Conversational interfaces for service portals have become reasonably common. The colloquial term for the software component that provides this interface is chatbot. Chatbots can either be built from scratch or, developed using one of many platforms available today. In either case, a chatbot usually functions as a part of a larger, Containing system, involving other software components such as Web Servers, Application Servers and Databases. Similar to any other software-intensive system, these systems also have architectural issues. This thesis studies the process of building a chatbot from an architectural perspective, highlighting the various design decisions associated with the process.

The first decision in the design process is to choose a development model. It includes decisions such as picking a development methodology (such as Agile Development) and deciding upon the building tools to use. The chatbot could either be
built in-house from scratch or; existing platforms can be used to aid the development. Figure 1 provides an overview of the available solutions. There are platforms which offer basic NLP services (termed as *NLP-as-a-Service* offering), that can be used to compose chatbot-specific functions. There are also specialised platforms that providing basic building blocks for creating a chatbot (grouped as *Conversation-as-a-Service* solutions). Some platforms also offer ready-to-deploy chat widgets for specific mediums (titled *ChatWidget-as-a-Service* platforms). Any of these platforms may be chosen to aid the chatbot-building process.

This decision, in turn, requires the understanding of the core elements of a typical chatbot and its Containing system. This thesis is focused mainly on platform-aided development. The chatbot-building platforms usually provide higher-level design abstractions to define the use cases for the chatbot, freeing up the developers’ minds from managing complex Natural Language Processing (NLP) tasks. These abstractions also help in building the chatbot in multiple, short iterations, as is the requirement with several IT solutions today. There are, however, some new design decisions which become a part of the development process, solely because of the use
of these platforms. One such decision involves designing a definition template for defining the chatbot over a platform. These templates can significantly affect the behaviour of the built chatbot. Three popular platforms were used for performing the case studies and experiments presented in this thesis - Google Dialogflow, IBM Watson Assistant and Amazon Lex. The relevant Research Questions, along with our contributions towards their solutions, are as follows:

1. **What is the architecture of a typical chatbot? What relationship does it share with its Containing system?**

We begin by providing a dissected view of a chatbot to show its major elements. We present five different elements of a chatbot which are part of a typical chatbot. The *Intent Classifier* classifies any query that the chatbot receives in a class. These classes called the *Intents*, are predefined during the chatbot building phase. The *Parameter Extractor* attempts to locate named entities in the query. Similar to Intents, these parameters, called *Entities*, are defined before the chatbot is placed in operation. The *Flow Manager* element manages the discourse with the user. Its job is to make sure that the conversation sounds coherent. The *Response Generator* performs the tasks required to process the query and prepare a response for the user. It may include replying with static messages or invoking a complex processing pipeline that requires interacting with elements of the Containing system. *Voice Utils* are used when a chatbot supports an audio interface in addition to text messages. They perform the Speech-to-Text and Text-to-Speech conversions.

Next, we discuss how the Containing system of the chatbot may affect its design. The *System Interface* refers to the collection of modes through which the Containing system interacts with its uses. Common interfaces maybe a website, an app or via messaging platforms like Messenger, Telegram, WhatsApp or Slack. The chatbot
may be required to face the users at some or all of these interfaces, which can shape its requirements. Actions are business operations that the chatbot needs to invoke as part of processing a query, e.g. create_new_order function of an e-commerce enterprise. Fulfilments are one or more intermediaries between Actions and the chatbot, and help in shielding the enterprise’s business operations from the chatbot.

We also suggest a Reference Architecture for any application that contains a conversational interface. The Reference Architecture puts the different pieces of chatbots and the Containing system that we discussed before, in a coherent perspective. As Concrete Architectures for the Reference Architecture, we highlight the variations when different chatbot-building platforms are used for the process.

2. How can chatbot-building platforms be evaluated for their effectiveness for a particular chatbot project?

There are several platforms which can help the chatbot-building process in different capacities. We present three categories of commonly available platforms in this area, as shown in Figure 1. First, The NLP-as-a-service platforms provide support for common NLP tasks. They can be used not only for building chatbots but other applications as well (those which process data in Natural Languages). The platforms which we term as Conversation-as-a-service solutions, “blackbox” the details of the background NLP tasks and directly provide features and services which are essential to the working of a chatbot, such as Intent Classification, Parameter Extraction and Flow Management. The platforms which we termed as ChatWidget-as-a-service offerings, often provide a visual editor to create a conversational flow graph and provide a “chat widget” which may be deployed directly on a medium (usually one or more messaging platforms).
In this thesis, we study the Conversation-as-a-service (CaaS) platforms in detail. They provide enough abstractions for the chatbot building process to take away the worries of implementing core NLP tasks while providing the developers with enough flexibility to compose chatbots for a wide range of use cases. As a part of our analysis, we compiled a list of desirable features in these platforms. The list is in coherence with the Reference Architecture that we proposed. As case studies, we also present relative rankings for the support of these features on three different CaaS platforms.

The Hospitality framework is a platform evaluation framework based on Software Architecture Body of Knowledge. It provides a methodology to compare two or more platforms based on their support for achieving project-specific quality goals, such as a Quality Attribute or an Architectural Tactics. We show how this framework can be used to compare candidate platforms for a chatbot project. As a case study, we apply the framework to three platforms for a sample chatbot, assuming that the chatbot needs to achieve high levels of Modifiability, Security & Privacy, Interoperability and Reliability Quality Attributes.

3. How can a chatbot be defined over a platform?

We discuss the process of defining the use cases that a chatbot is supposed to serve, over a CaaS platform. These platforms expect the definition of the use cases in terms of certain pieces of information. In essence, these platforms enforce a pattern on the definition of a chatbot, which we call as the Contextual Reactive pattern. The overall idea of the pattern is to express each use case in terms of a context, i.e. what the user expects the chatbot to do; and a response, i.e. what the chatbot must do when the context is encountered. The context is defined by declaring a set of Intents and Entities, and supplying some Examples of relevant user queries associated with
them. The response can be defined as static messages, or as a processing pipeline involving the execution of external business logic.

4. Does the process of defining a chatbot on a platform involve any design choices?

We present a design choice that arises because of employing the **Contextual Reactive** pattern for chatbot definition. The choice that we term **Intent Sets**, are essentially different instantiations of the pattern for the same set of chatbot use cases. We show that the same chatbot can be defined in multiple ways, and the behaviour of the built chatbot differs significantly for each case. As proof, we present the results of some experiments that we performed as part of a case study of three CaaS platforms. Our results show that the different versions of the chatbot often provide a different response for the same user query.

The contributions in this thesis are summarised in Figure 2.

---

**Figure 2: An overview of the contributions in this thesis**