POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Designing the Integration of Conversational AI with Web Architectures

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Gianluca Spadone**

Student ID: 946286
Advisor: Prof.ssa Maristella Matera
Co-advisors: Marcos Baez, Emanuele Pucci
Academic Year: 2020-21

Abstract

Everyone should be enabled to access the Web. Individuals who are blind or have visual impairments, elderly people, or users who are unable to use their hands or eyes in a specific context should access the Web. Nowadays, assistive technologies, and in particular screen readers can help users access Web information through channels that are different from the visual one, e.g., by voice and hearing; nevertheless, the resulting access paradigm is very complicated even for those users, e.g., blind and visually impaired individuals, who spend years and many efforts to learn how to use these technologies. In addition to the intrinsic complexity that characterize the fruition of Web content through screen readers, accessibility guidelines are not completely fulfilled in the majority of websites, and this lack causes severe problems when a screen reader has to interpret and read the website content. Voice-based conversational assistants, e.g., Alexa, could aid in the discovery of punctual Web information responding to basic inquiries. However, they are not able to fully support navigation of a website.

Given these lacks, this thesis aims to provide a new way to access the Web and navigate it using voice commands. The idea is to enable users to navigate content and services accessible on the Web by "talking to websites" instead of browsing them visually, by expressing their goals in natural language and accessing websites through a dialog mediated by a Conversational Agent (e.g., a voice-based browser plugin or a Web interface). This paradigm is enabled by a Web platform, the Conversational Web Framework (ConWeb) that, thanks to the integration of Conversational AI technologies, is able to handle a conversational user experience for browsing the Web. The dialogues supported by ConWeb are based on a set of conversational patterns defined with and for blind and visually impaired users. Nevertheless, the framework aims to be accessible and useful to a wide range of people with varying requirements and situations. The goal of this work is to propose an approach that can benefit people universally, and has a potential that will impact Web Engineering in the coming years.

Keywords: Conversational AI, Web architectures, Voice user interfaces

Abstract in lingua italiana

Chiunque dovrebbe avere la possibilità di accedere al Web. Individui con disabilità visive, persone anziane, o utenti che in un determinato momento non sono in grado di usare i propri occhi o le mani, dovrebbero averne accesso. Oggigiorno, le tecnologie assistive per il Web, e in particolare gli screen readers, possono aiutare gli utenti ad accedere al Web in modo differente da quello visuale, e.g., con la voce e l'udito; tuttavia, risulta essere complicato anche per coloro, e.g., persone senza vista o con disabilità visive, che spendono anni a imparare come usare queste tecnologie impiegando numerosi sforzi. Inoltre, in aggiunta alla complessità che caratterizza questi strumenti, le linee guida di accessibilità non sono quasi mai soddisfatte dalla maggioranza dei siti Web, e questa mancanza causa numerosi problemi agli screen readers che devono interpretarne il contenuto. Agenti conversazionali basati sull'utilizzo della voce, e.g., Alexa, possono aiutare a ottenere informazioni dal Web rispondendo a semplici domande. Tuttavia, questi agenti non sono in grado di supportare la navigazione su questi stessi siti.

Lo scopo di questa tesi è fornire un nuovo paradigma di accesso al Web per navigarlo tramite comandi vocali. L'idea è di consentire di navigare il contenuto e i servizi presenti nel Web "parlando coi siti Web" invece che navigandoli visivamente; esprimendosi in linguaggio naturale e accedendo tramite un dialogo gestito da un Agente Conversazionale (e.g., plugin di un browser o un'interfaccia Web apposita). Questa idea è possibile grazie a ConWeb, un framework conversazionale in grado, grazie all'integrazione con Intelligenza Artificiale Conversazionale, di gestire un'esperienza di navigazione orientata a una conversazione con l'utente. I dialoghi supportati da ConWeb sono basati su linee guida identificate da persone non vedenti e con disabilità visive. Inoltre, il framework vuole essere accessibile e utile per numerose persone con diverse esigenze. Lo scopo del nostro lavoro è di proporre un approccio che possa aiutare tutte le persone a poter accedere al Web e abbia un potenziale per influenzare l'ingegneria del Web negli anni a venire.

Parole chiave: Conversational IA, architetture per il Web, interfacce vocali per gli utenti

Ringraziamenti

Desidero ringraziare la Professoressa Maristella Matera per la sua disponibilità e i suoi consigli durante lo sviluppo della tesi e per la sua grande passione nell'ambito universitario e di ricerca.

Ringrazio profondamente la mia famiglia e in particolare i miei genitori che mi sono stati vicini e mi hanno supportato in questo percorso universitario, sia moralmente che economicamente.

Ringrazio in particolare Antonella, che mi è sempre stata accanto, ascoltandomi e consigliandomi, soprattutto durante i momenti più difficili.

Ringrazio coloro che hanno partecipato e seguito questo grande progetto, con passione e perseveranza: Marcos, Emanuele, Cinzia, Antonella, Isabella e Claudia.

Ringrazio infine tutti gli amici, che hanno sempre creduto in me e hanno dimostrato il loro sostegno durante tutti i bei momenti passati insieme.

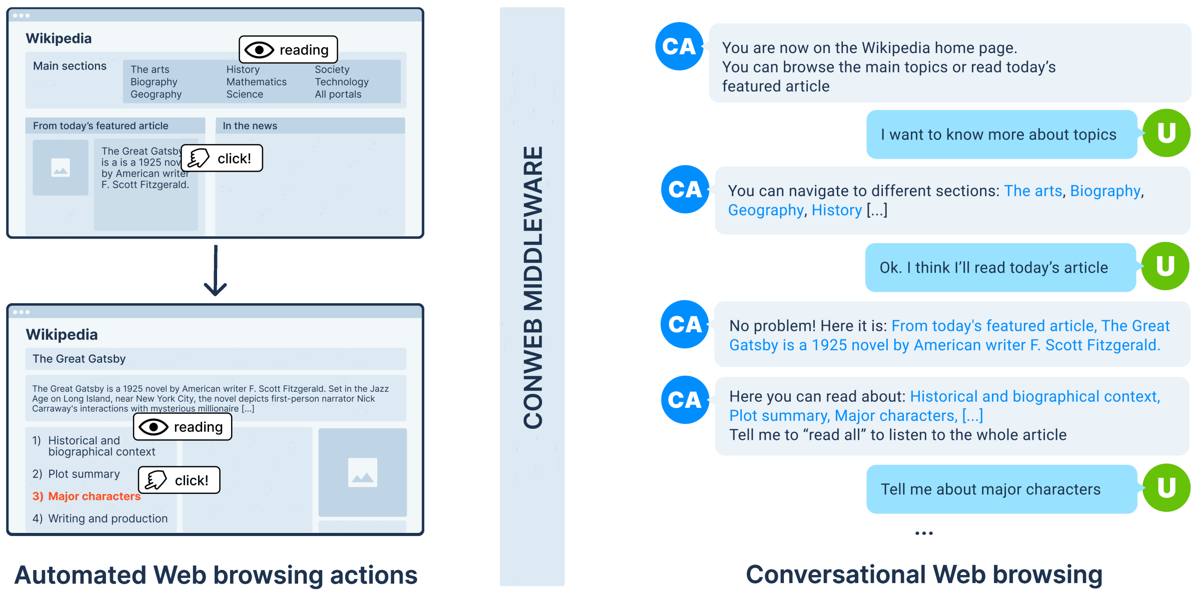
Contents

1 | Introduction

1.1. Context: Conversational Web Browsing

Web engineering is always evolving and expanding its content and functionalities. As a result, Web accessibility is a fundamental priority for each of us, and we must ensure that we all have this capability without significant differences in navigation experience. For this reason, people with visual impairments or those who are unable to access the Web using their hands or eyes must be granted alternative approaches yielding to the same result. Different users can take advantage from this extended capability: from blind people [2, 24], to individuals who have to look up information while driving [29] or cooking [33], or in general when user's hands and eyes are busy [31], to the elderly who can have difficulties while using an electronic device [25, 27].

Currently, to some extent assistive technologies can help accessing the Web. However, a severe issue is that the resulting user experience is the transposition of a linear visual scanning of Web pages, e.g., from the top to the bottom, into other fruition paradigms, for example based on the use of voice for conveying the Web page content. Sometimes this experience can be frustrating. Voice interfaces sometimes suffer from a lack of discoverability: unlike graphical User Interfaces, where the interaction can be easily grasped by recognizing visual cues, users frequently are unaware of what voice interfaces are capable of or what they can say [10, 36]. The adoption of Conversational voice-based Agents (CAs) can be a valid path to improve the accessibility of the Web. An example of CA for accessing Web content and provide simple voice queries is Firefox Voice [8], a Web automation tool able to serve Web content upon voice requests. By CA we mean a voice assistant who navigates the site on behalf of the final user. This thesis focuses on the design and development of a Web platform that enables the automatic generation and the execution of CAs, that can handle users' request expressed through voice commands. As a result, the thesis investigates aspects at the intersection of Web Engineering and Conversational AI, specifically *Conversational Web Browsing*, to identify a suitable solution to the previously highlighted challenges. While a vocal assistant may appear to be a viable solution, several and intricate concerns regarding how and why it must be built must be taken into account.


Wikipedia
Main sections
The arts
Biography
Geography
History
Mathematics
Science
Society
Technology
All portals
reading
From today's featured article
In the news
The Great Gatsby is a is a 1925 novel by American writer F. Scott Fitzgerald.
click!
Wikipedia
The Great Gatsby
reading
1) Historical and biographical context
2) Plot summary
click!
3) Major characters
4) Writing and production
Automated Web browsing actions
CONWEB MIDDLEWARE
CA
U
Conversational Web browsing

The Great Gatsby is a 1925 novel by American writer F. Scott Fitzgerald. Set in the Jazz Age on Long Island, near New York City, the novel depicts first-person narrator Nick Carraway's interactions with mysterious millionaire [...]

CA: You are now on the Wikipedia home page. You can browse the main topics or read today's featured article

U: I want to know more about topics

CA: You can navigate to different sections: The arts, Biography, Geography, History [...]

U: Ok. I think I'll read today's article

CA: No problem! Here it is: From today's featured article, The Great Gatsby is a 1925 novel by American writer F. Scott Fitzgerald.

CA: Here you can read about: Historical and biographical context, Plot summary, Major characters, [...] Tell me to "read all" to listen to the whole article

U: Tell me about major characters

...

Figure 1.1: Illustrative example of Conversational Web Browsing on Wikipedia

More specifically, a parallel study [5] on users' needs for Web accessibility was conducted concurrently with this thesis and suggests some design patterns for sustaining the notion of Conversational Web Browsing.

1.2. Scenario and Problem Statement

Let us consider a scenario in which a person with visual impairments wants to look for information on Wikipedia (see Figure 1.1). This would be possible thanks through current assistive technologies, such as screen readers, which is *a form of assistive technology that renders text and image content as speech or braille output* [35]. According to the literature [30], screen readers, for as widely used as they are, have several limitations and issues. For example, if blind users want to browse a long Wikipedia page, they would be unable to do that unless they do it in a convoluted and difficult manner. Our aim instead is to browse a Wikipedia page by dialoguing with a Conversational Agent (e.g., a smart speaker, a Web-client or a voice-based browser plugin). For example, starting from a home page the user can be introduced with a short description along with the main organisation of the website. The user could also at any point get oriented by inquiring about the content available in a given context, e.g., by uttering "What can I find on this page?". The user can then navigate the website by following up on one of the available options (e.g., "I want to navigate to [...]"). These requests can trigger navigation within or across pages in the website (e.g., from the Home to an article page). Ultimately, the user can browse the structure of the content or directly read the available content.

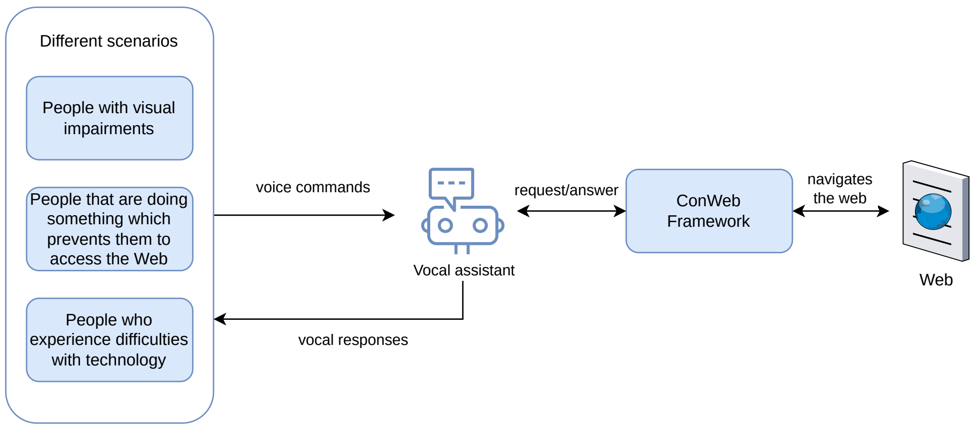

Different scenarios
People with visual impairments
People that are doing something which prevents them to access the Web
People who experience difficulties with technology
voice commands
vocal responses
Vocal assistant
request/answer
ConWeb Framework
navigates the web
Web

Figure 1.2: Vocal assistant to navigate the Web

As illustrated in Figure 1.2, to enable such interaction, a framework sitting between target users and the website must be able to identify the offerings and content of the website that can be accessed through the conversational medium, interpret user requests, and automatically perform related actions (e.g., click, extract information).

This paradigm is one of the few emerging approaches exploring the integration of conversational capabilities into the Web [8, 12, 26]. Previous work explored basic issues posed by a tight integration of Conversational AI with the Web, related to automating Web browsing actions to respond to natural language user commands [9], with a focus on technical feasibility. Our work tries to give a further contribution by discussing how to support more articulated design patterns for Conversational Web Browsing that are identified through an extensive user research [5]. Incorporating conversational patterns is fundamental to support recurrent sequences of human-bot interactions [21] serving expected browsing tasks. The reason why we want to help people with visual impairments and improve Web accessibility is to set bases for a new way of using and accessing information. Previous works [4] provide an interesting perspective and useful ideas on how to develop a voice assistant for Web access, following the definition of HTML annotations to retrieve Web content. As a result, the personal assistants that are the object of this thesis, aim to solve the mentioned issues and provide the groundwork for a new paradigm for accessing the Web based on voice commands. As a consequence, this thesis proposes the design of a platform that supports the access to the Web with voice commands that capitalize on a recently conducted user studies [5] in order to provide solid foundations for voice-based accessibility.

1.3. Methodology

The development of a voice assistant capable of solving the aforementioned issues is the result of numerous tasks and approaches. Our goal is to create a software platform that demonstrates the feasibility of using voice commands to access the Web. To reach this goal, we started from the results of an extended user study [5] that identified how to design conversations for Web browsing by means of several sessions with visually impaired users. The gained results are analysed and expanded upon in order to determine how to reconcile that work with the development of the software platform. In particular the design patterns identified through the user study are translated into architectural choices for the design of a Web platform focusing on the following aspects:

1. A conversational-browsing model must be built when the website is first accessed, to index and present to the users the available conversation nodes and the navigation structures that can sustain conversational browsing.

2. A conversation node does not necessarily correspond to an entire Web page; it can be a content paragraph, a navigation menu, a link, or any other element in the Web page that can be presented independently from the others and has a role in the progressive exploration of the website content.

3. A context representation characterising the navigation status must be handled to let the users to move easily backward, i.e., along previous conversation nodes, and forward, i.e., to identify and explore new reachable nodes.

4. To understand browsing-relevant requests from the user utterances, a Natural Language Processing (NLP) engine must be adequately trained starting from the website content.

5. Recognized requests must be matched with navigation and content reading actions as deriving from the conversational-browsing model.

6. The resulting Conversational Agent should be able to recognise website-specific requests as well as general scaffolding ones related to auxiliary commands enhancing the user control on the conversation.

Along with this effort, we examined past works [4] on how to browse the Web using a conversational assistant and extract information from Web pages by using specific annotations. These annotations are additional HTML attributes that identify the content of the Web page and allow the voice assistant (particularly the logic behind it) to interpret and render it vocally. These annotations allow for the creation of appropriate data structures that allow the framework to manage the navigation and the rendering of the information on the Web page through the voice-based paradigm. The data structure also guides the training of a Conversational AI model, which is fundamental to interpret the voice-based user requests and build related responses. In our work, we will draw inspiration from these modeling components, and will extend them in order to create new ones with extended functionality and stability.

Our platform must support browsing on Web pages, which implies requesting the HTML code from Web server and also enacting navigation actions. Therefore it utilizes a Headless Browser to retrieve and manage Web content: our framework must be indeed able to perform navigation without the need for the user to perform actions on the visual layout of pages.

Our vocal assistant must also be able to understand the user's intent (the action the user wishes to perform), which we will refer to as `intent` in this context, and words relevant to the context of navigation, which we refer to as `entities`. To understand the user's intent and distinguish it in the content of a Web page and extract relevant entities, we need to interpret the user utterance and understand it also by voice. It is important to note that our process for developing the ConWeb platform is based on the results of user researches and is finally validated by an additional study involving people with visual impairments; in this way, we aim to produce a viable solution that will have a real influence on Web access.

1.4. Contribution

Our work, and in particular the developed software framework, aims to be a foundation for Web accessibility through conversational assistants for everybody without distinctions. For this reason, we develop a software platform that aims to exhibit a new and different approach to access online information. We aim at demonstrate how interpreting voice utterances and subsequently act on Web pages is possible and could lead to a new navigation experience. To interpret user utterances we use Natural Language Understanding (NLU) APIs to demonstrate how it is possible to integrate Conversational AI in Web technologies to respond to user requests. This platform follows and brings together various studies from different fields and real-world cases in order to be developed following the mentioned guidelines and to be the start of a new way of accessing information. In this way we want to develop a platform with the goal of helping our target users in browsing Web through voice commands in an innovative way.

1.5. Structure of the thesis

The thesis is organized as follows:

- Chapter 2: the state of the Art is provided, evaluating different methods for Web accessibility and the starting points of our work.
- Chapter 3: a more in-depth examination of the difficulties stated, as well as the relevant aspects of our solution and their requirements are provided. Furthermore a detailed view of the architecture of our software solution is presented to give a general view of all of its functionalities and aspects.
- Chapter 4: an explanation of the Conversation-oriented Navigation Tree, one of the most important aspects of our framework, is provided.
- Chapter 5: an in-depth examination of the adoption of the NLU engine and its integration in the framework is provided.
- Chapter 6: an explanation of how the software architecture and its components are designed in accordance with the results of the guidelines and patterns derived from a user-based research [5] is provided. In particular, a description of how our framework may take appropriate actions based on the user's request is provided, with a focus on the Intent Handlers (components responsible for handling user requests) of our architecture.
- Chapter 7: an in-depth description of the implementation, including the adopted technologies, is reported, along with a full explanation of the different components. An evaluation conducted through a users study with a sample of users with vision impairments is presented and described.
- Chapter 8: this chapter concludes our work and outlines possible future works.

2 | State of the Art

Web content is designed to be accessed visually and interacted by keyboard, mouse and touch, rather than conversed with. A survey on users highlights how browsing the Web with accessibility tools such as screen readers can be difficult and frustrating and that Web pages often contain inaccessible content that is expressed only visually or that can be accessed only with the mouse [7]. In addition to this, a recent study cites how the Web content is designed to be seen, not heard [37]. Despite these facts, it is certainly possible to access the Web in ways different than using a mouse or a keyboard, each with its own set of strengths and weaknesses. In this chapter, after an introduction on Web accessibility, various methods of accessing the Web are introduced, described, and criticized in order to focus on their characteristics, as the basis to fulfill our goal of solving existing problems. Their description aims at showing the main existing problems regarding accessibility and usability on the Web. Following that, the current state of the Art regarding primary technologies employed to overcome the existing problems is outlined. A summary of this content is provided later in this chapter to have a thorough view of the current state of the Art in Web accessibility and how we intend to develop an alternative method of accessing information following and extending the presented technologies.

2.1. Web accessibility

The Web is a valuable information resource that practically everyone uses on a regular basis. However, different people have different ways of accessing the Web, which can depend on the device they are using, their ability to use it, or any disabilities they may have. For this final problem in particular, new methods of accessing the Web has been developed to overcome these difficulties, in other words, make the Web accessible (accessibility). The World Wide Web Consortium (W3C) in particular began projects for the Web encouraging a high level of usability for people with disabilities.

W3C defines accessibility as the design and development of websites, tools, and technology so that people with disabilities can use them. In particular, people should be able to:

- perceive, understand, navigate, and interact with the Web
- contribute to the Web

In addition, Web accessibility encompasses all disabilities that affect access to the Web, including: auditory, cognitive, neurological, physical, speech and visual. The W3C Web Accessibility Initiative (WAI) is a solution who develops technical specifications, guidelines, techniques, and supporting resources that describe accessibility solutions. In this way, WAI aims to standardize Web development under guidelines that will allow websites to be accessible. However, the reality is that Web developers are either uninformed of accessibility guidelines or prefer to ignore them; as a result, Web accessibility may not always fulfill its aims, and individuals may encounter a variety of difficulties while browsing. Furthermore, the reality is that 98% of websites do not support full accessibility as explained in a study [13]. In addition to this, a recent study [34] shows how 68% of users who require accessibility and find it difficult or not supported will likely leave the website they are navigating. For the arguments stated thus far, we may conclude that accessibility is a complex problem that must be handled, especially given the large number of websites that can be browsed.

2.1.1. Voice-based Assistive Technologies

There exist different methods of accessing Web information in addition to those that rely on physical devices with hands and eyes. With our research, we specifically focus our study on employing a new method of Web access while keeping accessibility in mind and on behalf of our target users.

Screen Readers

One method for accessing the Web, but especially for using a computer for people with visual impairments, is to use a screen reader. This tool allows users to comprehend the text on their computer screen and communicate vocally or with a braille device [35]. People with visual impairments frequently use screen readers and are used to their capabilities. For as much as they are accustomed to using them, they lament the numerous disadvantages that screen readers possess and are open to new technologies (especially the youngest ones). In particular, screen readers, while capable of reading text on a screen, are incapable of filtering content in a practical manner, managing navigation, or enabling the quick deduction that can be derived when visually inspecting a Web page (even involuntarily). As a result, although being a viable solution, screen readers are now encountering a slew of issues, especially when it comes to websites with a lot of content on a single page; instead we aim at demonstrating how our framework could solve these kind of issues. Our effort is to provide people with an alternative method of accessing the Web, which could aid people with visual impairments (in general our target users) and substitute almost entirely screen readers in retrieving Web content.

2.1.2. Voice assistants

Even though the Web can be accessed using properly configured accessibility tools, it is also possible to access it using alternative devices that individuals with disabilities may utilize. Voice assistants, such as Google Assistant, Amazon Alexa or Apple's Siri, have achieved significant popularity over the last decade, with more than half of adults in the United States stating that they have used a voice assistant, as cited in a recent report [22]. Two exploratory studies show how voice assistants are used by people with disabilities (especially people with visual impairments) including for unexpected cases such as speech therapy and support for caregivers [24]. Simultaneously, another study demonstrates how people with disabilities utilize voice assistants, but they face several problems when doing so [1]; by misspellings words or misunderstandings. In addition to this, the use of voice assistants follows predictable patterns; according to multiple surveys [3, 28], browsing the Web for content such as music and recipes, as well as conducting simple informational questions, are among the most common use-cases for voice assistants. Furthermore, we must state that estimates [14–16] imply that a third of all Web searches are now initiated by a voice inquiry rather than a typed query, showing a significant use of voice in technologies. On the other hand, while voice assistants can get information from the Web, they can only do so with simple searches and do not enable pages navigation. Moreover, the use of voice assistants has a significant impact on how people perceive them, particularly the notion that the assistant is always listening is an important aspect cited in recent studies which could change the way people use them [11, 17]. For the reasons stated thus far, voice assistants, while useful, are normally used for quickly and simple tasks or as complementary assistants to screen readers but do not support navigation through Web pages; for this aspect, we want to exploit a new way of accessing the Web and information in order to allow it to be accessible only through voice but with all the functionalities as a user may use a mouse and a keyboard and with a fluent flow of information and control.

2.2. Non-visual browsing

We have identified what accessibility is and its major issues together with different method for accessing the Web; however, the way people browse the Web through voice introduces the literature to the non-visual browsing concept. Our target users while browsing the Web with voice commands are blind or partially sighted people, elderly people, and people whose hands or eyes are busy at certain times. As a result, it is critical not only to allow these target users to acquire access to the Web as they are using a mouse and keyboard, but also to allow them to explore it in a non-visual manner; in other words, they must be able to understand, through mental visualization, the structure of Web pages. As a result, the concept of non-visual browsing evolved as a method of browsing that does not rely on its graphical interface (GUI). Even with the various tools mentioned in this chapter, such as screen readers or voice assistants, establishing a mental view of a Web page to assist the user is a further process that must be examined. A recent study on exploring challenges to visually impaired people aims to conduct this research so that we could create an architecture that meets their guidelines [6]. We must increase usability while browsing the Web in order to improve the user experience for visually impaired individuals on the Web. The literature concentrated on two primary concepts, in particular:

- `Segmentation`: which refers to a body of work that focuses on extrapolating Web entities from online pages. The CSurf browser [20], for example, can extrapolate the context from a page that is available via a link, allowing it to extrapolate content from the Web and reveal it to the final user, who may then select whether or not to follow the proposed link.

- `Skimming and summarization`: these methods seek to expose the content and extract the most important information from it. A long text may be difficult to understand in terms of meaning for a visually impaired individual if not read in its entirety. As a result, skimming and summarization seek to highlight the most significant aspects of a complex content. The AcceSS [23] system, for example, employs a transcoding process that includes summarizing and simplification processes to make Web pages more usable for non-visual browsing.

We are going beyond typical accessibility by revealing the content and extrapolating it in a way that traditional accessibility tools such as screen readers cannot. However, for our needs, this is insufficient since we must go a step further in order to provide effective navigation within a Web page.

2.3. Web exploration tools and NLU technologies

Accessing the Web and gathering various types of information is a wide-ranging process. To comprehend how to retrieve information from online pages and navigate or perform actions, various elements must be considered and assessed. The literature has progressed further in offering Web content, for example via voice commands adding more accessibility to it. To attain this result, developed tools must be able to automate online actions and supply content in response to voice requests. As a result, we must distinguish between two major technologies used in our project:

- Web automation tools
- Natural language understanding (NLU)

2.3.1. Web automation tools

Web automation tools aim to obtain information from the Web via voice commands. They attempt to expose the content of websites by posing questions and answering them in natural language. The information from Web pages could be seen as content upon request.

Firefox Voice

In this sense, Firefox Voice [8] is an example of such a system, which aims to provide end-users with a browser extension that allows them to access Web content and traditional voice assistant features (setting alarms, timers, asking the weather, and so on) without relying on any specialized machine learning technique for natural language processing or information extraction, but rather by leveraging various APIs supported by the Mozilla Community. Even though retired, Firefox Voice does not support navigation as intended if we use a mouse and keyboard, instead wishes to explore a website with simple and fundamental commands supporting different Web element as services available through voice requests.

CoScriper

CoScripter [19] is a collaborative scripting environment for recording, automating, and sharing Web-based processes. In this sense could be seen as a base for Web automation tool to perform actions on the Web. In particular, CoScripter enables the creation of macros that can be reused to accomplish Web automation processes or activities.

2.3.2. Natural Language Understanding - NLU

Web accessibility is important, and different approaches could produce varied results. Conversely, non-visual browsing must be considered in order for the final user to understand the structure of the page. Furthermore, the employment of Web automation tools as Conversational Agents (or simply by parsing natural language exposed utterances) may result in the usage of natural language by voice to specify the user intent. CoCo [18] is an examples of a natural language system that allows users to demonstrate series of Web-based operations and then invoke them using written natural language commands. In addition, the usage of conversational technologies could be exploited to extract relevant information from a user utterance conveyed in natural language using natural language processing (NLU). It is important to note that Conversational Agents may suffer from misunderstandings on speech recognition, resulting in incorrect natural language understanding as explained in a recent study [32]; in particular, different domain-specific terminology could lead to difficulties in surfacing content whose titles include non-standard spellings, symbols or other ASCII characters in place of English letters, or are written using a non-standard dialect. Therefore, this issue must be considered during the development of a voice assistant in order to accomplish proper utilization.

Rasa - NLU

An example of NLU is Rasa Open Source, which provides open source natural language processing (NLP) to turn messages from users into intents and entities. It is based on lower-level machine learning libraries like Tensorflow and spaCy, and provides natural language processing software that is approachable and as customizable as proper needs. Natural language processing is a category of machine learning that analyzes freeform text and turns it into structured data. Natural language understanding (NLU) is a subset of NLP that classifies the intent, or meaning, of text based on the context and content of the message. This is the concept we intend to apply in order to comprehend users intent and what they expect from an assistance in response to their requests.

Combining Web automation tools and Rasa NLU we want to develop a framework platform in order to comprehend user intent and perform automated actions on the Web providing content to the final user.

2.4. Summary

Our analysis leads us to a number of essential concepts that will be useful in our development and architectural design. We could specifically express the following:

- Web accessibility is an important factor to consider when providing information from the Web, given its existing problems and its goals of making the Web accessible.
- For what regards accessibility tools, screen readers are extensively used and a significant method of accessing websites; nevertheless, they lack browsing functionality and many websites do not support them (on in general, do not support accessibility).
- Voice assistants are a legitimate alternative method for quickly and simply querying on instant information or features (such as alarms, timers and weather queries), but not for navigating Web pages.
- Non-visual browsing must be considered in order for users to be able to develop a mental view of the website and not just supply them information on requests without the approach of navigation.
- Web automation tools are on the right track to discover a new method of accessing information (for example, Firefox Voice with different features). However, in order to address the needs of end-users, a more in-depth research on user experience is required to fully support browsing Web pages.
- Natural language understanding (NLU) is an important aspect of understanding voice-based commands, and it could be implemented with conversational technologies, as is the case with the majority of voice assistants, or without a machine learning approach (like Firefox Voice), but with a matching between the user's utterance and the given planned actions.

Following these aspects, we aim to provide our target users with a novel way of accessing websites information via a Conversational Agent built on a framework that leverages websites content and fulfills user requests in order to simulate mouse and keyboard browsing at the same time as building a proper mental view of websites structures. Starting with the challenges of accessibility and analyzing the existing issues, we intend to solve them, following the non-visual browsing concept, by providing a framework that can conduct actions on websites (such as Web automation tools) in response to user requests exposed by voice and interpreted by a NLU (in our case Rasa).

3 | Conversational Web Browsing: Approach

This chapter introduces the general concepts of ConWeb, as well as its architectural goals and requirements. In addition, we explain the approaches of our solution using a high-level architecture overview and an explanation of the main aspects.

3.1. Concepts

In this section, we will introduce terms and concepts that will help comprehend the overall perspective of ConWeb. In particular, the framework that serves the vocal assistant must recognize and manage the user's input; its understanding will be referred to as the `intent`. For example, if users ask for "Tell me about Mars", the intent, which can be referred to the words "Tell me", is to get them to a specific section of the website concerning a specific content (in this case "Mars", identified by a word, i.e. `entity`, "Mars"); this implies an automatic navigation to that section and the reading of the required content. If, on the other hand, users ask for "Help", the intent is that they require assistance. The framework includes all of the logic to generate and manage the dialog system, which strongly characterizes the assistant; hence, by "framework", "assistant", "Conversational Agent" or "ConWeb" we mean this logic and all of its properties.

The framework must extract all information from the page and then carry out the appropriate action based on the user intent. In order to analyze users phrases to determine their intents, the variety of possible requests and Web pages contents must be carefully examined in order to make the proper decision based on users needs. The fundamental framework characteristics supporting these analysis are presented in this following in the form of goals and requirements.

3.2. Goals and Requirements

The main goal is to help target users (people with visual impairments, old people or people who are unable to use a device with their eyes or hands at a specific time) to access information on a website and being able to browse it using a voice-based assistant; the framework supports the automatic generation of the assistant to provide an alternative way of accessing the Web.

The following list highlights the requirements we took into account when designing the framework.

Orientation - generic commands and functionality that every assistant must expose to facilitate orientation during conversation.

- The assistant must be able to explain to users the current behaviour of the navigation through voice.
- Users must be able to interrupt, ask and perform actions when they need them without waiting to the assistant to finish talking.
- Users must be able to use the assistant with reasonable waiting times between requests.
- The framework must expose a client to be used by final users whose requirements follow the ones for the ConWeb architecture.

Navigation - offering and interpreting commands through which the user can navigate a website:

- Users must be able to open a specific website using a voice command.
- The vocal assistant must be able to reveal the page's content in a logical and understandable hierarchy to the final users.
- Users must be able to request access to a specific content on the page simply by asking it.
- Users must be able to differentiate the content of a page using the information exposed by the assistant and therefore be able to request the preferred one by asking it with proper words following their natural reasoning.

Reading - offering and interpreting commands for presenting the page content to the user:

- Users must be able to understand how the textual content is structured in a certain section of the page.

- Users must be able to ask the assistant to read a certain text and receive an appropriate response.
- Users must be able to request the assistant to read various types of paragraphs and go through them.
- Users must be guided to the reading of paragraphs by the assistant in order to understand them without many difficulties.
- Users must be able to request if links are present in the text just read by the assistant and go through them.
- Users must be able to request how many paragraphs are present in the current section.

Linking - offering and interpreting commands for managing the user vocal selections or actions to navigate through another page:

- Users must be able to locate the links in a Web page and select them.
- The framework must be able to simulate users' click on a specific link and forward them to a new Web page.
- Users must be able to understand when a new Web page is loaded and therefore the navigation has changed.

Primary navigation commands - responding to the user requests for actions that are commonly performed in Web browsers (go backwards, return to the main page or request help):

- If users have not understood what has been spoken previously, they must be able to ask for it to be repeated.
- Users must be able to request a return to the previous navigation content depending on the current context.
- In the event of help is required, users must be able to request it.
- Users must be able to request to return to the beginning of the navigation.
- Users must be able to request the content of a Web page if they feel lost.
- Users must be able to ask where they are inside a page.
- The framework must be able to understand positive (yes) requests by users after having questioned them.

- The framework must be able to understand negative (no) requests by users after having questioned them.

The following list, on the other hand, shows the framework's non-functional requirements.

Non-functional requirements:

- Performance: the assistant must be able to respond to user requests in a reasonable amount of time. If users want to load a new page the assistant must inform them on the loading of the new website to prevent misunderstandings. In addition to this, the assistant and the framework behind it must exploit all the necessary features to achieve reasonable performance metrics as the final result (a usable assistant) is thought to give to the user a different way of accessing the Web and being accessible must also be usable in terms of performance.
- Scalability: the framework behind the assistant must be able to manage different instances of navigation in order to be used by multiple users.
- Availability: the framework behind the assistant must be developed and eventually deployed in order to be always available by users, maintaining their navigation sessions independently on their behaviours.
- Reliability: the framework behind the assistant and in particular the components responsible of parsing user utterance must understand it with a certain confidence level in order to avoid errors that could compromise proper usage.
- Maintainability: the framework must be designed with maintainability in mind, so that future approaches and extensions can be applied easily and with a wide range of functionalities without compromising its main features. In addition to this, the framework must be designed in such a way that it can be easily maintained and controlled while also understanding its own behavior.
- Usability: the assistant must be designed and developed in order to be properly used by the final user, respecting specific time frames to avoid long waiting times. It also must expose the main features and limits in order to let the user understand how to use it and what expect from it. At the same time the client configuration and usability must be properly developed in order to let final users to use it independently from the device or place where they are using it.

The previously specified requirements are intended to guide the development and acquire correct outcomes in order to construct a suitable platform. Based on previous works [4], our goal is to create a new and fully functional platform with new features and outcomes based on users research [5].

3.3. Design Decisions

In this section, we will explain the main aspects and approaches we use to achieve our goals. Despite their number, the developed voice assistant and framework include components and phases that will be examined sequentially. For a complete framework capable of demonstrating our goal, the following design decisions are required:

- Conversation-oriented Navigation Tree
- Intent and Entities Recognition
- Intent Handlers

3.3.1. Conversation-oriented Navigation Tree - CNT

Our framework must handle user utterances and carry out actions on the Web page (retrieving and exposing content, navigating through Web structure, and so on). To perform the aforementioned actions, we require an auxiliary structure that represents the content of a Web page (properly cleaned with only the content required for our purposes): the Conversation-oriented Navigation Tree (CNT). Our CNT is a redesigned tree of the Web page content made of multiple nodes representing different information and from which we could handle navigation between sections of the website as well as content retrieval. The CNT is a fundamental structure that is unique to each URL and is defined in detail in Chapter 4.

3.3.2. Intent and Entity Recognition

With `Intent and Entities Recognition`, we mean the ability to deduce the user intent from the given utterance and extract relevant entities (i.e. words relevant to the current context). To accomplish this goal, we must use a NLU (Natural Language Understanding) as support and which could distinguish from a list of intents that the user may use, the one of our interest through Conversational AI and subsequently extract relevant entities. The way we use NLU to extract user intent has many advantages, particularly the fact that the classification distinguishes with high confidence differences, the supported intents; however, there is a trade-off between correctly annotating the page and the number of intents the user could use in our framework. Chapter 5 goes into detail about this trade-off; how NLU works and how it is used in our work.

3.3.3. Intent Handlers

After determining the users intent, it is critical to take the proper action, such as updating the users navigation context and responding to help them comprehend the current situation. To achieve this goal, various bots (`Intent Handlers`) are built, each of which is in charge of a distinct type of action or CNT node to manage. A full explanation of all the bots and their scope could be found in Chapter 6. We choose to construct several bots in order to differentiate all of the users requests and manage them properly. We follow different Interaction-design patterns Integration coming from a specific user study [6] in order to achieve our goals on supporting possible user requests. The results of a rigorous user study include the many ways in which components are produced and why specific intents are chosen. A full explanation of how we include these behaviours could be found in Section 6.3. Additionally, there are bots in charge of different types of content on the page, and further ones are supported to enable scalability for future upgrades and extensions.

Our solution also includes a properly developed client that can be used and integrated with the framework application logic. To begin building this platform and making it accessible from everywhere, we opt to develop the client side on a Web server that could be accessed from various devices using a specific URL.

3.4. High-level architecture

The architecture of ConWeb is explained and demonstrated in this section. Because there are numerous components that interact with others, we provide a basic overview to comprehend a high-level abstraction of its internal architecture and the main pipeline the user utterance follows; a detailed explanation of all of its components could be found in Chapter 7. Figure 3.1 illustrates a high-level overview of the framework. We are able to identify one actor and two major components. The end-user is the actor, and the two primary components are the ConWeb Client and the ConWeb Server, which are discussed in the following sections.

3.4.1. ConWeb Client

The final user may use the ConWeb Client to communicate with the framework using only its voice; in practice, the client understands the user utterance and sends it to the ConWeb Server, which contains all of the application logic; it then waits for a response. To interface with the server and control all user configurations (for example which user is

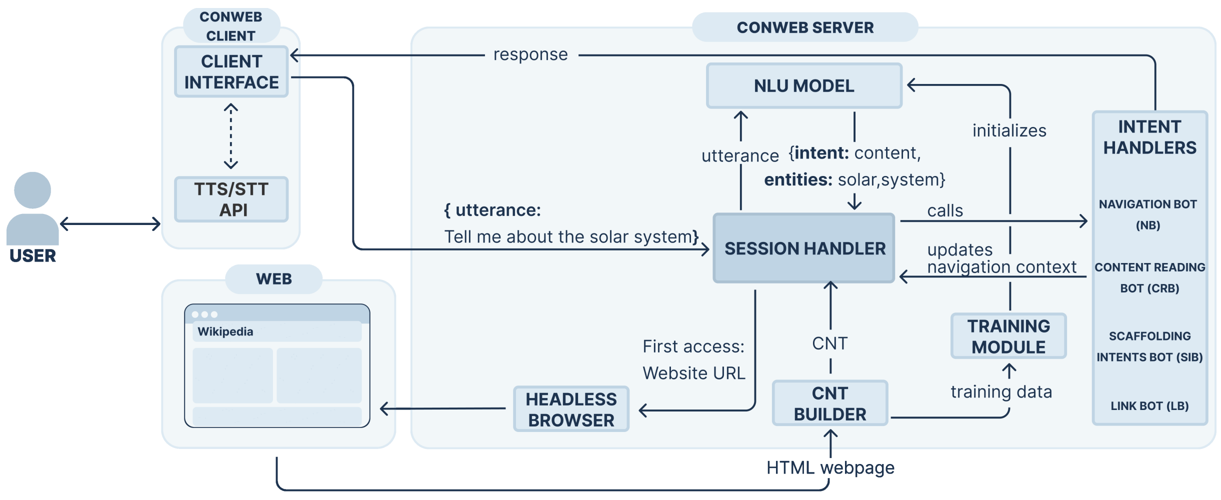


Figure 3.1: Framework architecture

using the application), the ConWeb Client has its own logic behind its interface. A simple front-end interaction directly with Speech Recognition/Synthesis APIs is used to permit vocal interface. In other words, when a user speaks to the client, its recorded voice is translated into a string using Speech to Text APIs and then transmitted to the ConWeb Server. The response from this last is received as a string, which is then translated into speech using Text to Speech APIs and spoken to the user. Finally, the ConWeb Client is deployed such that it can be accessed by a specific URL address. In this way, the final user could connect to it via a simple Web browser (for example Firefox or Chrome) and then the ConWeb Client could enable the user to communicate with it using simple voice commands.

3.4.2. ConWeb Server

The ConWeb Server is able to communicate with the ConWeb Client through a socket. When a specific user utterance is received, the Session Handler component is in charge of managing it. It specifically keeps track of all the different users sessions (such as navigation context, current page, and so on), therefore it gathers all the essential information to update them and perform the appropriate actions. A proper decision on a website is taken every time a user utterance is delivered to the ConWeb Server, however it is crucial to clarify what happens the first time a URL is opened by a user request (the user performs a first access to a specific URL). If the user utterance is to open a specific page (URL), the new URL is loaded and the navigation context is updated with the new page content. Through a Headless Browser and specific Web APIs the URL is loaded and a new CNT is created from the HTML annotations on the page. Web page data inferred from the

CNT is used to train an NLU model. Thus, the NLU model will be able to understand the user's utterance on the given page and provide information that will be used by the Intent Handlers to perform the required actions. The navigation context is then updated and the control is returned to the user along with an introduction to the new website (it is crucial to note that after opening a new URL, the framework implicitly assumes that the user request is an introduction to the new page content).

On the other hand, if the users utterance is a specific request inside a Web page they are browsing, it is forwarded to a NLU model trained on the specific visited URL. In Chapter 5, we go over how the NLU model parses the user utterance. After parsing the utterance, the framework is able to comprehend the user intent (extracting also relevant entities) and thus execute the proper action; specifically, the selection of the suitable action based on the navigation context is performed and the information obtained from the NLU model (intent and entities, i.e. substantives) are essential for this process. The appropriate Intent Handler is called to conduct the appropriate action; we can distinguish between the followings:

- Navigation Bot (NB): it finds the content the users are searching in the page and bring them to it (to the node corresponding to the content).
- Scaffolding Intents Bot (SIB): it performs the appropriate action from the ones that should be always present during navigation and that have been taken from the users study [5] (for example go back, repeat, help and so on).
- Link Bot (LB): it is able to open a link and update the navigation context.
- Content Reading Bot (CRB): it is able to read text content and expose it to the final user supporting different intents (read, select paragraph, go on and so on).

After the bots have taken the proper action, the control is returned to the user. It is important to note that the Navigation Bot and the Scaffolding Intents Bot are fundamental components of the framework that represent its main functionalities, whereas the Link Bot and the Content Reading Bot, while important components, are specific to the type of element found on the page, and thus other components similar to them (treating a different type of content) could be added to improve the assistant features in the future.

We presented the high-level pipeline of ConWeb from the point of view of the user utterance. Starting from the ConWeb Client, it is forwarded and analyzed thanks to Intent and Entities Recognition and subsequently appropriate actions are performed by Intent Handlers, supporting differnce features with Interaction-design patterns. Later, an answer is returned to the ConWeb Client and exposed to the final user.

4 | Conversation-oriented Navigation Tree - CNT

As previously stated in the preceding paragraphs, we must interact with Web pages in order to have a vocal assistant which navigates the Web in our place (based on our voice commands). When we navigate Web pages with a mouse and a keyboard (the same concept applies if we use a touch screen with a mobile device), we can understand and recognize their inner content, but this is not the case when we use a vocal assistant. As a result, we require an auxiliary structure that represents the Web page we are visiting and can be used by the framework to perform the necessary actions to support our requests. Drawing inspiration from previous works [4], we decide to use a tree structure in a conversational context to represent our navigation, as explained in the following sections.

The Conversation-oriented Navigation Tree (CNT) is a complex structure that will be explained further in the following sections. Consequently, an example of how to build a CNT on a Web page is shown.

4.1. CNT - structure

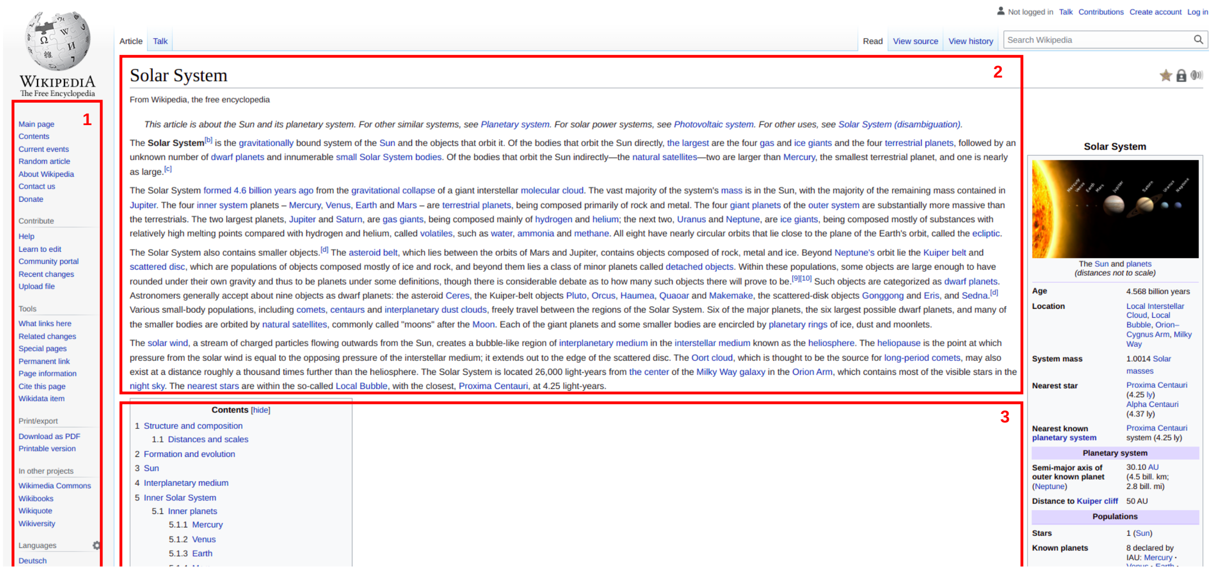
A CNT is a structure that represents the page's content. It is created by the framework using all of the necessary information to maintain the meaning and hierarchy of the content. In fact, the CNT's structure is made up of various nodes, each with a distinct content. The main feature of the CNT is that each node represents its own content, thus, the page follows a hierarchy between parent and child nodes; in other words, the page is divided into different sections, each of them is represented by a node. Because a node may contains additional contents, other nodes representing these contents are created and linked as children of the previously mentioned outer node which will be the parent. When a node representing its final content is created, it has no children and represents the actual content that it is describing. In fact, if a node has children, it only represents a concept useful for understanding what content their children manage and it lacks any real content to be used. Only leaf nodes contain the page's actual content (that could be a link, a text, an image and so on). To summarize what has been said thus far, nodes can be classified into three types:

- Root node: the node representing the whole page.
- Parent nodes: nodes that have children and therefore represent the information of these ones without having any real content.
- Leaf nodes: nodes that do not have any children and represent the final content of a part of the page (text, image, link and so on).

The CNT structure can be used to represent navigation. Users, with the support of the vocal assistant, can navigate through the page by simply going from node to node, and when they come across a leaf node, they could ask the assistant for information on it (if it is a text to read it, a link to click it and so on). It is important to note that the CNT structure is created by the framework following various annotations that are present on the Web page and are taken from a previous work [4] and redesigned to meet our requirements, as explained better in Section 4.2.

4.1.1. Extraction of Web Page Content

In this section, we exemplify how the content of a Web page can be divided to create a CNT. Figure 4.1 shows a page from Wikipedia about the solar system. The mentioned page is particular since it contains a lot of content, so it should be divided into many nodes with a proper hierarchy to help the user understand it just by hearing the vocal assistant. In the current example, we could divide the page into three main nodes: 1, 2, and 3, as shown in Figure 4.1. Node 1 represents all of the important links on the page's side (it will have other children nodes, each one representing a single link or a group of them), node 2 represents the page's introduction (it will be a leaf node because it does not contain any additional content other than an introduction text), and node 3 represents the rest of the page's content with all of its paragraphs (in this case it will be a parent node as the content is extensive). The aforementioned example is a design choice, and its purpose is to help the reader understand how the CNT can be created, however different decisions can change its structure. Node 3 is, of course, the most complex of the page because it contains a large number of paragraphs and texts; therefore, to better explain it, we will only refer to two of its children as shown in Figure 4.2. As we can see from the example, node 3 has all of its children, as well as nodes 3.4 and 3.5, which are discussing the "Interplanetary medium" and the "Inner Solar System" respectively. As we can see, node 3.4 has no further distinction in content and thus could be classified as a leaf node, whereas node 3.5 has a brief introduction and a subparagraph about "Inner planets". As



Solar System

From Wikipedia, the free encyclopedia

This article is about the Sun and its planetary system. For other similar systems, see Planetary system. For solar power systems, see Photovoltaic system. For other uses, see Solar System (disambiguation).

The **Solar System**[b] is the gravitationally bound system of the Sun and the objects that orbit it. Of the bodies that orbit the Sun directly, the largest are the four gas and ice giants and the four terrestrial planets, followed by an unknown number of dwarf planets and innumerable small Solar System bodies. Of the bodies that orbit the Sun indirectly—the natural satellites—two are larger than Mercury, the smallest terrestrial planet, and one is nearly as large.[c]

The Solar System formed 4.6 billion years ago from the gravitational collapse of a giant interstellar molecular cloud. The vast majority of the system's mass is in the Sun, with the majority of the remaining mass contained in Jupiter. The four inner system planets – Mercury, Venus, Earth and Mars – are terrestrial planets, being composed primarily of rock and metal. The four giant planets of the outer system are substantially more massive than the terrestrials. The two largest planets, Jupiter and Saturn, are gas giants, being composed mainly of hydrogen and helium; the next two, Uranus and Neptune, are ice giants, being composed mostly of substances with relatively high melting points compared with hydrogen and helium, called volatiles, such as water, ammonia and methane. All eight have nearly circular orbits that lie close to the plane of the Earth's orbit, called the ecliptic.

The Solar System also contains smaller objects.[d] The asteroid belt, which lies between the orbits of Mars and Jupiter, contains objects composed of rock, metal and ice. Beyond Neptune's orbit lie the Kuiper belt and scattered disc, which are populations of objects composed mostly of ice and rock, and beyond them lies a class of minor planets called detached objects. Within these populations, some objects are large enough to have rounded under their own gravity and thus to be planets under some definitions, though there is considerable debate as to how many such objects there will prove to be.[9][10] Such objects are categorized as dwarf planets. Astronomers generally accept about nine objects as dwarf planets: the asteroid Ceres, the Kuiper-belt objects Pluto, Orcus, Haumea, Quaoar and Makemake, the scattered-disk objects Gonggong and Eris, and Sedna.[d] Various small-body populations, including comets, centaurs and interplanetary dust clouds, freely travel between the regions of the Solar System. Six of the major planets, the six largest possible dwarf planets, and many of the smaller bodies are orbited by natural satellites, commonly called "moons" after the Moon. Each of the giant planets and some smaller bodies are encircled by planetary rings of ice, dust and moonlets.

The solar wind, a stream of charged particles flowing outwards from the Sun, creates a bubble-like region of interplanetary medium in the interstellar medium known as the heliosphere. The heliopause is the point at which pressure from the solar wind is equal to the opposing pressure of the interstellar medium; it extends out to the edge of the scattered disc. The Oort cloud, which is thought to be the source for long-period comets, may also exist at a distance roughly a thousand times further than the heliosphere. The Solar System is located 26,000 light-years from the center of the Milky Way galaxy in the Orion Arm, which contains most of the visible stars in the night sky. The nearest stars are within the so-called Local Bubble, with the closest, Proxima Centauri, at 4.25 light-years.

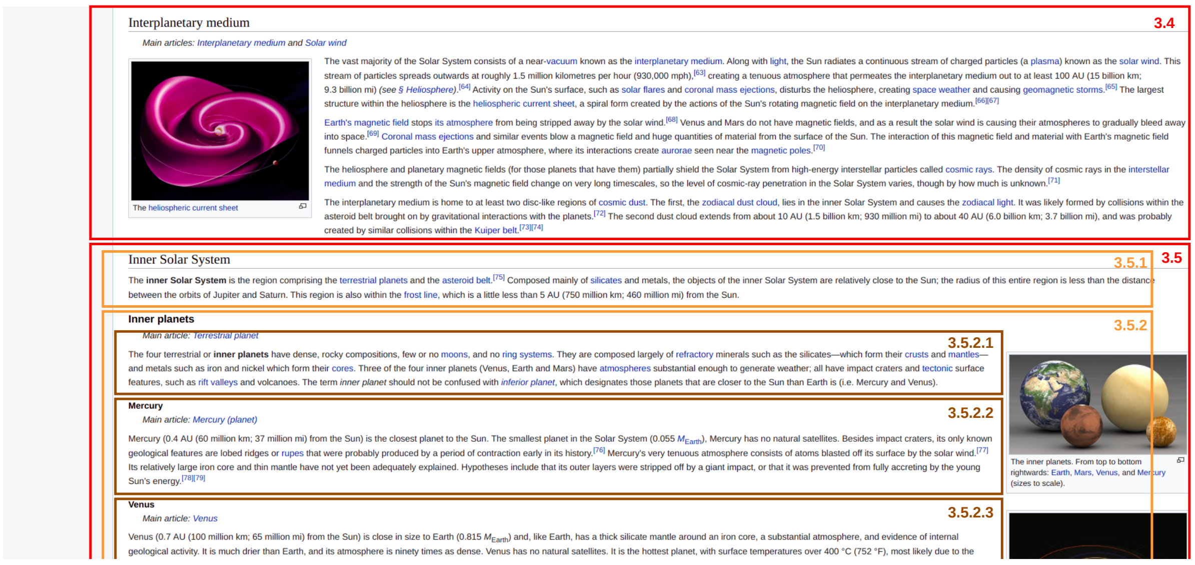
Contents [hide]

Solar System

The Sun and planets
(distances not to scale)

Age	4.568 billion years
Location	Local Interstellar Cloud, Local Bubble, Orion–Cygnus Arm, Milky Way
System mass	1.0014 Solar masses
Nearest star	Proxima Centauri (4.25 ly) Alpha Centauri (4.37 ly)
Nearest known planetary system	Proxima Centauri system (4.25 ly)
Planetary system	
Semi-major axis of outer known planet (Neptune)	30.10 AU (4.5 bill. km; 2.8 bill. mi)
Distance to Kuiper cliff	50 AU
Populations	
Stars	1 (Sun)
Known planets	8 declared by IAU: Mercury ·

Figure 4.1: Wikipedia solar system example



Interplanetary medium

Main articles: Interplanetary medium and Solar wind

The heliospheric current sheet

The vast majority of the Solar System consists of a near-vacuum known as the interplanetary medium. Along with light, the Sun radiates a continuous stream of charged particles (a plasma) known as the solar wind. This stream of particles spreads outwards at roughly 1.5 million kilometres per hour (930,000 mph),[63] creating a tenuous atmosphere that permeates the interplanetary medium out to at least 100 AU (15 billion km; 9.3 billion mi) *(see § Heliosphere)*.[64] Activity on the Sun's surface, such as solar flares and coronal mass ejections, disturbs the heliosphere, creating space weather and causing geomagnetic storms.[65] The largest structure within the heliosphere is the heliospheric current sheet, a spiral form created by the actions of the Sun's rotating magnetic field on the interplanetary medium.[66][67]

Earth's magnetic field stops its atmosphere from being stripped away by the solar wind.[68] Venus and Mars do not have magnetic fields, and as a result the solar wind is causing their atmospheres to gradually bleed away into space.[69] Coronal mass ejections and similar events blow a magnetic field and huge quantities of material from the surface of the Sun. The interaction of this magnetic field and material with Earth's magnetic field funnels charged particles into Earth's upper atmosphere, where its interactions create aurorae seen near the magnetic poles.[70]

The heliosphere and planetary magnetic fields (for those planets that have them) partially shield the Solar System from high-energy interstellar particles called cosmic rays. The density of cosmic rays in the interstellar medium and the strength of the Sun's magnetic field change on very long timescales, so the level of cosmic-ray penetration in the Solar System varies, though by how much is unknown.[71]

The interplanetary medium is home to at least two disc-like regions of cosmic dust. The first, the zodiacal dust cloud, lies in the inner Solar System and causes the zodiacal light. It was likely formed by collisions within the asteroid belt brought on by gravitational interactions with the planets.[72] The second dust cloud extends from about 10 AU (1.5 billion km; 930 million mi) to about 40 AU (6.0 billion km; 3.7 billion mi), and was probably created by similar collisions within the Kuiper belt.[73][74]

Inner Solar System

The **inner Solar System** is the region comprising the terrestrial planets and the asteroid belt.[75] Composed mainly of silicates and metals, the objects of the inner Solar System are relatively close to the Sun; the radius of this entire region is less than the distance between the orbits of Jupiter and Saturn. This region is also within the frost line, which is a little less than 5 AU (750 million km; 460 million mi) from the Sun.

Inner planets

Main article: Terrestrial planet

The four terrestrial or **inner planets** have dense, rocky compositions, few or no moons, and no ring systems. They are composed largely of refractory minerals such as the silicates—which form their crusts and mantles—and metals such as iron and nickel which form their cores. Three of the four inner planets (Venus, Earth and Mars) have atmospheres substantial enough to generate weather; all have impact craters and tectonic surface features, such as rift valleys and volcanoes. The term *inner planet* should not be confused with *inferior planet*, which designates those planets that are closer to the Sun than Earth is (i.e. Mercury and Venus).

The inner planets. From top to bottom rightwards: Earth, Mars, Venus, and Mercury (sizes to scale).

Mercury

Main article: Mercury (planet)

Mercury (0.4 AU (60 million km; 37 million mi) from the Sun) is the closest planet to the Sun. The smallest planet in the Solar System (0.055 M_{Earth}), Mercury has no natural satellites. Besides impact craters, its only known geological features are lobed ridges or rupes that were probably produced by a period of contraction early in its history.[76] Mercury's very tenuous atmosphere consists of atoms blasted off its surface by the solar wind.[77] Its relatively large iron core and thin mantle have not yet been adequately explained. Hypotheses include that its outer layers were stripped off by a giant impact, or that it was prevented from fully accreting by the young Sun's energy.[78][79]

Venus

Main article: Venus

Venus (0.7 AU (100 million km; 65 million mi) from the Sun) is close in size to Earth (0.815 M_{Earth}) and, like Earth, has a thick silicate mantle around an iron core, a substantial atmosphere, and evidence of internal geological activity. It is much drier than Earth, and its atmosphere is ninety times as dense. Venus has no natural satellites. It is the hottest planet, with surface temperatures over 400 °C (752 °F), most likely due to the

Figure 4.2: Wikipedia content division example

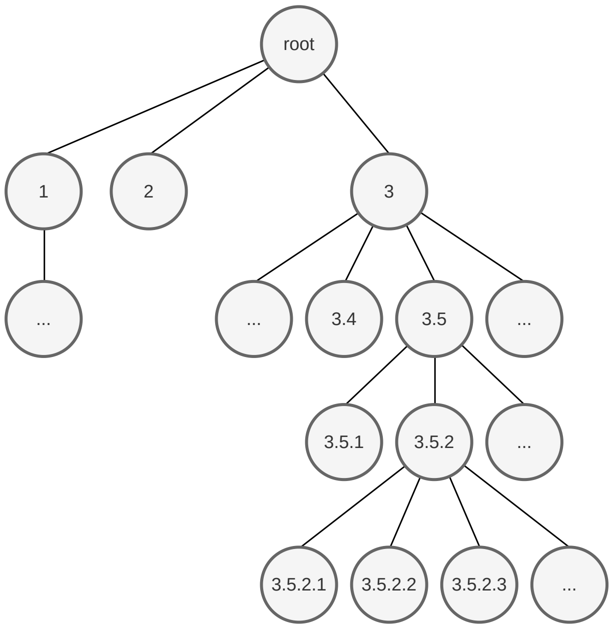


Figure 4.3: CNT example

a result, two nodes are created for the previously mentioned content as children of the node 3.5: 3.5.1 and 3.5.2, respectively. For the same reason as before, node 3.5.1 has no further content divisions and may be a leaf node, whereas node 3.5.2 has additional divisions. Figure 4.3 illustrates an example of a section of the CNT for the page described in this section.

Before concluding the CNT section, it is important to note that the content of the Web page should be divided to help the user understand properly the content; in this case, Wikipedia has a well-done division on content, which could be followed; however, different websites require different approaches.

4.2. Annotation format

The CNT is an important structure that can represent the content of a page and allows the framework to perform navigation on it based on the final user intent. The framework, on the other hand, constructs the CNT based on additional information on the page representing how the content should be managed. The additional information on the page is added using annotations, which are additional HTML attributes representing how the framework should manage the annotated content. The annotation must be done correctly in order for the framework to properly understand the content on the page.

4.2.1. Conversational Nodes annotation

First and foremost, it is critical to comprehend how annotation should be performed and how the framework will interpret it. According to the previous sections, the page is divided into different nodes in a hierarchical order, each of which is linked and represented with annotations. In other words, a properly annotated HTML component represents a node, and it can contain other nodes by containing other properly annotated HTML components. It is important to note that the annotation can be performed on existing HTML components or new ones can be created to group elements or contents according to the needs.

4.2.2. Annotation types

The annotation should follow specific tags in order to provide the framework with all of the necessary information. Following the previously specified CNT structure, there are three main tags to annotate for each piece of content (which will correspond to a node, the `cw` prefix stands for Conversational Web):

- cw-description: it refers to a description of the content (node) that will be spoken by the assistant; as such, it should be used properly in order to follow the proper way of expressing the content.
- cw-keys: this is a list of substantives that represent the content.
- cw-type: specifies the node type; currently, the node types are `navigation`, `link` and `content_reading`. The first refers to nodes that have children and thus are navigation nodes, while the other two are for leaf nodes (`content_reading` represents a node which have as content a text to be read by the assistant, whereas `link` represents a node which have as its content a link to be navigated through).

Figure 4.4 shows an example of an annotated paragraph, as well as the relationship between a node and the annotations.

Other types of leaf nodes could exist and be developed (our framework has been properly developed and studied to support its maintainability), but in order to demonstrate how to access the Web via a conversational assistant, we focus on the most important ones in terms of content: `content_reading` and `link`.

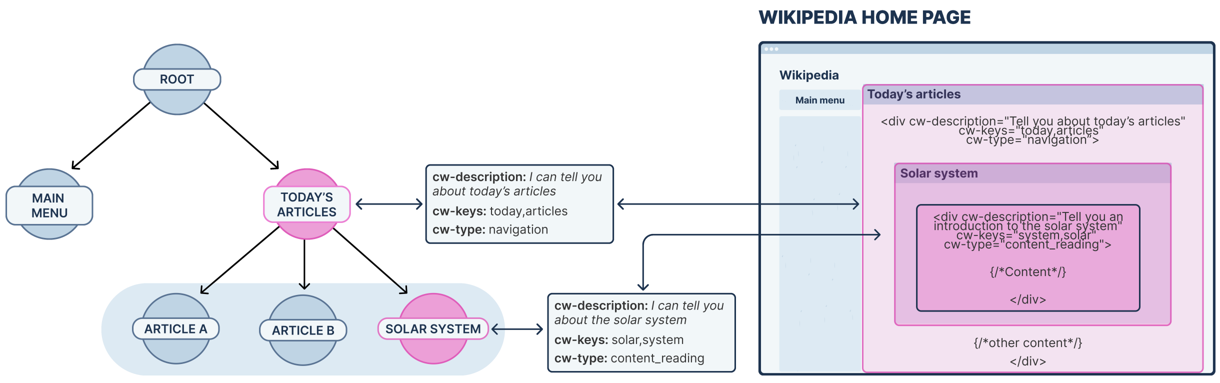

ROOT
MAIN MENU
TODAY'S ARTICLES
ARTICLE A
ARTICLE B
SOLAR SYSTEM
cw-description: I can tell you about today's articles
cw-keys: today,articles
cw-type: navigation
cw-description: I can tell you about the solar system
cw-keys: solar,system
cw-type: content_reading
WIKIPEDIA HOME PAGE
Wikipedia
Main menu
Today's articles
<div cw-description="Tell you about today's articles" cw-keys="today,articles" cw-type="navigation">
Solar system
<div cw-description="Tell you an introduction to the solar system" cw-keys="system,solar" cw-type="content_reading">
{/*Content*/}
</div>
{/*other content*/}
</div>

Figure 4.4: Annotation

```
<div cw-description="Tell you an introduction to the solar system"
        cw-keys="system,solar" cw-type="navigation">

              <div cw-attribute="text">
    <p> The solar system ... until the end of the galaxy </p>
  <p> Another concept ...  important for the community </p>
                        </div>

                        </div>
```

Figure 4.5: Annotation attribute

4.2.3. Annotations tag attribute

We have learned how to construct a CNT using annotations; however, we must also provide the framework with the content of the leaf nodes in order for it to be analyzed. The framework could parse the content of the leaf nodes simply by annotating it in the same way as described in the previous sections, with the exception of changing the HTML attribute. Instead of the provided HTML attributes, we only use the custom one: `cw-attribute`. This last attribute could be `text` if the content is a text to be read. Of course, additional attributes could be added and customized to add more functionality. The main reason for the framework use of this additional attribute is that there are leaf nodes that may contain final text to be read by the user but also a lot of non-useful content (ads, images and so on), therefore the `cw-attribute` tag can easily identify the final content. In particular, the framework will identify `<p>...</p>` tags as text content, therefore multiples tags could exist together inside an HTML element with the `cw-attribute` tag. The annotation attribute tag is relevant only for `content_reading` nodes as for `link` nodes the useful information is extracted automatically (see Section 6.2.6). Figure 4.5 illustrates an example with `text` content.

4.2.4. Annotations limit

We understand that using annotations could be a limitation for the framework and a challenge for website developers, but their use is critical since website pages vary in terms of content and structure and thus a properly annotated Web page could be used by the framework independently of its architecture. A future work or goal is to automate annotation of Web pages or make the framework capable of extracting important data without the need of annotations; however, this work will necessitate extensive researches on Web pages and machine learning techniques that are beyond the scope of this thesis. Another key point is that Web pages creation should follow common guidelines in order to provide effective accessibility; annotations want to build a solid groundwork to identify content on a Web page in order to give complete accessibility to it.

4.3. CNT - building

In this section we present a pseudo-algorithm to illustrate the procedure for building a CNT given a specific URL. The Algorithm 4.1 shows how a CNT is build starting from a given URL.

Algorithm 4.1 build_cnt()

```
Data: URL
Result: CNT structure
1: body <— headless_browser.retrieve_body()
2: root = Node(body)
3: return root
```

The CNT representation is a root Node containing the `body` element, extracted with the function `headless_browser.retrieve_body()` which will retrieve the element from the current Web page with all of its content. The recursion for the whole creation is explained in the initialization of a `Node` structure as explained in Algorithm 4.2.

Algorithm 4.2 Node()

Data: page_elem `/* generic element of the page, at first will be the root element, subsequently a inner annotated element and so on, until no other annotated elements are found */`

Result: recursion initialization of all CNT nodes

```
1:  children <— retrieveAllChildrenElements(page_elem)
2:  for child ∈ children do
3:      if child.cw_type == navigation then
4:          sub_elements <— retrieveAllChildrenElements(child)
5:          for grand_child ∈ sub_elements do
6:              children.remove(grand_child) /* remove from all the children, the
                ones under the first layer of children */
7:          end for
8:      end if
9:  end for
10: children_nodes = []
11: for child in children do
12:     children_nodes.append(Node(child))
13: end for
14: self.node_type = retrieve_attribute(cw-type)
15: self.description = retrieve_attribute(cw-description)
16: self.keys = retrieve_attribute(cw-keys)
17: self.children = children_nodes
18: self.content = retrieve_content()
```

`retrieveAllChildrenElements()` is a particular function able to retrieve all Web elements under a specific one (children) which have a `cw` attribute. A Headless Browser and particular APIs can be used to retrieve Web elements. The fundamental issue is that fetching Web elements does not adhere to the original hierarchy, and many websites do not follow proper structures when developing their pages. As a result, after retrieving all the children of a certain element, the code from line 2 to line 9 cleans these children, deleting any that are grandchildren of the first layer detected, or are on an additional layer that is not relevant to us. This is because `retrieveAllChildrenElements()` includes all possible children (even recursively) without providing us with a correct organization. An additional problem is that retrieving all the content with `retrieveAllChildrenElements()` does not lead to a useful result to be interpreted and build a proper CNT. Therefore we could state that the first 9 lines of code retrieve the first layer of children given a specific element, while from line 10 to 13 the children of the given element are initialized (using recursion). Last lines save in the current node all the necessary attributes retrieved thanks to the `retrieve_attribute()` function which retrieves through the Headless Browser needed tag attributes while `retrieve_content()` retrieves relevant content (such as texts, paragraphs and so on).

5 | Natural Language Understanding - NLU

We have seen in the previous chapters that understanding the proper intent is necessary for taking the appropriate action in response to a specific user's utterance. With the use of a natural language understanding (NLU), a subtopic of natural-language processing (NLP) in artificial intelligence that deals with machine reading comprehension, we are able to achieve this outcome. We need to build an NLU model that could parse user utterances and extract their intents and entities. The model for recognizing user intent is built for a specific URL that the user is visiting; this choice is made because the intents are dependent on the content of the page; therefore, a different URL will have different content, and thus a different model will be created with these new dependencies. Furthermore, even if certain intents may be the same for multiple pages, a new model must be trained each time because it is responsible for selecting the proper intent from all available ones (the train must be performed using all possible intents), and because some intents will certainly change, the train must be performed for each new URL. Before diving into the specifics of how the model is built and utilized, we will discuss and demonstrate the intents that our framework recognizes.

5.1. Intents

We here describe the intents that our framework understands and how they affect the user's request. A careful study on pattern integration, which is addressed in Section 6.1, is utilized to select the intents to be implemented. The following list summarizes all of the framework's intents, divided in different categories, with brief descriptions; a thorough explanation of how each intent is managed and which action is taken is provided in Chapter 6.

- Navigation intent (intent related to navigation)
  - *content*: it relates to the request of reaching a specific content on the URL and

could be achieve for example with "Tell me about Mars".

- Content-reading intents (intents related to text content in `content_reading` nodes)
  - *read*: it relates to the request of reading a specific text the assistant has found on the current node, for example with "Read me the first paragraph".
  - *read_paragraphs*: it relates to the request of knowing how many paragraphs are present inside a specific text.
  - *read_links*: it relates to the request of reading all the read links on the current `content_reading` node. The intent is also used to select one of these links.
- Scaffolding intents (intents related to default actions during the browsing of a Web page)
  - *help*: it relates to the request of help.
  - *page_info*: it relates to the request of having information on the content of the page.
  - *location*: it relates to the request of having information on the current position inside the Web page the user is visiting.
  - *repeat*: it relates to the request of repeating what the assistant has just said.
  - *back*: it relates to the request of returning to the previous content (i.e. node) during the navigation.
  - *top*: it relates to the request of returning to the starting point of the user navigation.
  - *affirmative*: it means an affirmative answer to the previous question asked by the assistant.
  - *negative*: it means a negative answer to the previous question asked by the assistant.

The list of intents just stated is the one identified through our integration with the NLU; specifically, we construct a model on a given dataset and then estimate the user's intent from its utterance, resulting in one of the preceding list.

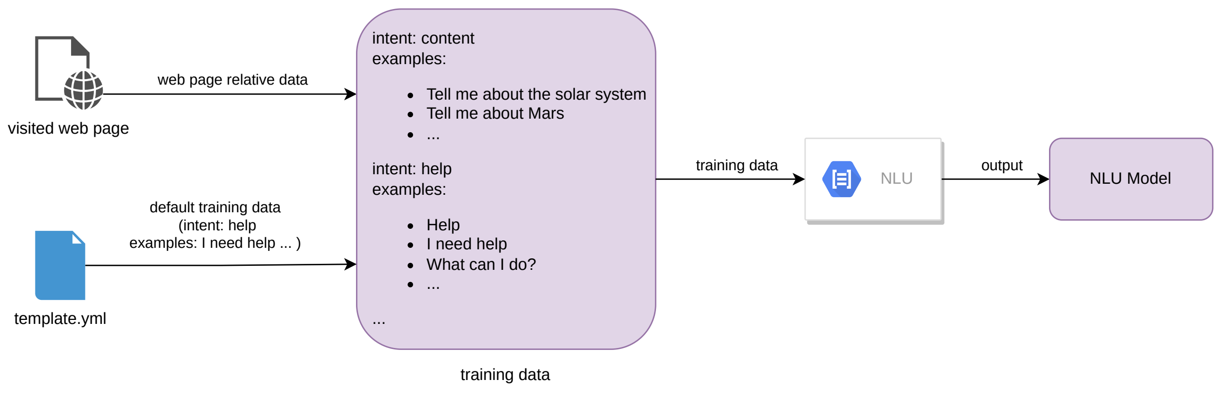


Figure 5.1: NLU model Initialization

5.2. Building the NLU model

We have stated that the NLU model is required to estimate the user's intent, and that each model is unique based on the page visited by the user. To understand how a NLU model is generated, we must first understand how one page differs from the others in terms of content. To begin training the model, we must first declare all of the intents to estimate (as previously described), and then offer some examples of data to the NLU in order for the model to be trained and ready for use. To detect the intent `help` we must provide the NLU with instances of sentences relevant to the given intent, such as "Could you help me?", "Help", "I need help", and so on. The NLU will take care of the rest. In our framework, all intents are always based on the same training data identified with a simple template; the only difference is the intent `content` which varies depending on the content of the page; for example, in a website page about the solar system, some examples could be "Tell me about the solar system" or "Tell me about Mars", whereas in another site about another argument, for instance, a biography, some examples could be "Tell me the private life" or "Tell me about his works". As a result, each time a new website is visited, a new model is created and trained on the given data, and each new utterance of the user is parsed by the NLU model (also known as Interpreter), recognizing the user's intent and performing the necessary actions. This process is illustrated in Figure 5.1.

5.3. Entity extraction

Each of the previously described intents could be related to a specific action; however, the first two intents, specifically the intent `content` and the intent `read`, require further explanation.

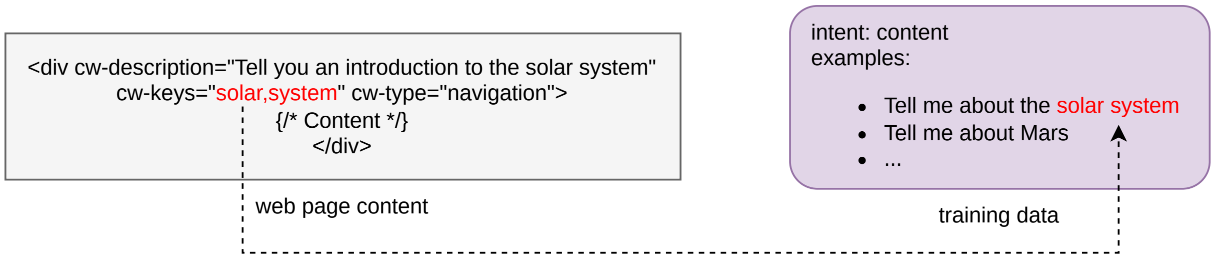

<div cw-description="Tell you an introduction to the solar system"
cw-keys="solar,system" cw-type="navigation">
{/* Content */}
</div>
web page content
intent: content
examples:
• Tell me about the solar system
• Tell me about Mars
• ...
training data

Figure 5.2: Entities - cw-keys relationship

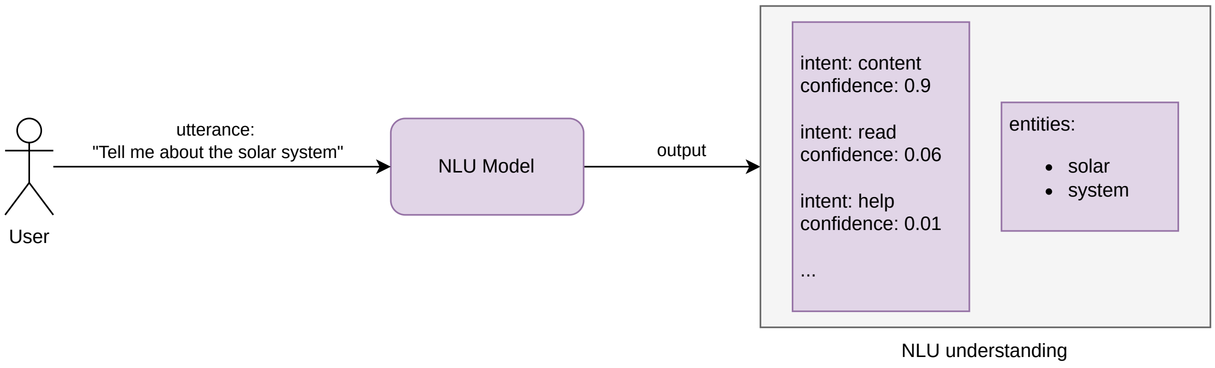


Figure 5.3: NLU - output

When our Interpreter extracts `content`, we must explain how to find the requested content on our page, thus we must extract useful words in our context. The NLU could assist by extracting entities for us. Entities are structured pieces of information included within a user message; hence, if a user requests "Tell me about the solar system", our Interpreter will understand the utterance's intent as `content` and will also add further information as extracted entities, in this case: "system" and "solar". Subsequently, we could extract the page's content using these two extracted entities (how specifically we could find the requested content is explained with more details in Section 6.2.3).

Our Interpreter has to be trained with proper data to understand our intents and to know which entities it can extract for us, thus we must include all of the entities available on the page in the training data. As mentioned in Section 4.2.2, we leverage the `cw-keys` present in the annotations to achieve this effect. In other words, each node is represented by a set of words (annotated in `cw-keys`), which we utilize as entities in our training data so that the Interpreter is able to understand when a user requests a specific content by simply extracting the mentioned words. The relationship between `cw-keys` and our Interpreter is shown in Figure 5.2, whereas Figure 5.3 illustrates the output of the NLU model (Interpreter) given an user's utterance.

Furthermore, the intent `read` requires entities to function properly; in fact, each user who wants to read may request a different content. For the cited intent we employed the fixed items stated below:

- first paragraph
- second paragraph
- ...
- ninth paragraph
- tenth paragraph

We believe that each separate entity for the intent `read` should be taken by the above-mentioned list, allowing for the potential of supporting a wide range of information independent of the visited website. For this first implementation, we only supported the specified entities; however, additional entities could be added and supported. A complete explanation of the `read` intent could be found in Section 6.2.5.

5.4. Intent selection

NLU is a powerful approach capable of understanding natural language, but it must be properly utilized to achieve the desired results. When an utterance is passed to the Interpreter, it is not accurate to claim that an intent is extracted since all intents are extracted, each with a specific confidence level ranging from 0 to 1. As a result, in our implementation, we have a Policy component (described with further details in Chapter 7) that will handle various confidences and choose the appropriate one. It is natural to believe that the intent with the greatest confidence is the correct one; nevertheless, we must also consider the extracted entities and the user's current context; as a result, a complex component must be constructed and tuned to fulfill our requirements. A thorough explanation of how this component operates can be found in Section 7.1.3. The goal of this chapter is to describe how NLU works in our context and how our framework can interpret the user's request with all of its content (i.e. entities) by using its outcomes. We attempted several configurations and models to get the previously described final result; as a consequence, in the following sections, we exhibit our past implementations and solutions and explain why they are not suited for our framework.

5.5. Alternative solutions for NLU integration

In this section we show different alternative solutions for the use of NLU withing ConWeb, which we explored. We also show why we discarded these approaches to follow the one presented in this chapter.

5.5.1. Intents per content

The first approach (inspired by previous works [4]) used a different intent for each different content of the page and additional intents for reading. It was a great starting point, although it had a few issues. At first, it lacks intents of support (such as `back`, `help`, and so on) and we believe that these intents are essential and fundamental to support the user's experience, and they are chosen in accordance with the different pattern's guidelines as mentioned in Section 6.1. Another issue was that, for what regards the content of the page, it did not employ entities and only different intents. In other words, for selecting which content to be reached a different intent is generated. The "intent" is the user's goal, and if the goal is to retrieve content, even if the content changes, the intent remains constant; this is why entities are used. The aim is always the same (retrieving a piece of content), and the only thing which changes is the content (entities). We opted to discard this implementation in order to better correspond to the NLU utilization, and also because entities extraction is a strong approach to searching the information in the CNT; a thorough explanation can be found in Section 6.

5.5.2. Multiple Interpreters

After introducing entities and new intents, we initially assumed that the best strategy would be to have various Interpreters for different intents, properly grouped. The reason behind this choice was that the content of the page varies each time, requiring us to train our model each time, while the majority of the intents and examples remain constant. As a result, we created three distinct Interpreters:

- Navigation Interpreter: an Interpreter able to handle the `content` intent.
- Scaffolding Interpreter: an Interpreter able to handle the intents that should be always present and not retrained like `back`, `top`, `help` and so on.
- Reader Interpreter: an Interpreter able to handle the intent `read`.

The Navigation Interpreter is the only one of the above Interpreters that needs to be trained every time a page is accessed. Each of the defined intents worked as expected, but the main issue we faced was the **out-of-context** problem. As previously stated, each Interpreter provides a list of intents with a specific level of confidence; therefore, when we fed the user's utterance to the Interpreters, we got a lot of "false positives". This occurs when there are few intents (not many for each Interpreter) and the training data examples for each intent do not change significantly (in fact, the only things that change in the Navigation Interpreter are the entities). We explored many ways to detect the out-of-context problem (see if an entity is present to predict the Navigation Interpreter's out-of-context), but they did not match our criteria giving us always confidence problems and "false positives". In fact, the out-of-context problem is well known in NLU, and having extra Interpreters contradicts proper usage. Having a single Interpreter (as we implement) is how various NLU approaches were thought, and the out-of-context problem could be detected by a threshold on the confidence level; the fact that we have different intents all together helps our Interpreter understand how to differentiate them, rather than having few intents for each single Interpreter.

6 | Intent Handlers: Bots

The framework is developed using the guidelines of a users study [5] in order to meet the need of being a real and valid solution for visually impaired people, providing them with an alternative and reliable way of accessing the Web. As a result, in this chapter, we briefly summarize the users study with all of its outcomes and then show how we apply these findings in our platform through the Intent Handlers.

6.1. User-based Requirements

In the time period from April to September 2021 two students at Politecnico di Milano performed an extensive user research to understand how users would navigate content and services accessible on the Web by "talking to websites" instead of browsing them visually, by expressing their goals in natural language and accessing the websites through a dialog mediated by a Conversational Agent (CA) [5]. With the help of 26 blind and visually impaired users, along different sessions of interviews and co-design workshops, they were able to identify and validate some prominent challenges and some related interaction-design patterns sustaining the notion of Conversational Web Browsing.

Firstly users were asked them to describe their experience with current voice-based assistive technologies. By using online tools for Conversational Agents rapid prototyping (e.g., DialogFlow), they were solicited to express their desiderata on the design of novel Conversational Agents for accessing websites. In the following we illustrate some of the identified conversation *patterns* from this work, which mainly refer to the structure of conversation for Web browsing. These patterns guided us in extending a preliminary version of the framework platform to: *i)* support an incremental, dialog-based exploration of the website, and *ii)* grant flexibility in the dialog organization, to fulfil the need of personalized browsing experiences.

Shaping-up the map of the navigable space. Users claimed that learning the structure of the website is a crucial initial step when they access a website for the first time. For this reason, they asked for strategies to identify high-level navigation mechanisms to support them in understanding the website structure, and fluidly move along the main areas (e.g.: "You can browse the main menu, [...]"). To identify how to move along different information nodes, they mentioned mechanisms for link predictability (e.g.: "Do you want to read a preview, or [...]") and for keeping track of the navigational context (e.g.: "You are now in the Wikipedia Home Page.").

Navigating through intelligible and quick mechanisms. Depending on their tasks and preferences, participants demanded for different navigation strategies. They described in-depth explorations to narrow down navigation options along the hierarchy of nodes, but especially punctual, fast-served requests were discussed as a means to locate a desired content (e.g.: "Tell me about [...]").

Summarizing and segmenting the page content. The research conversational paradigms should prevent unwanted and unneeded explorations resulting in poor user experiences. Segmenting contents and highlighting characterizing keywords could help localize the content of interest (e.g.: "Jupiter is [<short content preview>]. Do you want to know more or reading something else?").

Providing access to conversation-scaffolding intents. Users frequently expressed the need for scaffolding intents to help them identify possible actions at different navigation levels and for the provision of feedback on the system status, such as the use of landmark cues.

These patterns resulted in a specific methodology for conversation design that we adopted to structure the dialog system of the ConWeb framework.

6.2. Dialogue System

We learned in Chapter 5 how our framework can comprehend the user's intent by parsing its utterance with an NLU model. After understanding the user's intent, we must take the appropriate action in response to it. To reach this outcome, there are various Intent Handlers, which are components (bots) of our frameworks that can perform the appropriate action based on the provided intent. Intent Handlers follow the Interaction-design patterns already described; later in this chapter a explanation on which patterns and how they have been implemented and mapped is illustrated. There are different bots, each one handling a particular set of intents and we are going to describe how they are organized and built to accomplish this goal.

6.2.1. Intents mapping

Since different bots handle distinct sets of intents, Figure 6.1 illustrates a mapping between all of the intents stated in Chapter 5 and their respective bot. We can observe that there

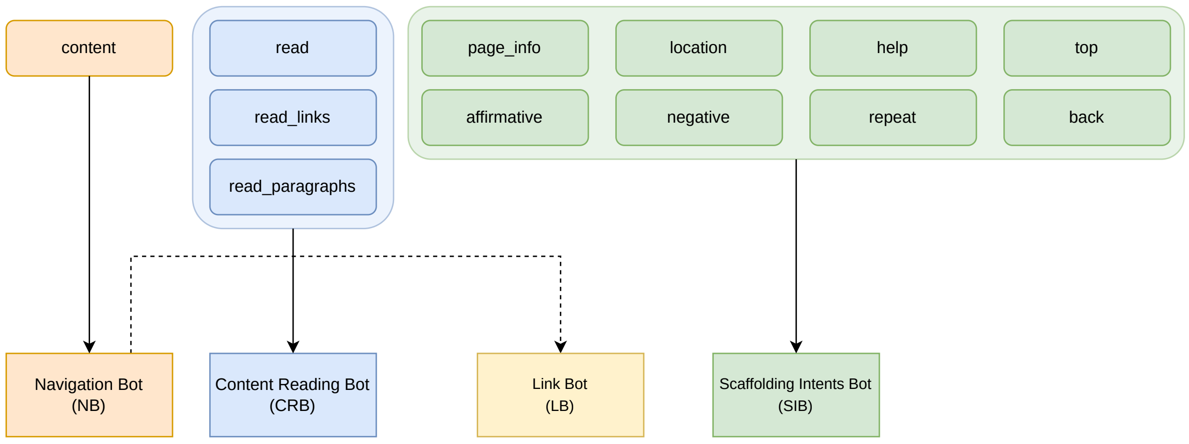


Figure 6.1: Intents mapping

are four bots, in specific:

- Navigation Bot (NB)
- Content Reading Bot (CRB)
- Link Bot (LB)
- Scaffolding Intents Bot (SIB)

The NB just handles a single intent (which, as mentioned in Section 6.2.3 necessitates sophisticated behavior), whereas the SIB manages a set of intents (those that should improve the user's experience when browsing the Web as default commands). The CRB handles only intents related to a node of type `content_reading` (related description could be found in Chapter 4) which will be explained in Section 6.2.5. The LB, on the other hand, does not handle a specific intent, but it is tied to the NB; a detailed explanation of its function is provided in Section 6.2.6.

6.2.2. Bots flow: Bot Manager

Before diving into the specifics of all the developed bots and their actions based on the selected intent, it is crucial to understand which bot must be invoked based on the chosen intent and the current navigation context. It is important to understand the flow regarding the bot selection each time a new utterance from the user is parsed, as well as why another component, the Bot Manager, is needed. Figure 6.2 illustrates the flow from the parsing of the user's utterance (extracting the intent and entities) to the call of the appropriate bot. At first, we could see that based on the user's intent, we might call one of the tree bots: NB, CRB or SIB. It is important to note that the CRB is called

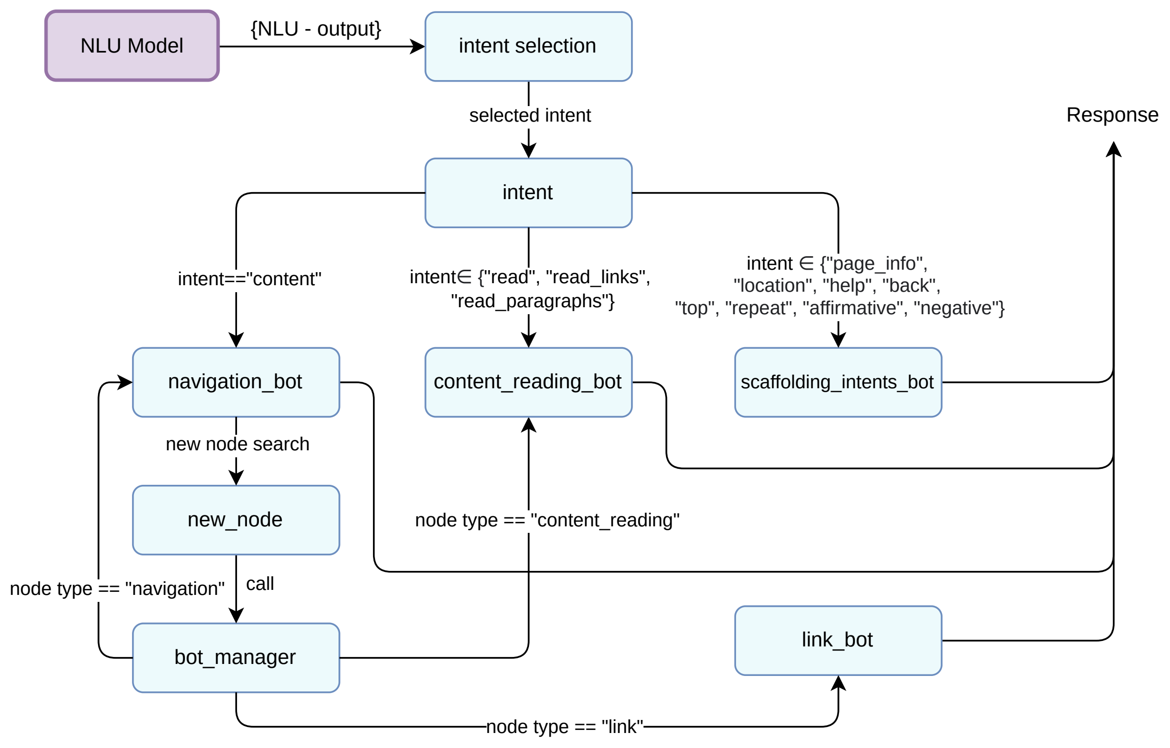


Figure 6.2: Bots pipeline

only if the user's current position in the page is a leaf node of type `content_reading` (see Section 4.2.2) where the CRB could perform actions (Section 6.2.5), in other words a navigation context check is performed. If the SIB is invoked, an appropriate action (described in Section 6.2.4) is performed and no other actions are required. The NB is the most challenging aspect since it will understand the content you are looking for and will discover the corresponding node in the CNT, a better explanation of how the node is found can be found in Section 6.2.3. However, once we reach the new node, we must take the proper action based on the type of node. As a result, the Bot Manager selects the right bot based on the node type:

- NB: the navigation continues, and the node's content is explained.
- CRB: the node's content is read or provided.
- LB: the Link Bot is invoked to redirect the user to a new URL.

The actions of all the bots are shown in the following sections, with further information and explanations providing a comprehensive overview of the provided flow.

6.2.3. Navigation Bot - NB

The Navigation Bot is the framework's most significant bot, assisting the user in navigating across the whole CNT. Indeed, we can see from Chapter 4 how the page's content is separated into nodes, each of which contains additional nodes with inner content. The NB key feature is its support for the intent `content`. In essence, when a user first enters a new page, it is placed at the root node, and the NB will obtain the description of the children nodes in order to let the user able to understand the content from the current position. As a result, the user will be able to select which content to select or, in addition, request another one that has not yet been presented but the user knows it is present (or maybe the user wants to check if it exists or not). When a user's utterance asking for a specific content is parsed and the intent `content` is extracted, the NB will bring the user to the requested node, updating the user's navigation context (for example, the current node during navigation) and other necessary structures relevant for the framework. After having brought the user to the requested node, as explained in Section 6.2.2, the Bot Manager will select the action to perform depending on the type of node where the user just landed. If the node is a `navigation` node (or a parent node), the NB is invoked once more to explain to the user the contents of its children nodes, as it did previously with the root node. To summarize the NB features, we could state:

- Bringing the user to the requested node.
- Explain to the user the content of the children nodes of the landing node (i.e. the node where the user arrived which is the root one if the user just landed in a new page).

An important specification must be done on how the NB is able to find the correct node of the whole CNT and could be found in the following.

CNT search

From Chapter 4, we learned how the CNT is constructed and how each node is represented. The relevant information about its nodes, in particular, are their keys (i.e. the entities representing the content of each node). The NB searches on the CNT using exactly the entities extracted from the user's utterance. In other words, after the user's utterance has been parsed by the NLU model and the intent `content` has been extracted, the entities from the utterance are extracted and used to search in the CNT for the requested content (node) using the following procedure:

- A depth first search is performed from the current position until all the CNT from the current node is explored. Thus, a structure is built with two elements; the node and the number of entities detected in that node from the ones specified by the user. The structure saved is of form: `{node, |entities|}`

- A check is made to see if there exist at least one node with a matching of entities strictly greater than 0. If it exists, the node with the most entities is picked (in the case of several nodes with an equal number of entities, the node encountered first is chosen, with a priority on the first layer of nodes); if it does not exist, the method is repeated, beginning from the root node rather than the current one.

The following algorithms show the pseudo-algorithms used for the aforementioned procedure. The Algorithm 6.1 shows how a recursive search is done starting from a specific node and the population of a structure `node_mapping` with a map between the encountered nodes and the matching keys with the entities extracted.

Algorithm 6.1 recursive_cnt_search()

```
Data: node, entities, node_mapping
Result: node_mapping between nodes and matched keys
 1: for $n \in node.children$ do
 2:     counter = 0
 3:     for $entity \in entities$ do
 4:         if $entity \in n.keys$ then
 5:             counter =+ 1
 6:         end if
 7:     end for
 8:     node_mapping[n] = counter
 9: end for
10: for $n \in node.children$ do
11:     recursive_cnt_search(n, entities, node_mapping)
12: end for
```

The Algorithm 6.2 instead, using the Algorithm 6.1 performs the whole search starting from a node and managing also the case in which a node is not found with the first search and therefore the whole procedure is repeated starting from the root node.

Algorithm 6.2 search_node_by_entities():

```
Data: node, entities
Result: node_mapping between nodes and matched keys
 1: node_mapping = {}
 2: recursive_cnt_search(node, entities, node_mapping)
 3: count_max = 0
 4: found_node = None
 5: for $key, value \in node\_mapping.items()$ do
 6:     if $value > count\_max$ then
 7:         found_node = key
 8:         count_max = value
 9:     end if
10: end for
11: if $found\_node \neq None$ then
12:     return found_node
13: end if
14: node_mapping = {}
15: recursive_cnt_search(root, entities, node_mapping)
16: count_max = 0
17: found_node = None
18: for $key, value \in node\_mapping.items()$ do
19:     if $value > count\_max$ then
20:         found_node = key
21:         count_max = value
22:     end if
23: end for
24: if $found\_node \neq None$ then
25:     return found_node
26: end if
27: return None
```

It is important to note that the algorithm just provided is created based on frequent user experiences when looking for content. All of the information used and the guidelines followed, in particular, could be summed up in the following points:

- The user usually requests something that has already been presented by the assistant or is related to it. As a result, the search begins at the current node in the CNT. If the user requests something unrelated, it will not be found, and the search will be repeated from the root node.

- The user normally asks for something presented by the assistant on its first layer of nodes, for this reason in case of multiple nodes with the same number of entities the first one encountered is chosen. Even if the search is performed with a DFS algorithm, the results of the first layer of nodes are stored before all the remaining ones.
- The framework cannot know the content asked by the user, for this reason the two searches (one from the current node and the other from the root) are performed on the entire CNT. It is important to note that the performance impact of these searches is negligible because the CNT should represent the content of a page, and even with hundreds of nodes, the performances do not have a significant impact, and a higher number of nodes indicates an inadequately annotated website or a not proper developed Web page.
- If a user wants to be more specific in asking for a specific content by using more words, for example if there exists identical content but with distinct peculiarities, the Navigation Bot can execute the search using all of the words (entities) provided by the user, therefore accomplishing its request on the entire CNT.

All the information on the CNT search stated before have meaning only and only if the annotation of the Web page has been computed or done correctly and in a proper way. The following points summarize the most significant guidelines to follow when using keys to annotate the website:

- The keys should be words (entities) that represent the content.
- The keys should be placed to support and follow the hierarchy; the framework will handle the rest. A node discussing solar planets, for example, should be annotated using the keys: planets and solar. If the referenced node contains a child node that discusses Mars, it should be annotated with: planets, solar and Mars. In this way the user is able to reach the Mars paragraph specifying also "solar" and "planets" even if there is another node of the website talking about Mars. Of course, the Navigation Bot can discover the appropriate content even if the annotation hierarchy is not properly annotated, but it cannot handle limit scenarios like the one stated before where the are multiple nodes talking about the same content in different places on the same page.
- The following entities should not be included in the keys: first, second,..., tenth,

paragraph, help, page, back, top, yes, okay, of course, where, link and repeat. The rationale for not including these items is the **out of context** problem, which is explained in Chapter 5.

6.2.4. Scaffolding Intents Bot - SIB

The Scaffolding Intents Bot (SIB) component is in charge of handling the user's default requests, which are identified by the intents already provided. This section explains which action is executed based on the chosen intent and the current context. Given that the intent has already been chosen and the bot has been invoked to handle it, we will dive into the management of all intents. All of these intents has been already presented in Chapter 5.

Help

The `help` intent is one of the main ones handled by the SIB. It can be accessed at any moment by a user who asks assistance. When the SIB is invoked, it returns a specified answer to the user, triggering the need for assistance. Following then, the client will be in charge of handling the response and composing a sentence to help the user; specifically, it will be a description of the assistant's features and how to use it. The goal of this intent is to provide users with an exhaustive answer in order for them to correctly use the assistant.

Page information

The SIB also handles the `page_info` intent. It, like the `help` intent, could be invoked by the user at any moment. When the SIB receives the mentioned intent, it will respond with a specific answer including the needed information. The description of the first layer of children nodes is retrieved from the root node and then added to the response. Finally, the client will be in charge of formulating the response to expose the content of the descriptions received to the final user. This intent's purpose is to offer users with a comprehensive answer about the major content of the current page they are visiting.

Repeat

The SIB handles a simple intent `repeat` of repeating what the assistant previously stated. `repeat` could be invoked by the user at any moment. When the SIB receives the mentioned intent, it will return the old response that was originally returned during the prior answer. The purpose of this intent is to allow users to listen again for something they may have misplaced or simply wish to listen again. The previous response is saved in a cache component and will be retrieved by the SIB. A more detailed explanation of how cache is implemented can be found in Chapter 7.

Top

If users become lost while browsing Web pages, it is essential to return them to the point where the browsing began. As a result, the SIB manages the intent `top`. It, like other intents, can be invoked at any time and will return users to the point where they began browsing. Specifically, upon receiving the `top` intent, the SIB will obtain from a cache component (details of which can be found in Chapter 7) the initial URL specified by users and redirect them to the indicated one; the current context will be updated. It is critical to note that the URL to which the intent `top` redirects is the one specified by the user to begin navigation. If the user navigates between link nodes, the URL to which the intent `top` refers will always be the same (the one stated at the beginning when opening the first Web page). If, on the other hand, the user opens a new website (without using any links on the page) and abandons the existing one, the URL to which the intent `top` redirects is changed to the one just used.

Affirmative and Negative

The `affirmative` and `negative` intents are handled by the SIB and should be invoked only after the assistant has asked the user a specific question. Obviously, a user could trigger the intents at any moment, but the assistant's response will be that it did not understand because the current context does not require a positive or negative response. The current assistant supports the question "Do you want to read the next paragraph?" asked by the CRB when the end of a paragraph is reached. If the SIB is invoked in the current context on the intent `affirmative`, the following paragraph will be read, calling the CRB. If the intent `negative` is invoked, no action is taken and a simple response is forwarded to the client.

Back

When users navigates the current page using the CNT, they explore all of the nodes. If there is the need of going to the preceding nodes, the intent `back` is triggered. The SIB can manage the `back` intent, and when triggered (which can happen at any time), it will take the user to the previously visited node, updating its current navigation context. The node queue is made feasible by a list of nodes handled by the framework that keeps track of all the visited nodes on the current Web page. When the `intent` function is invoked while the user is at the root node, the previously viewed page is loaded and the navigation context is updated. However, the aim of this intent is to navigate through nodes and not through pages for which there are links and if needed the intent `top`.

Location

The SIB can handle the `location` intent if users get lost while browsing a Web page and wants to know where they are under a certain URL. When the aforementioned intent is activated, the SIB will retrieve all the relevant information and forward them to the client, who will be responsible to formulate an exhaustive response to the user. The SIB, in particular, will examine the current node type. If the node is a `navigation` node, the SIB will retrieve its children's descriptions, along with the parent's, and transmit them to the client. If, on the other hand, the node is of the type `content reading`, the description of the node, as well as the type of text that is present, are transmitted to the client. The `link` node cannot be used as when users go through it, they are redirected to a new page, making impossible to activate the intent. Finally, the client will formulate an answer that describes the current position and what you can do from there (surf children nodes and which ones or read different type of content).

6.2.5. Content Reading Bot - CRB

In this section, we will go through the Content Reading Bot (CRB), which is in charge of reading and handling the text on the page. When users land on a `content_reading` node, the CRB is triggered; in other words, after asking the assistant to retrieve a specific content, they are brought to the node regarding the desired content, which is of type `content_reading` and subsequently the CRB is called. When the CRB is invoked, it is first initialized with all of the content of the selected node, specifically all of the text. We know from the Section 4.2.3 that each leaf node contains a `cw-attribute` identifying the content, which value is `text`, thus the CRB is initialized using this one. The text inside the `text` attribute is divided into `n` paragraphs following the tag <p>...</p>, so that the CRB can handle them. For this implementation, we limit the number of supported paragraphs to 10, because we discovered that in our test page of `solar system`, for a total of 84 paragraphs divided into 40 nodes, there is an average of 2.1 paragraphs per node, and the maximum number of paragraphs found in a single node is 6, for these reasons 10 is a sufficient number to cover the majority of cases. Furthermore, the reason why the number of paragraphs must have a definite maximum number is that a user could utilize an NLU model to request a particular paragraph, therefore we must train the Interpreter

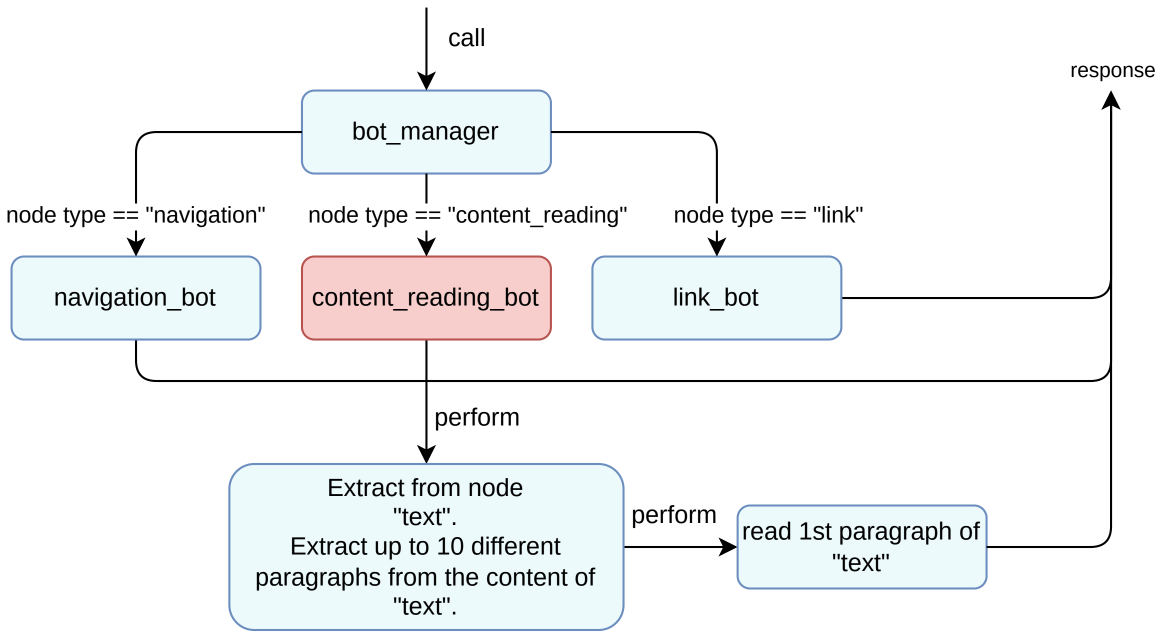

call
bot_manager
response
node type == "navigation"
node type == "content_reading"
node type == "link"
navigation_bot
content_reading_bot
link_bot
perform
Extract from node "text". Extract up to 10 different paragraphs from the content of "text".
perform
read 1st paragraph of "text"

Figure 6.3: CRB initialization

to comprehend all the numbers and cases to call a specific paragraph. Also, if the number of paragraphs is too high, it is possible that the annotation on the website was not done properly; nonetheless, the framework supports scalability, and the number of supported paragraphs may be easily expanded even if it goes against the hierarchical navigation principles and the features of supporting navigation by content. To summarize the CRB initialization when a node of type `content_reading` is encountered, we could state:

- 1. the attribute "text" from the node is extracted.
- 2. "text" content of multiple paragraphs identified by the tag <p>...</p> is extracted and saved for a maximum of 10 paragraphs.
- 3. an answer in provided to the user who just landed on the selected node.

The third point emphasizes that an answer is delivered to the user, but how it is formed necessitates a further explanation. It is assumed that the user wants directly to hear the content, therefore the text is returned and read by the client. In particular, only the first paragraph is returned with a final question if the user wants to continue the reading to the following paragraphs (only if they exist). Figure 6.3 illustrates the flow of the mentioned steps about the CRB.

What has been stated so far has been the initialization of the CRB for a specific node when the user is brought to it, but to continue the reading, the CRB also supports the intents `read`, `read_links` and `read_paragraphs`, therefore when users are inside a `content_reading` node, they can ask to read a specific content, to read all the links which are present (and have been already read) or to know how many paragraphs are present. In the following sections we explain better how each intent is managed by the CRB.

Read

The intent `read` aims at reading the specific content requested by the user. When the intent `read` is triggered, it is sent to the CRB along with its entity (thanks to the NLU model explained in Chapter 5). As a result, the CRB will read and return as text the exact entity specified by the user. The following entities are supported by the Content Reading Bot:

- first | first paragraph
- second | second paragraph
- ...
- tenth | tenth paragraph.

The corresponding content will be returned and read by the client, and a question about reading the next paragraph will be asked (only if it exists). To exit the node and proceed to another, a user could simply ask for another content or go back with the intent `back`, the use of the CRB will not interfere with the other intents due to proper NLU use and a proper implementation of the Policy component, which is better explained in Chapter 7.

Read links

If the text contains several links, which often happens in Web pages, users can retrieve them by using the intent `read_links`. In particular, when users are inside a `content_reading` node (and therefore the CRB has already been initialized), they could ask to know which links are present in the text that the assistant has just read. In this manner, the user can select one of the given links and be redirected to the appropriate page. To select one of the proposed links, users must use the term `link` to indicate to the Interpreter that they wish to select a specific one and easily trigger the `read_links` intent.

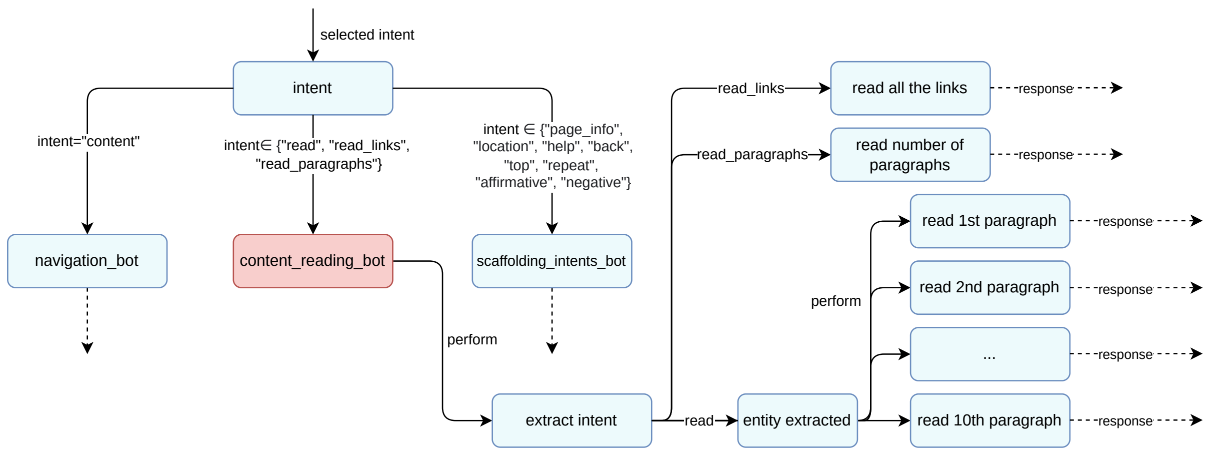


Figure 6.4: CRB features

Read paragraphs

When the user first lands on a `content_reading` node, the assistant reads the first paragraph and then asks if the user wants to read the next one (only if another paragraph is present). This choice is made for usability using the user's study guidelines [5], however if the user wishes to know how many paragraphs are there in the text, the `read_paragraphs` intent could be used. The CRB will return the number of paragraphs in the current node, and users can choose which one they prefer. Figure 6.4 illustrates a flow diagram of the CRB process including all of its features.

6.2.6. Link Bot - LB

The Link Bot (LB), which is in charge of dealing with nodes of type `link`, is another important component of the Intent Handlers. Because the LB is only triggered when the user lands on a node of type `link` as a result of the NB, it does not handle any form of intent. In other words, it is simply responsible for carrying out a certain action in response to a specified event. When a user arrives at a general node, the following actions are carried out:

- The node type is checked and depending on the type the corresponding bot is called.
- If the type is `link`, the LB is called.
- The LB is initialized retrieving from the node the link contained in it. The original link, is retrieved when the page is loaded checking inside the annotated content or for additional attributes as the `href` one.
- A response is then returned to handle a new URL with the one extracted before.

After these steps has been completed, the new URL is loaded, the navigation context is updated and a response is returned to users informing them about the new page where they just landed.

6.3. Interaction-design patterns integration

This section demonstrates how the Interaction-design patterns explained in Section 6.1 are implemented into the framework platform. Table 6.1 displays the mapping between the patterns and the architecture choices. It is important to note that the architectural choices may refer to a general architecture structure or a single component; the real purpose of the table is to provide the reader with an understanding of how the patterns are integrated and how our platform supports them. The rationale for this is that interaction-design patterns are abstract in terms of particular architectural decisions, therefore they might be supported intrinsically in the entire platform architecture. Another significant observation is that the requirements found for the framework platform from interaction-design patterns want to achieve the goals of these ones but are affected by technological limitations or implementation choices; to better understand what form of specific features are present in the framework the reader could refer to Chapter 6 and Chapter 7.

Interaction-design patterns	Conversational Framework Architecture choices
Shaping-up the map of the navigable space	CNT implementation is used to be the model of the website (Chapter 4) and create a mental map for the user. CNT nodes reflect website content in order to be available through navigation on the CNT.
Navigating through intelligible and quick mechanisms	The CNT structure represents navigation through its nodes, made possible by the Navigation Bot (see Section 6.2.3). In addition, the Link Bot makes it possible navigating through Web pages (see Section 6.2.6).
Summarizing and segmenting the page content	The CNT structure is made of numerous nodes which provide content segmentation in order to have a general view of different contents in a wide context. In addition, the Navigation Bot (see Section 6.2.3) provides description of nodes to let the user know in advance the content of a section; further works aim at retrieving the description automatically (future works are explained in Section 8.4).
Providing access to conversation-scaffolding intents	Scaffolding intents are managed by the Scaffolding Intents Bot (SIB) explained in Section 6.2.4. The SIB could be triggered at any moment by the interpretation of a user request by the NLU model (see Section 5).

Table 6.1: Interaction-design pattern Integration

7 | Framework Implementation and Evaluation

This chapter dives into the specifics of the ConWeb framework's implementation. First, we show a general overview of the technologies used, subsequently we illustrate the entire implementation as an architecture with different goals than the one described in Chapter 3. Following that, we go through all of the main components of the provided view in depth, resuming all of the functionalities discussed in the preceding sections. Finally, we describe the implementation evaluation through our users study with people with visual impairments after assessing our framework from a technical perspective.

7.1. Implementation

As already described in Chapter 3, the ConWeb platform is organized along different components which use different technologies:

- Languages:
  - `Python`[1] is used to develop the main logic of the ConWeb framework.
  - `JavaScript` and `HTML` are used to develop the front-end client.
- `NodeJS`[2] is used to extend the front-end client on being deployed on server-side together with supporting multiple and different connections by various users.
- `Rasa`[3] is used as NLU to parse user utterance and extract information from it, being an open-source Conversational AI giving us the information we need for our implementation.
- `Selenium`[4] is used as Web APIs to simulate browsing through a Headless Browser and retrieve content from Web page elements. Selenium APIs are properly supported

[1]https://www.python.org/
[2]https://nodejs.org/it/
[3]https://rasa.com/open-source/
[4]https://www.selenium.dev/

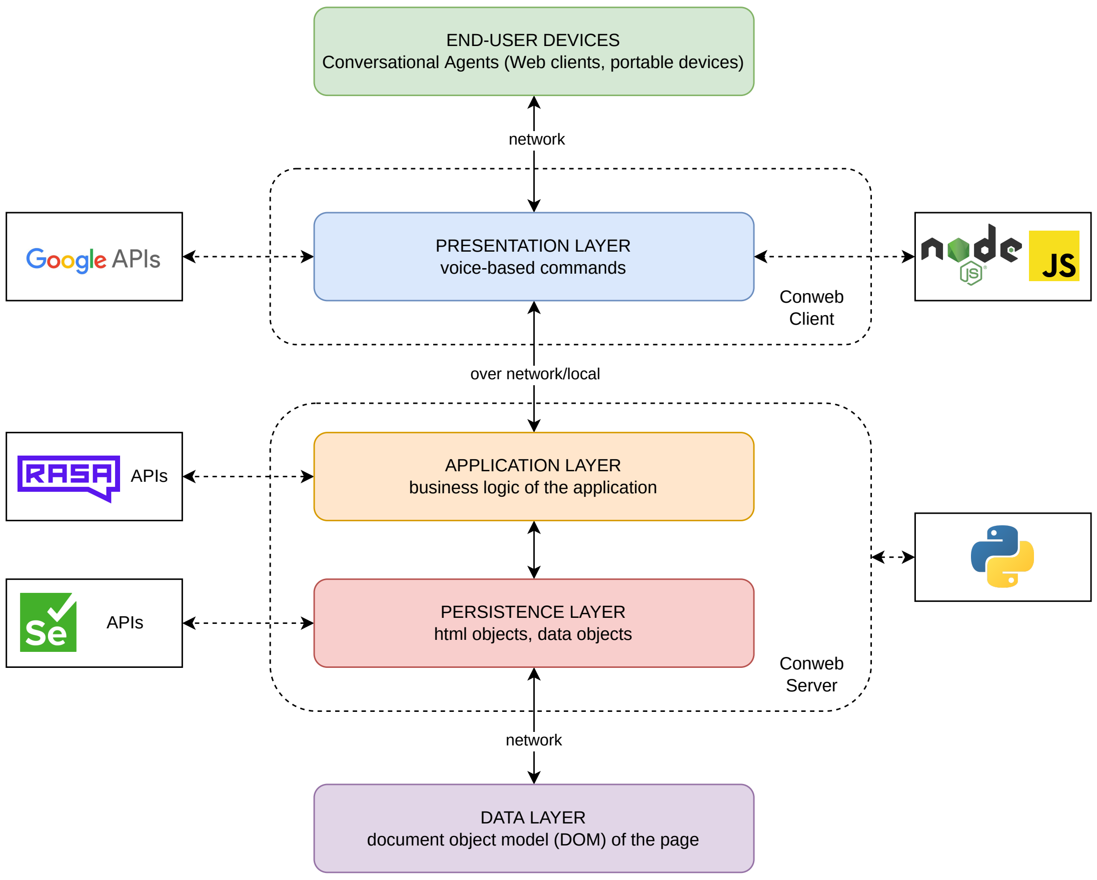


Figure 7.1: Layered architecture

in Python and therefore useful in our context.

- `Google Speech Recognition/Synthesis APIs` are used to translate voice into text and text into voice on client side. They result in being the most audible and accurate speech recognition/synthesis.
- `Linux Ubuntu 20.04 Server` is used to test the framework and deploy it in order to be always available.

Figure 7.1 presents a layered organization of the platform architecture, showing how the different technologies served the implementation of the different layers in the application stack.

The Presentation Layer provides the ConWeb Client from which the user can communicate with the ConWeb Server, whereas end-user devices are used by the final user to access the framework. Instead, the ConWeb Server represents two layers: the Application Layer (all of the framework's business logic) and the Persistent Layer (intermediate representation of HTML objects). The Data Layer, which represents the document object model (DOM)

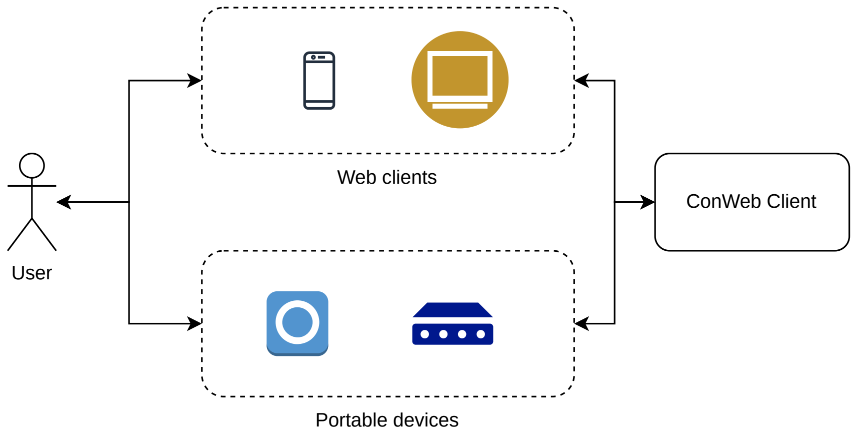


Figure 7.2: End-user devices

of the Web pages visited by the final user, is the last layer. All of these layers are part of the overall behavior of the pipeline started by the final user. We will go through each layer in detail in the following to see how the framework is built and which components are needed to manage its features.

The source code of the Conversational Web Framework could be found in the following:

- ConWeb Client: `https://github.com/spada397/conweb-client`
- ConWeb Server: `https://github.com/spada397/conweb-server`

7.1.1. End-user devices

The first layer, as previously stated, represents the end-user devices for accessing ConWeb services. To address all possible eventualities, several ways of accessing these services and hence the Web must be considered. As a result, we have identified two types of devices as show in Figure 7.2:

- Web clients
- Portable devices

With Web clients, we mean a specific URL that allows access to ConWeb services from any device using any browsers. Instead, by portable devices, we mean a client like Amazon Alexa that can be connected to a certain service (like our ConWeb APIs) and thus utilized anywhere. It is worth noting that mobile devices are included in the Web clients because they can be connected to our specific URL even if they are portable. The first type of devices provides extensive control over our services (the keyboard might be utilized as aid in addition to voice commands, as recommended by users study in Section 7.3),

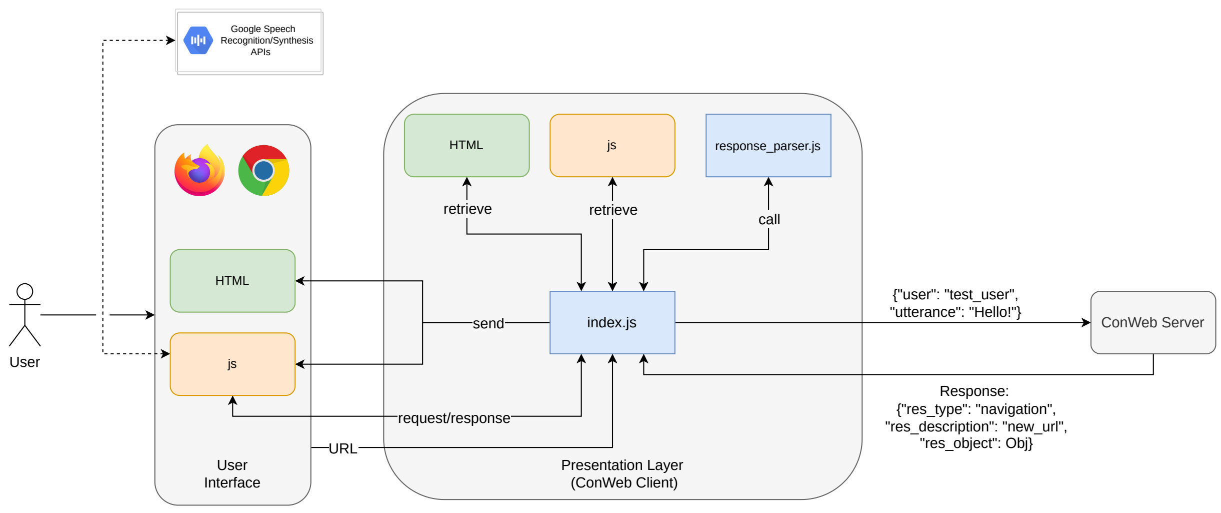

Google Speech Recognition/Synthesis APIs
User
HTML
js
User Interface
URL
HTML
js
response_parser.js
retrieve
retrieve
call
send
index.js
request/response
Presentation Layer (ConWeb Client)
{"user": "test_user", "utterance": "Hello!"}
ConWeb Server
Response: {"res_type": "navigation", "res_description": "new_url", "res_object": Obj}

Figure 7.3: Presentation Layer

whereas a portable device could be beneficial when traveling or being away from a personal computer. For our implementation, we focus on the first type of devices letting users able to use it through Google Chrome and Firefox. It is important to note that our solution includes APIs that could be accessed by any properly setup device in order to support all the mentioned ways of access.

7.1.2. Presentation Layer

As previously stated, the Presentation Layer refers to the ConWeb Client, which is accessible via end-user devices and allows communication with the public APIs exposed by the ConWeb Server. In our implementation, we concentrated on a client available via a specific URL using Google Chrome and Firefox, where the user could use it and explore all of the platform's ConWeb features. The ConWeb Client is implemented in Node.js and includes simple front-end HTML pages as well as some JavaScript to handle external APIs queries. Figure 7.3 illustrates the ConWeb Client's detailed structure. From a end-user device, the user could request a specific URL to access the services, and the ConWeb Client will retrieve the necessary files (HTML and JavaScript files) and transmit them to the interface via its core logic (index.js). The user will be able to utilize a client with voice commands and keyboard assistance in this manner; a thorough explanation of these functionalities can be found in Section 7.3. The JavaScript files will be able to record user voice and, using Google Speech Recognition APIs, translate it into text before sending it to the ConWeb Client. The ConWeb Client will then send the request to the ConWeb Server in the form of a JSON Object with two fields: `user` and `utterance`. The term `user` refers to the current session and user of ConWeb (useful for dealing with concurrency

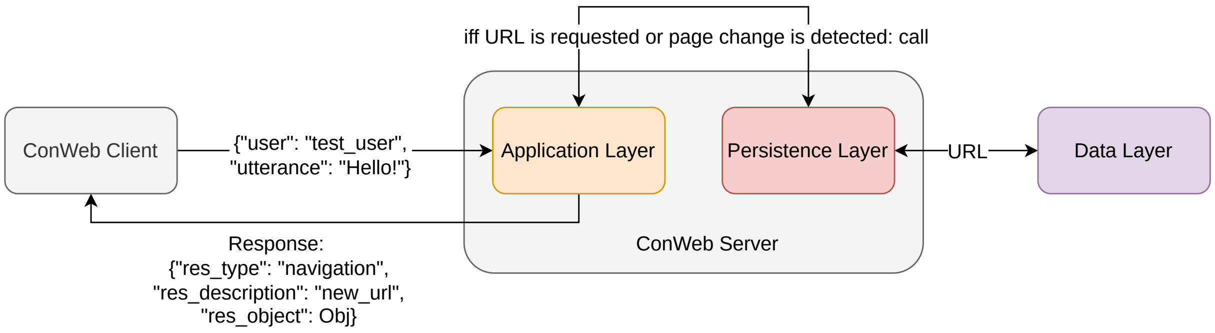


Figure 7.4: Application and Persistent Layers

requests), whereas `utterance` refers to the text recognition of the user request. After the ConWeb Server has parsed and analyzed the user request, a Response Object is returned, which will be parsed by `response_parser.js`. The Response Object elements (which are res_type, res_description and res_object) are required to allow the parser to reconstruct the response in natural language for being understood by the user. The final response is then provided to the end-user device, where it is translated into voice using Google Speech Synthesis APIs.

7.1.3. Application and Persistence Layers

This section introduces the Application Layer and the Persistent Layer, both of which are components of the ConWeb Server. We will first demonstrate how the two layers are structured within the ConWeb Server, and then we will go into detail of each of them. As shown in Figure 7.4 when the Application Layer receives a user request to navigate the Web on a new URL, the Persistence Layer is called, which retrieves the HTML page requested by the user from the Data Layer. The Persistence Layer will then handle all of the data inside proper structures before returning them to the Application Layer, which will handle them and return the required content to the user. In this manner, if users make a subsequent request that is related to the page they are navigating, the Persistence Layer will not be called since the Application Layer will have all of the necessary structured data for that page, allowing it to generate the answer directly for the users. In practice, the Persistence Layer is called only when a new page has to be parsed or a change in the page is detected; the Application Layer has all of the page's content when the Persistence Layer returns structured data (CNT). In the following sections, we will dive into the specifics of the two layers in order to comprehend their main functions.

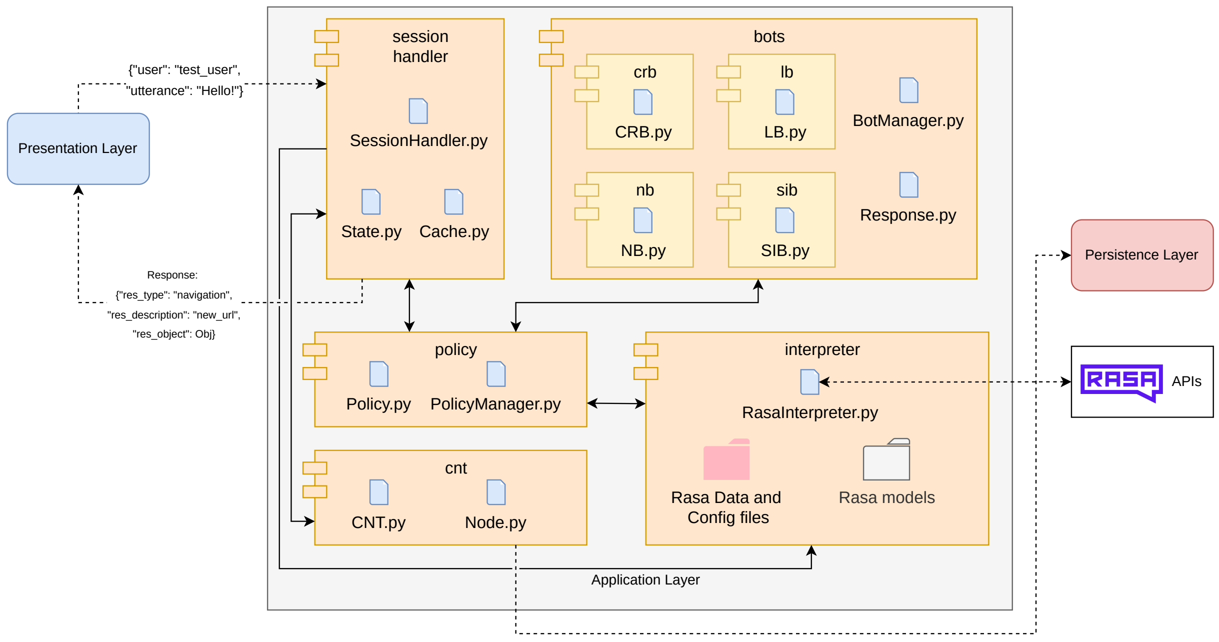


Figure 7.5: Application Layer

Application Layer

In this section we dive into the detail of the Application Layer which is shown in Figure 7.5 in a detailed architecture view. As previously stated, the Application Layer is located between the Presentation Layer and the Persistence Layer (which is only called when necessary) and is a component of the ConWeb Server. It is specifically developed in Python 3.8 and includes many modules, which will be discussed in this section following the real pipeline started upon the return of the user request.

Session Handler Module This module is in charge of managing the current user session and its navigation context. In particular, `SessionHandler.py`, upon receiving a user utterance, determines whether it is a request for a specific URL (opening a new website) or a request within the current page; this check is made by a simple control on specific words used by the user and its current navigation context; no NLU is used to perform this type of check because it is assumed the framework is used to first open a website and then navigate it. The `cnt` module is called if the request is to open a new URL; otherwise, the `policy` module is invoked to determine which action to do if the request is within a given URL. If the `cnt` module is called, subsequently is called the `interpreter` module in order to train the Interpreter for the new URL based on the extracted keys from the CNT. `State.py` is a particular class that is used to track the user's navigation context (current node, current URL, last action performed, and so on), whereas `Cache.py` contains some helpful data that may be utilized to speed up various actions or processes.

CNT Module When a new URL is passed to the `session handler`, it calls the `cnt` module, which is in charge of constructing a new CNT for the newly visited page. To accomplish this behaviour, it invokes the Persistence Layer to extract information from the actual Web page and then generate the CNT following the already presented Algorithm 4.1. In particular, to retrieve Web page elements and simulate navigation, the Persistence Layer is called using also Selenium APIs in an Headless Browser. `CNT.py` represents the CNT's class, whereas `Node.py` represents the CNT's node structure.

Policy Module When a new utterance has to be parsed and a correct action needs to be made on the current Web page, the `policy` module is called. To begin, it calls the `interpreter` module, passing the utterance to be parsed; in this way, the `policy` module will have all the intents and entities necessary to understand the user's goal. Following that, it calls the `bots` module arbitrarily in order to conduct the appropriate action and obtain the answer to forward to the `session handler` and then to the Presentation Layer. It is important to note that the correct call to the `bots` module is determined by the intent and entities collected, as well as a custom algorithm already briefly described in Chapter 6. The given algorithm is specified in `Policy.py` and is resumed in Section 7.1.3. `PolicyManager.py`, on the other hand, manages different or other policy files; this way, for future updates, if a new policy with different behaviors needs to be implemented, it can be done easily; using entities and intents, one can arbitrarily decide which action or call to the `bots` module perform.

Policy algorithm In this section, we will go through how the `policy` module decides which action to take given all of the data. After passing the utterance to the `interpreter` module, the list of intents with their confidence and entities is returned. As shown in Algorithm 7.1, we extract the intent with the highest confidence and check to see if it is one of the `read_links` or `read_paragraphs` types. In addition, we check if the user's current position is a leaf node of the type `content_reading` (i.e. last used bot is an instance of CRB). If both checks pass, the user most likely asked for one of the two intents inside a leaf node of type `content_reading`, thus we call the CRB. If not, we check how many entities have been extracted: if at least one entity has been extracted, the user most likely requested a specific content or text to be read, so we first check if the entities extracted are of a CRB (first paragraph, second paragraph,...) and if successful, we call the CRB. If entities are not of a CRB, they must be of a CNT for construction; thus, we search for the specific Node (Algorithm 6.2) using the extracted entities and then call the Bot Manager (explained in Section 6.2.2). If, on the other hand, no entities are detected, it is likely that the user requested a default action, in this case the SIB is called if the confidence of the extracted intent is above a given threshold (0.4 in our implementation). In all other circumstances, the user receives a `not_understand` response. It is important to note that the threshold, no matter how low, is intended to exclude intents that are most likely incorrect; additionally, with our Rasa implementation, all intents are classified with a high confidence (0.9 in most cases) and a very low confidence for others (0.01); as a result, we do not handle intents other than the one with the highest confidence and the value 0.4 wants to include misunderstandings with an high confidence in rare cases.

Algorithm 7.1 handle_utterance()

```
Data: utterance
Result: call to the proper Intent Handler
 1: parsed_utterance = rasa.parse(utterance)
 2: selected_output = parsed_utterance[0] /* the output is an array ordered by
    confidence*/
 3: if $selected\_output.intent \in [read\_links, read\_paragraphs]$ and $last\_used\_bot ==$
    $isInstance(CRB)$ then
 4:     CRB call
 5: else
 6:     if $|selected\_output.entities| > 0$ then
 7:         if $|selected\_output.entities| \in CRB\_entities$ then
 8:             CRB call
 9:         else
10:             new_node = search_node_by_entities(
                current_node, selected_output.entities)
11:             BOT MANAGER call
12:         end if
13:     else
14:         if $selected\_output.confidence > 0.4$ then
15:             SIB call
16:         end if
17:     end if
18: end if
19: return not_understand
```

To conclude this digression on the `policy` module, we note out that the current implementation with its algorithm is arbitrary and various other implementations might be done and have been attempted before arriving to the one shown in this chapter.

Interpreter module Another important module is the `interpreter` one, where the user's utterance is passed to be parsed by Rasa APIs. When the user visits a new page, via a call from the `session handler` module, the Interpreter for that URL is already trained. Rasa Configuration files (which specific Rasa Pipeline to utilize by the NLU) and Rasa Data files (intents with examples of utterances with all possible entities, taken from the CNT) are used to train the new Interpreter. `RasaInterpeter.py` represents the Interpreter class, whose real model is saved in a separate directory. When a user's utterance is passed, the appropriate model (depending on the current user and the URL) is selected and utilized to parse the sentence and return the result in the form of `intents`, `entities` and `confidence`. It is important to note that each Interpreter is saved in `Cache.py` and can be reused even if the user exits the current page (if the page changes content, the Interpreter need to be retrained).

Bots module This module is in charge of carrying out the appropriate action depending on the logic of the `policy` module. In specifically, the `policy` module will invoke the appropriate bot module (one of the `crb`, `lb`, `nb`, or `sib` modules, which represent the relevant bots) or `BotManager.py`, which will be in charge of picking the appropriate bot module if the user has just arrived at a new node. As mentioned in Chapter 6, each bot will handle different cases, and after completing the desired action, a response will be prepared and returned to the `policy`, which will transmit it to the `session handler` module and, finally, to the Presentation Layer. `Response.py` is used to model the response in order to have a correct structure that the client can understand (the response will contain fields like res_type, res_description, res_object and so on) and to change the way the agent will respond based on the needs without changing the logic of the `bots` module.

All of the modules in this part form the primary framework logic and pipeline, with the `interpreter` module utilizing Rasa APIs. It is crucial to note that other interactions between all the modules exist and are not stated for simplicity; the goal of the implementation description is to understand the modules' aims and their major interaction in order to comprehend the ConWeb business logic.

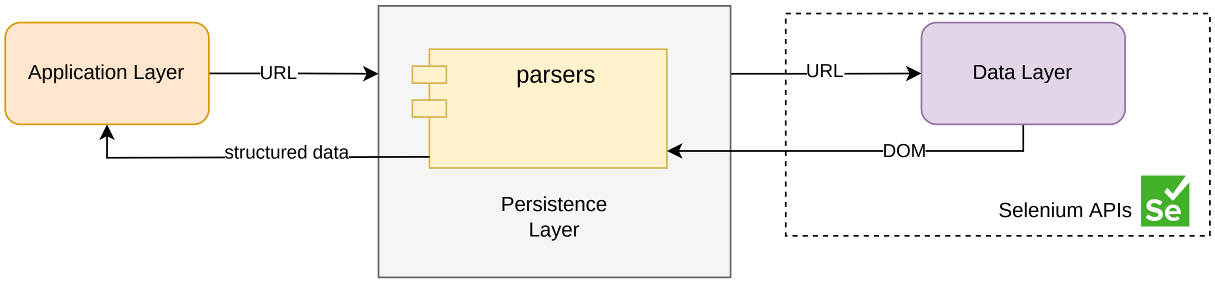


Figure 7.6: Persistence Layer

Persistence Layer

The Persistence Layer is another important component of the ConWeb architecture because it is in charge of processing data from the Data Layer. As shown in Figure 7.6, when necessary (see Section 7.1.3), a URL is sent to the Persistence Layer (specifically, to the `parsers` module), which is in charge of getting data from the Data Layer using an Headless Browser and Selenium APIs. Subsequently, the `parsers` module is responsible of transforming the data into proper structures so that the framework can understand them and build the CNT mentioned in the previous sections. In particular, part of the Algorithm 4.1 uses Selenium APIs. It is crucial to note that the Persistence Layer could also be used to handle navigation, namely the opening of new links.

7.1.4. Data Layer

The Data Layer is the last layer of the ConWeb architecture, and it represents the data (DOM) incoming from the Web. From the Persistence Layer, using an Headless Browser, it is possible to mimic Web page browsing and retrieve elements from loaded URLs with specified Selenium APIs (for example, `find_elements_by_xpath()`, referenced in Algorithm 4.2 as `retrieveAllChildrenElements()`, helpful for retrieving elements on the Web page based on the HTML attributes they contain). When the page elements are retrieved, the procedure is completed while keeping the hierarchy of these elements in order to allow the framework to have built the CNT.

7.2. Technical evaluation

We performed a technical testing to measure its response time to ensure that it met performance and usability requirements. In particular, long wait times for training Rasa Models occur during the deployment and testing of the Conversational Web Framework. The reason for this delay is that the deployment is done on a virtual machine with only the CPU enabled for processing and no GPU available, which should be present in order to minimize the waiting time for Rasa APIs. Table 7.1 illustrates our recently found performance metrics. When a new page is loaded, a new training must be performed, for this reason we divided the metrics distinguishing the cases. It is important to note that the high response time is present only during the opening of a new page and could be minimized changing hardware configurations. We also added metrics based on pages size since they are relevant for the response time. It is important to notice that with Response Time we mean the time from which the final user finish talking to the time when the assistant responses back and has been obtained as an average value from several executions of the framework upon different inputs (in particular, 50 executions for Response Time within a page and 10 executions for Response Time on new page).

Configuration	Response Time	Response Time on new page (with Cache)
CPU: Intel(R) Xeon(R) Silver 4116 2.10GHz, 6 physical cores. RAM: 16GB GPU: no Medium test page size: ∼250KB	32.2ms	∼30sec (9.7sec)
CPU: Intel(R) Core(TM) i7-10875H 2.30GHz, 8 physical cores. RAM: 32GB GPU: no Medium test page size: ∼250KB	30.1ms	∼15sec (8.9sec)

Table 7.1: Response Time metrics

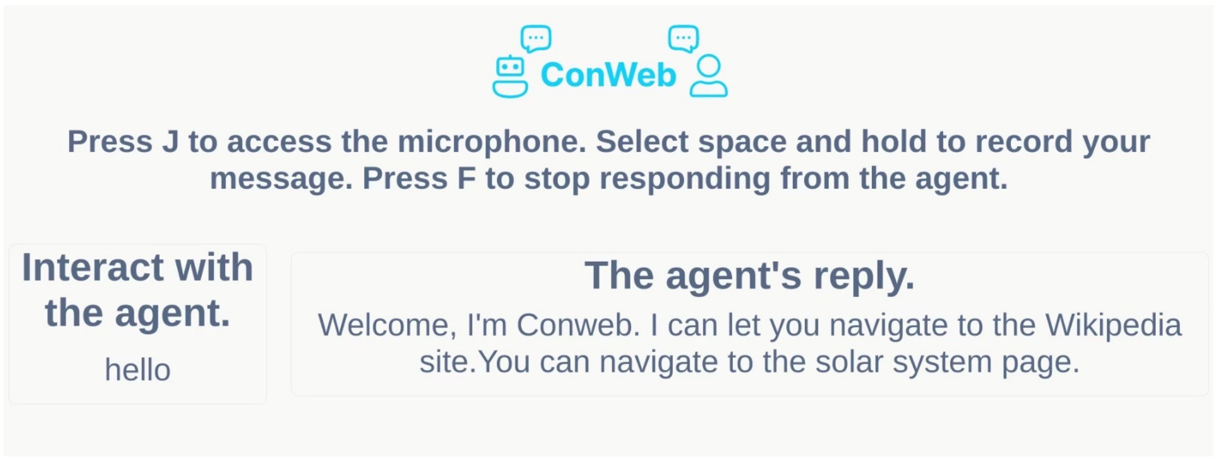
As can be seen, in a real implementation for wide use, including multiple GPUs and a proper configuration, the waiting time (in particular when loading new pages) could be acceptable and in range of few seconds.

7.3. User-based evaluation

This section explains how we tested our framework with a sample of blind and visually impaired users.

7.3.1. Study Set-up

The ConWeb framework is deployed on virtual machines at Politecnico di Milano in order to facilitate remote testing, and we used Web browsers: Google Chrome and Firefox as


ConWeb

Press J to access the microphone. Select space and hold to record your message. Press F to stop responding from the agent.

Interact with the agent.

hello

The agent's reply.

Welcome, I'm Conweb. I can let you navigate to the Wikipedia site.You can navigate to the solar system page.

Figure 7.7: ConWeb Client Interface

end-user devices. Figure 7.7 illustrates the interface we utilized for the ConWeb client, which is accessible via a specific URL. The use of large fonts and particular colors is intended to assist partially sighted people in using the assistant. To communicate with the ConWeb server, the final user must first connect to it by pressing the `j` key and then holding `space` so that it could register and transmit its message. Users can stop the assistant from talking by pressing the `f` key, and by pressing it again, the assistant will resume where it left off. The `f` and `j` keys are chosen because they are the primary keys with a little relief, essential for blind or partially sighted people, allowing them to rapidly find the relevant keys and utilize the client in a proper and easy way.

7.3.2. Results

Using the demo described in the previous section, we conducted users study with four distinct users, the results of which are presented in the following. It is important to note that our framework structure and architecture are also the result of the users study mentioned in this section and it is thus required for proper development.

Camillo, who has been blind since birth, was the first user to test our framework. He has been fascinated by our work and gave us many different recommendations on how to better develop our framework and continue our work. He advised us to register and send the message by pressing and releasing the `space` key. He also suggested that we use certain keys (`j` and `k`) that provide a little relief for blind people. In addition to this, with him, we ran into some issues with microphone permissions on its device, authorized via a pop-up not noticed by screen readers. We spent a significant amount of time resolving this issue, which demonstrated how screen readers can encounter issues in everyday use. In fact, pop-up authorization is required to access the client microphone (as a design and security measure), hence we must use browser APIs (in particular Firefox ones). The pop-up prompted by Firefox (version 97.0) was in an upper layer outside the browser and hence exceedingly difficult for the screen reader to catch, especially for a blind person from birth. With Camillo user study we experienced a real issue with screen readers and also we enrich our framework thanks to his suggestions on how utilize alternative keys (the one stated in Section 7.3.1) to make the demo more usable and understandable to the majority of people.

Alessandro, a person who has lost his sight, was the second user to test our framework. He was pleased to assist us with the development and advised that we make the framework (particularly the user interface) more customizable in order to fulfill the needs of various users. In fact, he would rather have an assistant who is comparable to screen readers than one who is extremely distinct from them. Because everyone has different needs and preferences, a customizable assistant with a variety of functions is the ideal method to address the aforementioned issue. The customization might range from the commands to use and interact with the framework, to the functions exposed, and even the way the assistant exposes Web content. To conclude this test, Alessandro assisted us from a more technical perspective in terms of the framework's usability.

Luca, a blind person from birth, and **Chiara**, a partially sighted person, were two students who were eager to assist us and test the framework. They were fascinated by the assistant and its capabilities, recommending that we speed up its voice because they are used to listening to screen readers at a high voice speed. At the same time, they admired the way the assistant proposed the content, particularly in a different way than a screen reader, which is very important for a complex website like Wikipedia (often used by students). In fact, contrary to Alessandro's viewpoint, students prefer an assistant different to a screen reader, and therefore customization is one of our future aims. Students prefer an assistant who exposes content in a new way, complementing our findings on how to develop a mental map of the website to aid navigation.

All three user tests were beneficial to our development and provided us with contrasting results that, in the end, represented the variety of users who could utilize our framework. To summarize the most relevant outcomes after further examination, we could mention the following aspects:

- The framework should fall somewhere between a traditional voice assistant (such as Amazon Alexa, Siri, and others) and a screen reader. In truth, our assistant wishes to aid people who are browsing online sites, but our target users are people who are used to dealing with screen readers. In this sense, people want complete control over what is happening on the website and to be able to traverse it quickly. As a result, we focused our future development on adding assistance features and customizations.

- Our framework must be completely configurable in order to suit the criteria of the preceding point as well as the requirements of other target users (who may use it while driving or cooking). As a result, even if totally customizable, the default configuration of our assistant should be completed in order to be an easily usable device to surf online pages in general. Advanced users, on the other hand, might customize and exploit it in other ways.
- During development, we focused our efforts on creating a framework capable of correctly navigating Web pages using voice commands, and users study results validated our efforts. However, as evidenced by users study, we did not prioritize the construction of the user interface with sufficient aid for visually impaired people. As a result, in a subsequent development, we focused our resources on designing a client that can be used by visually impaired people by following their suggestions (colors and keyboard keys).
- The alternate method of accessing Web page information was well perceived and confirmed the importance of obtaining website content as a priority regardless of the limitations mentioned by different scenarios.

7.3.3. Discussion

The results and conclusions of the users study highlight the necessity for a new method of accessing Web information as well as the importance of considering accessibility when developing Web content. The structure of the Conversational Web Framework has been confirmed by visually impaired people, allowing us to expand it further due to its scalability and maintainability. Our expectation of assisting people in accessing the internet has been thankfully validated not just through users study but also through real-world realities that anyone can encounter on a daily basis (for example helping people with disabilities on browsing Web or using technologies). The development of the Conversational Web Framework gives us a new perspective on Web content and development in order to suit the needs of all types of users, which should be the starting point for any system or application produced with a wide users base in mind.

8 | Conclusion and Future Works

In this chapter, we will conclude our study and analyze its limitations and outcomes. Meanwhile, we want to lay the foundation for future works and novel approaches to the Conversational Web context based on its limitations and known existing challenges.

8.1. Summary and Lessons Learned

Different people (blind, elderly or people who are unable to use hands or eyes at a specific moment) may need to access the Web for different information. Even if it is currently possible thanks to voice assistants or screen readers, the reality is that they can only do it for simple and quick information queries or without proper navigation (browsing) they would perceive as if they are using a mouse or a keyboard; in this sense, part of the navigation is missing for them. As a result, we wish to lay the groundwork for a new way of accessing information on the Web via Conversational Web Browsing by leveraging a software platform capable of navigating the Web via voice commands. We discovered through our evaluation that the need for a platform like the one we built is real and that it could assist a wide range of target users in accessing information. At the same time, we discovered how the Web is vast and that it is not fully supported for accessibility, therefore a significant amount of information cannot be easily accessed. In this way, our platform aims to not only solve the mentioned problems, but also to make people think about the issue of Web accessibility and lay a foundation on how to support it.

8.2. Outputs and Contributions

Our work resulted in a software platform for accessing Web information via voice commands; not only a simple client capable of receiving user voice and translating it into text and conversely, but also a framework capable of retrieving Web page content, understanding user input via an NLU, and simulating navigation through the entire page and different types of content. Our users study highlighted the need of having a new way of accessing information and navigating it through a mental map. We want to contribute to the state of the Art by giving end-users a new way of accessing information and laying the groundwork for accessibility, which is often neglected; always through our evaluation, we discovered how accessibility is not perfectly developed or supported, causing different problems in target users, and also how our voice assistant capable of navigating the Web for the final user could help them in finding information without difficulties.

8.3. Limitations

We evaluated our software platform by users study with three blind and one partially sighted people, while also developing the platform through users research with 26 volunteers [5]. Even if our users study and research provided us with a depth of knowledge and useful considerations that we did not expect, we must keep in mind that a small sample of people does not reflect the entire population of users may be utilize our platform. Furthermore, our users study and development focused on blind or partially sighted people; nevertheless, our target users include elders or people who are unable to use their hands or eyes in a specific situation; hence, even if the platform might be utilized by them, its development and research are biased. Even with the consideration done thus far, we aim to lay the groundwork for a new context of navigation through conversation with different aspects, and every user we interacted with contributed significantly to the work.

The use of annotations to retrieve content from the Web is one of our work's primary limitations. Because Web page data is presented in different ways and Web content is created using different (and sometimes incorrect) guidelines, developing a method for retrieving Web information useful for our framework would have necessitated a separate study that would have taken a significant amount of time and resources. For these reasons, we focused our efforts on testing with the hypothesis of retrieving information from the Web through annotations and demonstrating that we could achieve our objectives using this information. In fact, our framework demonstrates the ability to read content and navigate through pages without having to deal with forms, images, sounds, or even complex websites; in this sense, the developed platform aspires to demonstrate and serves as the foundation for a more advanced framework that can be extended and always supported in order to supporting even more the Web.

8.4. Future Work

The essential factors that are significant for future research and development are highlighted in the previous section. Therefore, we could list the following as future works:

- Improved annotations or automatic data extraction
- Extended bots
- Additional users study on different target users
- Extended customization

The initial goal is to find a technique to support websites without using annotations, eventually with an automatic mechanism to annotate Web sites or extract information. The reason for this is that not only websites pages are diverse from others, but not all websites support accessibility or annotations (or perhaps do not use resources to incorporate them). In this sense, a method for supporting the framework independently from Web pages could considerably increase its usage and coverage. With the same concept, the framework and, in particular, the Intent Handlers (bots) in charge of handling various types of content could be extended to handle other types of information such as explaining pictures, playing music, filling forms, and so on. Moreover, an additional users study to support other target users could enhance the platform so that it can be utilized by a wider range of people and in a variety of situations. To conclude, our users study showed how our framework should be customized to meet users needs. In this sense, another important future work is to make ConWeb customizable in order to be used by a variety of people having different needs and preferences.

Bibliography

[1] A. Abdolrahmani, R. Kuber, and S. M. Branham. "siri talks at you": An empirical investigation of voice-activated personal assistant (vapa) usage by individuals who are blind. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '18, page 249–258, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356503. doi: 10.1145/3234695.3236344. URL `https://doi.org/10.1145/3234695.3236344`.

[2] A. Abdolrahmani, K. M. Storer, A. R. M. Roy, R. Kuber, and S. M. Branham. Blind leading the sighted: Drawing design insights from blind users towards more productivity-oriented voice interfaces. *ACM Trans. Access. Comput.*, 12(4), jan 2020. ISSN 1936-7228. doi: 10.1145/3368426. URL `https://doi.org/10.1145/3368426`.

[3] T. Ammari, J. Kaye, J. Y. Tsai, and F. Bentley. Music, search, and iot: How people (really) use voice assistants. *ACM Trans. Comput.-Hum. Interact.*, 26(3), apr 2019. ISSN 1073-0516. doi: 10.1145/3311956. URL `https://doi.org/10.1145/3311956`.

[4] M. Baez, F. Daniel, and F. Casati. Conversational web interaction: Proposal of a dialog-based natural language interaction paradigm for the web. In *Chatbot Research and Design*, pages 94–110. Springer, 2020. ISBN 978-3-030-39540-7.

[5] M. Baez, C. M. Cutrupi, M. Matera, I. Possaghi, E. Pucci, G. Spadone, C. Cappiello, and A. Pasquale. Exploring Challenges for Conversational Web Browsing with Blind and Visually Impaired Users. In *CHI'22 Extended Abstracts*. ACM, 2022. ISBN 978-1-4503-9156-6/22/04.

[6] M. Baez, C. M. Cutrupi, M. Matera, I. Possaghi, E. Pucci, G. Spadone, C. Cappiello, and A. Pasquale. Supporting Natural Language Interaction with the Web. In *Proc. of ICWE 2022 (in print)*. Springer, 2022.

[7] Y. Borodin, J. P. Bigham, G. Dausch, and I. V. Ramakrishnan. More than meets the eye: A survey of screen-reader browsing strategies. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, W4A '10, New York, NY, USA, 2010. Association for Computing Machinery. ISBN

9781450300452. doi: 10.1145/1805986.1806005. URL `https://doi.org/10.1145/1805986.1806005`.

[8] J. Cambre, A. C. Williams, A. Razi, I. Bicking, A. Wallin, J. Tsai, C. Kulkarni, and J. Kaye. Firefox voice: An open and extensible voice assistant built upon the web. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445409. URL `https://doi.org/10.1145/3411764.3445409`.

[9] P. Chittò, M. Baez, F. Daniel, and B. Benatallah. Automatic generation of chatbots for conversational web browsing. In *Proc. of ER'20*, pages 239–249. Springer, 2020.

[10] E. Corbett and A. Weber. What can i say? addressing user experience challenges of a mobile voice user interface for accessibility. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '16, page 72–82, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450344081. doi: 10.1145/2935334.2935386. URL `https://doi.org/10.1145/2935334.2935386`.

[11] B. R. Cowan, N. Pantidi, D. Coyle, K. Morrissey, P. Clarke, S. Al-Shehri, D. Earley, and N. Bandeira. "what can i help you with?": Infrequent users' experiences of intelligent personal assistants. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350754. doi: 10.1145/3098279.3098539. URL `https://doi.org/10.1145/3098279.3098539`.

[12] M. H. Fischer, G. Campagna, E. Choi, and M. S. Lam. DIY assistant: a multi-modal end-user programmable virtual assistant. In *PLDI '21*, pages 312–327. ACM, 2021.

[13] Forbes. How website accessibility affects online businesses in 2019 and how to respond, 2019. URL `https://www.forbes.com/sites/ryanrobinson/2019/09/25/website-accessibility-online-business/?sh=73d1e8f79c19`.

[14] Google. Google app voice search insights, 2016. URL `https://thinkwithgoogle.com/consumer-insights/consumer-trends/google-app-voice-search`.

[15] I. Guy. Searching by talking: Analysis of voice queries on mobile web search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 35–44, New York, NY,

USA, 2016. Association for Computing Machinery. ISBN 9781450340694. doi: 10.1145/2911451.2911525. URL `https://doi.org/10.1145/2911451.2911525`.

[16] G. W. Index. Voice search: A deep-dive into consumer uptake of the voice assistant technology, 2018. URL `https://www.globalwebindex.com/reports/voice-search-report`.

[17] J. Lau, B. Zimmerman, and F. Schaub. Alexa, are you listening? privacy perceptions, concerns and privacy-seeking behaviors with smart speakers. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), nov 2018. doi: 10.1145/3274371. URL `https://doi.org/10.1145/3274371`.

[18] T. Lau, J. Cerruti, G. Manzato, M. Bengualid, J. P. Bigham, and J. Nichols. A conversational interface to web automation. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, page 229–238, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302715. doi: 10.1145/1866029.1866067. URL `https://doi.org/10.1145/1866029.1866067`.

[19] G. Leshed, E. M. Haber, T. Matthews, and T. Lau. Coscripter: Automating &amp; sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 1719–1728, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580111. doi: 10.1145/1357054.1357323. URL `https://doi.org/10.1145/1357054.1357323`.

[20] J. U. Mahmud, Y. Borodin, and I. V. Ramakrishnan. Csurf: A context-driven non-visual web-browser. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 31–40, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936547. doi: 10.1145/1242572.1242578. URL `https://doi.org/10.1145/1242572.1242578`.

[21] R. J. Moore and R. Arar. *Conversational UX design: A practitioner's guide to the natural conversation framework.* Morgan & Claypool, 2019.

[22] NPR and E. Research. The smart audio report (winter 2019), 2020. URL `https://www.nationalpublicmedia.com/insights/reports/smart-audio-report/`.

[23] B. Parmanto, R. Ferrydiansyah, A. Saptono, L. Song, I. Sugiantara, and S. Hackett. Access: Accessibility through simplification & summarization. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, pages 18–25, 01 2005. doi: 10.1145/1061811.1061815.

[24] A. Pradhan, K. Mehta, and L. Findlater. *"Accessibility Came by Accident": Use of Voice-Controlled Intelligent Personal Assistants by People with Disabilities*, page 1–13. Association for Computing Machinery, New York, NY, USA, 2018. ISBN 9781450356206. URL `https://doi.org/10.1145/3173574.3174033`.

[25] A. Pradhan, A. Lazar, and L. Findlater. Use of intelligent voice assistants by older adults with low technology use. *ACM Trans. Comput.-Hum. Interact.*, 27(4), sep 2020. ISSN 1073-0516. doi: 10.1145/3373759. URL `https://doi.org/10.1145/3373759`.

[26] G. Ripa, M. Torre, S. Firmenich, and G. Rossi. End-user development of voice user interfaces based on web content. In *IS-EUD 2019*, pages 34–50. Springer, 2019.

[27] S. Schlögl, G. Chollet, M. Garschall, M. Tscheligi, and G. Legouverneur. Exploring voice user interfaces for seniors. In *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '13, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319737. doi: 10.1145/2504335.2504391. URL `https://doi.org/10.1145/2504335.2504391`.

[28] A. Sciuto, A. Saini, J. Forlizzi, and J. I. Hong. "hey alexa, what's up?": A mixed-methods studies of in-home conversational agent usage. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, page 857–868, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351980. doi: 10.1145/3196709.3196772. URL `https://doi.org/10.1145/3196709.3196772`.

[29] R. Semmens, N. Martelaro, P. Kaveti, S. Stent, and W. Ju. Is now a good time? an empirical study of vehicle-driver communication timing. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300867. URL `https://doi.org/10.1145/3290605.3300867`.

[30] A. Sharif, S. S. Chintalapati, J. O. Wobbrock, and K. Reinecke. Understanding screen-reader users' experiences with online data visualizations. In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383066. doi: 10.1145/3441852.3471202. URL `https://doi.org/10.1145/3441852.3471202`.

[31] B. Shneiderman. The limits of speech recognition. *Commun. ACM*, 43(9):63–65, sep

2000. ISSN 0001-0782. doi: 10.1145/348941.348990. URL https://doi.org/10.1145/348941.348990.

[32] A. Springer and H. Cramer. *"Play PRBLMS": Identifying and Correcting Less Accessible Content in Voice Interfaces*, page 1–13. Association for Computing Machinery, New York, NY, USA, 2018. ISBN 9781450356206. URL https://doi.org/10.1145/3173574.3173870.

[33] A. Vtyurina and A. Fourney. *Exploring the Role of Conversational Cues in Guided Task Support with Virtual Assistants*, page 1–7. Association for Computing Machinery, New York, NY, USA, 2018. ISBN 9781450356206. URL https://doi.org/10.1145/3173574.3173782.

[34] WebAIM. The webaim million- an annual accessibility analysis of the top 1,000,000 home pages, 2021. URL https://webaim.org/projects/million/.

[35] Wikipedia. Screen reader— wikipedia, l'enciclopedia libera, 2022. URL https://it.wikipedia.org/wiki/Screen_reader. [Online; controllata il 7-febbraio-2022].

[36] N. Yankelovich, G.-A. Levow, and M. Marx. Designing speechacts: Issues in speech user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, page 369–376, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0201847051. doi: 10.1145/223904.223952. URL https://doi.org/10.1145/223904.223952.

[37] J. Zimmerman. Case for a voice-internet: Voice before conversation. In *Proceedings of the 2nd Conference on Conversational User Interfaces*, CUI '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375443. doi: 10.1145/3405755.3406149. URL https://doi.org/10.1145/3405755.3406149.

A | ConWeb source material and guide

A.1. ConWeb setup guide

This section provides a basic overview of how to install and utilize the Conversational Web Framework.

The ConWeb Server and the ConWeb Client are the two primary components. Assume we are going to install our framework on a certain machine `M` (Linux Ubuntu 20.04 is suggested). We may clone the ConWeb Server repository into a folder of our choice, and then use the `pipenv shell` command to build an environment in which to install the required dependencies. We could use `pipenv install` to install all the dependencies, however each dependency is specified in the `Pip file` if needed. To start our server, we could use the `python server.py` command in the ConWeb Server's root folder. It will listen opening a socket once it is launched.

Following that, we could clone the ConWeb Client to another folder of `M` and use `npm install` to install all the dependencies. Following that, we could launch the client using the `node index.js` command. The client will connect to the ConWeb Server via the socket while also starting its server at `localhost:5050/client`. Each URL will serve a separate interface for the demo, such as `localhost:5050/client/voice`, which was used in our evaluation. It is possible to map the `M` machine server so that at a given URL, `M` will route the request to `localhost:5050` in order to make the client available from another machine. In our case, the URL we used for our evaluation was `conweb.mateine.org/client/voice`, which was mapped to `localhost:5050`.

The example may be used intuitively with the keyboard and voice, but the client could have several configurations or even another interface. Furthermore, this appendix is to guide a future user through the installation and proper usage of the framework; however, it is always possible to change the configurations.

The source code of the Conversational Web Framework could be found in the following:

- ConWeb Client: `https://github.com/spada397/conweb-client`
- ConWeb Server: `https://github.com/spada397/conweb-server`

A.2. ConWeb video-demo

A video-demo has been recorded and made accessible to demonstrate the ConWeb framework's functionalities and features. The following link will take to a video demonstration of ConWeb:

- URL: `http://matera.faculty.polimi.it/wp-content/uploads/Conweb.mp4`

List of Figures

List of Algorithms

List of Tables