Pantic, K. (2012). A Wizard of Oz Tool for Android. *MobileHCI’12*, San Francisco, USA, September 21-24, pp. 65-70.

Molin, L. (2004). Wizard-of-Oz Prototyping for Cooperative Interaction Design of Graphical User Interfaces. *NordiCHI ’04*, Tampere, Finland, October 23-27, pp. 425-428.

Munteanu, C. & Boldea, M. (2000). MDWOZ: A Wizard of Oz Environment for Dialog Systems Development. In *LREC*, Athens, Greece, May 31-June 2, pp.104-107.

Nilsson, J. (2005). *Interaktionsdesign av pedagogisk programvara. En experimentell studie av demonstrationer som hjälpfunktioner i ett övningsprogram för mellanstadiebarn.* Master Thesis Information Systems. Karlstad: Karlstad University.

Nilsson, J. (2006). *Användbarhetsutvärdering av H-RIB XM – Ozlabprototyp 2 -.*

Pettersson, J.S. (2002) Visualising interactive graphics design for testing with users. *Digital Creativity*, 13(3), pp.144-156.

Pettersson, J.S. (2003). Ozlab – a System Overview with an Account of Two Years of Experiences. In Pettersson, J.S. (ed.) *HumanIT 2003*. Karlstad: Universitetstryckeriet Karlstad. pp. 159-185.

Pettersson, J.S. & Nilsson, J. (2011) Effects of Early User-Testing on Software Quality – Experiences form a Case Study. In Song, W.W et al. (eds.) *Information Systems Development*. Springer Science+Business Media, LLC. pp.499-510.

Pettersson, J.S. & Siponen, J. (2002) Ozlab – a Simple Demonstration Tool for Prototyping Interactivity. In *NordiCHI*, Århus, Denmark, October 19-23, pp.293-294.

Schlögl, S., Doherty, G., Karamanis, N. & Luz, S. (2010). WebWOZ: A Wizard of Oz Prototyping Framework. *EICS’10*, Berlin, Germany, June 19-23, pp.109-114.

Schlögl, S., Doherty, G., Karamanis, N., Schneider, A. & Luz, S. (2010). Observing the Wizard: In Search of a generic Interface for Wizard of Oz Studies. In *Proceedings of iHCI*, Dublin, Ireland, September 2-3, pp. 43-50.

Schlögl, S., Schneider, A., Luz, S. & Doherty, G. (2011). Supporting the Wizard: Interface Improvements in Wizard of Oz Studies. In *Proceedings of the 25th BCS Conference on Human-Computer Interaction*, Swinton, United Kingdom, pp. 509-514.

Schlögl, S., Chollet, G., Milhorat, P., Deslis, J., Feldmar, J., Boudy, J., Garschall, M. & Tscheligi, M. (2013). Using Wizard of Oz to Collect Interaction Data for Voice Controlled Home Care and Communication Services. In *Proceedings of the IASTED International Conference*, Innsbruck, Austria, February 12-14, pp. 511-518.

Segura, V.C.V.B. & Barbosa, S.D.J. (2013). UISKEI++: Multi-Device Wizard of Oz Prototyping. *EICS’13*, London, United Kingdom, June 24-27, pp. 171-174.

Serrano, M. & Nigay, L. (2010). A wizard of oz component-based approach for rapidly prototyping and testing input multimodal interfaces. *Journal on Multimodal User Interfaces*, 3(3), pp. 215-225.

Siponen, J., Pettersson, J. S. & Alsbjer, C. (2002). *Ozlab Systembeskrivning*. Arbetsrapport. Karlstads Universitet: Institutionen för informationsteknologi.

Zachhuber, D., Grill, T., Polacek, O. & Tscheligi, M. (2012). Contextual Wizard of Oz. In *Ambient Intelligence*, Springer Berlin Heidelberg, pp. 224-239.

# Appendix 1. Literature review strategy

Table A1.1 Literature review search strategy, various data base searches.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Search terms** | **Filters** | **Source** |
| Initial search | “WOZ” OR “WoZ” OR “WOz” OR “woz” OR “wizard of oz” |  | Google Scholar  ACM Digital Library Science Direct  Karlstad University OneSearch  IEEE |
| Elaborated | (—"—) AND (“technique” OR “method” OR “interaction” OR “data” OR “data collection” OR “prototype” OR “system” OR “tool”) | Publication date “2000 and forward” OR “2007-2013” OR “Peer reviewed” | Google Scholar  ACM Digital Library Science Direct  Karlstad University OneSearch  IEEE |

Table A1.2 Literature review search strategy, Science Direct, October 9th 2013

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Search terms** | **Filters** | **Results** |
| Initial search | wizard of oz  “wizard of oz” |  | 611  529 |
| Elaborated | “wizard of oz” | Article AND Reviewed Article | 424 |
|  | “wizard of oz” AND interaction |  | 368 |
|  | “wizard of oz” AND technique |  | 321 |
|  | “wizard of oz” AND technique AND interaction |  | 263 |

Table A1.3 Literature review strategy Google scholar, October 9th 2013

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Search terms** | **Filters** | **Results** |
| Initial search | wizard of oz  “wizard of oz” | Since 2012 AND do not include patents citations | 4090 |
| Elaborated | “wizard of oz” | Exact phrase AND Since 2012 | 3930 |
| Elaborated | “wizard of oz” AND (technique OR method OR interaction) | Since 2012 | 2880 |
| Elaborated | “wizard of oz” AND (technique OR method OR interaction) | Any time | 22500 |

# Appendix 2. Generic solutions besides the Ozlab system

There are several tools that incorporate the Wizard of Oz technique. Some are developed for testing on specific devices, some for simulating certain modalities or aspects of a prototype, while a few are more generic. Below the tools found in the literature review are presented. Only tools that can be configured and re-used for several studies are included in this section. The ordering of presentation in this section is alphabetic.

**ConWIZ** is a part of the Contextual Interaction Framework (CIF, see Zachhuber et al. 2012 for description and evaluation of the Contextual Wizard of Oz framework), for UbiComp environments (Grill, Polacek & Tscheligi 2012). Through an Android device tool, **Mobile Wizard**, the wizard can “send commands to external applications” (Z. et al. p. 231).

**DART** In an experiment by Dow, Lee, Oezbek, MacIntyre, Bolter and Gandy (2005) was used to give visitors an audio experience in a historic site using location tracking. DART is integrated in Macromedia Director.

**DiaWOz-II** is a configurable software environment for WOZ studies where a combination of mathematical input and natural language is used. DiaWOz-II, based on TEXMACS, is not an improved version of the predecessor DiaWoZ system. (Benzmüller, Horacek, Kruijff-Korbayová, Lesourd, Schiller & Wolska 2007)

**Jaspis** is adistributed software architecture that can be used for WOz studies on speech user interface and UbiComp (Mäkelö, Salonen, Turunen, Hakulinen & Raisamo 2001).

**LIVE** presented by Li and Bonner (2013) seems to simply duplicate (“mirror”) wizard’s screen onto something which the test subject(s) can see (and act upon).

**MDWOZ** is a spoken interaction dialog systems. (Munteanu & Boldea 2000)

**MultiCom** consists of an observation laboratory where WOz studies can be conducted. The laboratory also includes other software and hardware. (Caelen & Millien 2002)

**MuMoWOz** is a tool for conducting tests on multimodal mobile systems. (Ardito, Buono, Costabile, Lanzilotti & Piccinno 2009)

**NEIMO** is a platform for multimodal interaction. NEIMO supports a several wizards setup. (Coutaz, Salber, Carraux & Portolan 1996; Coutaz, Nigay, & Salber 1995)

**OpenWizard** is a “component-based approach for the rapid prototyping and testing of input multimodal interaction” (Serrano & Nigay 2010, p.224). OpenWizard incorporates the Wizard-of-Oz technique by replacing a component in a non-fully developed prototype with generic wizard component(s).

**Ozlab** was constructed as a GUI WOz system as there were no general graphics-supporting WOz systems at the time (Pettersson 2002). From 2013 this system is being replaced by a web-based version with similar features ([www.kau.se/en/ozlab](http://www.kau.se/en/ozlab)).

**SketchWizard** supports early prototyping of user interfaces incorporating pen-based interaction. (Davis, Saponas, Shilman & Landay 2007)

**SUEDE** (Klemmer, Sinha, Chen, Landay, Aboobaker & Wang 2000) is a tool for prototyping speech interfaces. All (simulated) system output needs to be added to the prototype on beforehand.

**Topiary is** a tool for prototyping location-aware applications. (Li, Hong & Landay 2004)

**UISKEI++**, reported by Segura and Barbosa (2013), is the intended evolution of the tool UISKEI (User Interface Sketching and Evaluation Instrument). UISKEI++ is envisioned to support WOz experiments and prototyping on multi-devices by providing multiple abstraction levels. The multiple abstraction levels are argued to allow the designer to compare the prototypes, regardless of device. (Segura & Barbosa 2013)

**WebWOZ** is web-based and focuses on flexible incorporation of Language Technology Components (LTC). (Schlögl, Doherty, Karamanis & Luz 2010) Experiments resulted first in sketches (Schlögl et al. 2010) and later in prototypes (Schlögl et al. 2011). Schlögl, Chollet, Milhorat, Deslis, Feldmar, Boudy, Garschall and Tscheligi (2013) report on their progress of offering voice controlled Home Care and Communication Services, vAssist, which in the future will be developed using WebWOZ.

**Wizard of Oz tool for Android** allows digitally created or scanned paper prototypes to be tested. The prototype must be developed beforehand because the tool automatically creates a folder with prototype specific objects that must be transferred to the Android phone. Communication TP – TL enabled by modified open-source VNC client for Android. (Linnell, Bareiss & Pantic 2012)

**WozARd** is a tool for WOz experiments on mobile phones, tablets and glasses which provides communication between the wizard and puppet Android device over wireless and Bluetooth. WozARd is location aware and accepts images, video and sound to be uploaded into the prototypes. WozARd logs test results and visual feedback on a SD card[[14]](#footnote-14). Content can be added during test. WozARd is planned to be released as open-source. (Alce, Hermodsson & Wallergård 2013)

1. Examples of research areas where the Wizard-of-Oz technique has been used: Automatic Speech Recognition (ASR), Text-to-speech synthesis (TTS), Multimodal Interaction, Human-Computer Interaction (HCI), Human-Robot Interaction (HRI), Augmented Reality (AR), Artificial Intelligence (AI), Ubiquitous Computing (UbiComp), Mixed Reality (MR), Intelligent Tutoring Systems (ITS). [↑](#footnote-ref-1)
2. The Ozlab webpages can be found at <http://kau.se/en/ozlab> (or the old version, accessed through [www.is.kau.se/~jsp/ozlab](http://www.is.kau.se/~jsp/ozlab)) [↑](#footnote-ref-2)
3. Wikipedia on Macromedia (2013-11-23) <http://en.wikipedia.org/wiki/Macromedia> [2014-01-04] [↑](#footnote-ref-3)
4. Wikipedia on ASP .NET MVC Framework (2014-02-05) <http://en.wikipedia.org/wiki/ASP.NET_MVC_Framework> [2014-02-18] [↑](#footnote-ref-4)
5. Wikipedia on Internet Information Services (2014-02-14) <http://en.wikipedia.org/wiki/Internet_Information_Services> [2014-02-18] [↑](#footnote-ref-5)
6. Wikipedia on WebSocket (2014-02-08) <http://en.wikipedia.org/wiki/WebSocket> [2014-02-18] [↑](#footnote-ref-6)
7. Wikipedia on JQuery (2014-02-14) <http://en.wikipedia.org/wiki/JQuery> [2014-02-18] [↑](#footnote-ref-7)
8. Wikipedia on Ext JS (2014-01-07)<http://en.wikipedia.org/wiki/Ext_JS> [2014-02-18] [↑](#footnote-ref-8)
9. Web browser engine can also be called layout engine or rendering engine. See Wikipedia on *Web browser engine* (2014-02-24) Available: <http://en.wikipedia.org/wiki/Web_browser_engine> [2014-02-28] [↑](#footnote-ref-9)
10. Stuart Dredge, The Guardian (2012-05-29). Google Chrome browser for iOS: first impressions. Available: <http://www.theguardian.com/technology/appsblog/2012/jun/29/google-chrome-browser-iphone-ipad> [2014-02-20] [↑](#footnote-ref-10)
11. The Chromium Blog (2013-04-03) *Blink: A rendering engine for Chromium*. Available: <http://blog.chromium.org/2013/04/blink-rendering-engine-for-chromium.html> [2014-02-28] [↑](#footnote-ref-11)
12. CNET, Stephen Shankland (2013-04-03) *Google parts ways with Apple over WebKit, launches Blink*. Available: <http://news.cnet.com/8301-1023_3-57577790-93/google-parts-ways-with-apple-over-webkit-launches-blink/> [2014-02-28] [↑](#footnote-ref-12)
13. Also called Four In a Row etc. <http://en.wikipedia.org/wiki/Connect_four> [2013-12-20] [↑](#footnote-ref-13)
14. SD is short for Secure Digital. https://www.sdcard.org/home/ [2014-03-20] [↑](#footnote-ref-14)