Using the Lively Web To Create Interactive Web Applications

by

Harsh Dawar
B.Tech., Amity University, India 2010

A Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Using the Lively Web To Create Interactive Web Applications

by

Harsh Dawar
B.Tech., Amity University, India 2010

Supervisory Committee

Dr. Hausi Müller, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Hausi Müller, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

ABSTRACT

This project presents the design and implementation of a web application called PALTask2.0 using the Lively Web programming environment. PALTask2.0 is the next major upgrade to the existing desktop-based version developed at the Rigi Research lab. One of the main motivations for pursuing this project is to demonstrate the capabilities of Lively and to inspire more developers to join and take part in the open source Lively community. Additionally, this report is intended to serve as a user guide, as it focuses on details of this Lively specific implementation. A few notable advantages of using Lively are: applications can be built entirely using a single programming language (i.e., JavaScript), Lively has an built-in Integrated Development Environment (IDE) and source code versioning scheme, and finally Lively applications can be built on the Cloud, thereby eliminating the hassle of deploying and managing web servers. The complete unified solution offered by Lively eases the developer's workload and helps increase productivity. This report outlines some of the identified weaknesses and lists suggestions to further improve the Lively Web Experience.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisors, **Dr. Hausi Müller** and **Dr. Sudhakar Ganti** for granting me the opportunity to do this degree, and providing continuous guidance and support.

Thank you to, **Rick McGeer**, **Marko Röder** and **Daniel Ingalls** for their endless support and encouragement throughout this journey.

A special thanks to everybody at **Rigi Lab** for the friendship, advice and support.

Chapter 1

Introduction

1.1 Motivation

One of the main challenges any emerging technology faces is the lack of community, documentation and resources. One or more of these drawbacks are enough to discourage even the most skilled software developers [11]. Lively Web faces a similar challenge, its documentation is limited to a single tutorial resource known as Lively 101 [20].

A possible way of encouraging users to join the Lively development community is by demonstrating and documenting the potential of the platform. We achieve this by participating in a larger research project [17, 19], which aims to determine the effectiveness of the Lively Web. We contribute by using the Lively platform to develop a relatively complex web application that involves features such as, communication using web sockets, transferring control messages, performing database interactions and making Restful API calls. Developing such an application using Lively could possibly unearth bugs, limitations and performance issues of the platform which can be used to improve the Lively Web Experience. Furthermore, the underlying report explains the project implementation details which can serve as a user guide for training purposes developers not familiar with Lively.

1.2 PALTask and its New Successor

PALTask [18] is a context-aware chatting tool developed at Rigi Lab with the intent to improve the chat user experience. The tool automates the process of retrieving

relevant resources dynamically based on analyzing the context of user chats. In particular, it uses the Rapid Automatic Keyword Extraction (RAKE) algorithm [7, 36] to extract important keywords. The combination of extracted keywords and the Personal Context Sphere [41] of the user is used to gather web resources from popular services such as Google and Youtube.

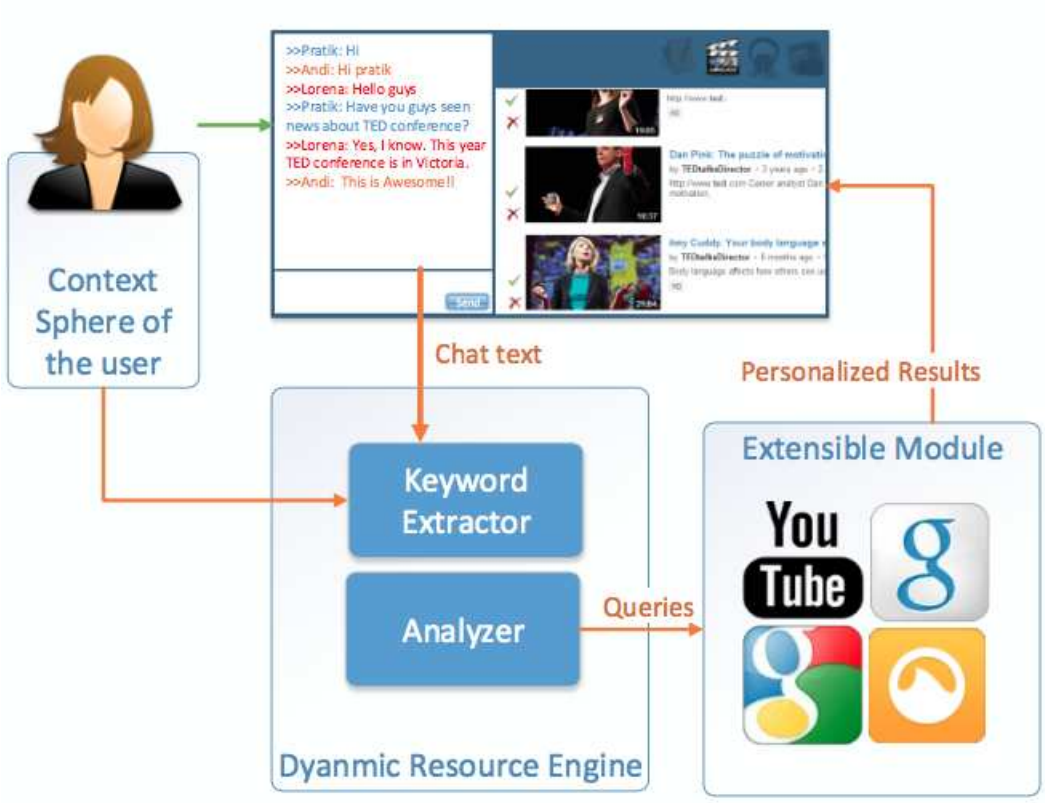


Figure 1.1: PALTask High Level Interaction

Since PALTask is a desktop-based application, it has to be manually installed and configured. With chatting applications increasingly moving towards a Software as a Service (SaaS) model [14], a web based software distribution scheme, it motivates the development of PALTask2.0 as a web-based application. A web-based environment makes the tool more accessible to the user [34, 39, 37] and creates opportunities to use web-based frameworks and technologies that are not available to a desktop based software system. PALTask 2.0 not only implements a web based version of the tool, but also integrates sophisticated machine learning services to improve the accuracy of the results. Furthermore, the user interface is significantly enhanced to attract users and encourage the use of Lively.

1.3 Approach

As mentioned in Section 1.1, this project has the following main goals that need to be accomplished

- Demonstrate the potential of Lively Web.
- Upgrade PALTask to make it more accessible.
- Improve the accuracy of the results.

The first two intents can be satisfied by implementing PALTask2.0 using the Lively Web programming interface. The idea is to construct a complex Morph that represents the chat window of the application. This main parent Morph can have Sub Morphs that implement specific actions. For example, a Sub Morph to display users online, display messages and pictures. The effectiveness of Lively can be illustrated by simply wiring these individual Sub Morphs together to perform as a unified chatting tool.

The third goal, which is to improve the accuracy of retrieved results, can be achieved by integrating PALTask2.0 with sophisticated machine learning services. Therefore, along with keyword extraction, PALTask2.0 is trained to understand certain intents and respond to them appropriately. The system is designed such that PALTask2.0 can be easily extended to recognize and handle more intents in the future.

Chapter 4 discusses the realization of the above mentioned goals in detail.

1.4 Contributions

Implementing PALTask2.0 in Lively has many contributions. A few have already been discussed in Section 1.1. Apart from them, multiple Morphs created during the implementation are available to developers to use freely in their projects. Hopefully PALTask2.0 can inspire developers to adopt Lively as their programming environment and contribute to this emerging open source community.

Chapter 2

Related Study

This chapter presents an overview of work done in a few different domains. The first is an overview of some interesting projects developed using the Lively Web. The second is an overview of chatting applications that exploit contextual information. Lastly, we study the components needed for Natural Language processing. We mention the components that can be reused from PALTask and also investigate a few machine learning services which can be integrated into PALTask2.0.

2.1 Past Projects built using the Lively Web

A number of interesting projects such as animations, games, collaborative tools such as video chat clients have been developed using the Lively programming environment [24, 23, 21, 22, 19]. Since it is not possible to discuss all conducted projects, we limit ourselves to notable contributions to the Lively Web.

Lincke et al. [24] used the Lively Web to develop a CPU Visualization tool. The tool uses a Node.JS server to extract CPU data using the *mpstat* command line utility. It then demonstrates the flexibility of Lively by integrating a third party JavaScript library known as Protovis [8], which is used to plot the CPU data extracted in a graphical representation.

Lincke et al., inspired by the interactive programming environment Fabrik [27], designed a rich web-based interface for Lively known as the Lively Fabrik [23]. In this interface applications are created by placing components in a graph and wiring them together. Lively Fabrik was designed to empower end-users to create dynamic web applications from within their Web browser instantly. To demonstrate the use

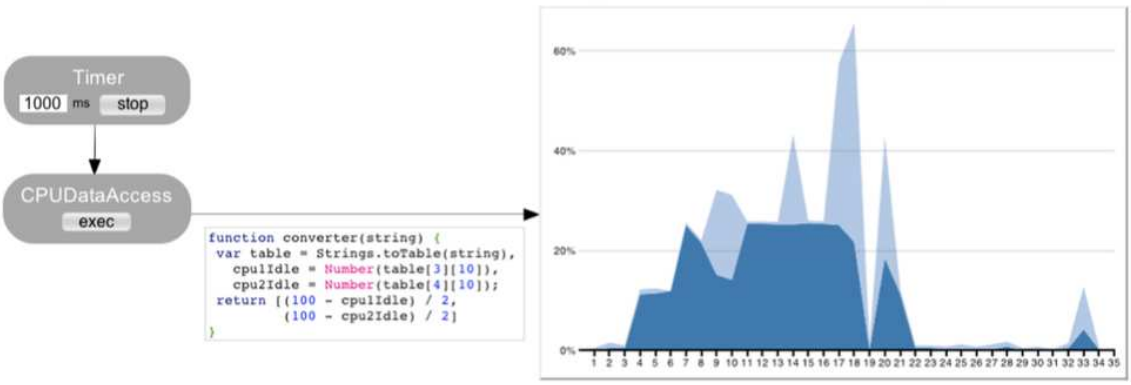


Figure 2.1: CPU Utilization Tool [24]

of Lively Fabrik the authors developed a Weather widget application. The widget allows users to type the name of the city or use the zip code to fetch the weather conditions and display them dynamically in a textual and visual form. The widget can be saved, modified and reused by Lively Web users in their projects.

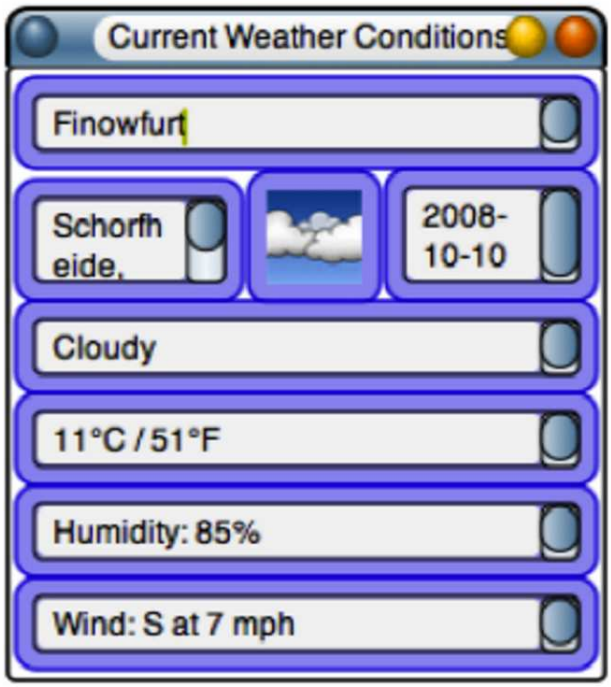


Figure 2.2: Weather Widget using Lively Fabrik [27]

Krahn et al. [21] present a calendar application built using the Lively programming interface. Although, calendar applications are now readily available on most

operating systems, the implementation in Lively can be customized to provide functionality as desired by the user. For example, the users of the calendar application can create notes and simply drag and drop them onto specific dates to save the entries.

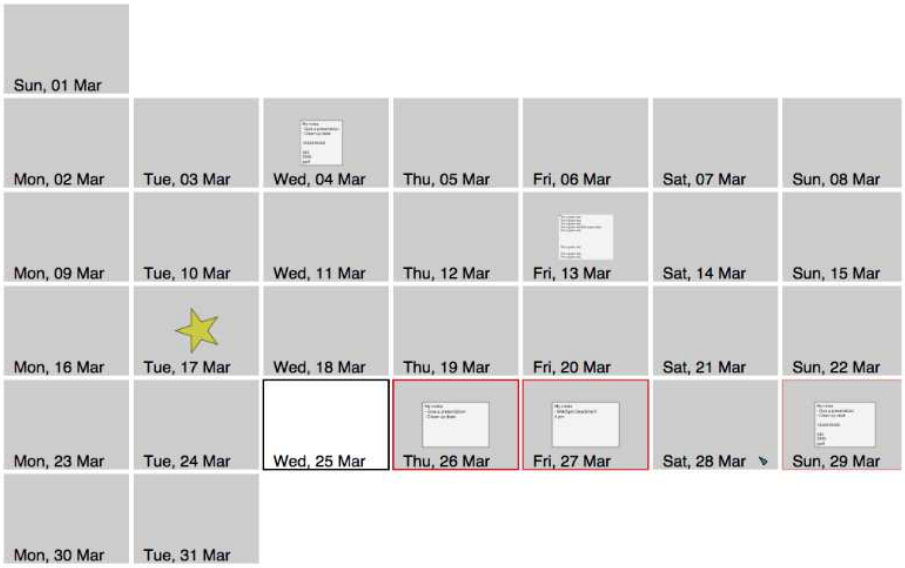


Figure 2.3: Calender Application using Lively [21]

A platform can only be truly collaborative when it supports the setup of a communication channel between users. This facilitates the exchange of data between the participating users and opens up endless opportunities to make distributed applications. Kuuskeri et al. [22] extended the Lively platform to support peer to peer communication between users accessing the same Lively server. They achieved the same by implementing remote wormholes, which are socket connections, allowing for data including JavaScript objects to be sent across to the destination peer. The authors also implement a multiplayer chat application known as *Dung Beetle* to illustrate the effectiveness of the wormhole. The wormhole connection is discussed in detail in Section 3.7, as it is used heavily in the development of PALTask2.0.

Apart from applications discussed above, music synthesizers, text animations and video player applications have been successfully created [19], demonstrating the effectiveness of Lively. This brings us to the following questions.

- **Why do we need to implement PALTask2.0 using Lively?**
  In my view, the applications developed using Lively do not utilize its full po-

Figure 2.4: Dung Beetle collaborative game built using Lively Wormhole [22]

tential as a unified platform to develop both, the back-end and the Graphical User Interface (GUI). We intend to create a highly user intuitive and responsive GUI for PALTask2.0 in the hope to inspire developers to join the Lively community. Additionally, since Lively is still in the research phase, development of a complex application such as PALTask2.0 would be a perfect test bed to discover bugs and shortcomings that can help improve this emerging JavaScript technology.

- **Has a similar application been developed using the Lively Web before?**
  An entity extraction utility [25] has been developed in the past using the Lively Web. The utility uses Apache Stanbol [1] to extract entities and automates the

[1] https://stanbol.apache.org/

retrieval of information based on the extraction results. PALTask2.0 builds on this utility by adding features such as chatting among peers, keyword extraction, sentiment analysis and machine learning.

2.2 Context-Aware Chatting Applications

Abowd et al. [3] define Context as the information gathered from users, environment, sensors, web interactions, devices and other systems that affect the user. Context-Aware chatting Applications are not restricted to but can greatly improve the user experience [18] by using the contextual information as a search key to automate the process of retrieving resources from the web. This section presents an overview of chat applications that extract context with the aim of utilizing it to improve the user experience of Instant Messaging (IM).

GaChat [15] developed by Satoshi et al., is designed to improve awareness among individuals participating in the chat. GaChat extracts nouns from the user chats and retrieves content from services such as Wikipedia and Google Image Search. Since the search is restricted to proper nouns, looking up generalized resources such as a restaurant nearby is not possible in this system. Figure 2.5 shows the data flow of Gachat.

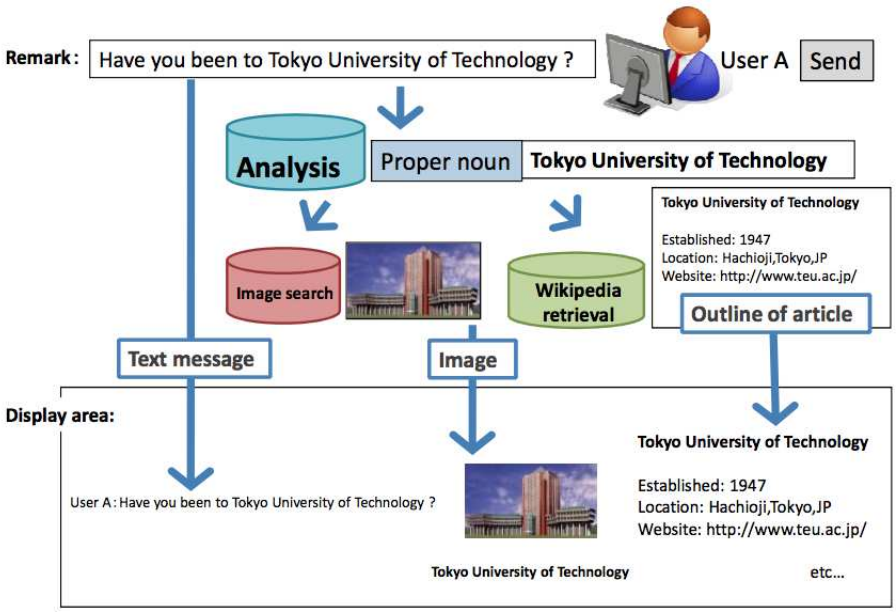


Figure 2.5: Data Flow of Gachat [15]

ConChat [35] mitigates semantic uncertainty between users participating in a chat session. Context is collected from sensors such as temperature, location and lighting in the room. It then resolves the semantic ambiguities in various areas such as time, data format and currency.

PALTask has several advantages over the applications discussed above.

- PALTask provides feedback messages in real-time.
- PALTask takes advantage of the Villegas' Personal Context Sphere [41] to provide personalized results to the user.
- PALTask uses various Natural Language processing techniques discussed in Section 2.3 to filter and refine the results for high accuracy [18].

2.3 Natural Language Processing Components

Natural Language Processing (NLP) [9] is a field concerned with the interaction between computers and human languages. Jain et al. [18] used various NLP techniques to improve the user chat experience in PALTask. In this section, we discuss the NLP components that are reused and introduce components that have been added in PALTask2.0.

2.3.1 Keyword Extractor

Trieschnigg et al. [40] state that keywords can be used as a simple technique to provide descriptive meta-data about a document or a conversation. Keywords extracted from a conversation can be used as search keys to look up content on the web. After investigating several Keyword Extractors, Jain et al. [18] chose to use a Python implementation of the Rapid Automatic Keyword Extraction for its accuracy and reliability [7].

In PALTask2.0 the two keyword extractor components are RAKE, reused from its predecessor, and Alchemy [5]. The addition of Alchemy gives users an option between two different keyword extraction services based on their preference.

2.3.2 Sentiment Analysis

Sentiment Analysis is an area of NLP that focuses on identifying the polarity opinion of a text. The subject of analysis could be any textual data such as a text document, tweets on social media and chat conversations. Jain et al. [18] use the Natural Language Toolkit [2] to filter the number of feedback messages received by the user.

We reuse the NLTK Sentiment Analysis service created for PALTask and also integrate the Alchemy Sentiment Analysis API [6] in PALTask2.0. Again, the user can select amongst the two services based on their preference. The use of Sentiment Analysis is explained in detail in Chapter 4.

2.3.3 Machine Learning

This section mentions a few shortcomings of using a keyword extractor and explores machine learning services that can be used to improve the accuracy of retrieved results. McCallum et al. [31] successfully designed a highly accurate domain specific search engine using machine learning.

- User chats in PALTask are accumulated to create sentence chunks, which are 140-160 characters long, which are then sent for keyword extraction. Analysing sentence chunks as opposed to analyzing each chat message helps filter the number of feedback results the user receives [18]. However, accumulating user messages to create sentence chunks can delay the feedback response, which is undesirable.

- The first step of the RAKE algorithm involves the removal of the stop words present in the text such as, *the*, *for* and *of*. Although this is ideal for analysis of generalized text, the results can be inaccurate when users talk about specialized topics, such as discussion on movies and songs. For example, listing 2.1 shows the response of the RAKE algorithm when analyzing the text *I want to watch The Lord Of the Rings*.

```
{
    '1':{
            'value': '1.0',
            'keyword': 'rings'
        },
    '2':{
            'value': '1.0',
            'keyword': 'watch'
        },
    '3':{
            'value': '1.0',
            'keyword': 'lord'
        }
}

```

Listing 2.1: Rake Algorithm response

None of the above keywords individually can pull up information about the movie *Lord of the Rings*.

One way to improve the efficiency of PALTask2.0 feedback retrieval is to integrate it with machine learning services. Since the primary goal of this project is to demonstrate the capabilities of Lively we do not dwell into the complex domain of machine learning. Instead, we investigate services that can enable PALTask2.0 to have a simple trained interface with minimal setup.

IBM Watson [16] offers a natural language classifier service which enables developers with little or no background in machine learning to create trained interfaces for their applications. Another web based service that offers a similar trainable interface is WIT.AI [42]. We chose WIT.AI to create a trained interface for PALTask2.0 since it is an open source service, which is simple to setup and provides a Restful interface that is easily accessible. Chapter 4 details the setup and integration of WIT.AI with PALTask2.0.

Chapter 3

Overview of Lively

Lively web [1] is an open source web-programming environment built entirely using the JavaScript programming language. It supports the development of a wide category of web applications from browser-based games, dashboards, potential SaaS applications and much more. Lively is accessible as a web page and one can begin working by simply loading the lively home web page in any modern browser such as Chrome, Safari, and Firefox. It requires no special installation of any tool or plug-in components. Daniel Ingalls et al. [17] state how modern computers capable of running billions of operations in a second still render pages using a decade old markup language. They believe that a web browser which supports a dynamic programming language such as JavaScript, sophisticated graphic systems and supports networking can do much better by supporting a world of active objects.

This section is intended to get the user familiar with some basic, yet important, Lively terminologies and concepts. A thorough understanding of this section is necessary since many of the described concepts are used in the implementation of PALTask2.0 discussed in Chapter 4.

3.1 The Morphic Architecture

The Lively Web is built on the Morphic architecture [28, 29, 30], which is inspired by the Self and Squeak programming environments. It is a user interface development technique which offers a live environment for designing applications. The term Liveliness in a Morphic architecture is used to describe Morph objects as active and interactive. For example, a rectangle Morph can be programmed to change color

when it is selected. The same morph can then be dragged and scaled. Figure 3.1 demonstrates multiple Morphs running as live objects on the screen.

Daniel Ingalls et al. [17] describe Morphs as having some or all of the following properties.

- A shape and size.
- Children Morphs also known as sub-morphs.
- A specific position and extent.
- An event handler to catch and react to events.
- A style editor.
- A layout manager to manage the layout of its sub-morphs.
- A stepping behavior.

To summarize, every entity visible on the screen including the internal lively tools is a Morph and hence forth a live object which can be manipulated as desired.

3.2 Creating a User Workspace

1. To get started, the user must simply visit the home page of a Lively server [1]. The user has a choice between using a server available online or launching their own instance of a Lively server [26]. The Lively server hosts all the necessary libraries required to develop web applications. For this project we use the Lively server hosted on lively-web.org.

2. The default home page loads when the users visit the Lively server URL. The user can click on the Login link (cf. Figure 3.2) and create a user-name. Since Lively is currently a research project, it does not support an authentication module.

3. On successful login, each user has a workspace created under the following URL http://lively-web.org/users/USER-NAME/start.html The user can simply visit the above URL and start implementing the web application. Unlike conventional Web development, the user does not have to spend any time configuring/installing web servers, installing an IDE/Code editor [38].

Figure 3.1: Multiple Morphs open in a Lively Screen

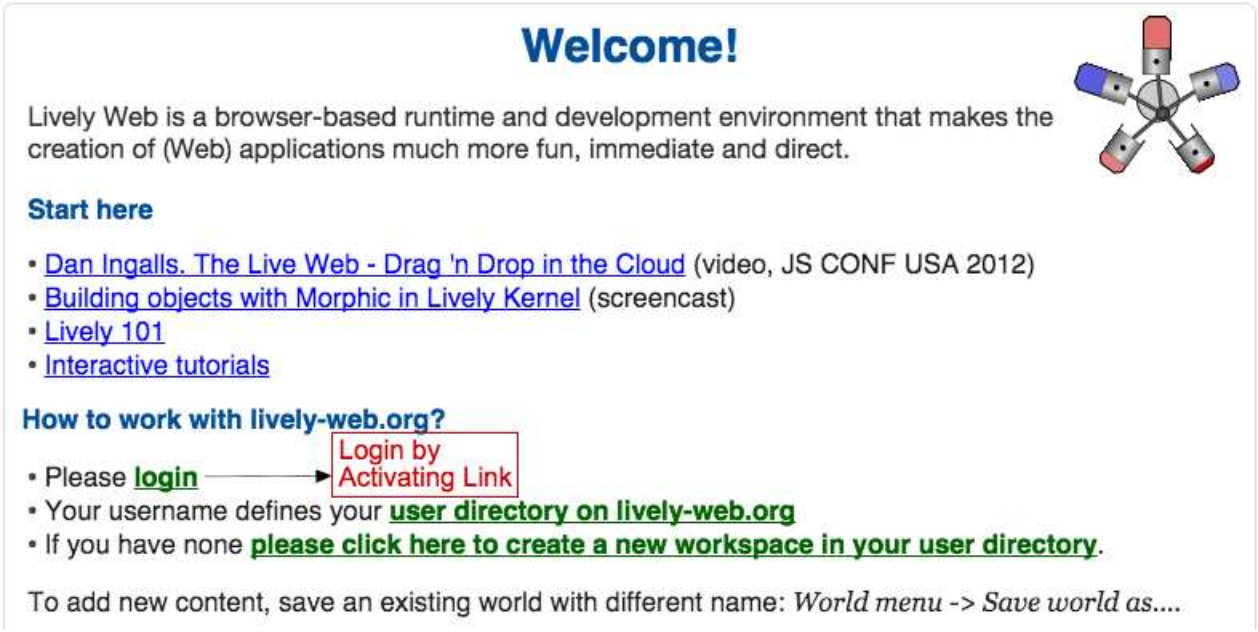


Figure 3.2: Login to Lively Web

3.3 What are World, PartsBin and Halos in Lively?

1. Kuuskeri et al. [22] define the term World as a workspace within the Lively platform. It is essentially a viewable web page in the Lively Web where application Morphs are placed. Users can share their World by simply distributing the URL of the workspace they created. The state of the World can be saved using the CMD/CTRL + S shortcut.

2. PartsBin is one of the key Morphs in the Lively Web. Lincke et al. [24] refer to it as the *"Repository of Parts"*. Since every Morph in Lively is a JavaScript object, it can be serialized and saved in JSON format. PartsBin takes advantage of this JavaScript feature to persist the Morphs and its associated functionality through serialization. When a Morph in Lively is published, it is essentially serialized and the state is saved in this repository. PartsBin contains all the published Morphs available in the Lively Web. To use a Morph, the user can launch PartsBin and drag it into the World. This allows for a collaborative setup to share, distribute and develop Morphs together. Once a user publishes a Morph to the Lively server, users across the world accessing the same server can simply access the Morph through this object repository. They can implement their own customizations to the Morph and publish it as well. Additionally, PartsBin also allows users to create their sub-directly/folders to manage their published Morphs.

   Aside:- Users can load any Morph from the PartsBin and use it in their world. The Morph then exists as a local copy on the user's system. However, all the changes to the Morph are lost if the user fails to save the world.

3. Halos in Lively is a menu option of possible operations that can be performed on a Morph. Selecting a Morph concurrently with the CTRL/CMD key will activate the Halo. The Halo appears as a set of buttons around the Morph, with each button having a specific action. Figure 3.4 illustrates a simple Rectangle Morph surrounded by an active Halos. The Halo menu conveniently enables users to perform all the key operations possible on a Morph such as dragging, adding scripts, styling, maintaining hierarchical relationships, inspecting, cloning and much more.

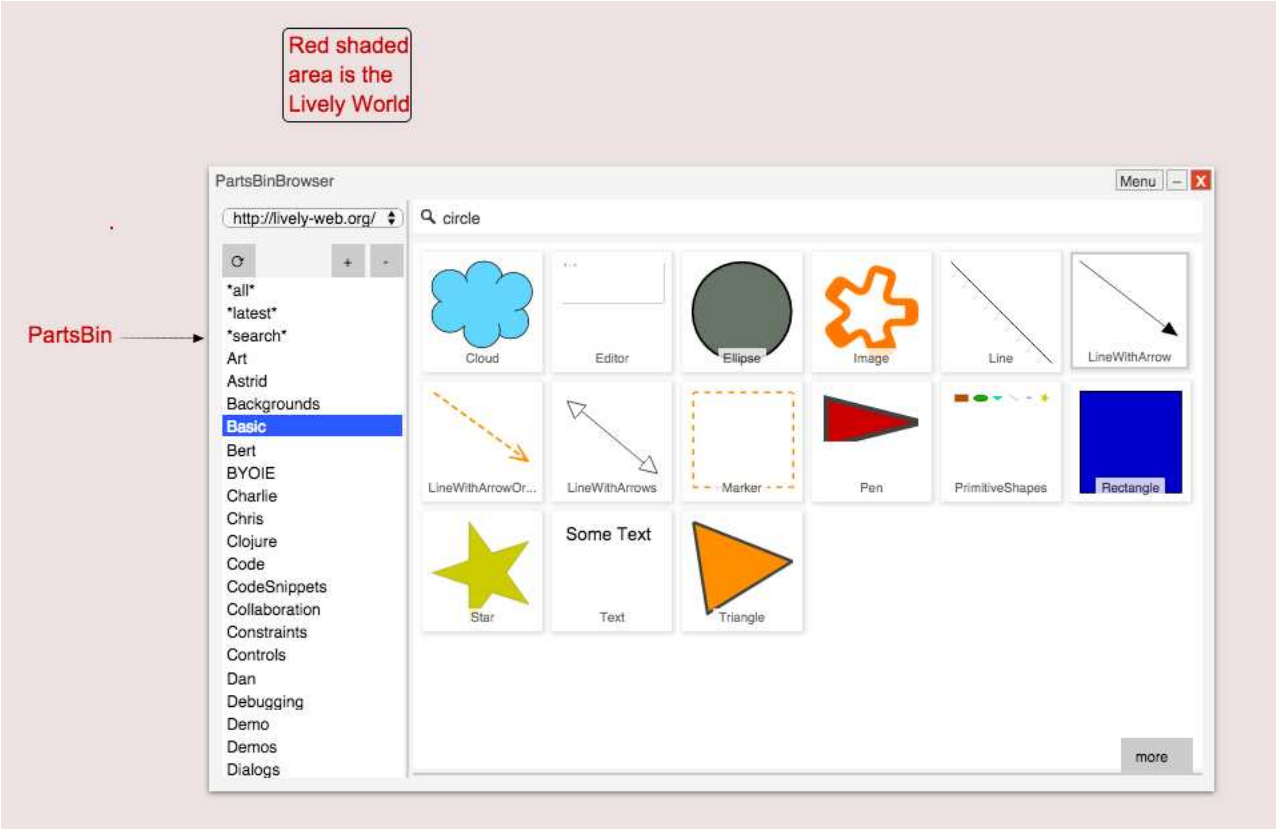


Figure 3.3: World and PartsBin in Lively

3.4 Debugging in Lively

Debugging is the technique of finding and reducing the number of defects in a software program. Often programmers utilize the process of setting up breakpoints and stepping the execution thread to inspect the logic of the software. JavaScript can be debugged using tools such as Chrome Developer Tools [12] and Mozilla Firebug [33]. Unfortunately, Lively does not support such tools at the time of writing this report. However, Krahn et al. [21] mention a Lively Stack Viewer, which is an effective debugging tool. We discuss the Stack Viewer along with other important debugging techniques thoroughly in the remainder of this section.

1. The Stack Viewer (cf. Figure 3.5) is a built-in Morph in Lively that prints the method call trace. It can be launched by simply inserting a call to the *halt()* method where the trace is needed. The Stack Viewer also offers In-line evaluation of code (i.e., the state of the variables can be inspected). Regrettably,

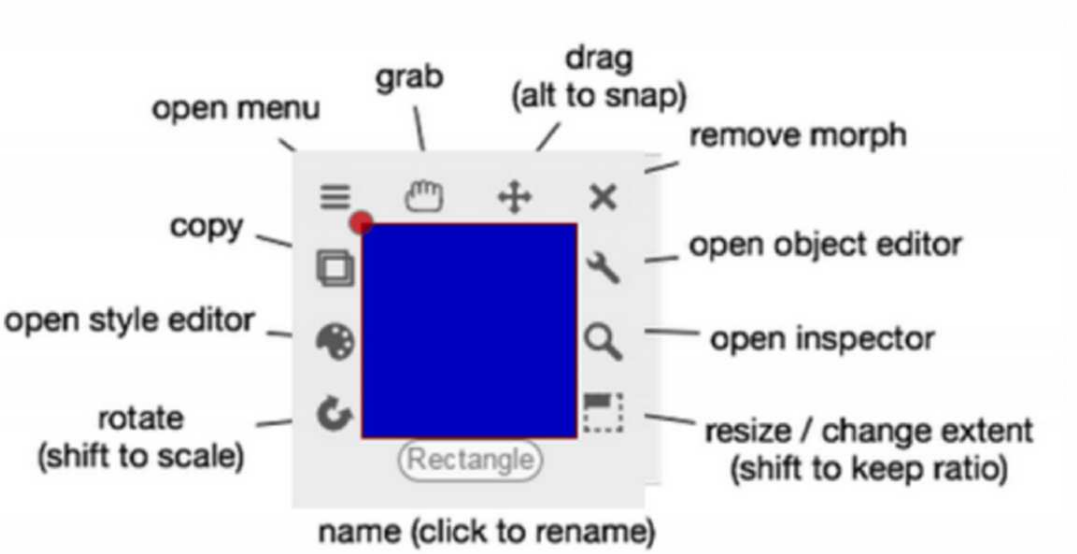


Figure 3.4: Halos Menu on a Rectangle Morph

at present the Stack Viewer does not support the stepping of the execution thread.

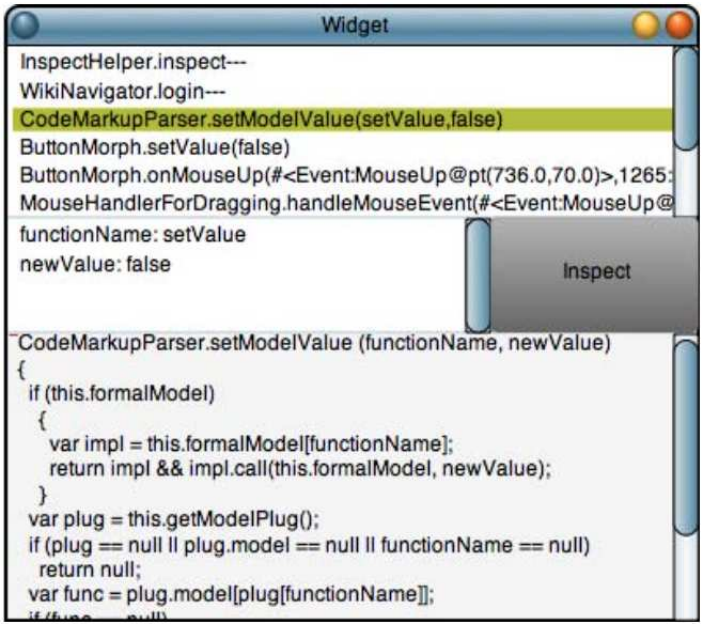


Figure 3.5: Stack Viewer

2. Each Morph in a Lively World is an object that can be inspected for its state. Users can simply launch the Object Inspector from within the Halo menu on the Morph.

3. Lively allows any JavaScript primitive type or an object to be inspected by simply invoking the *inspect* method and passing it as a parameter. Listing 3.1 illustrates the same concept.

```
1        var objectToInspect = {
2            name : 'Harsh',
3            email : 'Hdawar@uvic.ca'
4        }
5        Global.inspect(objectToInspect);
6
```

Listing 3.1: Using the Inspect Method

Figure 3.6 demonstrates the function of the Object Inspector.

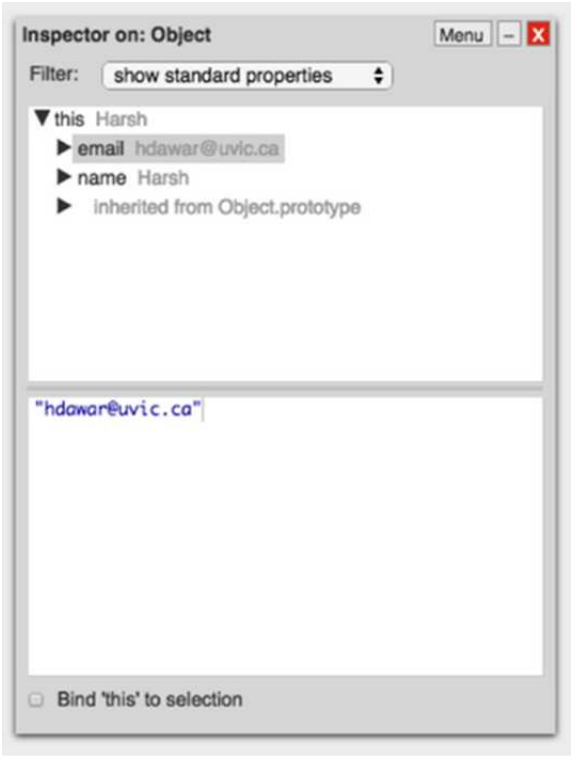


Figure 3.6: Object inspector

4. One of the most common used techniques to debug JavaScript written in Lively is to use *log()*and *alert()* methods. The user can simply pass the content to be logged as a parameter to these methods.

3.5 Designing Complex Morphs From Sub Morphs

As described earlier in Section 3.1, every entity in Lively is composed of Morphs. This Morphic setup can be hierarchical [20] (i.e., Morphs can have children), which in lively are known as Sub Morphs. This format encourages users to build complex Morphs that have immense potential to perform various functions. There are two ways to create such a hierarchical Morphic structure.

3.5.1 Using the Halo

1. Login and create a workspace as shown in Section 3.2
2. Launch PartsBin and search for a *Window* Morph.
3. Drag the *Text Window* Morph from PartsBin onto the world.
4. Search PartsBin for a *Button* morph and place it on the world.
5. Enable the Halo on the *Button* Morph and use the grab option to pick the Morph and place it on the *Window* Morph.
6. The *Button* Morph now becomes a child of the *Window* Morph

3.5.2 Using JavaScript and Lively libraries

1. A user can invoke the *getPart* method to get a Morph from the PartsBin.
2. Extract the *TextWindow* and the *Button* Morphs from the PartsBin.
3. Use the *addMorph* method to make the Button Morph the child of the *TextWindow*.

Figure 3.7: Parent-child morphs

```
var textWindow = lively.PartsBin.getPart('TextWindow', 'PartsBin/Charlie/');
this.world().addMorph(textWindow);
var button = lively.PartsBin.getPart('Button','PartsBin/Inputs/');
textWindow.addMorph(button);
```

Listing 3.2: using the addMorph method

3.6 Styling Morphs in Lively

HTML (Hypertext Markup Language) [13] and CSS (Cascading Style Sheets) [32] are two of the core technologies for building traditional web pages. JavaScript is added to make the behaviour of the web page dynamic. This is in contrast to the Lively Web, which was designed with the concept of uniformity [38]. The only programming language the developer needs to know to publish web applications is JavaScript.

Lively has its own styling environment which allows for the styling of Morphs. This section discusses two main techniques to effectively style Morphs.

3.6.1 Using the Style Editor

1. Enable the Halo menu on the Morph to be styled.
2. Click the Style Editor option from the Halo menu.
3. Use the CSS tab to apply styles using Cascade Style Sheet properties (optional).

4. Use Fill/Border to change color, border radius and border size.

5. Properties shown on the Style Editor depend on the type of Morph being styled (e.g., Text Label Morph has a property to change the text font and size).

6. Layout tab enables users to configure the default layout setting of elements within the Morph.

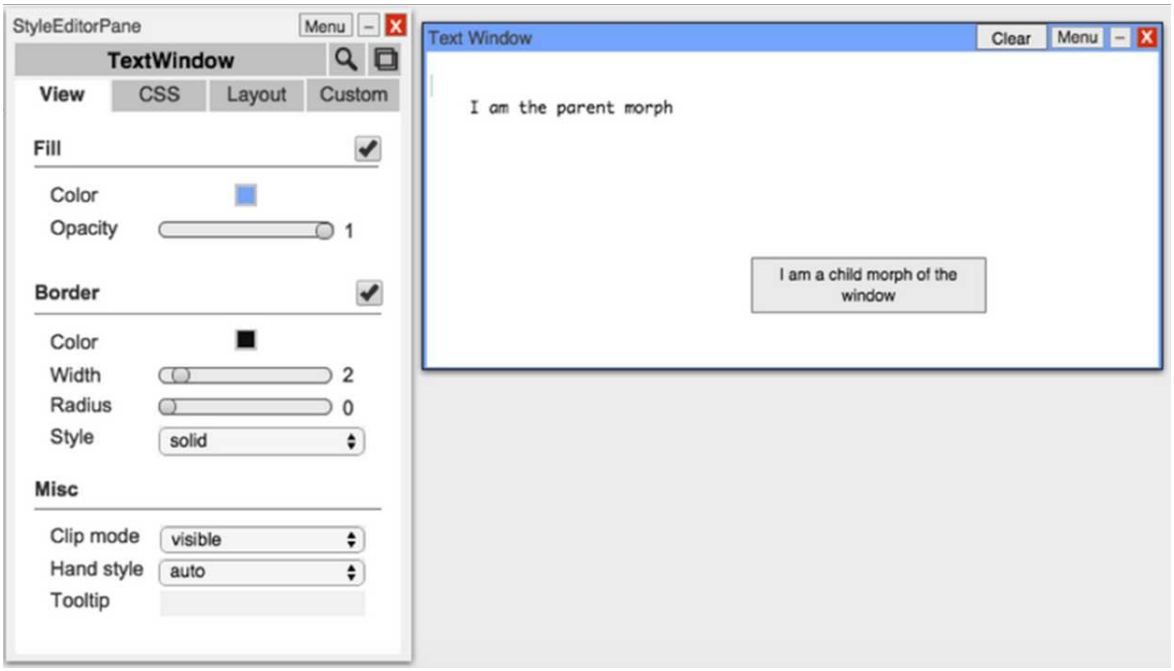


Figure 3.8: StyleEditor in Lively

3.6.2 Using Lively Libraries

1. The Morph class is the base class of all Objects in the Lively Web. Methods from this class can be used for styling.

2. The user can access all the methods made available to a Morph by using the shortcut CMD/CTRL + SHIFT + P on the morph object. Figure 3.8 shows the styling methods on the Morph object.

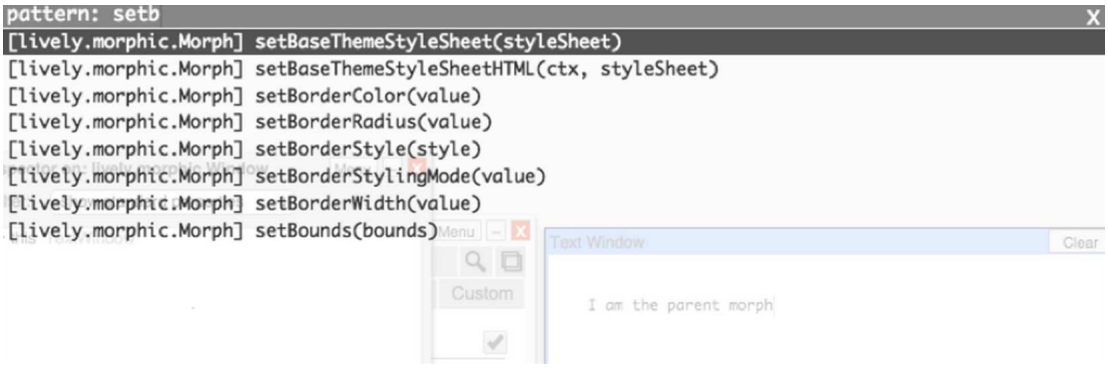


Figure 3.9: Methods for styling in the Morph class

3.7 Sending Messages across Lively Worlds

Lively Worlds that are hosted by the same server are connected to each other via an built-in socket connection known as L2L (lively-to-lively) [20]. Developers can take advantage of this L2L connection to transmit and receive messages between different worlds. Using L2L is a two phase process. This section demonstrates how L2L can be used to transfer messages between two Lively Worlds.

3.7.1 Registering an Action

This is the first phase of setting up a custom L2L message transfer scheme. The world has to be registered to receive content from a peer. The below mentioned steps will register the world with the *recieveMessages* action.

1. Invoke the *registerActions* method on the implicit SessionTracker object.
2. Pass the method a JavaScript object containing function objects.
3. The function objects should accept two parameters. The first parameter is the JavaScript object to be received. The second parameter is the Session object. The session object is used to send a response back to the sender.

```
lively.net.Sessiontracker.registerActions({
    recieveMessages: function(msg, session) {
        //Parse the msg object
        session.answer(msg, {error:null});
    }
```

```
});
```

Listing 3.3: Registering Actions

3.7.2 Transmitting a message over L2L

When a world is registered with an action it is essentially ready to receive messages for that specific action id. The following steps illustrate how to transmit messages to a world which is registered to listen to the *recieveMessage* action.

1. Since a user can simultaneously work on multiple worlds in different tabs, it is important to determine the world with the latest user activity.

2. The last active world of the user can be extracted by invoking the *getUserInfo* method on the Session object.

```
function getLastActiveSession(thisUser, thenDo) {
    this.session().getUserInfo((function (users) {
        Global.Properties.forEachOwn(users, (function (user,
   sessions) {
            if (user === thisUser) {
                var id = sessions.sortBy(function (sess) {
                    return sess.lastActivity;
                }).last();
                thenDo(id);
            }
        }).bind(this));
    }).bind(this));
```

Listing 3.4: Getting last active world

3. Invoking the *sendTo* method on the session object transmits the message to the destination world.

```
var dataToSend = {'Name':'Harsh', 'Email-id':'hdawar@uvic.ca'};
var userName = 'testuser';
this.getLastActiveSession(userName, function(value){
    session.sendTo(value.id, 'recieveMessage', dataToSend,
        function(error){/*Handle any response*/});
});
```

Listing 3.5: Getting last active world

Chapter 4

PALTask2.0

The architecture and working of PALTask are explained in detail by Jain et al. [18]. We also introduced PALTask in Section 1.2 of this report. It is useful to have some background information about PALTask since the primary motive of the application remains unchanged, and we reuse many components in the implementation of PALTask2.0.

This chapter covers the design and implementation of PALTask2.0 using the Lively Web. Readers who are not familiar with Lively are highly encouraged to review the concepts and terminologies discussed in Chapter 3. The lack of documentation in the Lively Platform was an inspiration to structure this chapter in a tutorial based format. We divide the implementation into six phases outlined below and cover each phase in detail with plenty of code snippets plus figures to ease the learning process.

1. Understanding PALTask2.0 architecture
2. Implementing the Chat Client
3. Integrating NLP Services
4. Integrated Search services
5. Training and Integrating WIT.AI
6. Styling the Morphs

Remainder of this chapter discusses each of the above stages in detail.

4.1 PALTask2.0 Architecture

PALTask2.0 has a completely different architecture as compared to its predecessor. The main motivation to pursue a different architecture is to utilize the asynchronous programming model of JavaScript, which promises inherent scalability and performance benefits when used correctly [10]. The architecture of PALTask2.0 comprises of five key Software components mentioned below.

- GUI Morph
- Keyword Extractor
  - RAKE
  - Alchemy Keyword Extraction API
- Alchemy Sentiment Analyzer
- Web Services
  - Google Custom Search API
  - YouTube API
  - Google Maps API
- WIT.AI

Each of the above components are discussed in detail in the following sub sections.

Table 4.1 lists the three subsystems that run asynchronously independent of each other in PALTask2.0. Figure 4.1 shows a high level interaction diagram of PALTask2.0. The following provides an overview of how the subsystems execute in an asynchronous environment.

Table 4.1: PALTask2.0 Sub Systems

Sub System	**Components Used**
Chat Subsystem	Lively-to-Lively (L2L)
NLP Subsystem	Keyword Extractor, Sentiment Analyser
Machine Learning Service	WIT.AI

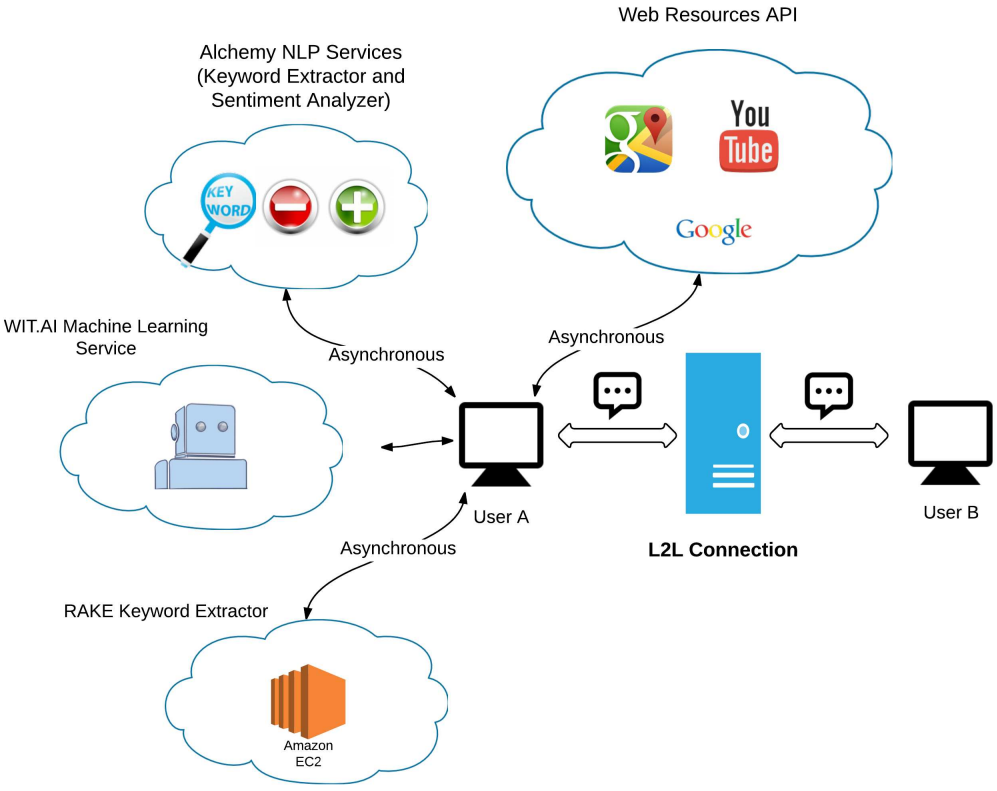


Figure 4.1: PALTask2.0 High Level Architecture

1. Chat Messages are forwarded to the Keyword Extractor, Sentiment Analyser asynchronously (i.e., without pausing the current execution thread).

2. Chat Messages are passed to WIT.AI asynchronously to gather Intents and Entities.

3. Independent of the above two steps, the Chat subsystem forwards the message to the destination World via the Lively L2L connection discussed in Section 3.7.

4. As the reply from Step 1 and Step 2 are returned from the API services their respective callbacks are invoked.

5. The callbacks from Step 1 and Step 2 asynchronously invoke the relevant web searches (i.e., Google and YouTube).

6. The chats continue to be sent independent of the above mentioned steps.

4.2 Implementing the Chat System

This section discusses the implementation of the PALTask2.0 chat server and client using the Lively Web. The chat server in Lively is implemented on top of the L2L connection that was discussed in Section 3.7. There are two notable advantages of the above design decision, of which one is that the L2L is a secure socket connection that has been thoroughly tested as a part of the Lively Implementation. Secondly, using L2L saves us from the hassle of implementing our custom chatting servers allowing us to focus on other important modules that handle retrieving dynamic feedback. The client or the GUI of PALTask2.0 is developed using the concept of Morphic Architecture discussed in Section 3.1. The idea is to have a parent Morph with multiple Sub Morphs, each having a specific functionality. For Example, PALTask2.0 has a main Window, which is the parent Morph of the tool. Components such as users online, buttons, and the message window are Sub Morphs of the parent window. The remainder of this section discusses the implementation of both components in detail.

4.2.1 PALTask2.0 Chat Server Using L2L

To implement a chat server using L2L, the PALTask2.0 Morph must register an action that corresponds to sending and receiving messages. Once registered, the Morph is ready to receive messages from the Session object for the registered action-id. The following steps demonstrate the registration of an action on the SessionTracker object.

- The *onLoad* method is invoked when a Morph is loaded in the World. We register the paltaskMessage action in the *onLoad* event using the *registerActions* method, as shown in Listing 4.1. The parameter passed is an array of function objects which correspond to the action behavior (i.e., code to be executed when the action is triggered).

```
1       lively.net.SessionTracker.registerActions({
2
3        paltaskMessage: (function (msg, session) {
```

```
4            //logic to cache msgs, forward to external services etc
   ..
5            this.addMessage(msg.data.user + ' : ' + msg.data.
   message, msg.data.user, 'left');
6
7            session.answer(msg, { error: null });
8
9        }).bind(this));
10
```

Listing 4.1: Registering the action paltaskMessage

- A user has the option to switch among worlds in a single session, which makes it necessary to determine the last active session of the user. Listing 3.4 shows how to get the last active world of a user.

- The *sendTo* method of the session object allows the transmission of messages. Any JavaScript object/primitive type can be sent over this L2L connection. Listing 4.2 demonstrates how to transmit a simple message using the L2L connection.

```
1        var dataToSend = {msg : 'hi'};
2        this.getLastActiveSession($morph('UsersOnline').selection.
   name, function (value) {
3            sess.sendTo(value.id, 'paltaskMessage', dataToSend,
   function () {
4                //handle response
5            }).bind(this);
6        }).bind(this);
7
```

Listing 4.2: Transmitting a message in PALTask2.0

4.2.2 PALTask GUI/Client Using the Morphic Architecture

Lively offers a unified solution to Web Development [38]. The GUI or the client of the application can be implemented using the Morphic architecture in the Lively Web. In this subsection, we design a basic GUI for PALTask2.0 with minimal features. We cover the styling of Morphs in detail in Section 4.6. Table 4.2 Lists the components and the associated Morphs needed to construct a basic PALTask2.0 GUI.

Table 4.2: Table with PALTask2.0 GUI components and associated Morphs

Component	Morph needed From PartsBin
Receive/Sent Message Box	List
Text Area to type Message	CodeEditor
Send Button	Button
Reset Button	Button
Users Online	List
Main Window(Parent)	Window

As explained in Section 3.5, search PartsBin for Window, List, CodeEditor and Button Morphs. Drag and place the Morphs onto the World. The steps below detail the operations to be performed on each of the Morphs to construct the PALTask GUI.

- **Main Window (Parent Morph):**
  - Enable Halo on Window Morph and use the *Change Extent* menu option to set the extent to pt (1000,550).
  - Using the *setTitle* option from the Halo menu, rename Window Morph to *PALTask2.0 Chat Client*.
- **Text Area:**
  - Enable the Halo Menu on one of the List Morphs and change the title to UsersOnline.
  - Use the *Change Extent* menu option to set the extent to pt (244,386).
  - Enable the *Grab* menu and place the Morph right-aligned to the Window Morph.
  - Use the *Add Morph to* option from the Halo to add this Morph to the Main Window Parent Morph.
- **Code Editor:**
  - Enable the Halo Menu on the Code Editor and change the title to *sendMessage*.
  - Use the *Change Extent* menu option to set the extent to pt (800,70).

  - Use the *Grab* option and place the Morph at the bottom of the Window Morph.
  - Use the *Add Morph to* option from the Halo to add this Morph to the Main Window Parent Morph.

- **Receive/Sent Message Box:**
  - Use the *Change Extent* menu option to set the extent to pt (800,350).
  - Use the *Grab* option to place the Morph on the Main Window above the Code Editor.
  - Use the *Add Morph to* option from the Halo to add this Morph to the Main Window Parent Morph.

- **Send/Reset Buttons:**
  - Use the *Set Label* Menu from the Halo to set the Label of the Buttons.
  - Grab the buttons using the Halo menu and place them under the Code Editor.
  - Use the *Add Morph to* option from the Halo to add this Morph to the Main Window Parent Morph.

Figure 4.2: PALTask2.0 GUI Layout

4.2.3 Wiring The Morphs Together

To create powerful applications utilizing the Morphic architecture, the system needs to be designed such that the Morphs interact with each other. In this subsection we wire or connect the Morphs created in 4.2.2.

Users Online: The Users Online Morph lists the users currently active in the Lively Web. The Morph should load the users as soon as the Parent Morph (i.e., PALTask2.0 is loaded into the world). The method shown in Listing 4.3 updates the users currently connected to the Lively servers. It is inserted into the *onLoad* Method of the Parent Morph.

```
1   function updateOnlineUsers() {
2       if (!this.session()) {
3           show('not online!'); return;
4       }
5       this.session().getUserInfo(function(users) {
6           var offlineList = $morph('UsersList').getValues();
7           Global.Properties.forEachOwn(users, function(user, sessions)
    {
8               if($morph('UsersList').getItem(user)) {
9                   offlineList.remove(user);
10              } else {
11              if(user.indexOf('unknown') < 0) {
12                  $morph('UsersList').addItem(user);
13              }
14          }
15          //adding here
16      }.bind(this));
17      offlineList.push(($world.getUserName(true)));
18      for(var i = 0; i< offlineList.length ; i++ ) {
19          $morph('UsersList').removeItemOrValue(offlineList[i]);
20      }
21  }.bind(this));
22
```

Listing 4.3: Updating Online users

Figure 4.3 shows the Users Online list populating the Users Online.

Receive/Sent Message Box: A message can be added to the Message Box in two different ways, we discuss the implementation of each in detail below.

- Return/Enter Key : The message in the CodeEditor should be transmitted when the Return Key is selected. To catch the event on the return key, the *onKeyDown* script needs to be added to the CodeEditor Morph. Listing 4.4 demonstrates sending a message when the return/enter key is selected.

```
1        function onKeyDown(evt) {
2            var keys = evt.getKeyString();
3            if (keys === 'Enter') {
4                this.get('PALTaskChatWindow').sendMessage(this.
   textString);
5            }
6            return;
7        }
8
```

Listing 4.4: onKeyDown Method

- Send Message Button : The typed message in the CodeEditor should be transmitted when the Send Message Button is clicked. The *doAction* script needs to be defined on the Button Morph as shown in Listing 4.5 in order to capture and handle button events.

```
1    function doAction() {
2        this.get('PALTaskChatWindow').sendMessage(textString,
   function() {
3            $morph('SendMessage').textString = '';
4        }.bind(this));
5    }
6
```

Listing 4.5: doAction Method

4.3 Integrating NLP Services

Retrieving resources from the web is one of they key tasks that PALTask2.0 performs. We use Natural Language Processing (NLP) services mentioned in Section 2.3 to extract dynamic context from user chats. The result from the NLP services forms the

Figure 4.3: PALTask2.0

search key that is used to retrieve resources from web services such as Google and YouTube. Jain et al. [18] investigate and find that using Restful API is a more efficient method for extracting context from the Web Services API. This section discusses the integration of the Restful Web Services into PALTask2.0 using the Lively Web.

The table below lists the Restful APIs used in conjunction with the URL to access them.

Table 4.3: List of Web API Services used with URL

API Used	**RESTful URL**
Keyword Extractor	access.alchemyapi.com/calls/text/TextGetRankedKeywords
Sentiment Analysis	access.alchemyapi.com/calls/text/TextGetTextSentiment
Rake Keyword Extractor	52.25.183.205:10002/

- **Alchemy Keyword Extractor/ Sentiment Analysis API:** The Alchemy API offers a variety of Web Services for NLP. PALTask2.0 uses the Alchemy Keyword Extractor as well as the Sentiment Analyzer NLP services. The keyword extraction API enables the extraction of keywords from any web-based content, HTML and text content. The extracted keywords are used as search keys to gather relevant resources from the web. The Sentiment Analysis API functions to the determine the sentiment/mood of the sentence. In PALTask2.0,

Sentiment Analysis is performed on user conversation. Section 2.3.2 explains the use of Sentiment Analysis.

The service can be assessed using a developer API key, which can be generated online [4]. Although the service is paid, it permits thousand free interactions a day. The query parameters and the API Key, which is not included due to privacy concerns, are appended as shown in Listing 4.6. A simple **GET** request using the jQuery object allows a Restful call to the Alchemy API. The result is returned by using a callback method where it can be parsed for keywords.

```
1    function getKeyWordsFromAlchemy(text, callback) {
2        var url = this.alchemyURL + '&text=' + text + '&outputMode=
   json';
3        Global.$.get(url, null, (function (msg) {
4            callback(msg);
5        }).bind(this));
6    }
```

Listing 4.6: Using Lively to use the Alchemy Keyword Extraction Service

Figure 4.4 shows the response of the Alchemy Keyword Extractor API when the Inspect was invoked on the response object.

The Alchemy Sentiment Analysis can be assessed by simply setting the appropriate URL mentioned in table 4.3.

- **RAKE Keyword Extractor:**

  The RAKE keyword extractor service discussed earlier in Section 2.3.1 is reused by PALTask2.0 as one of the Keyword extraction services. At the time of writing this report, the RAKE implementation in Python used by PALTask does not offer a Restful interface. We use the XMLHttpRequest API to make HTTP requests to the server. A script named getKeywords on parent Morph gets invoked when the text is ready for analysis. Listing 4.7 shows the implementation of *getKeywords*.

```
1    function getKeyWords(text, callback) {
2        var xmlhttp;
3        xmlhttp = new XMLHttpRequest();
4        xmlhttp.onreadystatechange = function () {
5        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
6            //document.getElementById("myDiv").innerHTML=xmlhttp.
   responseText;
```

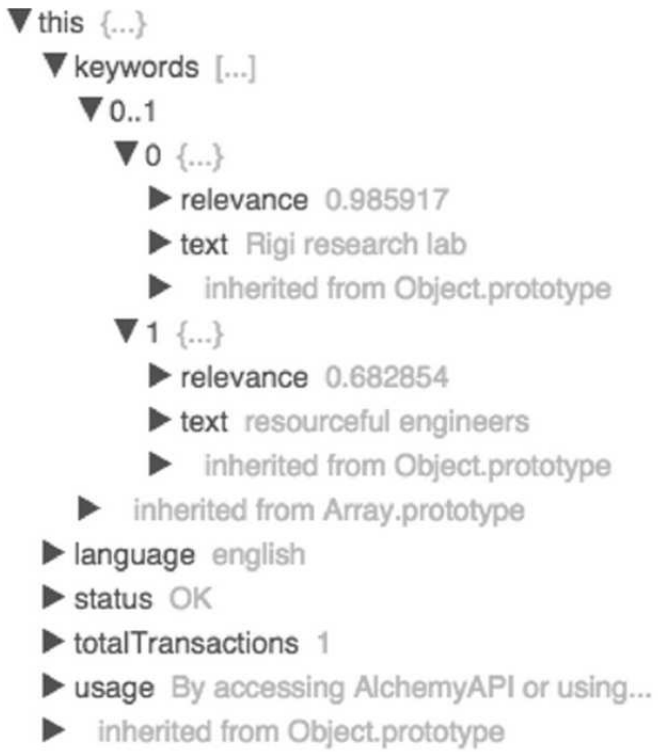


Figure 4.4: Object Inspector on Alchemy Keyword Extraction Response

```
7            var jsonResponse = JSON.parse(xmlhttp.responseText);
8            callback(xmlhttp.responseText);
9            }
10       };
11       var jsonobj = JSON.stringify({
12                   message: text,
13                   api_option: '1'
14               });
15
16       xmlhttp.open('POST', 'http://URL_OF_THE_SERVER/', true);
17       xmlhttp.setRequestHeader('Content-type', 'application/x-www
    -form-urlencoded;charset=UTF-8');
18       xmlhttp.send(jsonobj);
19    }
```

Listing 4.7: onKeyDown Method

4.4 Integrating Search Services with PALTask2.0

Lively Applications are built using the JavaScript programming language allowing us to design PALTask2.0 to execute the process of retrieving results from search services asynchronously. This enables the PALtask2.0 chat system to function independent of the modules responsible for extracting keywords and retrieving results. One notable advantage of the above design is under poor network conditions the primary task of the application (i.e., IM remains unaffected).

PALTask2.0 uses various keyword extraction services for retrieving web resources such as the Google Custom Search API, Youtube API, and the Google Maps API. The type of services to be invoked depends on the type of content the user is looking for. Table 4.4 shows the content type and the corresponding resources invoked.

Table 4.4: Content Type and Services Invoked

Content Type	**Services Invoked**
Text	Custom Search API
Video	YouTube API
Text and Video	Custom Search and Youtube API

This section describes in detail the services used and also the integration of these services with the Lively Web.

4.4.1 Google Custom Search API

Jain et al. [18] discuss how to create a Google custom search engine to explore the entire web. We reuse the same implementation for PALTask2.0. The RESTful API query requires the following parameters

- Custom Search Engine ID
- Google Developer API Key
- Search Query

The Google custom search API can be incorporated in PALTask2.0 using an asynchronous GET request. There are many ways to achieve this; we use the jQuery object since it comes integrated with Lively and satisfies our requirements. The following is

an example of a request that searches the web for the text *Soccer*.

https://www.googleapis.com/customsearch/v1?
cx=009717441976368975003%3Ajkuahk1zsso&key=API_KEY&q=Soccer

As described in Section 4.3, in the callback response of the keyword extraction, the keywords extracted are forwarded to the Google custom search API. The response is parsed to extract the content needed and returned to the callback method.Listing 4.8 demonstrates this concept.

```
function queryGoogleCustomSearchAPI(keyword, callback) {
 Global.$.get(url, numResults, (function (msg) {
     var queryResult = [];
     for (var i = 0; i < numResults; i++) {
         queryResult.push({
             title: msg.items[i].title,
             snippet: msg.items[i].snippet,
             link: msg.items[i].link,
             image: imageurl,
             keywordRequested: keyword
             });
         }
     callback(queryResult);
     }).bind(this)).fail(function (msg) {
     alert('Your Quota of Google API has exhaused!!');
 });
```

Listing 4.8: PALTask2.0 Google Custom Search

4.4.2 YouTube API

To gather video resources for the user, PALTask2.0 is integrated with the YouTube API. The API offers a Restful interface for queries and requires the following parameters.

- The type of content needed (video, audio)
- Google Developer API Key
- Search Query

Here is a sample YouTube RESTful API call query searching for the text *Sushi*,

https://www.googleapis.com/youtube/v3/search type=video& q=sushi&key=API_KEY

The *items* array in the response object contains the Youtube Video IDs. The array is extracted from the Response object and returned to the callback function. Listing 4.9 demonstrates a call to the Youtube API.

```
1  function queryYouTubeAPI(keyword, callback) {
2       Global.$.get(url, null, (function (msg) {
3           callback(msg.items, keyword);
4       }).bind(this));
5   }
```

Listing 4.9: YouTube API integration

4.5 Configuring, Training And Integrating WIT.AI

As examined in Section 2.3 keyword extraction, although very efficient, can lead to poor performance when dealing with specific domain discussions. We use machine learning and attempt to improve the accuracy of the results in domains such as watching movies and songs, searching restaurants and food recipes. The design is structured to support new domains as they are added and trained.

Keeping in consideration the aim of this project, we do not dwell on the details of machine learning. Instead, we use WIT.AI, a Web-based service that can be easily configured, trained and integrated with PALTask2.0. WIT.AI splits the Natural language into Intents and Entities. Intents can be understood as extracting the motive of the text, whereas, entities are words of interest within that Intent. Table 4.5 defines the Intents, and their respective Entities created for PALTask2.0.

Table 4.5: WIT.AI Intents and Entities

Intents	**Entities**
play_video	search_query
food_reciepe	food
restaurantSearch	search_query, location

Using WIT.AI allows the reuse of Intents and Entities created by the community along with their trained data set. We cover the training and integration of WIT.AI in detail in this section.

4.5.1 Configuring and Training WIT.AI

- Create an app by visiting the WIT.AI create app page (Login into GitHub will be required).
- Create the Intent and their corresponding Entities defined in Table 4.5.
- Enter an expression and train it by defining the Intent and the Entity for that expression. For example,
  Input Expression: do you know how to make sushi?
  Intent: *food_recipe*
  Entities: (food, *sushi*)
- Goto the settings tab and scroll down to API details.
- An API key is required to access the service from within PALTask2.0, copy the API key provided.
- Test the WIT.AI service created by using the curl command as shown in the home page.
  Response Received :

```
{"_text" : "Do you know how to make sushi?",
 "outcomes" : [ {
     "_text" : "Do you know how to make sushi?",
     "intent" : "food_reciepe",
     "entities" : {
             "food" : [ {
             "value" : "sushi"
                     } ]
         },
     "confidence" : 0.999 }

```

Listing 4.10: Sample Response From WIT.AI

4.5.2 Integrating WIT.AI with PALTask2.0

The process of configuration and training described in the previous section is a prerequisite to operating WIT.AI. PALTask2.0 asynchronously sends each chat message to WIT.AI to understand its intent; this is done independent of the Chat communication, Keyword Extraction and other Search queries to gather web resources. If a chat message has an intent that can be handled by PALTask2.0, its handler parses the response object and queries the relevant service for web resources. To improve the quality of the feedback received we consider responses only with a confidence of 80 percent and above.

Listing 4.11 demonstrated how an API call to the WIT.AI Service is made.

```
1           function getResultsFromWitAI(message, callback) {
2                Global.$.ajax({
3                    url: 'https://api.wit.ai/message',
4                    data: {
5                        'q': message,
6                        'access_token': TOKEN
7                        },
8                    dataType: 'jsonp',
9                    method: 'GET',
10                   success: (function success(response) {
11                       callback(response);
12                   }).bind(this)
13               });
14
```

Listing 4.11: WIT.AI Integration with PALTask2.0

4.6 Styling the Morphs

Considering the aim of the project, this section forms a critical stage of the development of PALTask2.0. We redesign the basic GUI layout created in Section 4.2.2. The intention is to create a design that accommodates user chat, feedback, and web resources while being user intuitive and accessible.

There are many User Interface designs possible for an application such as PALTask2.0. Figure 4.5 shows the design created for this project. This section covers the implementation details of styling various Morphs to accomplish the design shown.

4.6.1 Styling the Main Window

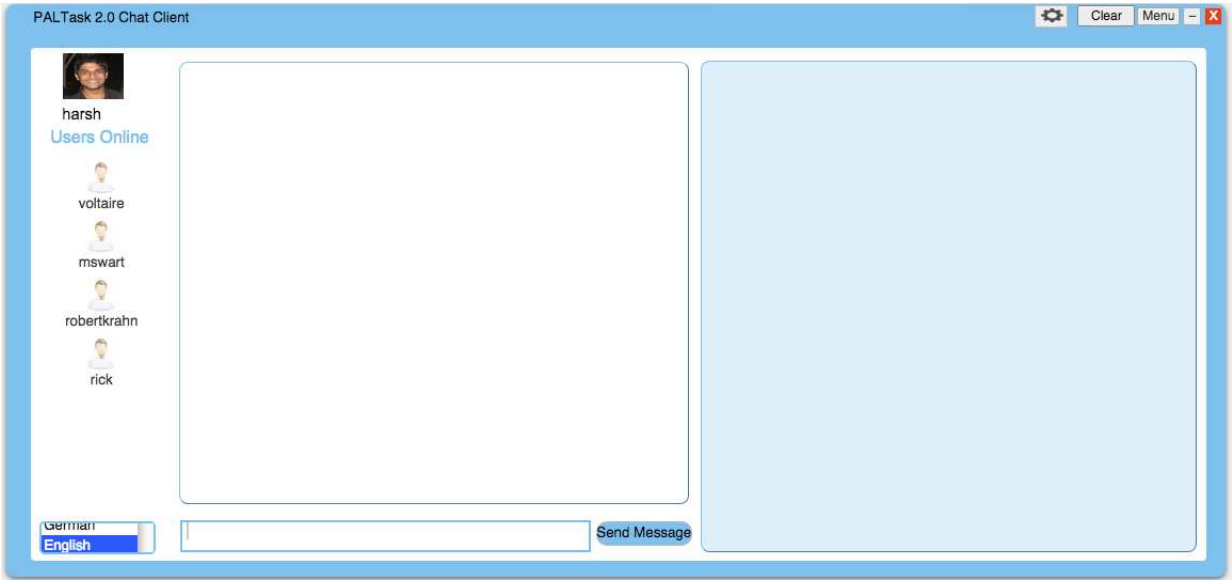


Figure 4.5: PALTask2.0 With Styling in Lively

- Activate the Halo on the *ChatMesages* Morph and re-size it to pt (470,410).
- Get a *MorphList* from the PartsBin and place it on position pt (620,10). Rename the Morph to *feedBack* and set the extent of the Morph to pt (460,460).
- Use the Style Editor on the *feedBack* Morph to set the Fill color to *DFEFFA*, Border Radius to 10 and Border Width to 1.
- Use the Style Editor on the *Send Message Button* Morph to set the Border Radius to 10, Border Width to 1 and the Fill Color to *7EC0EE*.
- We intend to show the Users with their display pictures, hence we need to replace the List Morph with the Morph List Morph to achieve the same.
- Grab the *MorphList* from PartsBin. Set the Extent to pt (120,330) and the Position as pt (6,90).
- To present a Display Picture of the user logged in, grab an *Image* Morph from PartsBin, set the extent to pt (50,50) and the position to pt (30,5).

- Enable Halo on the *Main Window* Morph and set the Border Radius to 8, Border width to 4 and the Fill Color to *7EC0EE*.
- Enable Halo on the Title *Text* Morph. Use the object Inspector to invoke the *setTextColor*method with the parameter to color white.

4.6.2 Re-implementing the UsersOnline Morph

The design implemented in Section 4.2.2 presents the users online in a textual format. In this section, we extend the existing system to support the display of user display pics along with the name, as seen in Figure 4.6. We reuse the concept of Morphic architecture to design a Morph with two Sub-Morphs, one which represents the Image, and the other with displays the user-name. Since the List of users online is updated real time by the application, the Complex Morph is created using the libraries provided by the Lively Web and not the Halo menu. The steps below describe in detail how such a Morph can be created using the Lively Web.

Figure 4.6: Morph representing a user in PALTask2.0

- Using the *makeRectangle* method we create a Rectangle Morph that behaves as the Parent Morph. The reference to the Morph is saved in a variable so it can be accessed again.
- The *fromURL* method in the Image class is used to create an Image Morph. The reference is saved in a variable named *img*.
- A Label to represent the user name is created using the *makeLabel* method in the Text class. The reference is saved in a variable named *label*.
- The Label and the Image Morph are made the children of the Rectangle Morph by using the *addMorph* method.

- The position of the Sub Morph is adjusted using the *setPosition* method to align the Image and Label Morph as required.

```
function createUserItem(userName) {
    var imgWidth = 30,
        imgHeight = 30,
        textHeight = 10,
        width = 90,
        height = imgHeight + textHeight;

    var img = lively.morphic.Image.fromURL(url, lively.rect(width / 2 - imgWidth / 2, 0, imgWidth, imgHeight));
    var label = lively.morphic.Text.makeLabel(userName, { fixedWidth: true, clipMode: 'hidden', extent: pt(width, textHeight), align: 'center' });
    var item = lively.morphic.Morph.makeRectangle(0, 0, width, height).applyStyle({ fill: null, borderWidth: 0 });
    //Make the Image and Label the child Morphs
    item.addMorph(img);
    item.addMorph(label);
    label.setPosition(pt(0, imgHeight));
    return item;
}
```

Listing 4.12: Creating Morph to Represent Users

4.6.3 Creating Dynamic Video Morphs

To present the user with Video Web Resources, a Morph capable of playing YouTube Videos, as seen in Figure 4.7 is developed. The Morph is created using the Lively Libraries since it has to be added dynamically to the *feedBack* Morph. The steps below outline how the Video Morph is created.

- Download an *HTMLWrapperMorph* from PartsBin and set the extent to pt (350,230).
- set the HTML to an empty page using the ObjectInspector and publish the Morph with the name *suggestionBox*.
- use the *setHTML* method of the HTMLWarpperMorph to set the HTML to an embedded YouTube URL link as shown in Listing 4.13 .

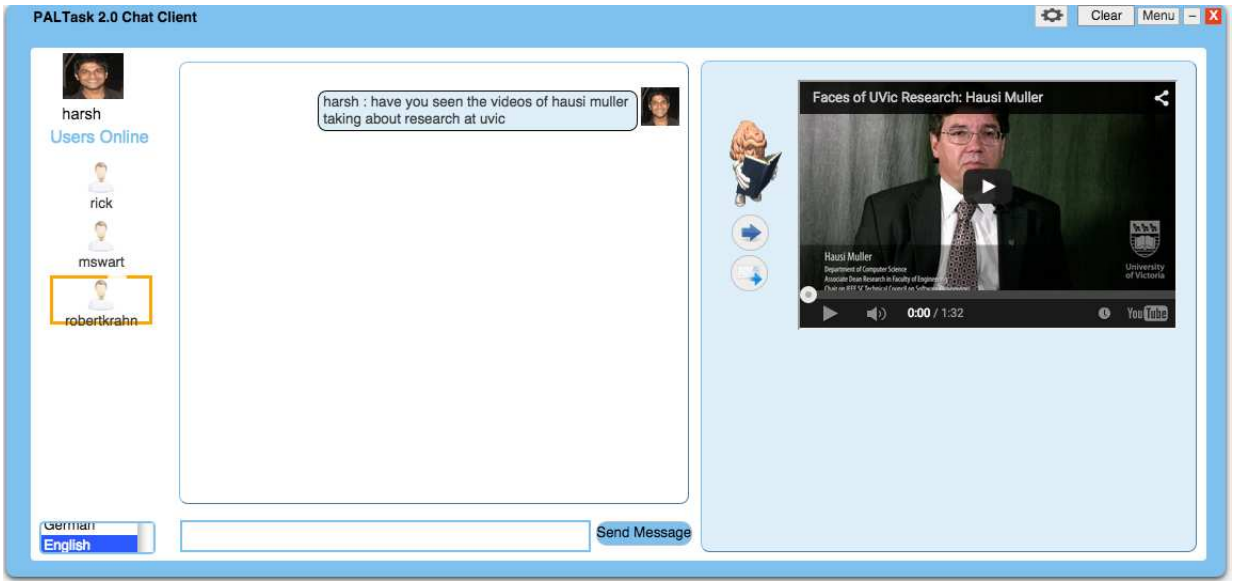


Figure 4.7: Video Morph in PALTask2.0

```
var item = lively.morphic.Morph.makeRectangle(0, 0, 432, 250).
applyStyle({ fill: null, borderWidth: 0 });
var suggestionBox = lively.PartsBin.getPart('Suggestions', 'PartsBin
/HarshDPalTask');
suggestionBox.setHTML('<iframe width = "352" height="230" src="https
://www.youtube.com/embed/fxd297SVVr0"></iframe>');
```

Listing 4.13: Creating a Video Morph

4.6.4 Creating Dynamic Google Map Morphs

As described in Section 4.5, WIT.AI is trained to recognize an Intent of locating or exploring restaurants. To represent the data visually to the user, we need a Morph capable of embedding a Google Map as shown in Figure 4.8.

Listing 4.14 demonstrates the creation of a Map Morph using the Lively Web libraries.

```
//searchResult is the Entity extracted by WIT.AI

var suggestionBox = lively.PartsBin.getPart('Suggestions',
                    'PartsBin/HarshDPalTask');
suggestionBox.setOpacity(0);
item.addMorph(suggestionBox);
```

```
6    suggestionBox.setHTML('<iframe width = "352" height="230"
                src="https://www.google.com/maps/embed/v1/search?q=' +
    searchResult +              '&key=API\_KEY">' + '"></iframe>');
```

Listing 4.14: Creating a Map Morph

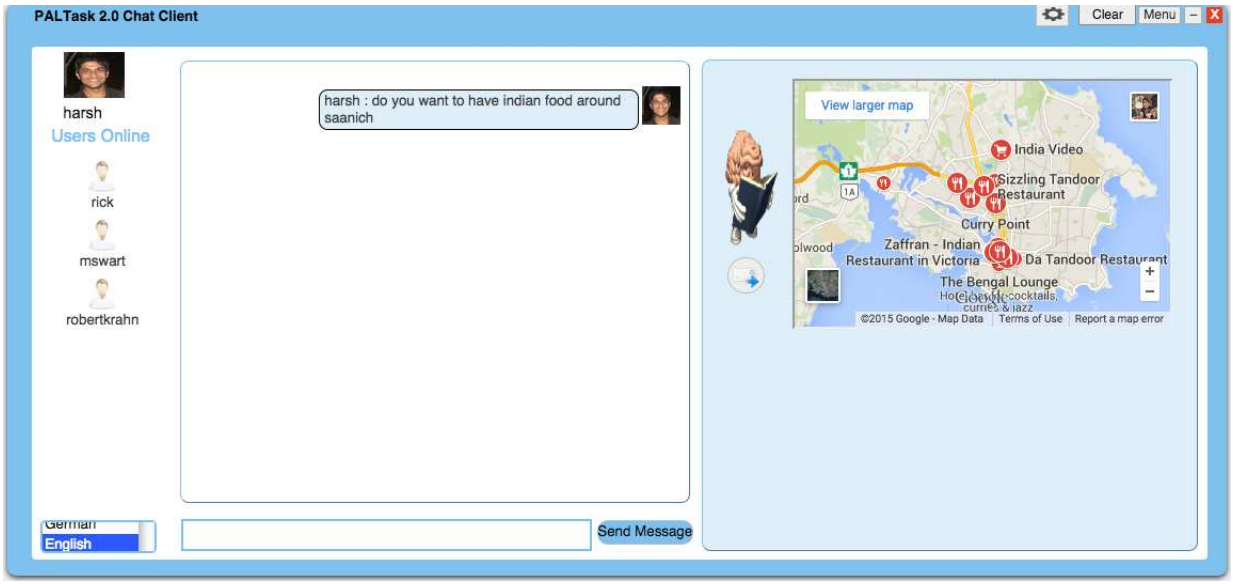


Figure 4.8: Map Morph in PALTask2.0

4.6.5 Adding Retrieved Web Resources to PALTask2.0

The sections above demonstrate how to create Morphs that can represent the various Web Resources retrieved by PALTask2.0. To present the Morphs created in Section 4.6.1 to the user, we need to add it to the *feedBack* Morph. The *addMorph* method on the *feedBack* Morph enables us to add the Video, Maps Morphs as children. Listing 4.15 demonstrates the above concept.

```
1            //item is the reference of the Morphs to be inserted.
2            $morph('feedBack').addMorph(item);
```

Listing 4.15: Adding Web Resources to PALTask2.0

Chapter 5

Evaluation

It is essential to evaluate PALTask2.0 as it aids in identifying strengths and weaknesses of the Lively Web programming environment, analyzing the efficacy of WIT.AI, validating the accuracy of retrieved resources and finally, creating road maps for future enhancements of PALTask2.0. This section further discusses the benefits mentioned above in detail.

5.1 Lively as a Programming Environment

In a conventional Web programming environment managing Web Servers, Integrated Development Environment, and product integration are some of the routine tasks performed by a web developer. Aside from the mentioned tasks, often developers are required to use multiple frameworks to handle the server side and GUI implementations, such as SPRING Framework[1] for Back-end Logic and Java Server Faces[2] for presentation. In contrast, the Lively Web development is based on the principle of uniformity (i.e., JavaScript is the only technology required to develop an interactive Lively Web application). This grossly reduces the overhead allowing the developer to focus on the functionality rather than the underlying technology.

Apart from inheriting the advantages that come with the JavaScript programming language such as improved scalability and performance, Lively Web provides the necessary tool-set required to further take advantage of JavaScript. We mention some of the built-in tools/features that greatly enhance the Lively programming experience below.

[1] http://projects.spring.io/spring-framework/
[2] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

- **Object Inspector**: Introduced in Section 3.4, the Object Inspector is a useful tool for a weekly typed programming language such as JavaScript. Any Object including Morphs can be inspected with ease using the Object inspector.

- **Liveliness**: In most of the conventional web programming environments when changes are made to the server side logic, the application server needs to be restarted with the latest compiled source code. Conversely, the Lively web is always active and changes made to the Morphs are visible in real time without the hassle of restarting or reloading the application.

- **Ease of Designing GUIs**: The simple drag and drop feature in the Lively Web enables users with little user interface development experience to create rich web-based applications. Whereas experienced developers have the flexibility to create powerful Morphs dynamically using Lively libraries.

- **Collaboration**: Lively is truly a collaborative environment. Many Morphs created as a part of this project are available in the PartsBin. Moreover, they are ready to be used freely by any user in their own application. For example, a user who needs a list of users online can simply use PartsBin to extract the *UsersOnline* Morph created for PALTask2.0 and customize it as per their requirements.

- **Lively-to-Lively**: The built-in L2L connection in Lively provides user with the potential to create relatively complex applications with ease. The only information needed to send a message across is the session id of the user, which is accessible using the Lively libraries. Applications such as video players, video conferencing applications have been built successfully on top of the L2L connection. PALTask2.0 heavily utilizes the L2L connection to transmit messages and web resources.

Along with the above mentioned strengths, there are a few recognized weaknesses of Lively Web. We discuss these drawbacks and make suggestions that can further improve the Lively Web Experience.

- **Documentation:** Unlike other APIs and programming environments the Lively Web is limited when it comes to the official documentation available. A general overview of Lively is provided in Lively 101, which is an interactive tutorial created using Lively itself. Although it helps users to get started with

Lively, it leaves gaps that can only be filled by tediously searching through the Lively source code. One of the key tasks in making Lively more accessible to users is by creating detailed API documentation such as JavaDoc.

- **Performance:** Overall the Lively Web is very responsive, however we did notice a lag and drop in performance with an increase in the number of Morphs loaded into the world. One possible reason for this could be that the entire Lively Library is loaded on the client side when the world is loaded. A possible solution to this problem is to download only the Basic Morphs while loading the world and fetch the Morphs lazily when requested by the user.

  Another concern noted affecting the performance of the application relates to the consumption of space. Lively web pages are relatively heavy and take up considerable RAM space. A single Google chrome tab with PALTask2.0 open takes up around 700 MB of RAM. The consumption goes up as more Morphs are created and added to the *feedBack* Window. For Lively to be successfully deployed in large scale applications the Lively team needs to consider testing the interface for performance benchmarks and improvements.

- **Security:** Since Lively is under active research and development, it currently does not support an authentication module. Users can simply login by providing a user-name, which gives them access to all the Morphs in the PartsBin. Users can then overwrite or modify Morphs without the permission of the author. To encourage users to contribute and join the Lively community, Lively needs an authentication module that protects the Morphs created by the users.

  Additionally, users can simply change a Morph by accessing the Halo menu. In a production environment, it will be beneficial to disable the capability of users to change the state of the Morph. This concern requires attention and prompt resolution by the Lively team to encourage developers to adopt the Lively Web as their development platform.

- **Collaboration:** In large scale projects, enterprises need sophisticated source code version tools such as GIT and SVN to manage and keep track of the changes. Currently, Lively does not support integration with such tools nor does it offer any built-in Morph such as the Object Inspector to manage Morph changes over time.

We believe that the Lively Web experience can be greatly improved if some or all of the suggestions mentioned above are implemented.

5.2 Evaluating PALTask2.0 Accuracy Improvements

Jain et al. [18] evaluate the accuracy of PALTask by conducting an experiment that evaluates user satisfaction. In their experiment, the results indicate that the Personal Context Sphere significantly increased satisfaction amongst users by personalizing the retrieved resources. PALTask2.0 reuses components from its predecessor, more specifically it uses the RAKE keyword extractor and the sentiment analyser and hence inherits the accuracy demonstrated [18]. In this section we focus on evaluating the components added to PALTask2.0 (i.e., the WIT.AI machine learning service discussed in Section 4.5).

5.2.1 Evaluating WIT.AI Video Retrieval

In this section we compare the accuracy of the results of the WIT.AI module compared to the RAKE Keyword Extractor. We achieve this by simulating general user conversations and evaluating the results obtained. The web resource retrieved is placed into the following grades.

(A) **Accurate and Relevant**: Signifies the web resource retrieved is the content the user would look up manually.

(B) **Accurate but non Relevant**: Signifies the web resource is related to the chat conversation but not of direct help to the user.

(C) **Inaccurate**: Signifies that the web resource will likely be ignored by the user.

Table 5.1 demonstrates WIT.AI improving the accuracy of retrieved results. Out of the 16 sample messages transmitted, PALTask2.0 was able to display 13 relevant and accurate web resources.

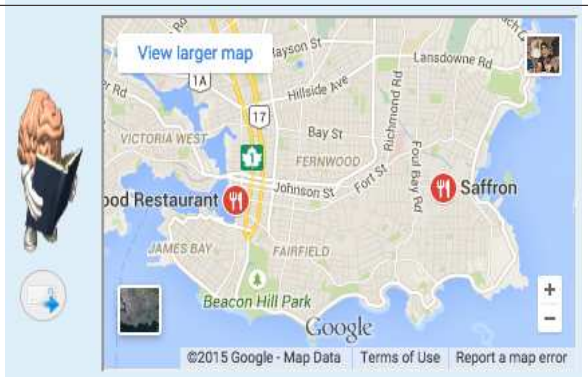
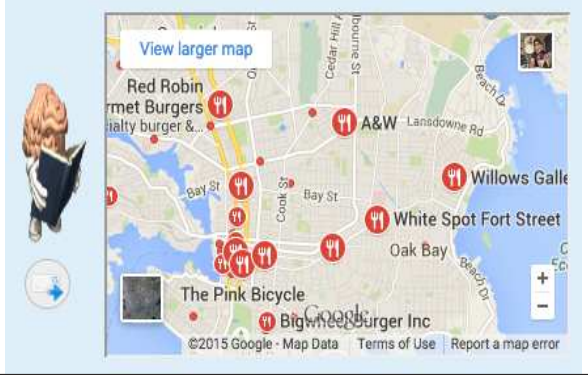
5.2.2 Evaluating WIT.AI Restaurant Finder

PALTask2.0 uses WIT.AI to automate the process of looking up restaurants on Google Maps. Table 5.2 demonstrates the effectiveness of WIT.AI to extract web resources related to finding restaurants.

Table 5.1: Evaluating WIT.AI

Chat Message Passed	**WIT.AI**	**Keyword Extractor**
Have you seen the American Sniper?	A	A
Do you want to watch Harry Potter and the chamber of secrets?	A	B
The Bourne Ultimatum movies are so good!!	C	A
Check out the trailer of the titanic	B	C
Check out the trailer of the lord of the rings.	A	C
Do you want to see any of the rambo movies?	A	B
See the movie ash vs evil dead..	A	B
Watch the video of summer by calvin harris.	A	A
Listen to the uptown funk by bruno mars.	A	A
Did u see the wimbledon 2015 finals?	A	C
Can we please see the highlights of the fifa world cup?	B	B
Have you seen videos of Yvonne Coady taking about uvic?	A	A
Hausi mullers' videos on research at uvic are interesting!!	A	B
You have to watch the batman and the dark knight.	A	B
Lookup the Bruno mars song Treasure.	A	B
Did u see the Grammy performance of Bruno mars it was really good!!	A	B

Table 5.2: Evaluating Map Morphs

Chat Message Sent	Map Morph Presented	Grade
Have you been to the Lemongrass Thai restaurant in Victoria?		A
Do you want to eat Persian food? I live close to Oakbay.		A
I want to try a sushi restaurant tonight somewhere close to Duncan.		A
Any Burger restaurants around? I am hungry.		A

5.3 Evaluation Summary

This chapter evaluated the effectiveness of the Lively Web as a programming environment. It greatly reduces the developers work load due to its live nature and the uniformity principle. During the implementation of PALTask2.0, we identified a few shortcomings of Lively that need to be addressed urgently. We also compared the performance of WIT.AI and the RAKE keyword extractor by running some simple simulation tests and found an improvement in the accuracy of the retrieved web resources.

Chapter 6

Conclusions and Future Work

This chapter summarizes the report by outlining the contributions and presenting ideas for future improvement of PALTask2.0 and the Lively Web.

6.1 Summary

This project investigated the effectiveness of the Lively Web as an effective web programming environment by developing a complex web application. We undertook a comprehensive study in Chapter 2 to analyze some notable applications built using Lively. We then concluded that none of the applications presented in the literature fully utilize the potential of Lively (i.e., to create rich web applications). Next, we discussed why implementing an application such as PALTask2.0 would be an ideal test bed to evaluate the capabilities of the Lively Web. We studied in detail the components used by PALTask and reviewed the ones that can be reused in PALTask2.0.

One of the secondary goals of the project was to contribute to the available documentation regarding the usage of the Lively Web. This was inspired by the lack of available resources. Currently, Lively only has a single resource known as, Lively 101. Chapter 3 provides detailed information about Lively terminologies and also discusses the key concepts in a step by step tutorial. Users who have gone through this chapter can further advance their practical skills by following Chapter 4, which presents in detail the implementation of PALTask2.0 with plenty of code snippets and diagrams.

In Chapter 4, we contribute to PALTask by integrating a machine learning service known as WIT.AI. We used this service to train PALTask2.0 in understanding certain intents and respond appropriately. This relieves the user from the responsibility of

selecting the type of resources needed (i.e., talk about making sushi and PALTask2.0 retrieves a video of sushi recipes, whereas a user talking about sushi restaurants would be presented with appropriate restaurants located on a map). In Chapter 5, we determined the accuracy improvements of integrating WIT.AI into PALTask2.0 by running an experiment with simulated user chats and determining the relevance and accuracy of results received. We concluded that WIT.AI considerably improved the accuracy and also automated the process of web resource type selection.

6.2 Future Work

This section covers the potential future work for improving Lively Web and further enhancing PALTask2.0.

6.2.1 Lively Web

The Lively Web architecture is suited for applications that have a drag and drop model. It would be interesting to observe Lively's performance and effectiveness when creating a web application such as Lucid charts.[1] One could also investigate how a Lively Application can be integrated with JavaScript Libraries, such as Socket.IO .[2] In my opinion, a users guide on integrating third party libraries would attract users to the Lively community. Browser based games are one of the best examples of an interactive web application. Lively Web due to its live nature can be an effective tool to create such games as well. Overall, there is significant scope for future work that awaits the Lively Web to comprehend fully and reveal its potential.

One of the best examples of interactive software is computer games [3], it would be interesting to evaluate how Lively performs in context to browser based games similar to Flash Player. Overall, there is a huge scope of future work that awaits the Lively Web to fully understand and demonstrate its potential.

6.2.2 PALTask2.0

PALTask2.0, a complex context-aware chatting system developed using the Lively Web, is presented in this project. PALTask2.0 builds on its predecessor by adding

[1] www.lucidchart.com
[2] http://socket.io/
[3] http://zyngagames.com/

features such as, improved accuracy, higher accessibility, user intuitive GUI and automatic resource type retrieval. The following is a list of features that can be incorporated into PALTask2.0 to further improve the user experience.

1. Using the JavaScript WebRTC API[4] to integrate audio and video communication among peers.
2. Extending WIT.AI to enable speech recognition.
3. Integrate the Personal Context Sphere.
4. Using SOCKET.IO to implement a chatting server instead of L2L. This would open options for integrating PALTask into mobile applications.
5. Training WIT.AI to identify and handle more Intents.
6. Integrate Login using social accounts such as, Facebook and Twitter.

[4] http://www.webrtc.org/

Appendix A

Additional Information

PALTask2.0 was developed on the following Lively Server[1], it can be assessed and used freely using any modern browser. Users can access PALTask2.0 source code by loading the PALTask2.0 Morph in their world and launching the script editor from the Halo Menu. Most of the Morphs created for this project can be reused by accessing them from the PartsBin.

Users looking to extend PALTask2.0 are highly encouraged to read Chapters 3 and 4 of this report. The training data set used for WIT.AI is available on the following URL[2] and is open to extension.

Since the web services used by PALTask2.0 are paid services, users are requested to create your version of developer keys as described in Chapter 4 for using or extending this application.

[1] www.lively-web.org
[2] https://wit.ai/harshuvic/PALTask2.0

Bibliography

[1] Lively web [Online]. http://www.lively-web.org/.

[2] Apache 2.0. Natural language tool kit. http://www.nltk.org/.

[3] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and Ubiquitous Computing*, pages 304–307. Springer, 1999.

[4] Alchemy. Generating alchemy API key. http://www.alchemyapi.com/api/register.html.

[5] IBM Alchemy. Alchemy keyword extraction API. http://www.alchemyapi.com/api/keyword-extraction.

[6] IBM Alchemy. Alchemy sentiment analysis API. http://www.alchemyapi.com/api/sentiment-analysis.

[7] Aneesha. Rake implementation. https://github.com/aneesha/RAKE.

[8] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *In Proceedings IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009.

[9] Gobinda G Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003.

[10] Flaviu Cristian. Synchronous and asynchronous. *Commun. of the ACM*, 39(4):88–97, April 1996.

[11] Uri Dekel and James D. Herbsleb. Improving API documentation usability with knowledge pushing. In *Proceedings of the 31st International Conference on Soft-*

ware Engineering (ICSE 2009), pages 320–330, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Google. Google chrome dev tools: Debugging javascript. https://developer.chrome.com/devtools/docs/javascript-debugging.

[13] Ian S Graham. *The HTML sourcebook*. John Wiley & Sons, Inc., 1995.

[14] Md Mahmudul Hasan and Woakil Uddin Ahamed. Saas: A new era for call center based on cloud computing. *ULAB Journal of Science and Engineering*, 3:44–50, 2012.

[15] Satoshi Horiguchi, Akifumi Inoue, Tohru Hoshi, and Kenichi Okada. Gachat: A chat system that displays online retrieval information in dialogue text. In *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW 2009), Sanibel Island, Florida*, 2009.

[16] IBM. IBM Watson natural language classifier. http://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/nl-classifier.html.

[17] Daniel Ingalls, Krzysztof Palacz, Stephen Uhler, Antero Taivalsaari, and Tommi Mikkonen. The lively kernel a self-supporting system on a web page. In *Self-Sustaining Systems*, pages 31–50. Springer, 2008.

[18] Pratik Jain, Andreas Bergen, Lorena Castaneda, and Hausi Müller. PALTask chat: A personalized automated context aware web resources listing tool. In *Proceedings IEEE Ninth World Congress on Services 2009*, pages 154–157. IEEE, 2013.

[19] Alan Kay, Dan Ingalls, Yoshiki Ohshima, Ian Piumarta, and Andreas Raab. Steps toward the reinvention of programming. Technical report, National Science Foundation, 2006.

[20] Robert Krahn. Lively 101: Getting started with Lively. http://lively-web.org/users/robertkrahn/Lively-101.html.

[21] Robert Krahn, Dan Ingalls, Robert Hirschfeld, Jens Lincke, and Krzysztof Palacz. Lively wiki a development environment for creating and sharing active web content. In *Proceedings of the 5th International Symposium on Wikis*

and Open Collaboration (WikiSym 2009), pages 9:1–9:10, New York, NY, USA, 2009. ACM.

[22] Janne Kuuskeri, Janne Lautamäki, and Tommi Mikkonen. Peer-to-peer collaboration in the Lively kernel. In *Proceedings of the 2010 ACM Symposium on Applied Computing SAC 2010*, pages 812–817, New York, NY, USA, 2010. ACM.

[23] J. Lincke, R. Krahn, D. Ingalls, and R. Hirschfeld. Lively fabrik a web-based end-user programming environment. In *Proceedings Seventh International Conference on Creating, Connecting and Collaborating through Computing (C5 2009)*, pages 11–19, Jan 2009.

[24] Jens Lincke, Robert Krahn, Dan Ingalls, Marko Röder, and Robert Hirschfeld. The Lively partsbin a cloud-based repository for collaborative development of active web content. In *Proceedings 2012 45th Hawaii International Conference on System Science (HICSS)*, pages 693–701. IEEE, 2012.

[25] Lively Team. Stanbol implementation. http://www.lively-web.org/users/cschuster/semantics.webm.

[26] Lively Web. Setting up an own lively server instance. https://github.com/LivelyKernel/LivelyKernel.

[27] Frank Ludolph, Yu-Ying Chow, Dan Ingalls, Scott Wallace, and Ken Doyle. The Fabrik programming environment. In *Proceedings IEEE Workshop on Visual Languages*, pages 222–230. IEEE, 1988.

[28] John Maloney. Morphic: The self user interface framework. *Self*, 4, 1995.

[29] John Maloney and Walt Disney Imagineering. An introduction to morphic: The Squeak user interface framework. *Squeak: OpenPersonal Computing and Multimedia*, 2001.

[30] John H Maloney and Randall B Smith. Directness and liveness in the morphic user interface construction environment. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, pages 21–28. ACM, 1995.

[31] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. A machine learning approach to building domain-specific search engines. In *Proceed-*

ings International Joint Conference on Artificial Intelligence, volume 99, pages 662–667, 1999.

[32] Eric A Meyer. *Cascading style sheets: The Definitive Guide*. O'Reilly Media, Inc., 2004.

[33] Mozilla. Mozilla Firebug: Web development evolved. http://getfirebug.com/errors.

[34] Tim O'reilly. What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*, (1):17, 2007.

[35] Anand Ranganathan, Roy H Campbell, Arathi Ravi, and Anupama Mahajan. Conchat: A context-aware chat program. *Pervasive Computing, IEEE*, 1(3):51–57, 2002.

[36] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. *Text Mining*, pages 1–20, 2010.

[37] Antero Taivalsaari. Mashware: The future of web applications. 2009.

[38] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. Web browser as an application platform: The Lively kernel experience. 2008.

[39] Mark J Taylor, J McWilliam, H Forsyth, and S Wade. Methodologies and website development: a survey of practice. *Information and Software Technology*, 44(6):381–391, 2002.

[40] Dolf Trieschnigg, Mariët Theune, and Theo Meder. Learning to rank folktale keywords. In *Book of Abstracts of the 23rd Meeting of Computational Linguistics in the Netherlands: CLIN 2013*, page 92, 2013.

[41] Norha M Villegas. *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems*. PhD thesis, Department of Computer Science, University of Victoria, 2013.

[42] Wit.AI. Natural language for developers. https://wit.ai/.