Trinity College
The University of Dublin

Notification Abstraction: An info-bead modeling approach to personalised notification management

Author:
Kieran Fraser

Supervisor:
Prof. Owen Conlan

A thesis submitted in fulfillment of the requirements for the degree of

M.A.I. in Computer Engineering

to the

University of Dublin, Trinity College

Submitted to the University of Dublin, Trinity College,
May, 2016.

Declaration of Authorship

I, Kieran Fraser, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signed: _____

Date: _____

Summary

With the evolution of the social web and ubiquitous computing, there has been an unprecedented increase in the amount of notifications being pushed at mobile users. Research has shown that the incoming notifications distract the user severely from their current task, regardless of whether the notification is read or not. A significantly high influx in notifications has also been shown to resonate negatively with those receiving the notifications.

In this project, a framework is proposed to manage the incoming notifications of a user by delivering them in a contextually relevant manner. This involves the creation of an intelligent system which attempts to predict whether a user should receive a notification immediately or at some other contextually relevant time. In order to achieve this, both the notification and the user are modeled and the contextual relationship between them is analysed.

For the purposes of this project, the development of the Notification Management System (NMS) was split into three parts, which together form an end-to-end solution: notification capture, notification uplift and delivery simulation. The decision to split the NMS into three parts was to enable the project to focus on the development of the intelligent framework which would be implemented in the NMS. This intelligent framework is comprised of an info-bead model implementing a *Mamdani* Fuzzy Inference System (FIS).

The first application, *NAbsMobile*, was developed for the first part of the project, notification capture. Its main function was to capture real-world notification data which could be later used to test the NMS. It was important that the data being used to test the framework was extracted from real-world scenarios, as this would aid in the feasibility study of the proposed solution. *NAbsMobile* was built as an *Android* application and was deployed on the mobile phones of two volunteers for a significant period of time which enabled a sufficient amount of rich data be captured.

The second application, *NAbsUplift*, was developed for the second part of the project in order to maximize the privacy of the collected data-sets and prepare the data for simulation in the NMS. The main functionality of *NAbsUplift* is comprised of an interface to allow a user to create and update a set of uplift terms for their notifications, a means to extract the notification data from an *SQLite* database and convert it to an easily editable format for uplift, and finally, an interface for tracking a user's personal rankings of the uplift terms, which is later used in the inference mechanism of the NMS. The *NAbsUplift* application enables the owner of the notification data-set to manually uplift their own notifications thus ensuring sensitive data is kept private. The application was developed in *Java* and implemented an *SQLite* database for storing the ranking data, the *Apache POI* library for exporting the notifications to an *Excel* spreadsheet and *JavaFx* for the development of a Graphical User Interface (GUI).

The final application, *NAbsDesktop*, was developed for the final part of the project which was comprised of managing incoming notifications on behalf of the user. This involved simulating incoming notifications using the uplifted notification data-sets provided by the two volunteers and evaluating the results output by the NMS. The development of the *NAbsDesktop* application first involved the creation of a generic info-bead library with which to model the notification and user. It was then necessary to implement a FIS within each uniquely developed info-bead attribute. This was accomplished using *jFuzzyLite*, a *Java* fuzzy logic control library. Once again an *SQLite* database was implemented for persisting data within the application and *Apache POI* was used for importing the uplifted notification data-sets. JavaFx was used for creating a GUI through which simulation results could be viewed. The performance of the NMS in handling the notification data was different for both volunteers suggesting bias in the system. The evaluated results for the author's data-set were quite good with a high percentage of notifications being delivered at user-expected contextually relevant times. In contrast, relatively few of the supervisor's notifications were being effectively managed as many notification delivery times failed to meet user expectations. Three reasons were proposed for the low percentage success rate of the supervisor, the first being an insufficient amount of data available in their *Google Calendar*, the second being that the static ranking system was incapable of effectively modeling a dynamic user, and the third being the static fuzzy membership functions were failing to effectively reflect the user's expected behavior.

It was concluded that the combination of the info-bead modeling approach and fuzzy inference system was effective on the author's data-set, but further research is required in order to enable the framework to scale effectively and manage the notifications of a wider range of users'.

Abstract

Managing the growing influx of notification data being pushed at mobile phone users is an increasing necessity in today's world of social media and ubiquitous computing. The challenge of effectively managing a diverse range of notification types, and analysing their current context with relation to an individual user, is a difficult task.

This dissertation proposes a notification management framework which hopes to contextually deliver notifications at the peak opportune time of the user, thus relieving them of distractions caused by contextually irrelevant notifications.

The proposed design is implemented using an *info-bead modeling approach and fuzzy inference system* and the final solution is evaluated through a comparison between simulated and expected results for two real-world notification data-sets.

This dissertation concludes that the solution is effective when rich data sources are available and the fuzzy inference system is adequately personalised to the user.

Acknowledgements

To begin I would first like to thank my supervisor Owen Conlan for his constant help, guidance and enthusiasm throughout the course of this project. His unwavering support ensured I remained focused and motivated to completion.

I would also like to acknowledge the huge support of my family and friends throughout the year. In particular my parents, Barry and Edel, and my aunt, Therese, without whom success would not have been possible.

Finally, I would like to thank everyone who took the time to discuss and/or proofread my thesis with me. Your contributions were invaluable.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The modern era of technology has expanded increasingly towards smart devices making decisions on behalf of a user (Knijnenburg et al., 2012). This shift in the responsibility of decision making has caused expectations of technology to rise exponentially [*Gartner's Hype Cycles for 2015*]. Consumers now demand services which not only equal human expertise, but also surpass it. *Artificial Intelligence* (AI) has seen a huge surge in popularity in recent times, as AI agents are now mastering and defeating humans in age-old games such as *GO* and *Chess* [Silver et al., 2016]. Recent breakthroughs from massive corporations, such as *Google* and *Facebook*, have inspired developers to once again push the boundaries between man and machine so that tasks which were once impossible to entrust to a computer, now seem achievable. For example, these companies are investing in smart technology in order to give the consumers the exact information they are looking for at a given time, in order to maximize their interest and productivity on their social platforms. This form of personalisation is borne through recommendation systems that analyse patterns in consumer behavior and attempts to anticipate their desired needs. Through interacting with technology such as this on a regular basis, consumers are quickly adjusting, classifying it as normal, and expecting it in every other technological interaction.

One such task that has yet to be fully entrusted to a machine to operate in an intelligent way, is the management of incoming notifications [Fischer et al., 2013]. In a world which is being increasingly dominated by data, how can we expect a machine to identify the information which is contextually useful and necessary for a user at a particular point in time, from that which is superfluous and distracting? Until recently this would have been deemed impossible, as a machine wouldn't have sufficient data about a person to make an informed decision. However, with the development of the *Social Web*, people now pour out their lives onto the internet through social media platforms where their data accumulates and is made readily available for use through open standard API's [Murugesan, 2007]. The task at hand, however, is still a daunting one. With the rapid rise in mobile usage throughout the world, it has become normal for vast amounts of information to be at our fingertips 24 hours a day,

365 days a year. It is not sufficient that this information be available whenever we call on it. It is now being physically "pushed" at us multiple times a day and from multiple sources, commanding our attention and our time. Push notifications from various social media sites, email accounts and other mobile apps, fight continuously for precedence. The problem with this scenario is that it occurs in real-time, with no awareness or empathy for the consumer of the information [Pielot, Church, and Oliveira, 2014]. The point is such that, while the information being sent may be something that the consumer wants to see, it may not be necessary for them to see it at the particular time the notification was "pushed" to them. Intercepting, contextually categorizing, and adapting notification delivery based on current consumer needs, is therefore the key motivating factor behind this project.

1.2 Research Question

The key problem to be found with notification delivery is the context. The nature of notifications are to alert a user to a particular piece of information. A user will have normally subscribed to these notifications and so the user will generally have a vested interest in the information they contain. However, the information is not always of interest to them at the particular time the notification was sent. A person's mood, interests, activities and physical position in the world (to name but a few), are all fluid and dynamic characteristics of a user. This constant change of users needs and desires, and changes in consummation of information, remains fluid day by day, hour by hour, minute by minute.

The notifications being sent to consumers, lack this fluidity. The sender of the notifications dictates *what* is sent to the user and *when* it is sent. While the *what* of the notification is perfectly valid, as the user will have subscribed to the information being sent to them (giving them a certain amount of control over what information they are receiving), the *when* of the notification lacks any dynamic input from the user at all, rendering the system unintelligent. Short of blocking all notifications completely, or choosing a fixed time of day for a particular notification to be sent, a user has very little control over the delivery of notifications. This project aims to provide a solution to this issue by addressing the following question:

How can notifications be effectively managed in a real-world context, such that their delivery is at the peak opportune time of the user?

The **peak opportune time of the user** relates to a number of variables which are constantly changing: For example, the current physical position of the user, their current activity or their current mood. While all these aspects of a user are constantly changing, conversely, the notification remains static. Like the user, it also has a number of characteristics which define, it such as it's subject matter, it's sender and

the time it was sent, but unlike the attributes of the user, these will always remain unchanged once the notification is sent. There will therefore be, over a period of time, a particular moment when the relationship between the user's attributes and the notification's attributes are at the highest possible level of harmony. So take, for example, a user who is currently involved in an activity that correlates directly with an incoming notification. The notification could be delivered to the user straight away. Conversely, a notification's greatest harmonization with a user's attributes may occur at a future point in time, such that the notification would need to be withheld until that point in time - for example, a social message sent to a user while they are in a meeting, can wait until the user breaks for lunch.

There are a number challenges which arise by addressing this question such as the means of gathering sufficient data on the user in order to infer the peak opportune time of the user, verifying the validity of this data as the user's characteristics change over time and even evaluating the ethical implications involved in allowing a machine dictate when a user receives their information.

1.3 Goals and Objectives

The main goal of the project was to develop an end-to-end solution that captures and manages real-world notifications effectively, such that their delivery is at a contextually relevant time for users. This meant developing a framework for modeling user and notification data, as well as a means to combine both models and infer the delivery time of a notification. As the proposed solution is quite broad and expands into many areas, the main goal is split into a number of smaller goals, each with corresponding objectives, which relate to a particular stage in the Notification Management Pipeline (NPM) (figure 1.1).

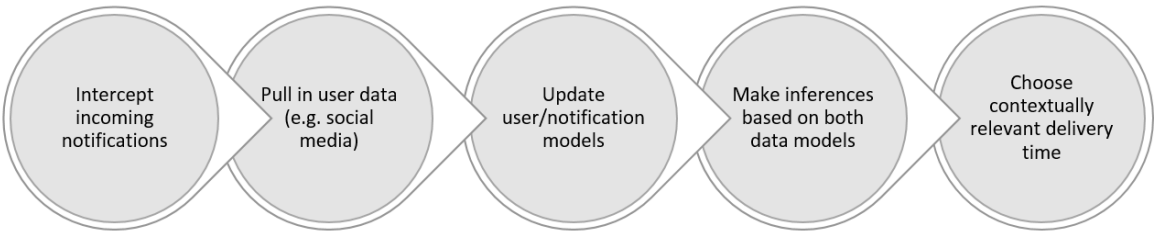


FIGURE 1.1: Notification Management Pipeline

The goals are:

1. **Gather real-world notification data**

   The first goal is the accumulation of notification data. This goal is of pivotal importance, as without a sufficient amount of good quality data to test with, the Notification Management System (NMS) cannot be built. It is also important for

evaluation purposes that the test data used be captured in the real-world so as to accurately simulate real-world conditions for the system to run against. This will effectively test the performance of the system in an environment close to reality. This goal relates to the first step in the NMP whereby the notifications must first be intercepted before they reach the user. The objectives associated with this goal:

- Create a mobile application for capturing notifications.
- Integrate the application onto the mobile phones of a number of test users for a fixed period of time.
- Export the accumulated data from the mobile phones, to an easily accessible and editable format, for development purposes.

2. **Harvest user data from various sources**

   The second goal is the harvesting of user data from various social media outlets. This goal involves identifying useful sources of information that can be used to link a user with a notification, and integrating the relevant API's with user/notification models. The data harvested must be sufficient enough to accurately describe the user's characteristics at a point in time. For example, an activity a user is currently involved in at a point in time could be identifiable from their *Google Calendar*. Their current relationships (family, friends, colleagues) could be identified through their *Facebook* profile. Their current interests or work habits could be derived from *Instagram* and *LinkedIn* accounts. This goal is associated with the second stage of the NMP. The objectives associated with this goal are:

   - Identify relevant social media outlets.
   - Integrate the chosen API's into the Notification Management System (NMS).
   - Select the minimum necessary and relevant information from each social graph and store the information in the NMF for use in the user and notification models.

3. **Develop a user/notification model**

   The third goal is the development of both a user and notification model. The user model describes the user at a particular moment in time. The notification model will describe the incoming notifications that are received. The success of notification delivery is dependent on the accuracy of modeling the user at a particular point in time. This goal encapsulates experimentation into the amount of data necessary to sufficiently model a user for notification management. The question of whether or not the selected social media outlets provide enough relevant data on the user to accurately model them at a particular point in time,

will become more apparent at this stage. This goal is associated with the third stage of the NMP, where the data gathered from social media in the previous stage, is integrated with the developed info-bead model of the user/notification. The info-bead model is the state-of-the-art modeling concept, which is being experimented with throughout this project, and it is through this concept that the user and notification models will be expressed. The objectives associated with this goal are:

- Create a library for the various info-bead model components.
- Develop a framework of interconnected custom info-beads necessary for building the NMS.
- Uplift and classify the notification and social media data where necessary.
- Integrate the sensor input data with the info-beads (social media API's and accumulated notification data)
- Persist the data such that experimentation and testing can be carried out.
- Develop a GUI which can manipulate the beads for ease of testing and visualization.

4. **Develop an inference mechanism to contextually deliver notifications**

   The fourth goal is comprised of the development of an inference mechanism which combines various attributes of both the user and notification models, to infer the importance and relevance of a notification at a particular time. It is also used to determine the next best contextually relevant moment the notification should be delivered. The *Mamdani* Fuzzy Inference System (FIS) is chosen for this task and, as it has never before been combined with the info-bead model for notification management, is part of the novel solution this project attempts to explore. This goal maps to the fourth and fifth stages of the NMP, and is the intelligent part of the framework. The objectives associated with this goal are:

   - Create a heuristic knowledge base for each info-bead component based on its purpose.
   - Map relevant user data to incoming notification data and determine contextual relevance.
   - Integrate a fuzzy inference library to carry out *fuzzification*, *inference*, *composition* and *defuzzification*.
   - Map the dufuzzified result to a contextually relevant time for delivery.

5. **Scale the developed framework to ascertain effectiveness**

The final goal of the project is to scale the NMS by testing it with multiple users in order to evaluate its performance in a real-world environment, and ascertain whether

or not the framework works for users of differing characteristics and differing incoming notification types. The objectives of this goal include:

- Capture real-world notifications from additional users.
- Integrate the new user data from various sources into the info-bead model.
- Uplift user and notification data using the same predefined term set.
- Run simulations with the same inference mechanism but with the new data integrated in the info-bead model to evaluate the performance.

1.4 Overview

So far the main concepts defining the foundations of the project have been discussed, such as the motivation, the underlying research question and the goals/objectives set to be achieved. The following chapters will build on this foundation, a notification management framework, which is to be the novel contribution of this project.

Chapter 2 discusses the state-of-the-art which provides a solid grounding in the current state, progress and performance of similar systems to date. This chapter analyses current modeling methods and explores the info-bead model in depth with a particular focus on previous implementations.

Chapter 3 describes the design methodology behind the various stages of the project. This therefore covers the plan of harvesting and persisting data across the various applications to be built, as well as ensuring the highest standards of ethics and privacy are maintained throughout the process. It also details the functionality necessary for each component of each application in each stage of the project.

Chapter 4 outlines the implementation of the design, and details the technical aspects of the framework in greater depth. The topics discussed in this chapter span from the development of the mobile application capturing notifications, to the final desktop application which is powered by the info-bead model framework and FIS.

Chapter 5 evaluates the results of the NMS, as well as the methodology used to develop the it. This chapter critically analyses the experiments and tests, in order to ascertain whether the system can set high standards in a real world environment, and whether the venture of notification management through the combination of an info-bead model and fuzzy inference engine is possible.

Chapter 6 has some final concluding thoughts on the project as a whole. This chapter discusses the achievements of the project, as well as the limitations and obstacles which occurred. Future work to be explored and, recommendations for doing so, is also discussed.

Chapter 2

State of the Art

At the beginning of this project, a number of design choices regarding technologies and methodologies had to first be made in order to structure a feasible solution for the problems surrounding contextual notification delivery, discussed in the previous chapter. Discovery of these technologies and methodologies was accomplished through identifying applicable trends within the realms of user modeling, artificial intelligence, fuzzy logic and data privacy. This chapter aims to discuss various state-of-the-art approaches to problems similar to those found in this project, through analyzing the technologies and methods applied, as well as the results which they produce.

An important aspect of this project will revolve around modeling data - for instance, the incoming notification and the user to whom the notification is to be delivered. User modeling dates back to the late 1970's, in which user information was stored and contained completely within an application, with little distinction between user-modeling and other general application functions [Kobsa, 2001]. In the early 90's Kobsa authored the term "User Modeling Shell System", sparking the development of reusable user models [Kobsa, 2007]. More recently, agent-based user models are being developed to address situations where more information is needed to make certain inferences on a user (for example in highly adaptive user-models which require large amounts of user specific data), and so collaboration between agents occurs as the need arises in real-time. An example of this would be *I-Help*, a collaborative environment for seeking help among peers, implemented using distributed agents which gather individual models on a user in different contexts [Vassileva, Mccalla, and Greer, 2003]. The design from which *I-Help* was derived, stemmed from the argument that in modern times it is no longer applicable to have a single user model for a user, as it would fail to remain consistent in the current world of distributed computing. This is especially true in the context of this project, as users, subscribed to a number of different notification sources, generally have different user accounts for each source, all of which contribute to a different side of the user and therefore model the user differently under different conditions and in different scenarios. *I-Help* also implemented a distributed architecture (made up of multi-agent "matchmakers"), which enabled the scalability of the system to remain intact, no matter

the additional load. Similarly, the advantages of using a decentralized model for independent functionality such as data gathering, is discussed by Kobsa and Yimam [Yimam and Kobsa, 2000] in their design of DEMOIR[1] [(Yimam-Seid and Kobsa, 2003)], which implements a number of "Expertise Indicator Source Gatherers" which act as independent agents sourcing data in order to "inform" the modeling components of their framework. Also concluded from the design of DEMOIR, is a process of choosing from either a centralized, decentralized or hybrid model architecture, for applications derived through previous experiences with expert recommender systems. The process, involving three steps, enabled a flexible solution with regard to the DEMOIR framework, to become apparent:

- identify system requirements and tasks.
- analyze architectural alternatives to centralized and decentralized approaches.
- specify the central features of the proposed application and identify the flexible solution that accommodates these features.

This process raised questions in the design stage of this project whereby, for the purposes of the simulation software being developed for notification management, a centralized user modeling approach was implemented. The underlying framework being used, the info-bead model, still enables a decentralized expansion to still be possible however. In contrast with the hybrid architecture implemented in the DEMOIR framework, and the distributed architecture implemented in I-Help, many user modeling systems tend to use a standalone centralized approach [Fink and Kobsa, 2000]. Two such examples would be *Personis* [Kay, Kummerfeld, and Lauder, 2002] and CUMULATE [Brusilovsky, Sosnovsky, and Shcherbinina, 2005].

CUMULATE is a user modeling server which stores information regarding student actions generated from an e-learning platform and, through inferences made by agents with access to the data on the server, forms a user "student" model. Every component of *KnowledgeTree*, a distributed e-learning architecture, that has a direct interaction with the student, sends information pertaining to student actions to the server. The server then processes every "event" and sends any inferred information regarding the student model, back to any interactive component wishing to adapt/personalize itself to the user.

For this purpose the server has two protocols - one to receive incoming information in the form of "events" about the user, and one to send information regarding the user to the components who initiated the requests. Uniquely, the CUMULATE server doesn't discard the "event" information once it is processed, but stores it, and makes it available to a number of inference agents who assert various attributes of a user

[1]DEMOIR - Dynamic Expertise Modeling from Organizational Information Resources - framework to develop and test expertise modeling algorithms

- such as their "interests" or "motivation level". This architectural design is quite similar to the design implemented in this project, and is illustrated in figure 2.1.

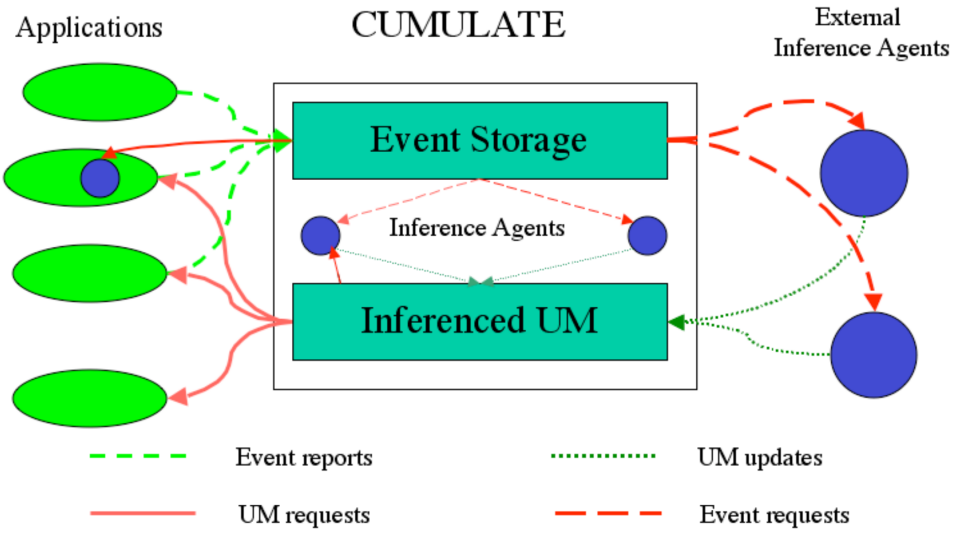


FIGURE 2.1: The structure of the CUMULATE server [Brusilovsky, Sosnovsky, and Shcherbinina, 2005].

User model servers have become increasingly more important as growth in ubiquitous computing occurs and an important factor to consider with this is the privacy of user data and the amount and type of data that is being used in order to model users [Brar and Kay, 2004]. *Personis*, similar to CUMLATE, is a user modeling server. However, *Personis* places an emphasis on the need for transparency in the modeling process. The *Personis* server empowers the user to monitor the data which is being gathered on them, and it also gives them control over it. The processes involved in gathering the data, and the purposes for which it is being used, are also available to the user through the system which strives for complete openness between user and modeling application. An emphasis is also placed on the security of the system, as the user modeling component of any adaptive hypermedia system is arguably, according to Kay and Brar, most vulnerable due to the sensitive "evidence" data which it gathers. *Personis* also aims to act as a central server for all applications requiring personalisation for an individual, such that when a user enters a new application, there is no delay due to there being no information on the user yet available. A *Personal Jazz Channel* was developed to test the concept of the *Personis* server through which users would be recommended various songs, artists and albums depending on their tastes. The key aspect of the developed application was a *scrutability* interface, accessible via a "Profile" button, where the user could view and control the personal data which was being gathered on them.

Hand in hand with increasing the security and privacy of personalisation systems

comes difficulties with maintaining performance. Kobsa and Wang discuss the implications of providing user modeling servers which both provide an optimal level of personalisation to requesting applications and yet still adhere to privacy laws in the various constituencies through which they are accessed [Wang and Kobsa, 2007]. The proposed architecture is one which encapsulates a number of personalisation methods in different components [Wang et al., 2006]. The system chooses the component with a personalisation method which adheres to the current privacy laws in place but also performs the personalisation best for the user out of all other valid components. Figure 2.2 is an illustration of the implemented architecture developed using an architecture-level configuration management system, *ArchStudio*.

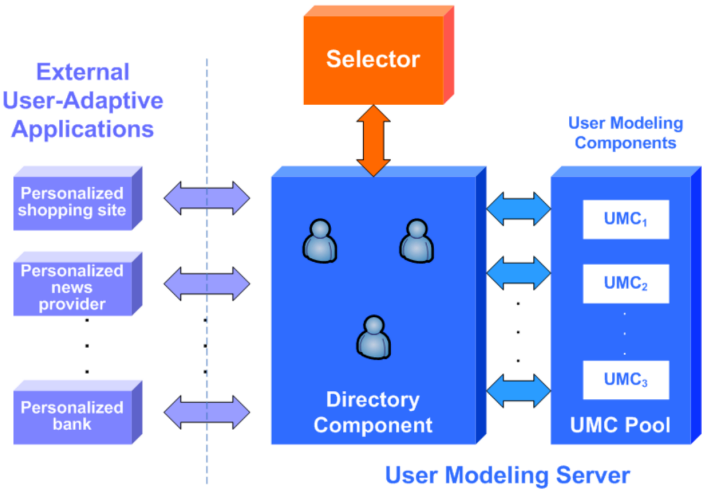


FIGURE 2.2: The dynamic privacy-enabling personalisation infrastructure [Wang et al., 2006].

The external user-adaptive applications in figure 2.2 query the User Modeling Server (UMS) and request personalised user information while also providing updates to the particular user model of the current user. The UMS is made up of a directory component comprised of two systems, Scheduler and Representation, a Selector and a number of User Modeling Components (UMCs). The Scheduler system manages communication between the external application and the UMC. The Representation system tracks the user information passed to the UMS. The Selector makes a decision, based on the privacy constraints of the connected user, which UMC is chosen.

An example of the privacy-enabling user modeling server is found integrated with a social networking website, *UniversalFriends*, where users are offered a personalised list of potential friends located globally around the world [Wang et al., 2006]. As users span different countries and continents, privacy laws differ greatly from user to user. Individuals also may have restraints on what data they wish to offer to the application for use. The privacy-enabling UMS therefore manages the privacy expectations of the users. Different UMC's require different types of data pertaining

to the user, such as demographic information or on-site behavior for example, and also implement different inference mechanisms, such as fuzzy controllers, rule base reasoning and machine learning. Depending on the data required and method of inference used the selector chooses only the UMC which adhere to the particular users privacy constraints and as such the privacy of the user is maintained. Figure 2.3 illustrates the process whereby 3 users from different countries, and with differing privacy constraints, have access to different selections of UMC's from the full pool.

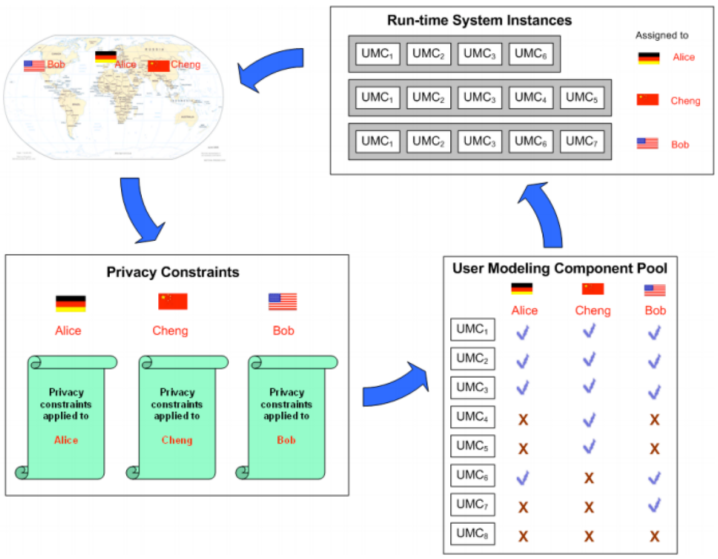


FIGURE 2.3: Privacy-enabling personalisation process [Wang et al., 2006].

So far a number of user modeling applications having been discussed, the key aspects of which revolve around privacy, accessibility, scalability, reusability and the use of inference agents. The info-bead user model, which is implemented in this project as the user model approach of choice, has the flexibility to incorporate all of these aspects, and more, into one coherent modeling system ["User Modeling Criteria and the Info-bead User Modeling Approach"].

The info-bead user modeling approach is a component-based software development approach whereby a user model is made up of atomic elements named info-beads which hold single attribute values of a user [Dim, Kuflik, and Reinhartz-Berger, 2015]. Info-beads can be joined through connections named info-links to form info-pendants which are made up of multiple info-beads connected by info-links to infer a single attribute value held by the root info-bead. Info-links are the protocols used to exchange attribute data between info-beads. The combination of info-beads and info-pendants make up the attributes used to generate generic user models. Combinations of user models, additional info-beads and info-pendants make up group models. In the context of this project, contextual notification delivery, an info-bead could be created to hold a particular attribute of a notification, such as the *Sender* of

the notification. A users current location might also be an attribute held by another info-bead. The location info-bead could be converted to an info-pendant as there are multiple means to determine a users current location. For instance, there could be separate GPS, WiFi and calendar info-beads all of which are connected to the "user location" info-bead to collectively infer the users location. By this means not all the data need be present for the "user location" info-bead to be able to infer the users location (e.g. if the GPS info-bead couldn't get the users location via the GPS sensor for instance, the information provided by the WiFi info-bead would still be enough to infer the users location. Kobsa discusses this aspect of ubiquitous computing, and in particular user modeling in mobile devices, where the model needs to be able to perform with uncertain or partial data [Kobsa, 2007].

Dim, Kuflik and Reinhartz-Berger describe the info-bead to be made up of three parts:

1. Operational

   The operational part of the info bead is responsible for the input and output of "evidence" data. This is achieved through internal interfaces, which other info-beads can invoke to pass information, or through external API's, such as those provided by *Facebook* or *Google* for instance. The operational part of the info-bead also contains an inference mechanism which is used to convert the "evidence" data to an atomic/composite value. The info-beads pass information in the form of *Triplets* which are made up of an id, a detection-time and an information item. The information item encapsulates a number of attributes describing the evidence/inferred data which is being exchanged such as the type, accuracy, confidence value, the units used, the previous evidence it was derived from, the time of inference, the length of time it is valid for and so forth.

2. Metadata

   This part of the info-bead contains information describing the purposes of the info-bead, the data it holds, the developer who built it and varying other attributes which could be harnessed by internal or external artificial agents to determine the use of the info-bead in a specific context and apply it.

3. Control

   The control part of the info-bead is responsible for a number of housekeeping functions such as the activation or deactivation of the info-bead, authorizing access to other info-beads, defining info-link communication settings (comprised of pull, push and notify) and various other maintenance tasks.

The internal logical flow of the info-bead is illustrated in figure 2.4.

An example application of the info-bead user model in practice is also discussed by Dim, Kuflik and Reinhartz-Berger whereby a social behavior analysis application

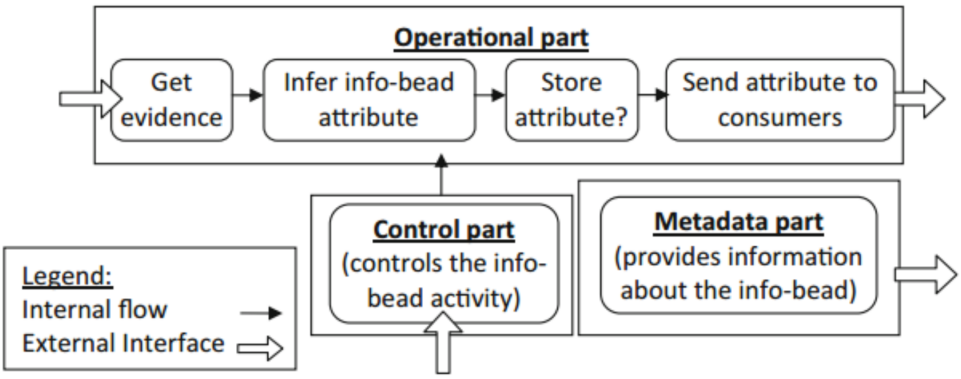


FIGURE 2.4: Internal structure of an info-bead [Dim, Kuflik, and Reinhartz-Berger, 2015].

is implemented in the Hecht archeology museum. This museum is equipped with various proximity sensor technology which the info-bead user model leverages as sensor evidence data in order to infer a particular behavior type about the visitors in the museum [Dim and Kuflik, 2014]. There are three info-beads collecting evidence data from the various sensors located around the museum: the "location" info-bead, "proximity to other visitors" info-bead and "azimuth" info-bead. Data is gathered and inferred in these three info-beads and pushed onward to subsequent info-beads. The location data is pushed to the "TOA" info-bead in order to derive the time of arrival of visitors at certain locations within the museum. The azimuth data is used to determine the visitors orientation at exhibits. All the inferred data is pushed to a final info-bead which infers the behavior type of the visitor(s). The example given is that of a pair of visitors classed as "Geese" (one person of the pair tends to lead the other from exhibit to exhibit). This is inferred using the time of arrival data, as one of the pair will arrive at a location just before the other, and the proximity data, as both visitors will remain in close proximity to each other. Figure 2.5 illustrates the simplified info-bead Group Model (GM) for a pair of visitors classified as "Geese".

Dim and Kuflik also describe the potential of using the info-bead model approach in mobile applications in a ubiquitous environment [Kuflik, Mumblat, and Dim, 2015]. The info-bead model is flexible such that, if info-beads are developed generically, they can be reused depending on the context of the application which wishes to use it. Info-beads can be added and removed depending on the context. They can also be searched by artificial agents due to their rich metadata parts. The inference system within the bead is also very flexible as it supports any number of mechanisms the developer wishes to deploy. Subsequently the info-bead user model approach can easily integrate the privacy-enabling personalisation approach proposed by Kobsa and Wang, previously discussed, by simply removing info-beads from the user model if their inference mechanism or data collection method conflicts with a users privacy constraint.

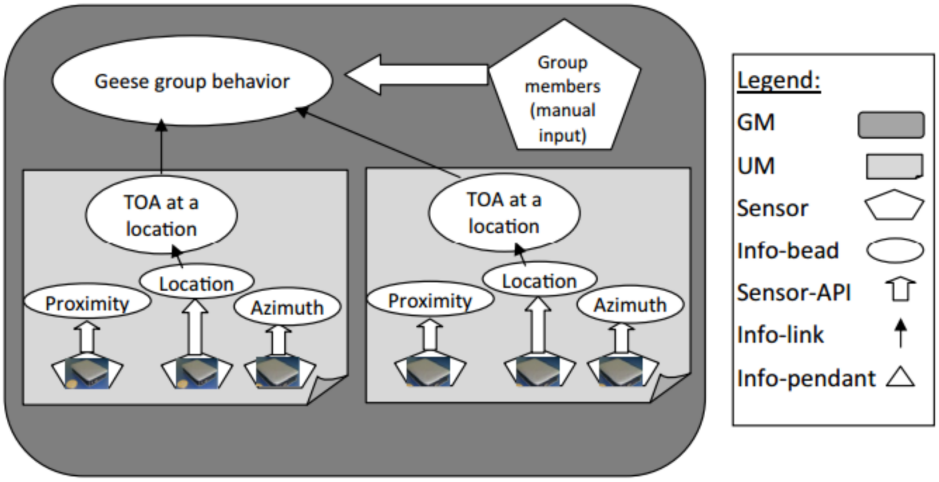


FIGURE 2.5: Group Model for "Geese" classification of visitors [Dim, Kuflik, and Reinhartz-Berger, 2015].

In the context of this project, employing a user model alone won't enable the management of notifications. Intelligence is required if the system is to actively decide whether notifications should be sent to the consumer now or at a contextually relevant time later. The info-bead model approach has been designed such that each info-bead contains an inference mechanism, essentially classing the info-bead model as a distributed artificial intelligent system [Iancu and Popirlan, 2010]. Each info-bead can be viewed as an artificial agent as it can sense various aspects within its environment and also act upon them [Russell and Norvig, 2003]. For instance, a particular info-bead could be deployed in a house to detect temperature levels. The temperature would be sent to the info-bead via "sensor" evidence data. An inference would then be made by the info-bead to determine whether the temperature is above or below "comfortable". The info-bead then might instruct an air-conditioner to speed up or slow down based on the result.

In order to make decisions the info-beads must assert a decision based on the evidence data it receives. Fuzzy logic controllers are one such inference mechanism that may be used as a solution to the problem of notification management. The classification of whether or not a notification is important enough to send to a user is a vague concept. Fuzzy logic deals with managing uncertainty in expert systems [Zadeh, 1983]. Fuzzy expert systems have been most notably been implemented in the medical sector to aid in the diagnosis process of patients [Phuong and Kreinovich, 2001]. Within the realms of medicine the knowledge which is used to diagnose patients is generally uncertain as is the relationship between symptoms and diseases [Vetterlein and Ciabattoni, 2010]. Linguistic terms are used to record the current physical states of patients for example, as they are sometimes difficult to express quantitatively. Consequently, the only data available for diagnosis could be a linguistic variable comprised of: "patient is suffering from a strong abdominal pain". This is

a vague concept as different people have differing views on what constitutes "strong". CADIAG-IV (Computer Assisted Diagnosis) is a medical consultation system which aids internal medicine by implementing fuzzy logic concepts to deal with the uncertain knowledge surrounding medicine [“Patient specific adaptation of medical knowledge in an extended diagnostic and therapeutic consultation system”]. CADIAG-IV uses fuzzy sets and membership functions to transform observed data and test results into linguistic variables. Medical knowledge is then expressed in the form of antecedent-consequent rules in a knowledge base as illustrated in figure 2.6.

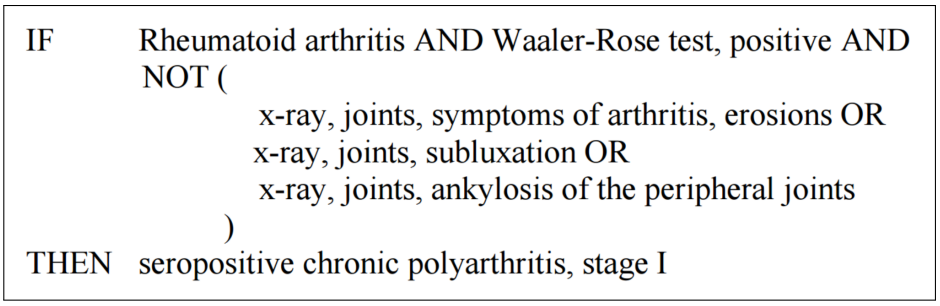


FIGURE 2.6: Example heuristic rule of the knowledge base of the medical consultation system [Dim, Kuflik, and Reinhartz-Berger, 2015].

Subsequently, under the compositional rules of fuzzy inference [Zadeh, 1973], rules in the knowledge base, which were true to a certain degree depending on the vague inputs, are aggregated and a fuzzy number is output. This fuzzy number can then be mapped to a proposed examination to be carried out on a patient. MedFrame [Sageder et al., 1997] is one such application which implements this medical consultation system. The system is also composed of an explanation tool which illustrates how it came to a specific diagnosis.

Fuzzy inference systems can also be used to guide medical practitioners during the treatment of cancer [Saleh, Barakat, and Awad, 2011]. The fuzzy Decision Support System (DSS) suggested by Saleh, Barakat and Awad implements a Mamdani inference method to identify the individual patient risk status in treatments for breast cancer. Six input variables undergo Fuzzification, Inference, Composition and Defuzzification in order to obtain a subsequent output variable representing risk.

In this chapter various state-of-the-art technologies were discussed accompanied by real-world applications of those technologies. Of those discussed, this project aims to implement the info-bead model approach to manage notifications contextually. A Mamdani fuzzy inference system is proposed as the inference mechanism of choice for the info-bead model. This is a novel approach toward notification management and the performance of the proposed framework design discussed in the following chapter is the main focus of this project.

Chapter 3

Design

3.1 Introduction

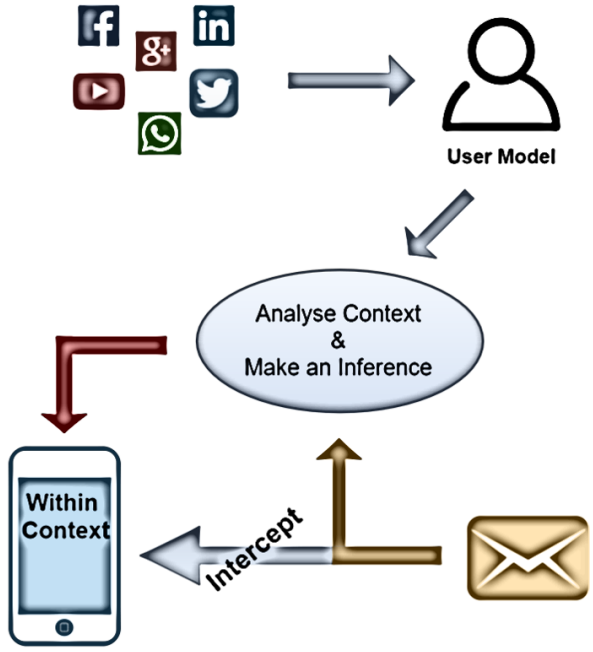


FIGURE 3.1: High level notification management system design.

The main goal of this project is to deliver notifications in a contextually relevant manner, minimize disruption, and maximize transparency and control to ensure the user never feels their privacy is invaded, but also receives notifications at the most contextually relevant moment. There are a number of steps involved in achieving this goal. These steps span from data collection, data uplift and user modeling, to context mapping, inference and notification delivery. The main design element is that of the Notification Management System (NMS), which contains the info-bead model framework that functions as the brain of the system. It also acts as a pipeline which guides the incoming notifications through a specific path which results in a delivery at the right time for the user. The high level design of this framework can be seen in figure 3.1.

3.2 Mobile Application (NAbsMobile)

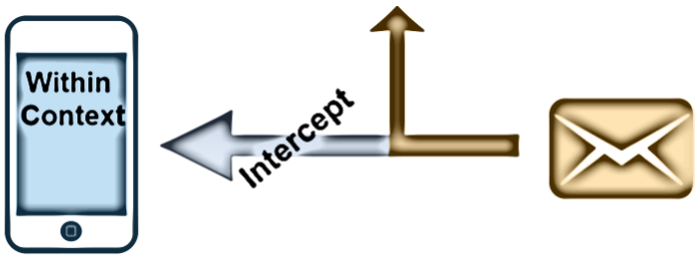


FIGURE 3.2: Notification interception.

3.2.1 Overview

The first step in the design process is to intercept and redirect incoming notifications (figure 3.2). In order to achieve this a mobile application was developed which has access to users incoming notifications as well as permission to read the multiple layers of data it provides. In conjunction with having access to the notification data, user data can also be gathered from the phone at this stage which will later aid in the inference process.

The proposed design of this mobile application doesn't include integration with the underlying NMS. It was decided that in order to keep the privacy of data to a maximum and to ensure completion of the project in full, that it would be better to use the mobile application as a separate entity with which to capture the data necessary to test with, and then develop the intelligent framework in a desktop application where security and control could be exploited to a higher degree. Therefore the data which is collected in the mobile application must be transferred to a desktop environment and plugged into the NMS and subsequent info-bead model framework for the system to be complete. Sending of notifications can then be simulated and subsequently intercepted by the system mimicking real time scenarios. For development purposes, this allows refinements to be made to the model and inference algorithms under stable and repeatable conditions so that performance can be evaluated and improved upon.

3.2.2 App Design

The design of the mobile application (NAbsMobile - **N**otification **Abs**traction Mobile) is as follows:

- NAbsMobile senses an incoming notification.

- The notification details are logged and stored in a database on the phone - this includes various fields which describe the notification: the time and date the notification is received, the sender, the application through which the notification is received, the subject line of the application (named ”ticker text” in the *Android* environment) and the body of the notification message.
- The notification is then allowed to proceed to alert the user. (Note - in a real world scenario, with the NMS integrated onto the phone itself, destroying the notification would be the logical course of action at this point as it can be later recreated and sent to the user at a contextually relevant time. However, in this circumstance, as destroying the notification would result in the user never getting the notification, it is allowed to continue to alert the user).

It is also worth noting at this point, that data such as a user's current location could also be pulled from the phones GPS coordinate's and stored for later use. This would require the user to have their GPS turned on at the time the notification is delivered to the phone, which is not always the case. While the location wasn't logged in this particular project (although it would be recommended for future studies as it provides an additional data source to use in the inference process), it was considered, and it raised an interesting point with regard to the NMS. It must be able to operate even if data is missing. It cannot simply rely on all the data being available all the time as this will vary from user to user. If the info-bead model framework is to be scalable to multiple users (which is the final goal of the project) then it must be flexible enough to deal with missing data. This is also true of the incoming notification data. Each notification is unique, as they are sent at different times, from different people, through different vendors and with different subjects and context. There is no guarantee that the information within the notification is complete. The framework must therefore be built to handle discrepancies which may occur.

The mobile application falls into the first stage of the Notification Management Pipeline which is related to the goal of gathering real-world notification data. Volunteers must be recruited at this stage so that real world notification data can be harvested from them and subsequently used to test whether their notifications can be effectively managed. Two volunteers were used throughout this project, the author and the supervisor. Both subjects had the *NAbsMobile* application installed on their phones for a period of time whereby the application was able to observe all incoming notifications and log the details in a database located on each individual's phone.

3.2.3 Design Ethics

The use of volunteers and the act of harvesting their personal data via notifications raised a number of ethical issues which needed to be addressed before proceeding.

1. Informed consent

This was the initial task to complete before integrating the mobile application on both phones. Both volunteers had to be aware of exactly what data was going to be harvested from their phones and how it was going to be used. Also, the installation of the application itself would require modifying the users' phones in an irreversible way (*Android* debugger mode needed to be switched on), and hence, all this information had to be made clear before proceeding, so as to give the volunteers a fair opportunity to decline participation. As both volunteers have a vested interest in the project and were aware of the risks involved, this step was straightforward, but nonetheless important.

2. Integrity and disruption

   Also to consider before continuing with the *NAbsModile's* installation on the supervisor's phone, was the functionality of the application which resulted in destroyed notifications. The first version of the application (installed and used by the author on their phone) had the functionality of logging and destroying notifications completely (which resulted in an interesting psychological experiment). The supervisor however required the notifications to still get through once their details were logged - hence the functionality of the application had to change slightly. Ethically, it was important that the notifications were not tampered with in any way and were delivered in full to the user in the same manner as though no logging process took place at all. If this was not the case the application could have caused major inconveniences to the volunteer.

3. Security

   The third and final point to consider before installing the application was the security of the users' data being collected and stored on the phone. Ethically, the responsibility of ensuring the users' data remains private sits with the developer - hence it was important to design the application such that the data could not be easily stolen or even viewed through the phone itself (in the unlikely event that the phone was stolen). The application is designed to store the data within the *NAbsMobile* application database on the phone. The application has been designed to have no internet permissions set - hence it cannot communicate via the internet which limits the amounts of online attacks which could occur. It also has no physical sensory permissions, meaning data cannot be transferred via NFC or *Bluetooth*. It has been designed so that the only means of exporting the data is through the application itself via an "Export to Database" button. This exports the data to a database accessible via USB.

3.3 Data Extraction & Uplift

3.3.1 Overview

The next step in the design process is to extract the data collected in step one and integrate it into the NMS. In an ideal world this would be a simple matter of allowing the system access the database exported from the mobile application however, the notification data at this point is in a raw form and so is of little use. Passing the raw information to the inference mechanism would make it extremely difficult to come to a solution regarding the notifications delivery date and time. It would require additional steps of parsing the text of the notification and performing multiple processes of semantic analysis on it, in order to ascertain the meaning behind it. This, as shall be discussed later in Chapter 6, will be a task for the future on this project. For the time being a certain number of assumptions can be made in order to concentrate fully on the end-to-end framework which is being developed and in particular the key aspect of the project, which is the exploration of the info-bead model combined with the fuzzy inference mechanism. By assuming that future work will be done on automating the processes of finding meaning behind the notifications, it allows for notifications to be manually classified by a predefined terminology, in order to both abstract away the sensitive raw data from which they've been derived, but also to give them context and meaning which can be related to the user model derived from social media outlets.

Before carrying out the uplift on the notification data, it first must be extracted from the two individual mobile phones on which it has been gathered. The mobile application was designed to export its data via USB. This means that the database can be stored on a computer. However, it still lacks the functionality to be easily viewed or edited. In order to carry out the uplift, a simple desktop application, *NAbsUplift*, was therefore developed to extract the data from the database and insert it, along with a drop-down of predefined terms, into an *Excel* spreadsheet. Through this application, the uplift process on notifications can be done quickly and with minimum effort and error. The list of predefined terms is also then saved for future reference.

3.3.2 Predefined Terminology

In order to abstract away the sensitive data within the notifications and provide a means to map notification data to user attributes, a predefined set of terms was created and used to uplift the notifications (the full set of terms can be found in Appendix A). This set of terms was created by first studying the gathered notifications, and subsequently categorizing them in general terms, in a manner which described them at a high level and did not give away any personal information pertaining to the user. This process was done individually by the author and supervisor on their own data-set of notifications respectively in order to get an overlapping set of terms

which covered all notifications in both data-sets. It also ensured that the notification data gathered was only worked upon by the owner of the notification data which aided in keeping the intrusion of privacy to a minimum. It was also necessary as the best person to classify the notifications was the owner themselves as they were the only person who knew the full context of the notification and its potential meaning. Of course, if a different person were to classify the notification data, they could come up with differing results, hence a certain amount of bias is present in both data-sets.

As both the author and the supervisor are unique, with different interests, family commitments and age profiles, it is to be expected that the notifications being classified separately will end up having a different terminology. For example, the author, a student, will have "college" down for a majority of notification subjects, while the supervisor, a member of the academic community within the college, won't have a need for college, but will instead have "work". It is necessary to include both terms in the completed terminology, as without both terms a data-set wouldn't be properly uplifted.

3.3.3 NAbsUplift Application Design

The design for the application which carries out the process of uplifting the data-sets is simple. The user is first able to import their notification database (exported from the mobile application, *NAbsMobile*) into the *NAbsUplift* application which parses the data and stores it locally. It then provides an interface for the user to browse their notifications one by one so that the user can accumulate a list of terms that describe their notifications in a general sense. There is then an interface for the user to also add and remove terms from the terminology data-set. The application then has the functionality to convert the notification data into an easily editable form - an *Excel* spreadsheet - along with the list of predefined terms, which the user can set to each notification and hence apply the uplift.

3.3.4 Design Ethics

The uplift application was a necessary aspect of this project as it ensured the data being used by both volunteers remained absolutely private. It also aids in the scalability of the project as, if and when more test subjects are needed, the application can be distributed to the applicants for them to use and prepare additional test data, without sensitive data ever being passed to the developers of the project. This section of the project, enhances the ethical integrity of the project, while also enabling the project to scale to more test users with minimum effort.

Two main ethical questions arose within this section of design:

1. Who should carry out the uplift on the notification data?

   The first question is one which sparked the design of the uplift application. It was argued that the developer of the project couldn't be the person to classify the notifications of the supervisor, as the context surrounding the notifications would be unknown - hence the accuracy of the uplift would be jeopardized. Subsequently, it was agreed that the best person to carry out the uplift would be the owner of the notification data, for they would know the context of each notification best. It would also mean that no other person would view their sensitive data which would keep intrusion on their privacy to a minimum. Once the data would be uplifted, the sensitive data could be erased leaving only the uplifted values behind, which is the only data necessary for the NMS to work. In this manner, no-one but the owner of the data would have access to the sensitive information. However, this method also relies heavily upon the owner of the data to correctly classify and uplift their data. It requires them to be completely honest about the context of the notifications, regardless of the subject matter or adverse knock-on effects it could have. This is a tough ask for volunteers who are not completely invested in the project (as the author and supervisor are). This leads to the second ethical question.

2. How is the uplift integrity to be verified?

   As discussed in the first question, keeping the privacy of the volunteers notification data to a maximum is a high priority. However, in doing so, there is a drop in the accuracy of the uplift process. Ideally there would be a second reader observing the notification uplift process, verifying the accuracy of the results, or giving a different point of view on the uplift (to avoid any bias which may occur when choosing from the predefined term list for example). Having a second reader, one unrelated to the notifications, would breach the privacy of the user however - hence, it is avoided. This means that complete trust is placed in the volunteer to uplift the data to the best of their ability and to the fullness of their knowledge. It is clear that there is a give and take between maximizing the privacy of the user's notifications and maximizing the accuracy of uplifted values chosen.

3.4 Notification Management System (NAbsDesktop)

3.4.1 Overview

As discussed earlier, the NMS is designed as an end-to-end solution. However, some assumptions are made in order to effectively make this so. We are assuming for example that the raw notification data which the NMS is pulling from, will be uplifted. For

this assumption, the *NAbsMobile* application was created for gathering the raw notification data. The *NAbsUplift* application was then created for manually uplifting the raw notification data and preparing it for simulation. While these two elements were created outside of the actual notification management application, they are still an integral part of the overall end-to-end application. Ideally they would be part of the application itself (discussed in greater detail in Chapter 6), but by segregating the components in this manner it allows for complete concentration to be placed on the main aspect of this project: the info-bead model and fuzzy inference process.

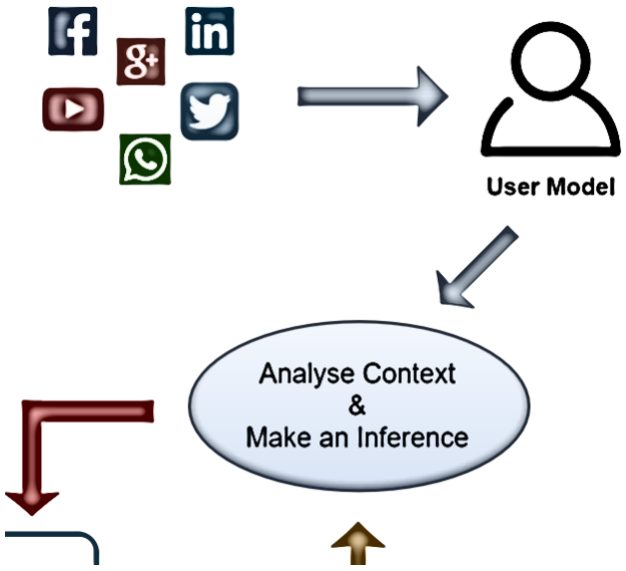


FIGURE 3.3: The high level design of info-bead functionality.

Figure 3.3 illustrates the functional elements in the high level design of the NMS (the prototype application of which is named *NAbsDesktop*) which are to be encapsulated by the info-bead model framework. This diagram shows that the user model, which is derived from social media plugins, and the notification data, which is taken from the mobile application (*NAbsMobile* in this case), are plugged into a system which carries out the analysis and contextual mapping of the data. Calculation of the contextually relevant delivery time is then the output of this system. All these elements are broken down into basic functions which are slotted into inter-connected info-beads and info-pendants. Figure 3.4 illustrates how the info-bead model framework, in the case of this *NAbsDesktop*, is functionally designed.

As discussed in Chapter 2, the info-bead model is made up of a number of components. The info-beads themselves are comprised of operational, control and metadata parts, and can be combined together to form info-pendants and group models. Through use of info-beads, entities with a range of attributes can be modeled and interlinked. In this case there are two entities which must be modeled. The first entity is the (incoming) Notification. The *Notification* entity is made up of a number

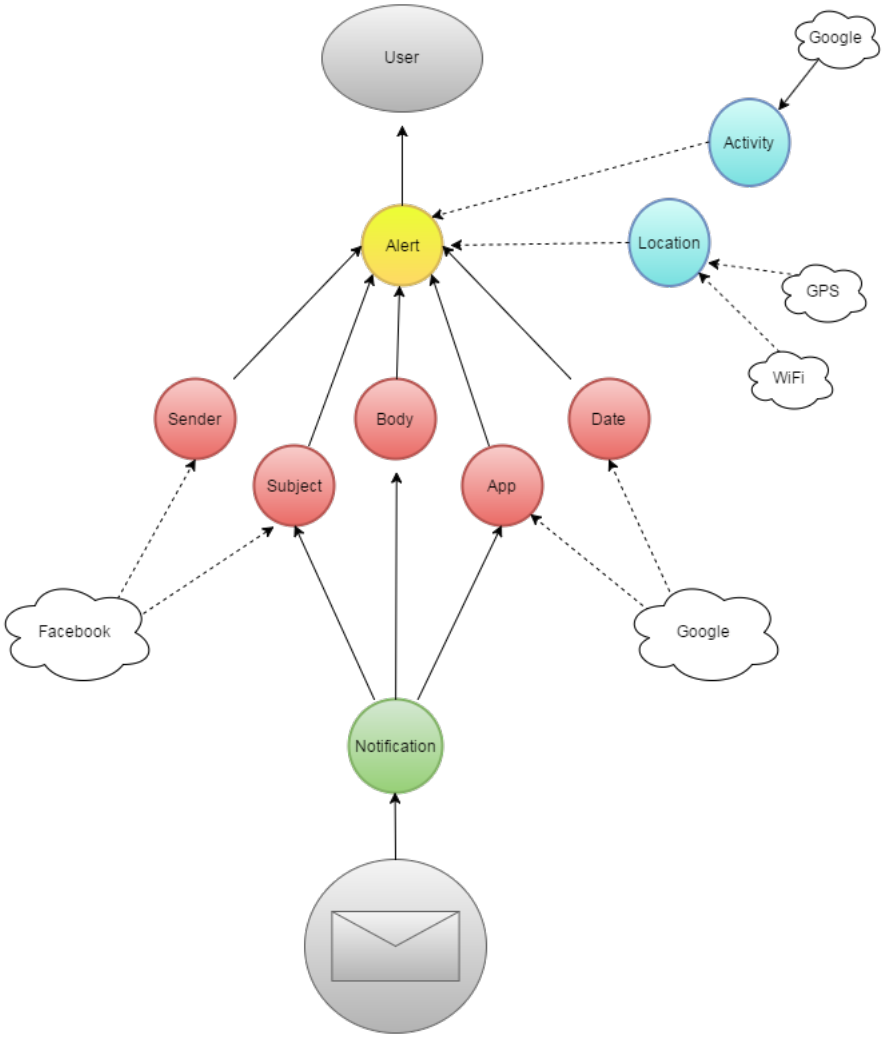


FIGURE 3.4: Info-bead model functional design.

of attributes such as: the sender of the notification, the application which the notification was received through, the subject, the body of the notification (the raw text) and the time and date which the notification was received. Hence, from figure 3.4 the notification model is made up of the red beads. The second entity is the *User*. The *User* entity is made up of a number of attributes such as: the users current location and the current activity they are engaged with. Of course additional attributes can be added to provide an even greater level of detail. However, this requires a greater number of data sources to be integrated in the system. The blue info-beads in figure 3.4 model the *User*.

Again from figure 3.4 it can be seen that there are a number of beads connected to the notification-attribute info-beads[1] via inward and outward directional arrows. These arrows are the info-links between the info-beads. The info-links connect beads

[1]notification-attribute info-beads - these are info-beads which describe the various attributes of the (incoming) *Notification* entity, such as the *Sender* info-bead or the *App* info-bead.

together and are defined by a communication protocol of either push (solid arrows), pull (dotted arrows) or notify (not currently used within the developed framework). In the case of the notification-attribute info-beads the incoming info-links come from three main sources: *Facebook*, *Google* and the *Notification* info bead. The *Facebook* and *Google* "clouds" are both, in this case, classified as sensor data[2] which the notification-attribute info-beads pull upon when necessary, in order to make inferences. The info-beads are also connected to a *Notification* info-bead. The communication method between the *Notification* info-bead and the notification-attribute info-beads is a push from the *Notification* info-bead to the notification-attribute info-beads. The *Notification* info-bead simply contains the raw data from the notification data-set collected from the *NAbsMobile* application and is uplifted via *NAbsUplift* application. To simulate an incoming notification, this raw data is sent to the *Notification* info-bead, which activates the rest of the info-bead system. It then proceeds to push the data of the incoming notification outwards to the notification-attribute info-beads. In each notification-attribute, info-bead inferences are made based on the data it is programmed to receive, and it then pushes this information onward to any info-bead which is subscribed to the info-beads push events. Through this method, the inferences make there way through the network and end up in the *Alert* info-bead.

The user-attribute info-beads[3] (blue beads) are also connected via info-links to sensor information and other info-beads. The location info-bead for instance takes in GPS coordinates and a *WiFi* connection which could be found via the users mobile phone. This information can then be used to generate the users location. The user attribute info-beads are connected to the alert info-bead via a pull communication method. This means that the beads only offer their data to the alert bead when asked for it. As the *Alert* info-bead is only activated when an incoming notification occurs, this information is only pulled upon when all data from the notification attributes has been inferred and pushed to the alert info-bead. With the combination of both notification inferences and user inferences, the alert info-bead carries out additional inference logic to calculate at what point the notification should reach the user.

3.4.2 Info-Beads & Info-Pendants

As previously discussed, the info-beads are made up of three significant parts, each responsible for a certain functionality. The operational part of the info-bead is responsible for receiving evidence data[4], programmatically using it to make an inference and subsequently creating an atomic or composite attribute. It consequently sends the inferred attribute as a *Triplet* to other info-beads or an external consumer.

[2]sensor data - an info-bead model concept which regards any data provided to an info-bead, which is not coming directly from an info-bead, as sensor data.

[3]user-attribute info-beads - these are info-beads which describe the various attributes of the User entity, such as the *Activity* info-bead or the *Location* info-bead.

[4]evidence data - an info-bead model concept of the raw data used to make inferences.

As illustrated in figure 3.4, the process as described above, corresponds to the operational part of the *Notification* info-bead (the green bead), which is invoked by the incoming notification. The evidence data from the simulated incoming notification is passed to the *Notification* info-bead and is comprised of the uplifted sender, subject, time/date, body and application values of the notification. The *Notification* info-bead infers which value is which and sends the information as triplet via push info-links to the appropriate notification-attribute info-beads (red beads).

A similar scenario occurs again for each notification-attribute info-bead. The operational part of each info-bead receives the incoming evidence data from the *Notification* info-bead, however, in this scenario, the info-bead also pulls additional evidence data from another sensor source: *Facebook* and/or *Google*. The *Notification* info-bead activates the "getEvidence" function of each notification-attribute info-bead, and, from within this function, the notification-attribute info-beads also pull data from *Facebook* and/or *Google* using the relevant API's.

For example, a typical scenario would be that a notification has been sent by a particular entity, a family member for instance. In this scenario (illustrated in figure 3.5) the notification sender attribute would have been uplifted to "family" from within the *NAbsUplift* application. This notification would be simulated as being sent in the NMS and all the evidence data of the notification (sender, subject, application etc.) would be fired at the info-bead model framework entry point - the *Notification* info-bead (Step 1 in figure 3.5). The *Notification* info-bead would parse this data (Step 2) and send the uplifted sender data - which would be "family" in this case - to the *Sender* info-bead (Step 3). The *Sender* info-bead would store the evidence data received from the *Notification* info-bead, and also pull information from the *Google* API (Step 4), such as calendar information which would help the info-bead determine if the current event a user is attending is related to the sender of the notification. Using the information received from both the calendar and the *Notification* info-bead, the *Sender* info-bead would then make an inference as to the importance and contextual relevancy of the sender of the notification (Step 5). This value of importance and relevancy would then be sent as a triplet to the *Alert* info-bead (Step 6), which would save this piece of evidence data for its inference process (Step 7).

The control part of the info-bead is responsible for the activation/deactivation of the info-bead, the authorization for other info-beads to access its data, the authorization to send information to other info-beads, the definition of its communication protocols (push/pull/notify) and the continuous updating of the info-beads internal settings.

The control parts of the info-beads in figure 3.4 are invoked continuously as information filters through the info-beads and the settings and states of the info-beads change. The first time the control part of the *Notification* info-bead is invoked is when an incoming notification is fired at the *Notification* info-bead during the simulation

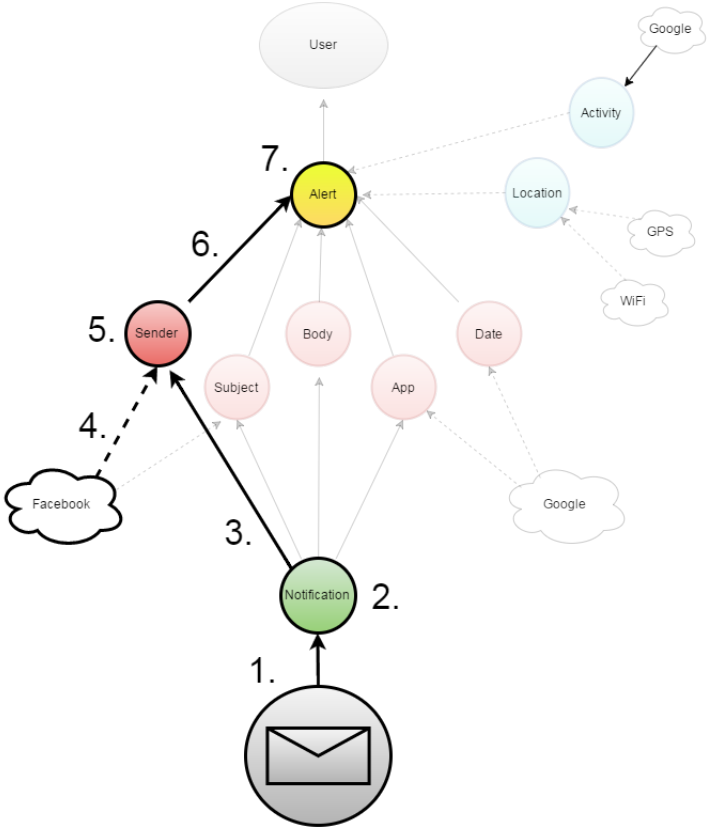


FIGURE 3.5: Info-bead example scenario (partial).

process. At this point the *Notification* info-bead sets its activation mode to "ON" and continues its inference process. The same occurs for each remaining info-bead. They are only activated when they are contacted or touched by the incoming notifications path filtering through the info-bead model. Potentially this means that there could be many info-beads inactive at any time if the information isn't relevant to the info-bead in question.

The final part making up the info-bead is the meta-data part. This part describes the functionality of the info-bead and allows it to be easily understood by developers, as well as easily searched. Use of the meta-data part of the info-bead hasn't been exploited to its full potential in this project. However, Chapter 6 discusses the future possibilities that this offers in terms of scalability and expansion of the info-bead model.

Currently the meta-data part of the info-bead supports various fields such as the name of the info-bead, the version, the resources used within the info-bead, the trustworthiness of the info-bead and contact details of the creator of the info-bead. This information is being set for each info-bead within the NMS's info-bead model framework however it is not being used for any inference functionality within the NMS but for ease of understanding how a particular info-bead fits into the info-bead model

framework as a whole. Future plans will include intelligently searching for an appropriate info-bead based on the evidence/inferred data provided in order to further automate the system.

3.4.3 Social Media Data Harvesting

The previous section outlined in some detail the design for the info-bead model and the means by which information flows through the framework to eventually reach a user in the form of a contextually delivered notification. This is the backbone of the NMS. An integral part of the process is accessing additional data pertaining to the user for use in making inferences about a particular attribute of a notification. This is an important step in the design of the NMS as it provides the much needed additional data necessary to contextually link the notification with the user. For example, the *Sender* info-bead makes an API request to the users *Google* calendar to find events the user is currently subscribed to. Based on the relevancy between current or upcoming events and the "sender" evidence data received by the info-bead, an importance/relevancy value attributed to the sender of the notification will be inferred. This inferred value is set to be between 0 and 1 (0 being a less important or irrelevant notification, and 1 being very important or relevant) and is pushed to the next info-bead(s) from within a *Triplet*. The scale of 0 to 1 was chosen simply because it suited the fuzzy inference mechanism implemented in the info-beads.

The data sources chosen for use within the system are easily accessible, with open API's readily available, and they contain a large amount of rich data from which information can be gleaned about a user's attributes. For example, using *Google Calendar* information, such as the users location, activity and who they're with at any given time, can be accounted for. This of course is dependent on the engagement a particular user has with social media. For the purposes of this project it is assumed that a user has a high level of engagement with their *Google* calendar, updates their schedule regularly, and adds information such as the subject of the event and the people it engages with in the description of the event. These assumptions are quite specific and may not translate completely to a real world situation. However, for the purposes of testing the info-bead model and the FIS, these assumptions are adequate.

The *Facebook* API was also integrated into the info-bead model framework of the *NAbsDesktop* application, however it was not used as the project dealt simply with using the users *Google* calendar for inferring the importance of the notification attributes. Future work on the project will include adding a greater number of inferences based on *Facebook's* social graph.

3.4.4 Inference Mechanism

Overview

The inference mechanism integrated into each info-bead of the info-bead model, is perhaps one of the most important logical functions within the NMS. The inference mechanism is the intelligent part of the system which links data together in a contextual manner.

Within the operational part of each info-bead there is an inference function which applies logic to the evidence data to infer an atomic or composite value. One of the advantages of the info-bead model is the flexibility to implement any number of inference mechanisms. There is no restriction on the type of inference mechanism that can be used and this is a powerful characteristic of the info-bead model as it enables developers of info-beads to use whichever inference mechanism works best for the particular info-bead being developed, and also whichever inference mechanism they are most comfortable implementing.

There are a number of inferences that need to be made throughout the system in order to achieve the main goal of delivering a notification contextually to the user. The greater amount of data available to the system, the greater number of inferences can occur, meaning that the delivery of the notification can be judged to a finer degree. For this project, data used in the inference mechanism was gathered from 3 main sources, some of which have already been discussed in previous sections:

1. Incoming notification data-set - the notification data gathered by the *NAbsMobile* application on both the author and supervisor.
2. *Google* calendar data-set - the *Google* calendar data of both the author and the supervisor.
3. Ranked uplift term-set - the perceived importance of each uplifted term value.

The incoming notification data-set provides 5 attributes which can be used in the inference process. These are:

- Sender - the person who sent the notification.
- Subject - the subject line of the notification (named "ticker text" by *Android*).
- Body - the body of the notification (the actual message contained in the notification).
- Application - the application the notification was received through.
- Date - the date and time the notification was received.

The *Google* calendar provides data on the user's current schedule. Accessed via a *RESTful* API, *Google* provides the user's schedule in the form of *Event* objects. Queries can be made to get the next *N* events for a particular user. In this case, the key data used from each *Event* object is:

- Start Date - the start date and time of the event.
- End Date - the end date and time of the event.
- Description - the description of the event (as discussed in the previous section, it is assumed the user will have added the subject of the event and the people known to be associated with the event, to this field in their uplifted value form).
- Location - the location where the event is to take place.

The third and final data source is the ranked uplift term set. This is comprised of the set of terms created by the author and supervisor when assessing their respective notification data-sets. These terms are given a ranked value dependent on their perceived (by the author and supervisor) importance.

The perceived importance of each uplift term value for the author and supervisor can be found in Appendix A.

Fuzzy Logic

Fuzzy logic is a multi-valued logic derived from fuzzy set theory to deal with reasoning which is approximate as opposed to precise. It is a system for dealing with vague concepts. Fuzzy set theory involves classifying things as part of a set to a certain degree. In contrast with traditional logic which can describe things only in binary forms of true or false, fuzzy logic has the ability to use approximate values, such as a degree of truth or a degree of false.

As some decision making and problem solving tasks are difficult to understand quantitatively, it makes sense to tackle them instead in such a way that it is easy for humans to understand. In this case, identifying which notifications should be allowed to reach a user straight away, and which notifications should wait for a later time, is a difficult problem to solve quantitatively. A binary method to tackle the problem would be to classify notifications as either "important and deliver now" or not "important so deliver later". But this throws up a number of problems. To what degree of "later" should the notification be delivered? Do we need to set a fixed number of minutes to wait for the notification to try and reach the user again? This is also not a realistic view of notifications. While there will be notifications which definitely should be classified as either "important" or "not important", there will also certainly be notifications which fall into the grey area of being neither fully important or fully unimportant. Consequently, this means that the problem of classifying notifications

is a vague or fuzzy problem. We could, naturally, create a third set, named "neutral", in-between the important and unimportant sets, which would catch the remaining types of notification, however if we instead allow membership of all these sets be a continuous function such that a notification can be part of a class to a certain degree, the accuracy in describing notification importance will be greater, and consequently, the notification can be delivered more precisely.

The proposed inference mechanism to be implemented in each info-bead is as follows:

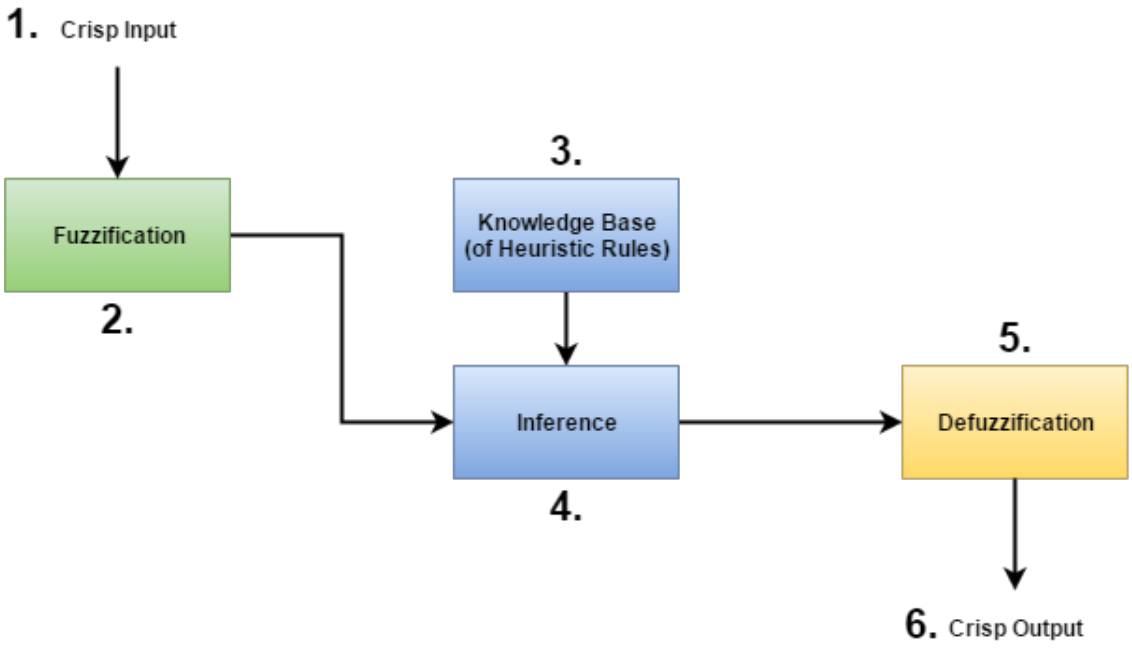


FIGURE 3.6: Fuzzy Inference System (FIS).

Figure 3.6 illustrates a Fuzzy Inference System (FIS). A FIS has 4 main steps which contribute to crisp input values being mapped to an inferred crisp output value. These 4 steps are as follows:

- Fuzzification
- Inference
- Composition
- Defuzzification

Note: Every implemented membership function and knowledge base within all the info-bead FIS's are illustrated in Appendix D.

Fuzzification

Due to the fuzziness of the problem in identifying which notification is contextually important, the inference system of each info-bead is designed in such a way that heuristic and common sense rules are the significant drivers. In order to be able to implement these rules, the quantitative problem must first be converted to a linguistic one. Hence, the crisp values[5] associated with the data sources mentioned in the *Overview* of section 3.4.4, must first be mapped to linguistic variable sets. This process is known as "fuzzification". *Fuzzification* is achieved through the use of membership functions[6] of predefined shape and parameters.

Recalling figure 3.4 from section 3.4, there are a number of info-beads which are modeling various attributes of the notification. These are the red beads in the illustration - "Sender", "Subject", "Body", App" and "Date". Each of these info-beads contain an implementation of a *Mamdani* Fuzzy Inference System (FIS) and the crisp input values that are used for the inference process are contained within the evidence data received from the *Notification* info-bead and the *Google* calendar API. Recalling that the value pushed from the *Notification* info-bead to the notification-attribute info-beads is an uplifted term value, and the data received from the *Google Calendar* API is an *Event* object, these two data types clearly won't work with the FIS as they are not crisp inputs. Therefore, some work first needs to be done to convert these values (string and object) to a crisp (real number) value (step 1 in figure 3.6).

The uplifted term value received from the *Notification* info-bead is easy to convert to a crisp value as it has already been prepared. This is the data-set which was created by the author and supervisor ranking the importance of the uplifted terms. The info-bead simply searches the received uplifted term value in this data-set to find the corresponding importance value (a ranking between 1 and 10), and it then has a crisp value for the uplifted term that can be used with the FIS. The significance of using this value in the system is due to it holding contextual meaning between the notification and the user, as it was chosen by the user as a mark of importance on the uplifted value. A highly ranked uplifted value would increase the chances of the notification making it through to the user sooner. For example: the *Notification* info-bead would send the uplifted value of "family" to the *Sender* info-bead. The info-bead would search the ranking data-set for the importance of "family" perceived by the author (appendix A), and would receive the value of 10. This value is then divided by 10 in order to convert it to a crisp input value between 0 and 1 (the purpose of which is to satisfy the fuzzy logic library implementation of the FIS used in the project - *jFuzzyLite* [Rada-Vilela, 2014]). As 10 is the highest possible ranking, the ranking

[5] crisp value - a precise value. In contrast with a fuzzy value which would be imprecise or vague and may have a range of values.

[6] membership function - this is a curve mapping input crisp values to a membership value between 0 and 1. The membership value determines the strength of association a crisp value has with the particular set the membership function was created to describe.

value found to correspond to the uplifted value (which is provided as evidence from the *Notification* info-bead), will always be divided by 10. The name of the crisp value described in this example would be "senderImportance", as it signifies the importance of the sender from the user's perspective. Similarly for the remaining notification-attributes, the crisp values would be called "subjectImportance" or "appImportance" and so on. As "sender", "subject" and "app" are simply the notification attributes which are being checked for contextual relevance through inferred importance, we will call this first crisp input value *attributeImportance*.

The reason for querying the *Google Calendar* API from within the notification-attribute info-bead is to ascertain whether or not the evidence data received from the *Notification* info-bead is contextually relevant to the user. For example: if a user gets a notification regarding a social event that is happening next week, and has a schedule comprised of work related events all day, then the importance of the "subject" attribute of the notification should drop significantly, leading to the notification being delivered at a later stage (perhaps when the work related events are over). This requires the concept of the subject's contextual importance in relation to scheduled events to be represented as a second crisp input value in the FIS. In this example the crisp value is called *eventRelevance*. This value is derived by completing the following steps:

- Query the API to receive the next 10 events in the users schedule (this was found to approximate two days of events).
- Check if the description of each event contains the uplifted term set (provided by the *Notification* info-bead as evidence data pushed to the notification-attribute info-bead). If there is a keyword match, then the event is contextually relevant.
- Rank all the events (on a scale from 0 to 1) based on their importance and their relevance to the time and date the notification was delivered (events which may have matched contextually with the uplifted value may still be much later. Hence importance would decrease).
- The value for the maximum ranked event is the new crisp input describing the contextual importance of the uplifted value with respect to the user's scheduled events.

As the two crisp values are now in the correct form and scale, a rational number between 0 and 1, the first step of the FIS, *fuzzification*, can begin (step 2 in figure 3.6). In this step the two crisp values are mapped to a linguistic variable. The set of linguistic variables for the first crisp input, *attributeImportance*, is comprised of:

- Not Important (NIP)
- Important

- Very Important (VIP)

The set of linguistic variables for the second crisp input, *eventRelevance*, is comprised of:

- Not Relevant
- Relevant
- Very Relevant

The relationship between these two input variables has yet to be defined (this will be done in the following step), however the result of combining both input variables will result in a number of output membership functions corresponding to additional linguistic variable sets. The output after combining a particular *attributeImportance*, "senderImportance" for example, and its corresponding *eventRelevance* would result in a specific output of *attributeContext*, or in this case "senderContext". This output value is the inferred value describing the contextual relevancy and importance of the sender of the notification. For example - a ”senderImportance” of ”VIP” and an *eventRelevance* of ”Very Relevant” might result in a ”High” ”senderContext” meaning that the sender of the notification is both important in terms of the ranking the user gave them, and also contextually relevant to the user at this point in time, based on their calendar schedule. The output linguistic variables, are as follows:

- Low
- Medium
- High

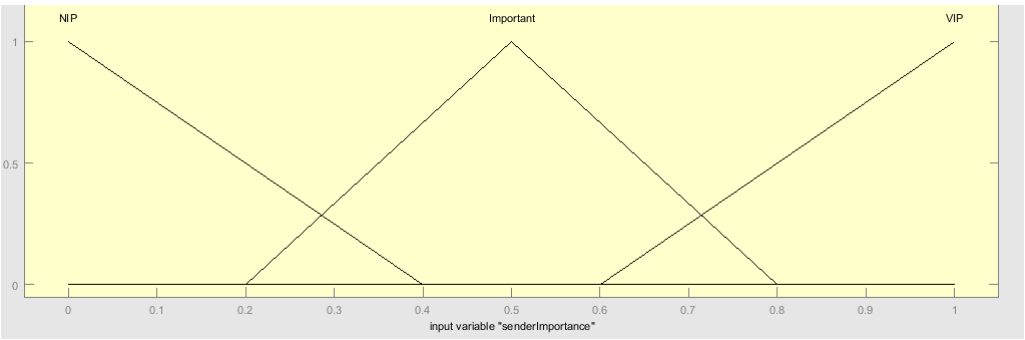


FIGURE 3.7: Membership function for "senderImportance" (Note: an updated membership function was used in the NMS and can be found in appendix D.

The mapping of crisp values to linguistic terms occurs in the *fuzzification* process through a membership function which classifies the crisp input value as belonging to the linguistic term sets to a certain membership degree μ (a value between 0 and

1). A membership of 0 would signify that the crisp value is not a member of the set while anything greater than 0 signifies the value is a member of that set to some degree. The membership functions can be shaped to describe the linguistic term they are associated with, which gives the developer the scope to use any mathematical model which makes sense for the linguistic term in question. An example of the membership function used for "senderImportance" is illustrated in figure 3.7:

In figure 3.7 the crisp input value is translated to linguistic variables *NIP*, *Important* or *VIP*. Figure 3.8 illustrates an example of how this occurs. The red line has a crisp value input of 0.1 on the x-axis. This intersects the *NIP* function at a y-axis value of approximately 0.7, hence this crisp input value belongs to the linguistic variable set of *NIP* with a membership degree of 0.7. The green line has a crisp value input of 0.25 on the x-axis. This intersects both the *NIP* and the *Important* membership functions, therefore this input value is a member of both these linguistic variable sets, the degree of which can be found by finding the corresponding y-axis value at the point of intersection: μ(NIP) $\approx$ 0.2, μ(Important) $\approx$ 0.4.

Once *fuzzification* of the input values is complete, the system now has linguistic variables which correspond to real world values with which to manipulate and make inferences. This is a key point, as it allows heuristic knowledge to be applied easily in the following steps.

The complete portfolio of membership functions for all variables used throughout the info-bead model framework can be found in Appendix D.

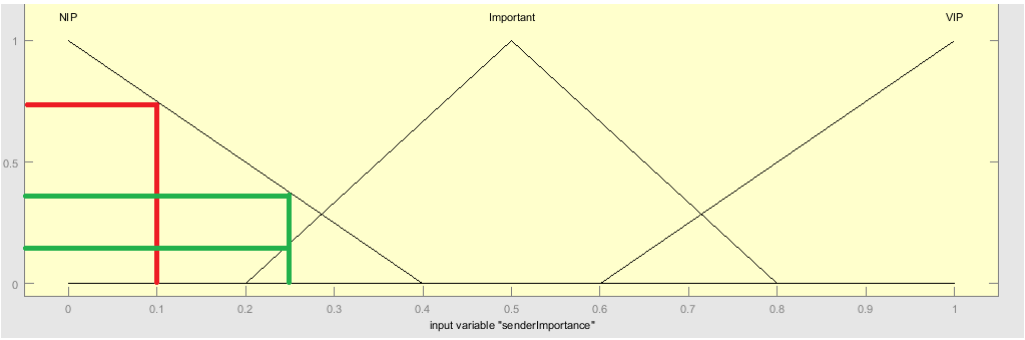


FIGURE 3.8: Example *fuzzification* for ”senderImportance”.

Inference

Recalling from the illustration of the FIS in figure 3.6, once *fuzzification* is complete, the next step in the process of fuzzy inference is creating a knowledge base of heuristic rules (step 3). The outputs from the *fuzzification* block are the linguistic variables which the crisp inputs have been mapped to. However, in order for the inference block to relate the input and output variables together and form an inference, it must be provided with a rule block which defines the relationships between the variables.

There were a number of options to choose from when selecting which type of FIS to implement. Subsequently, it was chosen to implement a *Mamdani* FIS as it would enable the use of linguistic rules. This meant that the relationships between the input and output variables within the FIS could be defined using natural language - the advantages of this of course being that the heuristic rules could be expressed easily when developing the system. Another option here would have been to use the *Takagi-Sugeno-Kang* FIS which has an advantage over *Mamdani* due to its increased computational efficiency. However, as the NMS is not yet embedded in a mobile device, it was deemed unnecessary. This choice should be revisited when the NMS is to be installed on a mobile device and increased computational efficiency becomes a necessity.

The knowledge base is comprised of a number of linguistic rules which are used to perform approximate reasoning, based on fuzzy set theory. These rules are in the form of antecedent-consequent (IF-THEN) clauses, both of which are "fuzzy" (both antecedents and consequent have a degree of certainty), and they contain the human intelligence that govern the various situations which can occur through incoming notifications. For example, figure 3.9 illustrates the knowledge base integrated into the *Sender* info-bead FIS. The individual rules are easy to understand, due to the use of linguistic variables within IF-AND-THEN clauses. The antecedents of rule 1 would be the membership value "senderImportance" has with the *NIP* set, combined with the membership value *eventRelevance* has with the *Not Relevant* set. The resulting consequent will then result in "senderContext" having a certain membership value for the *Low* set. One of the key points of the fuzzy inference process is that more than one rule in the knowledge base can fire at a time. In traditional Boolean logic all the antecedents would have to be true for a rule to fire. However, with fuzzy logic the antecedents can be partially true resulting in a number of rules firing to a certain degree (this will be discussed in the *composition* stage).

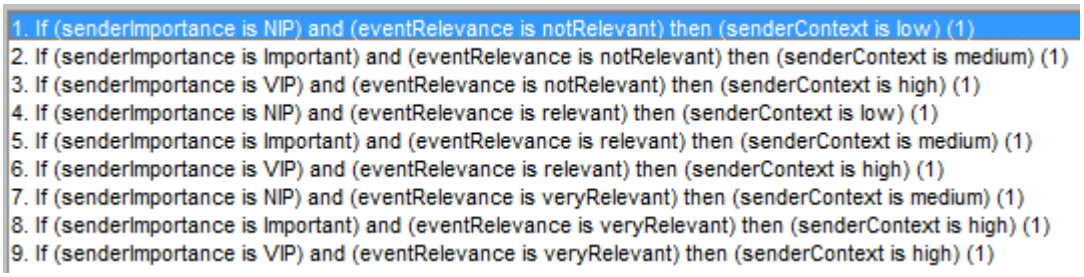


FIGURE 3.9: *Sender* info-bead FIS Knowledge Base (from Matlab's Fuzzy Logic ToolBox).

The *inference* stage of the FIS therefore, involves determining which rules in the knowledge base are to fire (step 4 in figure 3.6). It accomplishes this using the membership function (μ) values determined from the *fuzzification* stage. It evaluates the degree to which the antecedents are true. For example: rule 1 would only fire if

"senderImportance" had a membership value greater than zero for the *NIP* set, and *eventRelevance* had a membership value greater than zero for the *Not Relevant* set. Recalling the membership functions for both *NIP* and *Not Relevant*, illustrated in Appendix B, this would occur whenever the crisp input value for "senderImportance" is less than 0.4, and also when the *eventRelevance* crisp input value is less than 0.4.

The logical AND linking both antecedents determines the form of the consequent output function. Logical AND translates to taking the minimum membership value from the antecedents and applying it as an alpha-level cut on the output function. An alpha-level cut is a restriction on a function to ensure it does not exceed a given "alpha" value on the y-axis. This means that by applying an alpha-level cut to an output function, the maximum membership (μ) a value can have for that particular linguistic variable set, is limited. Figure 3.10 is an example of an alpha-level cut being applied to rule 1 of the knowledge base, occurring in the inference process of the *Sender* info-bead FIS. The first graph is the membership function for *NIP* fuzzy set, the second graph is the membership function for *Not Relevant* fuzzy set, and the third graph is the membership function for the fuzzy output set *Low* (the dotted blue line being the original *Low* membership function). As illustrated, there are two crisp inputs of 0.3 and 0.25 for ”senderImportance” and *eventRelevance* respectively. The two crisp values are fuzzified to a μ value of 0.2 and 0.6 respectively. As both antecedent μ values are greater than zero, rule 1 can fire. Hence the output is calculated by applying a logical AND to both antecedents. This translates to taking the minimum μ value of 0.2 and 0.6, 0.2, and applying an alpha-level cut to the output *Low* membership function. This creates a new output membership function capped at 0.2 for this particular rule (illustrated by the green line in figure 3.10). The next step is to combine all the output membership functions of all the rules which were found to fire, in order to create the output membership function of the FIS.

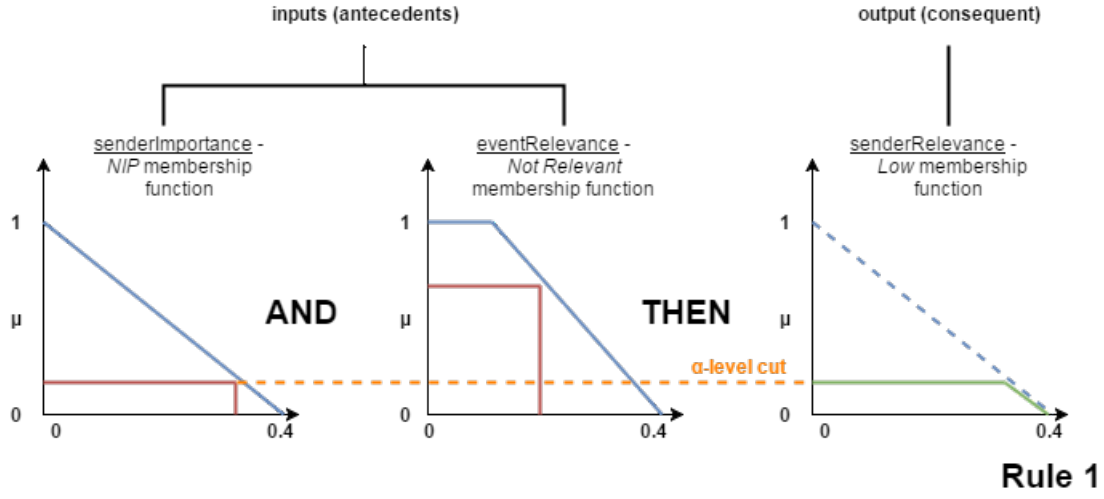


FIGURE 3.10: An example of the logical AND alpha-level cut process.

Composition

Due to the fuzzy nature of the problem, notifications can fall into many different categories of importance and relevancy. The previous steps provide solutions for describing the notifications in terms of a degree of importance and relevancy, and drawing up a set of rules that dictate what should occur for various situations. However, notifications can fulfill the requirements of multiple rules at a time meaning multiple rules will fire (at different strengths) leaving the system with multiple output functions. In order to be able to convert the inferences made back into a real world value that can be used in the NMS, these multiple output functions need to be combined into one single output membership function - one which describes the full extent of fired rules and strengths and which can be used by the system to calculate a tangible result.

Composition is the second part of the inference step (step 4 of figure 3.6), determining the degree to which each rule in the knowledge base is fired. In essence, the *composition* process calculates the contribution and influence that each rule has on the overall output membership function of the FIS. Figure 3.11 demonstrates once again the *fuzzification* and alpha-level cut, but for rule 2 (which would have also fired for the given crisp input values from the example in figure 3.10). Figure 3.12 then illustrates the *composition* process involved for combining both fired rules into one output fuzzy set. Hence, *composition* is achieved by aggregating the output functions of all the fired rules. In this manner the individual strengths of all the fired rules are described by the final output fuzzy set.

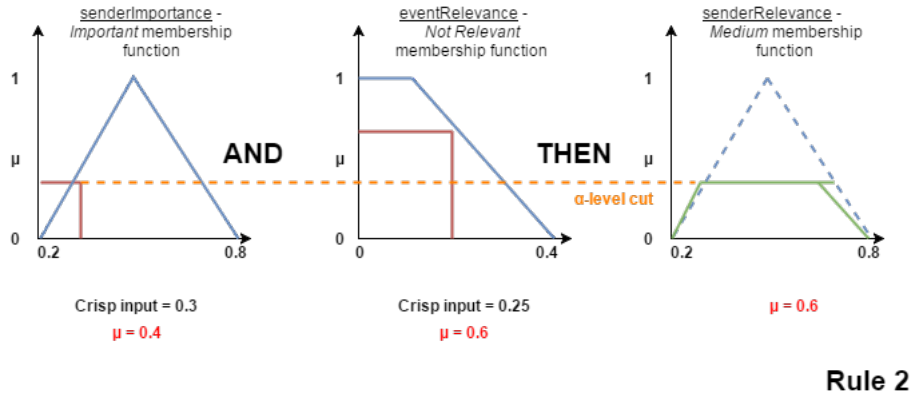


FIGURE 3.11: An example of the *fuzzification* and alpha-level cut of rule 2 found in the *Sender* info-bead knowledge base.

Defuzzification

The final step of the FIS is to defuzzify the output fuzzy membership function into a real number which can be used by the NMS for identifying the importance and

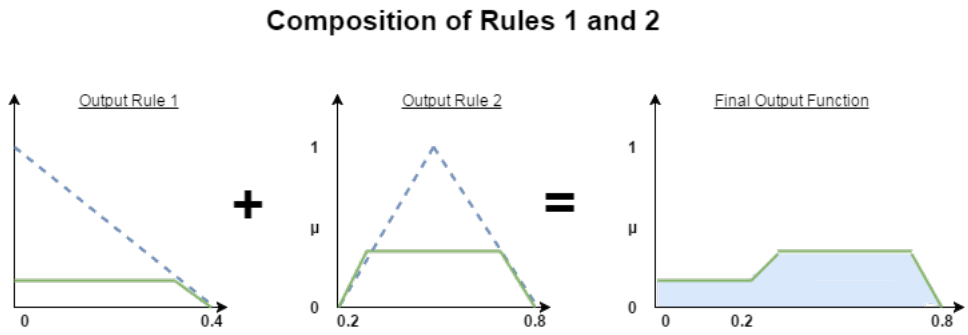


FIGURE 3.12: An example of the *composition* process of output fuzzy sets from rules 1 and 2 found in the *Sender* info-bead knowledge base.

relevance of the notification. This process can be achieved by a number of means, two such popular methods are:

1. Mean of Maxima - this method calculates the crisp output value to be the mean of the maximum values of the output fuzzy set.

2. Centroid - this method calculates the crisp output value to be the mean of the entire range of output fuzzy set values - essentially getting the center of gravity of the area under the curve created by the output fuzzy function. This method requires integration over the curve. Hence it takes greater computational power than the *Mean of Maxima* method.

Both methods are illustrated in figure 3.13. For the purposes of this project the *Centroid* method was chosen, as it yields results with greater accuracy than that of the *Mean of Maxima* method when used with the *Mamdani* inference implementation [Wang and Chen, 2014]. The crisp value that is returned for the illustrated example in figure 3.12, is 0.4. This value is saved in the *Sender* info-bead as the inferred value. The info-bead then provides this as evidence data to the *Alert* info-bead which carries out further inferences based on all the notification-attribute info-bead inferred values, as well as the user information it receives.

The examples used to step through the stages of the FIS system in this section were mainly specific to the *Sender* info-bead. There are similar systems deployed in the remaining notification-attribute info-beads. There is also a final FIS deployed in the *Alert* info-bead, which carries out the final inference on all the individually inferred data attributes to discern whether the notification is important and relevant. There is an extra step in the *Alert* info-bead to translate the crisp output value received from the FIS to a contextual delivery time. This process will be discussed in the next section.

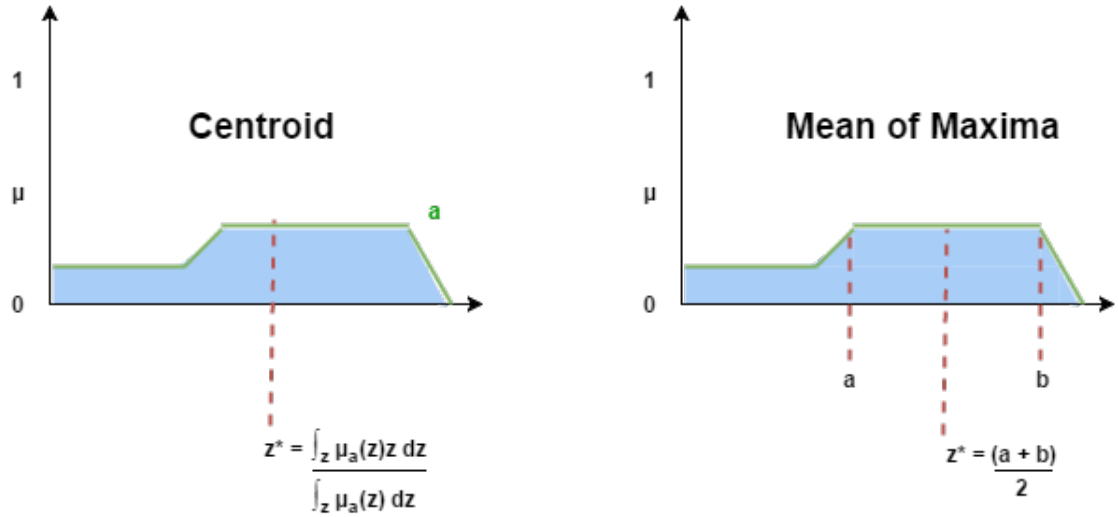


FIGURE 3.13: An illustration of two *defuzzification* methods - *Centroid* and *Mean of Maxima*.

3.4.5 Contextual Delivery

The previous sections outline how the value of a notification's importance and relevance can be determined by the FIS and expressed as a crisp value. The final crisp value output by the *Alert* info-bead however, must be translated to an actual delivery time for the notification. This is achieved by creating five categories which the *Alert* info-beads FIS output can fall into, each with a unique delivery method (these five categories were implemented as the output linguistic variables in the FIS of the *Alert* info-bead, and can be found in figure D.13 in Appendix D):

- Now - Interrupt the user and deliver the notification.
- Very Soon - At the next break in the users schedule, deliver the notification.
- Soon - At the next free period (a break consisting of a few hours) in the users schedule, deliver the notification.
- Later - At/Before the next contextually relevant event in the users schedule that the notification is linked with, deliver the notification.
- Much Later - Identical to the Later category.

The above delivery mechanism is dependent on the users *Google Calendar* and in order for the contextual deliveries to be accurate, the user must engage with their calendar and keep it up to date. If, in the above rules, it is found that the users current location doesn't match with their calendars information, or if there is nothing scheduled for the user at the time of the incoming notification, it is assumed the user is currently free and notifications which previously fell into the *Very Soon* or *Soon* categories are delivered to the user instantly.

3.4.6 Design Ethics

During the design of the NMS a number of ethical dilemmas arose, mostly surrounding the privacy and use of the users data. The NMS required a number of input data sources in order to function, most of which contain sensitive information pertaining to the user, which was necessary to be protected - for instance: the uplifted notification data provided by the two volunteers (the author and the supervisor), the data provided via the *Google Calendar* API, and also the ranking data provided on the uplifted terms. While some of this information is more sensitive than others, it is all private data belonging to the user, and although the two volunteers agreed for it to be used for the purposes of this project, the design of the application has to ensure that the data is sufficiently protected, and that any sensitive information used is only openly accessible to those directly involved in the project.

In addition to data privacy, the use of the data is also of ethical importance. For instance, will the inferences made within the NMS cause the user to miss some of their notifications? Could incorrect inferences occur due to misinterpretation of the data resulting in an inaccurate model of the user being used to deliver notifications? Could the system be biased towards some notifications and skew delivery in favour of a particular sender or application? These questions all raise legitimate concerns especially as some of these scenarios could occur accidentally. The key to resolving any issues such as these is to keep the process of the inference and use of the data as transparent and open as possible, giving the user the option to intervene, and have an input to the system if they deem it necessary. The ranking system used to infer the importance of uplifted terms is an example of the users input into the system. In this manner, the system can be fine tuned by the user themselves to suit there preferences.

The use of social media integration with the NMS means that users personal accounts must be accessed. *Google* provides a secure API for accessing the data. However, they also require user credentials to be provided. To minimize the risk of using multiple user credentials, just one account was used for the *Google Calendar* integration. This account was set up for the purposes of this project and contains separate calendars, one each for both volunteers. The volunteers personal/work calendar data was then added to the relevant account calendar, allowing any sensitive event information to be first removed, so as to protect the user's privacy.

3.5 Summary

This chapter discussed the design methodologies behind the various components that make up the full end-to-end notification management solution. This included:

- *NabsMobile* - The mobile application that captures the real-world notification data.

  In this section the motivation behind the design decision to separate out the capture of real-world notifications and the management of the notifications, was discussed. The approach of gathering real-world notification data through the use of volunteers was also explained, as were the ethics involved with this process.

- *NabsUplift* - The application used to aid in the notification uplift process.

  In this section the necessity of uplifting the notification to a predefined terminology, was examined. The design behind how the application would maximize users privacy and scale easily was also analysed, as was the method behind the design of the predefined terminology. Finally the ethical implications of developing this application were also studied.

- *NabsDesktop* - The desktop application implementation of the Notification Management System.

  In this section the novel aspect of the project design, the info-bead model and fuzzy inference combination, was explored. This included a discussion on the info-bead model framework design for the NMS, the FIS design for the individual info-beads within the framework(the *Sender* info-bead FIS taken as an example), the social media integration design and the ethics surrounding all aspects of the NMS.

Chapter 4

Implementation

4.1 NAbsMobile

As discussed in Chapter 3, the motivation behind the *NAbsMobile* application was to gather a data-set of real-world notifications for use in testing the performance of the NMS, and in particular, to aid in the evaluation of the effectiveness of the info-bead model and fuzzy inference combination.

In order to capture real-world notifications, the *NAbsMobile* application had to be installed upon the mobile phones of volunteers willing to have their notifications observed on an ongoing basis. Two volunteers were used for this project - the author and the supervisor.

NAbsMobile was designed as an *Android* application using *Android Studio*. In the first version of the application the following requirements, set out during the design phase, were implemented:

1. The application should log the key attributes of the *Notification* entity.
2. The application should store the data gathered in a secure database locally on the phone.
3. The application should have the functionality to export the database of notification data to a computer.
4. The application should delete notifications once the data has been logged so that the user does not receive any notifications.
5. The application should provide the user with a means to view all the captured notifications.

The first requirement involves enabling the application to observe all incoming notifications and extract the necessary data from them. In order to, achieve this the *NotificationListenerService* had to be implemented. This is an *Android* service which is notified by the system when a new incoming notification is delivered to the

phone. This service is first declared in the application's *Manifest*[1] file with a number of permissions necessary for the application to use the service on the phone it has been installed. An *Intent filter*[2] is also added to the services declaration, so that when a notification is sensed by the system, the *NotificationListenerService* is found within the *NAbsMobile* application on the phone and is started. Figure 4.1 illustrates the declaration in the *Manifest*. Note that the permission being set ensures that the only entity that can bind to the applications service, is the *Android* system on which the phone is being run. Hence, no external force (e.g. from the internet) can take control of this service and read or modify the incoming notifications.

```
1 <service android:name=".NotificationListener"
2 android:label="@string/service_name"
3 android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
4 <intent-filter>
5 <action android:name="android.service.notification.
      NotificationListenerService" />
6 </intent-filter>
7 </service>
```

FIGURE 4.1: Declaration of the *NotificationListenerService* in the *Manifest*.

With the *NotificationListenerService* now declared, the next step is to implement the service within the application and apply the functionality to log the notification data. A *NotificationListener* class is created, which extends the *NotificationListenerService* for this purpose. Within the *NotificationListener* class, the *onNotificationPosted()* function belonging to *NotificationListenerService*, is overridden and the incoming notification (in the form of an object called *StatusBarNotification* in *Android*) is sent to a database helper class for data extraction and storage. Once the notification is stored it is then deleted from the phone. Hence, the notification is no longer available for viewing by the user. It also no longer alerts the user - hence removing all notification functionality from the users phone.

In order to store the notification data within the phone securely over a long period of time (which was necessary in order to gather enough data for the project experiments) a long term storage solution needed to be implemented. For this reason an *SQLite* database was created within the application. The schema for the database contains only a single entity, the *Notification* (illustrated in figure 4.2)

The *packageType* reference in the schema relates to the application through which the notification was received. For example, *Facebook Messenger* notifications would appear as "com.facebook.orca", and *WhatsApp* messages would appear as "com.whatsapp".

[1]AndroidManifest.xml - This is a mandatory file included in the application's root folder which provides the *Android* system with essential information pertaining to the developed application (e.g. permissions).

[2]Intent Filter - an *Android* term - specifies the types of intents that an activity, service, or broadcast receiver can respond to.

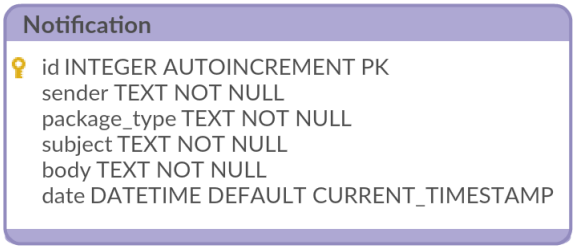


FIGURE 4.2: NAbsMobile SQLite schema.

It was not always clear from the string alone as to which application was being referenced. Hence, some research into application names also had to be carried out when the project reached the uplift stage.

Once the information pertaining to the notification was stored within the applications *SQLite* database, there needed to be a method of extracting the data onto a computer for continued manipulation. This required the *SQLite* database to be made exportable to the phone's external SD card. Again, for this functionality, permissions first have to be defined in the applications *Manifest* (figure 4.3).

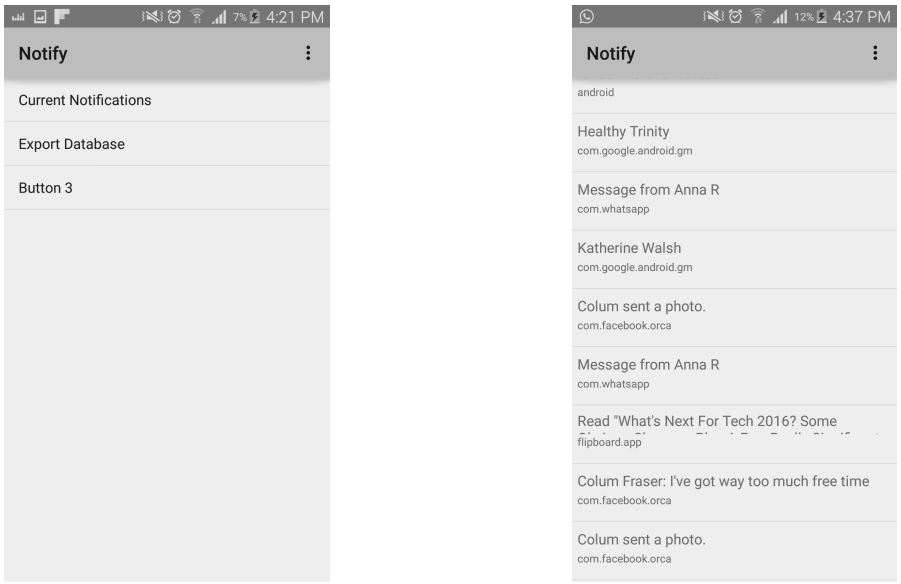
```
1 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

FIGURE 4.3: Permission declared in the Manifest for access to external storage.

```
1  private void exportDB(){
2    File sd = Environment.getExternalStorageDirectory();
3    File data = Environment.getDataDirectory();
4    ...
5    String currentDBPath = "/data/"+ "intercept.notification.notify" +"/
         databases/"+DatabaseHelper.DB_NAME;
6    String backupDBPath = DatabaseHelper.DB_NAME;
7    ...
8    try {
9    source = new FileInputStream(currentDB).getChannel();
10   destination = new FileOutputStream(backupDB).getChannel();
11   destination.transferFrom(source, 0, source.size());
12   }
13   ...
14  }
```

FIGURE 4.4: Exporting the local *SQLite* database to the external SD card.

Figure 4.4 illustrates the export function which converts the current local *SQLite* database to a *File* object, and "streams" it to the external SD card destination. By implementing this functionality, the database is now accessible on a computer via a USB connection with the phone.



(A) NAbsMobile home screen. (B) NAbsMobile list screen.

FIGURE 4.5: Screenshots of the NAbsMobile application.

A final piece of functionality was also added to the application for viewing the list of logged notifications in the database from within the application itself. This was used mainly to ensure that notifications were being correctly captured from the all possible applications stored on the phone. This required the creation of a simple fetch query fired at the local *SQLite* database to retrieve all attributes of the notification, and an *Android ListView* layout for displaying all the notifications to the user with an interactive scroll-bar.

This concludes all necessary implementation necessary for version 1 of the *NAbsMobile* application. The only difference in features in version 2 of the application, was to remove the deletion of notifications after storage of the notifications attributes occurred. This allowed for notifications to alert the user and still be viewable in the status bar of the phone. Screenshots of the application are illustrated in figure 4.5.

4.2 NAbsUplift

Following from Chapter 3, the next step in the projects design was to prepare the data for input into the NMS. This requires the development of the second application of this project, *NAbsUplift*.

The motivation behind *NAbsUplift* was that it would aid the user in the uplift process and maximize the security and privacy of the user's data. This application is designed with scalability in mind as it enables the project to span to additional test users' if necessary while keeping sensitive information secure. Therefore, the

application needed to be able to carry out a number of specific functions for this purpose:

1. Import the *SQLite* database of notifications exported from the *NAbsMobile* application.
2. Provide the user with an interface to explore the notifications within the database, so they can note what, if any, new uplift terms are needed for their data-set.
3. Provide the user with an interface enabling them to view, add and remove terms from the uplift terminology.
4. Export the data in an editable form, along with the terminology set to be used by the user for uplift.

It was chosen to create this application using the *Java* programming language and, as the application is required to have user interaction, it was decided to use *JavaFX* for the user-facing *GUI*. *JavaFX* is a set of graphic and media packages that adds rich design and functionality components to an applications front-end. *Apache Maven* is also implemented as the build automation tool of choice for the application in order to ease the management of the applications dependencies.

In order to implement the first requirement of the *NAbsUplift* application, an *SQLite JDBC*[3] driver had to be added to the project in order for the application to be able to establish communication with a database. Once added, a connection is then made to the exported *SQLite* database containing the notification data using the *SQLite JDBC* driver (line 5 of figure 4.6). Once a connection is successful, a simple SELECT query is run against the database to retrieve the necessary data. In order to temporarily store the data locally within the context of the *NAbsUplift* application a *Notification* class is first created to hold the notification attributes which are extracted from the database (sender, subject, app, body and date). The result set returned from the query is then iterated over and, for each row in the set, a *Notification* object is created and its attributes are set according the the data values extracted (lines 9-19 of figure 4.6). The *Notification* object is then added to the list of notifications.

The second requirement of the application which involved creating the user-interface for viewing the notification data was implemented using the *JavaFX* platform. A number of text fields were created to show the data values of each *Notification* attribute and navigational buttons were implemented for the user to navigate through the list of notifications. Based on the values which appeared they would then be able to decide which uplift terms might be necessary for their data-set. Figure 4.7 is an illustration of the applications front-end implementation.

[3]JDBC - Java Database Connectivity, an API defining how an application accesses a database (implemented specifically for Java).

```
1  private static void loadFromDatabase(ArrayList<Notification> list){
2    ...
3    try {
4    Class.forName("org.sqlite.JDBC");
5    c = DriverManager.getConnection("jdbc:sqlite:NotificationDB.db");
6    stmt = c.createStatement();
7    String sql = "SELECT * FROM notifications";
8    ResultSet rs = stmt.executeQuery(sql);
9    while(rs.next()){
10     Notification notif = new Notification();
11     notif.setId(rs.getInt(1));
12     notif.setSender(rs.getString(2));
13     String packageType = rs.getString(3);
14     notif.setPackageType(packageType);
15     notif.setSubject(rs.getString(4));
16     notif.setBody(rs.getString(5));
17     notif.setDateTime(formatDate(rs.getString(6)));
18     list.add(notif);
19   }
20   ...
21 }
```

FIGURE 4.6: Function for extracting the notification data from SQLite database.

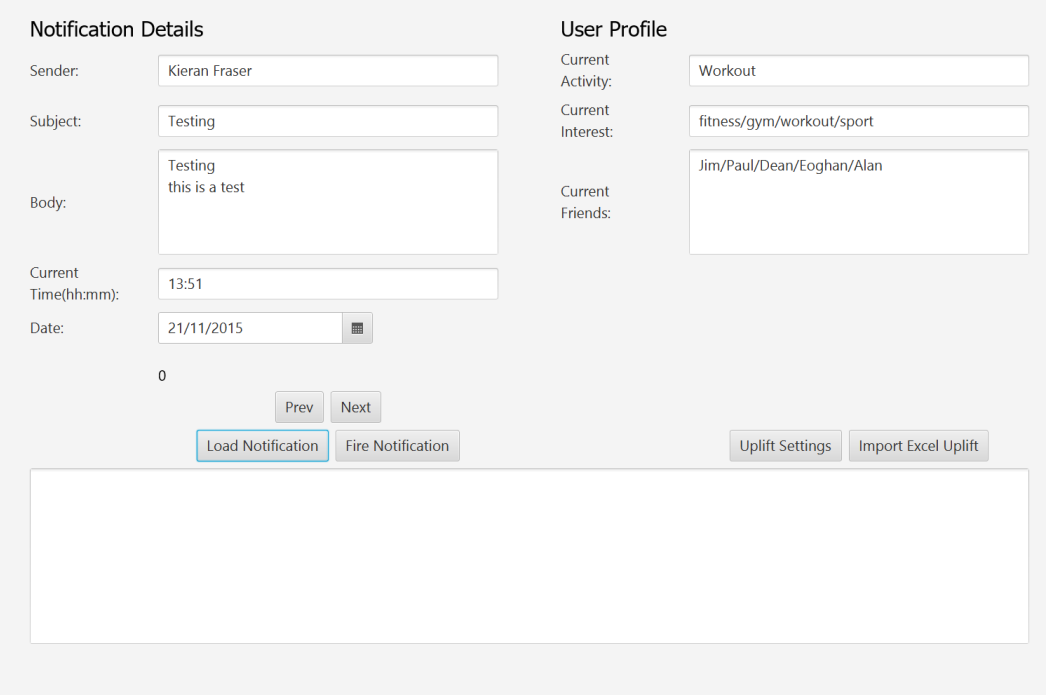


FIGURE 4.7: Screenshot of the NAbsUplift application through which notification data can be viewed.

At this point of the project, a prototype version of the info-bead model framework was also being developed and refined from within the data uplift application (as all the data necessary for the model's development was available for the first time in the

project). For this reason additional fields pertaining to user data were also created for the user to input various values and test the performance of the info-bead model. This was a good platform and starting point for developing the info-bead model as certain failures came about in implementing the model which were later rectified when developing the *NAbsDesktop* application. For example, the implementation of info-links were incorrectly developed, meaning the communication between the info-beads broke down: the push, pull and notify methodologies were not working correctly. This was later rectified by applying the observer design pattern (which will be discussed in greater detail in the next section).

The third requirement for the application which enables the user to edit the uplift terminology was implemented by creating an additional *JavaFX* window which contains editable text fields for all the uplift term categories: *Sender*, *Package* (which translates to the application the notification was sent through), *Subject*, *Body* and *Date*. The user can add or remove uplift terms for each category as they see fit. In order to persist the uplift terminology an *SQLite* database is created for the *NAbsUplift* application containing tables for each category. Each table has a column for uplifted terms and also a column for a corresponding ranking for a terms importance relevant to the particular user. This is effectively generating the source of data used within the inference mechanism in the *NAbsDesktop* application. The schema for the *SQLite* database are illustrated in figure 4.8 and the front-end GUI is displayed in figure 4.9.

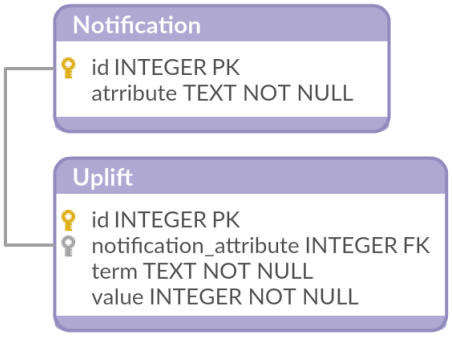


FIGURE 4.8: Database schema for the uplift terminology term set.

The final requirement of the data uplift application is to provide the functionality to export the notification data in an editable format to enable the user to uplift the data. This is achieved in the *NAbsUplift* application through the implementation of the *Apache POI* library which enables the reading and writing of *Microsoft Office* formats, through *Java*. The library is added as an external dependency using *Maven* and an additional *JavaFX* "Export" button is added to the user interface. Once clicked by the user the list of *Notification* objects is iterated over and the attributes of each notification are added to rows in an Excel spreadsheet, each attribute in an individual cell (lines 9-17). Adjacent to the cell with the notification attribute value a

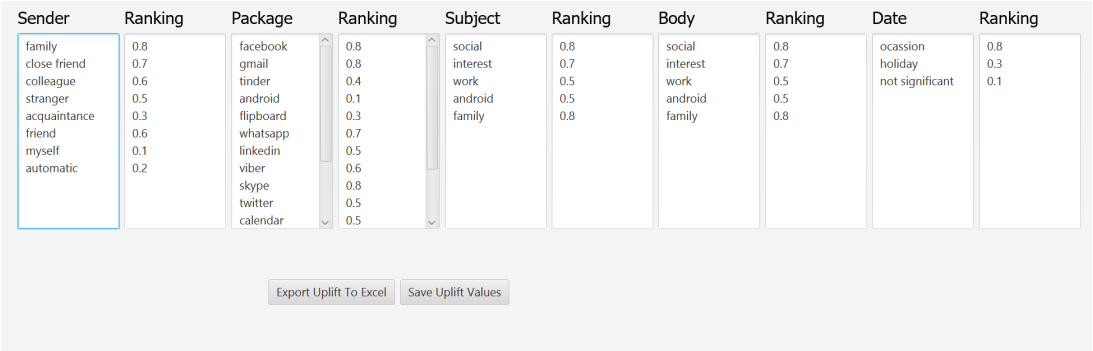


FIGURE 4.9: Screenshot of the NAbsUplift application through which the uplift terminology can be edited.

drop-down box is created and populated with the uplifted terminology for the particular category the attribute falls under (lines 18-23, figure 4.10). This is accomplished by querying the local *SQLite* database containing the uplifted terminology. The file is then exported as an *Excel* file (line 25).

```
1  private static void exportToExcel(ArrayList<Notification> notifications){
2    ...
3    XSSFWorkbook wb = new XSSFWorkbook();
4    XSSFSheet sheet = (XSSFSheet) wb.createSheet("Notification Uplift");
5    XSSFRow row;
6    int rowId = 0;
7    Cell cell;
8    CreationHelper createHelper = wb.getCreationHelper();
9    for(Notification nab : notifications){
10     row = sheet.createRow(rowId++);
11     cell = row.createCell(0);
12     cell.setCellValue(nab.getSender());
13     cell = row.createCell(2);
14     cell.setCellValue(nab.getPackageType());
15     cell = row.createCell(4);
16     ...
17   }
18   validationHelper = new XSSFDataValidationHelper(sheet);
19   CellRangeAddressList addressList = new CellRangeAddressList(0,
         notifications.size(),1,1);
20   constraint = validationHelper.createExplicitListConstraint(senderArray);
21   dataValidation = validationHelper.createValidation(constraint, addressList
         );
22   dataValidation.setSuppressDropDownArrow(true);
23   sheet.addValidationData(dataValidation);
24   ...
25   FileOutputStream fileOut = new FileOutputStream("uplift.xlsx");
26 }
```

FIGURE 4.10: Function for extracting the notification data from SQLite database.

This concludes the implementation of the functionality required for the *NAbsUplift* application. The user can subsequently uplift the notification data from the

resulting exported *Excel* file with minimum effort. Once the user has set each notification attribute an uplifted value, the original data value of the notification can be deleted, as it is now no longer needed. In this manner the user's sensitive information is kept private and never has to be sent to developers or those involved with the project.

4.3 NAbsDesktop

The main application to be developed in this project is the *Notification Management System* (NMS) which aims to effectively manage a users notifications by delivering them at a contextually relevant time. *NAbsDesktop* is the application discussed in this section and it is this project's implementation of a NMS.

The previous sections of this chapter discussed the development of two applications which were necessary to prepare the notification data which is pivotal for the *NAbsDesktop* application to work. Without the notification data, the application has no data to use for simulating incoming notifications. Similarly without the uplifted term-set and rankings, the application can't accurately infer the delivery times of the notifications. For this reason, the NMS implemented in this section is completely dependent on the accomplishments of the previous two applications. One of the future goals of this project would be for the capture and uplift of the notifications to occur in a single NMS application. However, this was not the aim of this project as the focus is largely on the performance of the info-bead model and fuzzy inference framework. Therefore, the requirements of this application focus on developing an application for exploring the performance of this framework, and are as follows:

1. Simulate incoming notifications.
2. Provide an interface for viewing and editing the current incoming notification.
3. Provide an interface for viewing and editing the current users state.
4. Display the final delivery decision to the user.

These are simply high level requirements for the *NAbsDesktop* application to function but they are dependent on the development of the underlying info-bead model and fuzzy inference framework, which will be discussed in detail throughout this section.

As with the previous application, *NAbsDesktop* is developed with the *Java* programming language and uses *Apache Maven* for managing external dependencies. The additional dependencies added to the *pom.xml*[4]:

[4]POM - *Project Object Model* are, an *XML* representation of a *Maven* project. All the dependencies of the project are defined in this file for example.

- EclispeLink - An open source persistence framework which enables *Java* to access, persist and manage the data of a database using *JPA*[5]. Within the *NAbsDesktop* application a number of objects need persisting, such as the individual info-beads for example. *EclipseLink* enables a POJO[6] to be created and mapped to the info-bead entity of the applications database.

- FasterXML - A *JSON* library for *Java* issued by the *Jackson Project*. This library is used within the *NAbsDesktop* application for converting *Java* objects to *JSON* and vice versa.

- Apache POI - A *Java* library for reading and writing *Microsoft Office* formats. This library is used within the *NAbsDesktop* application for importing the uplifted notification data from an *Excel* spreadsheet provided by the user.

- Google API Client - A library which provides all the functionality necessary to integrate any *Google* service with an application. In this application, *Google Calendar* is queried using this library in order to get a users schedule and subsequently create inferences for contextually relevant notification deliveries.

- jFuzzyLite - A fuzzy logic control library for *Java* [Rada-Vilela, 2014]. This library provides the components necessary for building the *Fuzzy Inference System* within the *NAbsDesktop* application.

- JavaFX - A GUI library for *Java* applications. This library is again, as with the previous application, used for implementing the front-end of the application.

The *NAbsDesktop* application can be broken down into two segments. The front-end of the application, built using *JavaFX* to provide the user with the results of the system, and the back-end of the application in which the info-bead model and fuzzy inference framework is implemented.

4.3.1 Back-end

The back-end of the application is made up of a number of components. The first component is that which handles the importation of the notification data which has, at this point, been uplifted and stored in an *Excel* spreadsheet. The function which handles this process implements, once again, the *Apache POI* library. Figure 4.11 illustrates an excerpt from the import function which iterates over all the notifications in the spreadsheet and adds them to a list of *UpliftedNotification* objects. The *UpliftedNotification* object entity describes the incoming notification which will be fired at the info-bead model.

[5]JPA - *Java Persistence API*, a *Java Object to Relational Mapping* (ORM) standard used for accessing and managing data between *Java* objects and relational databases.
[6]POJO - Plain Old Java Object.

```
1  ...
2  while (row!=null)
3  {
4    UpliftedNotification notif = new UpliftedNotification();
5    notif.setNotificationId(i);
6    Cell cell = row.getCell(1);
7    notif.setSender(cell.getStringCellValue());
8    notif.setSenderRank(getRank(sender, notif.getSender()));
9    ...
10   list.add(notif);
11   i++;
12 }
13 ...
```

FIGURE 4.11: Excerpt from the function which imports the notification data from the *Excel* spreadsheet.

Once the notification data has been loaded into the NMS, the next step is to create the applications info-beads. Recalling figure 3.4, the necessary info-beads for the system include the initial *Notification* info-bead, the notification-attribute info-beads (*Sender*, *Subject*, *Body*, *App* and *Date*) and an *Alert* info-bead.

As the info-beads must be persisted throughout the application, the first step regarding their development is to create a library of generic entity classes which each individual info-bead class can subsequently extend and/or implement. In this manner, the info-bead structure, main functional components and associating data (all discussed in Chapter 2) can be standardized and stored.

First created is a generic info-bead class which contains data attributes such as the info-beads name, communication mode, activation status, list of beads it is authorized to send or receive information from, keywords associated with the particular info-bead and its evidence/inferred data. The full database schema for the info-bead model is illustrated in figure 4.12 (an *SQLite* database is implemented for the *NAbsDesktop* application in order to persist the info-bead data). Also added to the info-bead class are the functions which are common to all info-beads such as:

- *inferInfoBeadAttr()* - the function which contains the logic for making inferences based on the provided evidence data and subsequently creating new atomic attributes within the info-bead.

- *storeInfoBeadAttr()* - the function which contains the logic for persisting the info-bead attributes in the database. Any operations that must be carried out on the data are done in this function before the info-bead is saved in the database.

In addition to the generic info-bead class, two separate *Java* interfaces[7] are also created:

[7] *Java* interface - similar to the *Java* class, but an interface can only contain function definitions. It cannot implement functions.

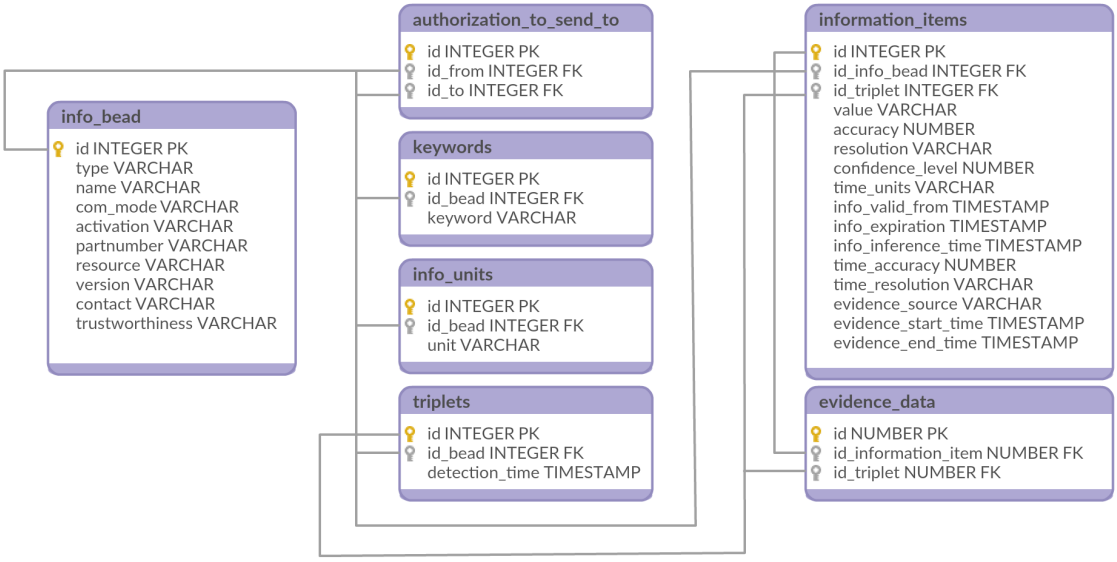


FIGURE 4.12: NAbsDesktop (info-bead model) database schema.

- *BeadInputInterface* - this interface contains one method definition, *getEvidence()*, which is used by each info-bead for receiving data (this could be via a push from another info-bead or additionally it could be called by the info-bead itself, in order to gather information from sensor sources such as the users *Google Calendar*).

- *BeadOutputInterface* - this interface also contains one method definition, *sendToConsumer()*, which is used by each info-bead for sending data to other info-beads or external sources.

The next step in development, is to create the individual classes for each info-bead illustrated in figure 3.4. For example, a *SenderInfoBead* class is created for the *Sender* info-bead. This class extends *InformationBead*, the generic info-bead class, and implements *BeadInputInterface*, *BeadOutputInterface* and *Runnable*[8](figure 4.13).

```
1 ...
2 @Entity
3 @DiscriminatorValue("Sender")
4 public class SenderInfoBead extends InformationBead implements
      BeadInputInterface, BeadOutputInterface, Runnable{
5 ...
```

FIGURE 4.13: Excerpt from *SenderInfoBead.java*.

All remaining individual info-bead classes are created in the same manner, giving each info-bead the basic functionality required for the info-bead model framework to

[8]Runnable - this is *Java* interface implemented by classes whose instances intend to be executed by a thread.

function, but enough flexibility that each individual info-bead can implement different logic and achieve its own unique task. Common functionality shared throughout all info-beads includes the *storeInfoBeadAttr()* function, the logic of which is applied generically, as it simply persists the *JPA Entity* (the info-bead class in question), and the *sendToConsumer()* function , which simply pushes inferred data to all other info-beads subscribed to updates from the invoked info-bead. Both functions are illustrated in figure 4.14.

```
1  @Override
2  public void storeInfoBeadAttr() {
3     EntityManager em = App.getEntityManager();
4      em.getTransaction().begin();
5      em.persist(this);
6     em.getTransaction().commit();
7  }
8
9  @Override
10 public void sendToConsumer(String senderId, Date sentTime, Triplet
       outputData) {
11    for(BeadInputInterface listener : senderListeners){
12       listener.getEvidence(senderId, sentTime, outputData);
13    }
14 }
```

FIGURE 4.14: Functions to save the info-bead values in the database and to push the inferred data to all other subscribed info-beads (*SenderInfoBead.java*).

EclipseLink handles the persistence of the POJO's. Hence, once a *persistence.xml*[9] file containing all individual info-beads is defined within the application, saving the info-bead objects can be concisely implemented via lines 3-6 in figure 4.14. Communication between info-beads is instrumented by the *sendToConsumer()* function. Once again recalling figure 3.4, it can be seen that, in this model, all illustrated info-links connecting the individual info-beads have a directional communication protocol of *push*. In order to implement this in the info-bead model framework, the *Observer* software design pattern is used. This pattern essentially propagates changes in one object through to a list of dependent objects. The *Observer* design pattern is implemented in the info-bead model framework as follows:

- A list of "listeners" is first added to each info-bead. These "listeners" are other info-beads that have subscribed to receive updates from the invoked info-bead. For example, the *Notification* info-bead has five other info-beads subscribed to its push events (figure 4.15). The type of object the list is defined to hold is one of *BeadInputInterface* which results in polymorphism[10] being achieved as multiple

[9] *persistence.xml* - the deployment descriptor file for persistence using JPA. It declares the classes to be persisted as well as the database connection details.

[10] Polymorphism - the ability to process objects differently depending on their data-type or class.

object types can now be added to the list as long as they have implemented the *BeadInputInterface*. This is necessary as, in the case of the *Notification* info-bead, there are five types of object being added to the list: *SenderInfoBead*, *SubjectInfoBead*, *BodyInfoBead*, *AppInfoBead* and *DateInfoBead*.

- At the point of instantiation of an info-bead, the list of subscribed "listeners" are added via an *addListener()* function defined within the info-bead (figure 4.16).
- A push event is executed from the info-bead's *sendToConsumer()* function, figure 4.14. Within this function, the list of subscribed "listener" info-beads are iterated over and for each info-bead, their *getEvidence()* function is invoked and data is exchanged through the function's input parameters.
- Depending on the purpose of the "listener" info-beads, once their *getEvidence()* function has been invoked, it can then carry out additional inferences on the evidence data received and pass it on to its own "listener" info-beads via the same method. In this manner the data is pushed through the info-bead model until it reaches an info-bead with the functionality to alert the user, if the inferences so permit.

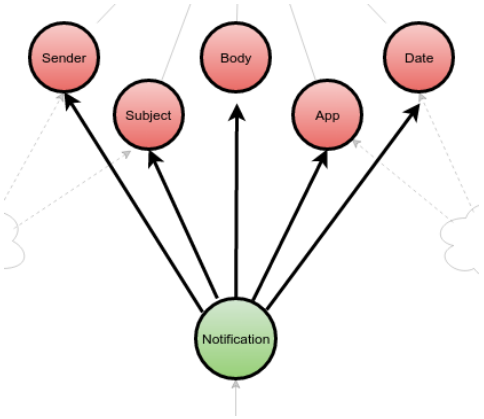


FIGURE 4.15: Info-bead model design, initial push pattern.

This concludes the generic functionality common to each info-bead in the framework. Individual info-beads have additional functionality contained within their *inferInfoBeadAttr()* and *getEvidence()* functions which are unique to the particular info-bead in question. For this project there was a limit to the number of info-beads that could be implemented completely within the scope of the project. Hence, the *Notification*, *Sender*, *Subject* and *Alert* info-beads were given priority as they complete a path from the uplifted notification through to the user with a well grounded inference on importance and context being made.

```
1  ...
2  private List<BeadInputInterface> senderListeners = new ArrayList<
       BeadInputInterface>();
3
4  public void addListener(BeadInputInterface bead){
5  this.senderListeners.add(bead);
6  }
7
8  public void removeListener(BeadInputInterface bead){
9  this.senderListeners.remove(bead);
10 }
11 ...
```

FIGURE 4.16: The *SenderInfoBead* "listener" list and functions for the *Observer* design pattern implementation.

Notification info-bead

The *Notification* info-bead is the uplifted notifications first point of contact with the info-bead model framework. The purpose of this info-bead is to distribute the uplifted notification to all other info-beads which require the information. In order to achieve this, additional purpose-specific logic is implemented within this info-bead:

- *receivedNotification()* is an additional function added to the *Notification* info-bead so that the NMS can simulate an incoming notification by passing it to the uplifted notification data. Within the *NAbsDesktop* application, the uplifted notifications are saved as *UpliftedNotification* objects - hence this is the input parameter to the *receivedNotification()* function. However, in the future, this can be as flexible as the system requires.
- Within the *receivedNotification()* function the uplifted notification data is converted to a *JSON* string which can then be stored in the *InformationItem* of the *Triplet* to be distributed to all other subscribed info-beads (figure 4.17).

Sender & Subject info-beads

These notification-attribute info-beads both make inferences on the evidence data, the incoming uplifted notification, provided by the *Notification* info-bead. The inferences made are specific to the attribute of the notification the info-bead has been created to model. Hence, within the *Sender* info-bead, inferences will be made surrounding the importance and context of the sender of the notification. Both info-beads make inferences using a similar Fuzzy Inference System (FIS) which is invoked within the *inferInfoBeadAttr()* function. As an example, the logic within the *SenderInfoBead* object is as follows:

```
1  public void notificationReceived(UpliftedNotification notification){
2
3      Date receivedNotificationDate = new Date();
4      Triplet operational = new Triplet();
5      operational.setDetectionTime(receivedNotificationDate);
6
7      InfoItemFields information = new InfoItemFields();
8      ObjectMapper mapper = new ObjectMapper();
9      String notificationString = null;
10     try {
11     notificationString = mapper.writeValueAsString(notification);
12     } catch (JsonProcessingException e) {
13     e.printStackTrace();
14     }
15     information.setInformationValue(notificationString);
16     information.setInfoValidFrom(receivedNotificationDate);
17
18     operational.setInformationItem(information);
19     this.setOperational(operational);
20
21     storeInfoBeadAttr();
22     sendToConsumer(this.getId(), receivedNotificationDate, operational);
23 }
```

FIGURE 4.17: Function used by the NMS for pushing an incoming notification into the info-bead model framework (*NotificationInfoBead.java*)

```
1  @Override
2  public void getEvidence(String senderId, Date sentTime, Triplet inputData) {
3      ObjectMapper mapper = new ObjectMapper();
4      try {
5      notification = mapper.readValue(inputData.getInformationItem().
           getInformationValue(),
6      UpliftedNotification.class);
7      } catch (IOException e1) {
8      e1.printStackTrace();
9      }
10
11     try {
12     events = GoogleCalendarData.getNextNEvents(10, notification.getDate());
13     } catch (IOException |  ParseException e) {
14     e.printStackTrace();
15     }
16     this.run();
17 }
```

FIGURE 4.18: Function invoked when an info-bead is pushed data (*SenderInfoBead.java*)

- As the *SenderInfoBead* is subscribed to push events from the *Notification* info-bead, its *getEvidence()* function will be invoked through the *Notification* info-beads *sendToConsumer()* function. Once this occurs the notification data, contained within the *Triplet* passed to the info-bead, is accessible (lines 4-10, figure 4.18). This evidence data is stored locally in the *SenderInfoBead* and additional

sensor sources are polled for additional contextual data. In this case, the users' *Google Calendar* is queried and the users' current schedule is pulled down and stored in the form of a list of *Event* objects (lines 12-17). All the data necessary for making inferences is now stored within the info-bead and the FIS can function. The FIS is started via the *run()* method. Hence, the computational overload involved in making the inferences is distributed to a separate thread.

- Within the *run()* method, four main tasks are dealt with in a specific order.

  1. If there have been no errors with getting the data (recall that if data isn't available then there is no need for the info-bead to be active as inferences can't be calculated) then the state of the info-bead is set to active.
  2. The info-bead *inferInfoBeadAttr()* method is invoked.
  3. The info-bead *sendToConsumer()* method is invoked so as to send the inferred information to all subscribed info-beads, thus pushing the data further up the info-bead model chain.
  4. The info-bead *storeInfoBeadAttr()* method is invoked so as to persist the state of the info-bead.

The *storeInfoBeadAttr()* and *sendToConsumer()* functions remain unchanged from implementations previously discussed, but the *inferInfoBeadAttr()* function will differ for each info-bead depending on the information they attempt to infer. The *SenderInfoBead inferInfoBeadAttr()* function is illustrated in figure 4.19.

Recalling from Chapter 3 - Section 3.4.4 - Fuzzification, the two crisp inputs of the FIS, *attributeImportance* and *eventRelevance* must be in a specific format: a real number between 0 and 1. Lines 4-10 of figure 4.19 illustrates the logic which converts the *Event* objects retrieved from the *Google* calendar query to the correct crisp input format. This is done through the *getEventImportance()* method which is passed the list of *Event* objects, the attribute the info-bead is making inferences on (the "Sender", in this case), and the uplifted notification data. Within this function the complete list of *Event* objects is iterated over and a value is assigned to each event based on its:

1. contextual relevance with the sender of the notification - found by attempting to match the uplifted sender value from the notification with a value in the events description (lines 1-6 , figure 4.20). A value of 0 is assigned to the event if there is no match. If there is a match, the value is assigned using point 2.

2. contextual relevance between the time the notification was delivered and the start time of the event - a value is derived for this using the following equation (implemented in lines 8-15, figure 4.20):

$$1 - \frac{currentEventDiff}{maxEventDiff} \tag{4.1}$$

```
1  @Override
2  public void inferInfoBeadAttr() {
3
4      double eventInput = EventInference.getEventImportanceValue("Sender",
           events, notification);
5
6      if(eventInput == 0.0){
7      eventInput = 0.00001;
8      }
9
10     SenderFuzzy senderFuzzy = new SenderFuzzy();
11     double senderInput = (double) notification.getSenderRank()/10.0;
12     double inferredValue = senderFuzzy.processSender(senderInput, eventInput
           );
13
14     Triplet operational = new Triplet();
15     InfoItemFields info = new InfoItemFields();
16     info.setInformationValue(String.valueOf(inferredValue));
17     operational.setInformationItem(info);
18     operational.setDetectionTime(new Date());
19     this.setOperational(operational);
20 }
```

FIGURE 4.19: The *Sender* info bead inference function (*SenderInfoBead.java*)

In equation 4.1 *currentEventDiff* is the difference in time (minutes) between the incoming notification and the event start time, and *maxEventDiff* is the difference in time (minutes) between the incoming notification and the last event received (the event furthest away in the users schedule. **Note**: a fixed number of events can only be retrieved from the *Google Calendar*, and in this project the number is set at 10 - which relates to approximately 2 days for the two volunteers. This value should be dynamic depending on the sparsity of a users calendar).

By placing a value on each event using the above two methods, events are essentially ranked depending on how far into the future they are, and whether or not they are relevant to the sender of the notification. Therefore, the crisp input value is the value of the event with the maximum ranking (*cel* = contextual event list: those events which had a match between the uplifted sender value of the notification and the event description):

$$max_{cel}(1 - \frac{currentEventDiff}{maxEventDiff}) \tag{4.2}$$

Line 11 of figure 4.19 illustrates the remaining crisp input, *attributeImportance*, being prepared for the FIS. This is the ranking given to the uplifted value by the user and is stored within the *SQLite* database within *NAbsDesktop* (editable via the user-interface provided, which is discussed in section 4.3.2). With both input values

```
1  private static boolean hasContextMatch(String calendarDescr, String
       notification, String attribute){
2      if(calendarDescr.contains(notification)){
3        return true;
4      }
5      else return false;
6  }
7
8  private static double applyRating(CalendarEvent event,
9      long maxTimeDiff, long incomingTimeDiff){
10     if(incomingTimeDiff < 0 ){ // ensure that the ongoing event is
           calculated correctly
11       incomingTimeDiff = 0;
12     }
13     double result = (double) (maxTimeDiff - incomingTimeDiff)/maxTimeDiff;
14     return result;
15 }
```

FIGURE 4.20: Two functions which contribute to converting a users schedule to a crisp input value for the FIS (*EventInference.java*)

now in the correct format, they can be passed to the FIS through function *senderFuzzy.processSender()*, in order to determine the relevance and importance of the sender attribute (line 12).

The FIS is implemented in the project using the *jFuzzyLite* library. A fuzzy inference class is created for each info-bead that requires a FIS (e.g. *SenderFuzzy*). On initialization of the *SenderFuzzy* class for instance, the FIS is initialized and parameters for the 4 stages of a FIS (*Fuzzification*, *Inference*, *Composition* and *Defuzzification*) are configured:

- *Fuzzification* - the membership functions for the two crisp inputs and the corresponding *senderRelevance* output are implemented as illustrated in figure 4.21. Lines 4-6, 11-12 and 20-22 define the shapes and limits of all the linguistic variable functions. These correspond to the membership functions developed with the *Matlab Fuzzy Logic Toolbox*, which were illustrated as examples in Chapter 3 (also found in the Appendix).

- *Inference* - the knowledge base containing the heuristic rules is implemented as illustrated in lines 1-8 of figure 4.22.

- *Composition* & *Defuzzification* - these two stages of the FIS are configured on line 10 of figure 4.22. The fourth input parameter to the *engine.configure()* method, is the accumulation configuration (defining how the output fuzzy sets of all the fired rules are aggregated). The parameter is defined as "Maximum", which is an S-norm (models the union of fuzzy sets demonstrated in the examples of Chapter 3 - figure 3.12). The fifth parameter to the *engine.configure()*

```
1  senderImportance = new InputVariable();
2  senderImportance.setName("SenderImportance");
3  senderImportance.setRange(0.000, 1.000);
4  senderImportance.addTerm(new Triangle("NIP",0.000,0.000,0.400));
5  senderImportance.addTerm(new Triangle("IMPORTANT",0.200,0.500,0.800));
6  senderImportance.addTerm(new Triangle("VIP",0.600,1.000,1.000));
7
8  eventRelevance = new InputVariable();
9  eventRelevance.setName("EventRelevance");
10 eventRelevance.setRange(0.000, 1.001);
11 eventRelevance.addTerm(new Rectangle("NOTRELEVANT", 0.000, 0.5, 1.0));
12 eventRelevance.addTerm(new Triangle("RELEVANT", 0.5, 1.001, 1.0));
13
14 senderRelevance = new OutputVariable();
15 senderRelevance.setEnabled(true);
16 senderRelevance.setName("SenderRelevance");
17 senderRelevance.setRange(0.000, 1.000);
18 senderRelevance.fuzzyOutput().setAccumulation(new Maximum());
19 senderRelevance.setDefuzzifier(new Centroid(200));
20 senderRelevance.addTerm(new Triangle("LOW",0.000,0.250,0.500));
21 senderRelevance.addTerm(new Triangle("MEDIUM",0.250,0.500,0.9));
22 senderRelevance.addTerm(new Triangle("HIGH",0.400,1.000,1.000));
23 engine.addOutputVariable(senderRelevance);
```

FIGURE 4.21: The *attributeImportance*, *eventRelevance* and *senderRelevance* membership function implementations (*SenderFuzzy.java*).

method is the *defuzzification* method implemented, and in this case, for reasons discussed in Chapter 3, the "centroid" method is chosen.

```
1  RuleBlock ruleBlock = new RuleBlock();
2  ruleBlock.addRule(Rule.parse("if SenderImportance is NIP and EventRelevance
       is NOTRELEVANT then SenderRelevance is LOW", engine));
3  ruleBlock.addRule(Rule.parse("if SenderImportance is IMPORTANT and
       EventRelevance is NOTRELEVANT then SenderRelevance is HIGH", engine));
4  ruleBlock.addRule(Rule.parse("if SenderImportance is VIP and EventRelevance
       is NOTRELEVANT then SenderRelevance is HIGH", engine));
5  ruleBlock.addRule(Rule.parse("if SenderImportance is NIP and EventRelevance
       is RELEVANT then SenderRelevance is LOW", engine));
6  ruleBlock.addRule(Rule.parse("if SenderImportance is IMPORTANT and
       EventRelevance is RELEVANT then SenderRelevance is HIGH", engine));
7  ruleBlock.addRule(Rule.parse("if SenderImportance is VIP and EventRelevance
       is RELEVANT then SenderRelevance is HIGH", engine));
8  engine.addRuleBlock(ruleBlock);
9
10 engine.configure("Minimum", "", "Minimum", "Maximum", "Centroid");
```

FIGURE 4.22: The knowledge base of heuristic rules implemented in the FIS (*SenderFuzzy.java*).

Once the FIS has been configured, the crisp inputs are passed to the *senderFuzzy.processSender()* function (line 12, figure 4.19). This function is defined in *SenderFuzzy.java* and simply passes the two input parameters through the FIS, and

returns the final crisp inferred output value to the info-bead which sends the value on-wards through the info-bead model via the *sendToConsumer()* method. This concludes the description of the additional functionality implemented in the notification-attribute info-beads. Each FIS is unique to the specific info-bead it is modeled for. However, the methodology used does not change. A similar FIS is also implemented in the *Alert* info-bead, but in contrast to the example above, it has a greater amount of crisp input values (these will be the crisp inferred output values from each individual notification-attribute info-bead). Hence also a greater amount of rules in its knowledge base.

Alert info-bead

The final piece of functionality to discuss with regard to the back-end of the *NAbsDesktop* application and info-bead model framework, is the logic which deals with converting the final crisp output value from the FIS in the *Alert* info-bead to a contextual delivery point in the users schedule. This is done by once again iterating over the user's scheduled events which were drawn from their *Google Calendar*. Depending on the threshold the inferred value falls under, a different logic will apply (discussed in Chapter 3, section 3.4.5). The threshold values map to linguistic variables of *now*, *very soon*, *soon*, *later* and *much later* which are implemented in the FIS - hence it is the FIS which determines which bracket the notification falls under and ultimately how the notification is delivered. Figure 4.23 illustrates the implementation.

Finally, line 26 of 4.23, sends the result to the front-end of the application. In the case of *NAbsDesktop*, the front-end of the application is simply for testing and evaluation purposes, the functionality of which will be discussed briefly in the following section.

4.3.2 Front-end

The front-end of the application was implemented using the *JavaFx* platform to provide an interface for simulating notification delivery. This includes functionality for loading a data-set of existing notifications stored in an *Excel* spreadsheet, as well as creating ad-hoc notifications from within the application interface. An interface for editing the data-set of uplifted term rankings is also provided, as well as functionality for the application to provide feedback on the notifications inferred delivery time once a notification has been passed through the NMS.

Figure 4.24 illustrates the home screen of the *NAbsDesktop* application whereby a user can choose from a number of notification data-sets provided by both volunteers participating in the project.

Once a data-set is chosen, the user can navigate to the simulation screen, figure 4.25, where a notification's attributes can be edited if desired before submitting it for

```
1  // now - interrupt
2  if(inferredValue<6.0){
3     result = result + "Notify now "+this.getPartNumber()+"\n";
4  // verysoon - next break
5  }else if(inferredValue<60){
6    if(userLocation == 1.0){ // if there's an event on
7      result = result + "at next break - "+App.getNextBreak()+" - "+this.
           getPartNumber()+"\n";
8    }
9    else{
10    result = result + "Notify now "+this.getPartNumber()+"\n";
11   }
12 // soon - next free period
13 }else if(inferredValue<90){
14   if(userLocation == 1.0){
15     result = result + "Notify next free period - "+App.getNextFreePeriod()+"
           - "+this.getPartNumber()+"\n";
16   }
17   else{
18       result = result + "Notify now "+this.getPartNumber()+"\n";
19   }
20 // Later & Much Later
21 }else{
22   result = result + "Notify next contextual relevant event - "+App.
       getNextContextRelevant()+" - "+"\n";
23 }
24 System.out.println(result);
25 App.result = result;
```

FIGURE 4.23: The implemented logic for finding a point in a user's schedule where a delivery should be made (*AlertInfoBead.java*).

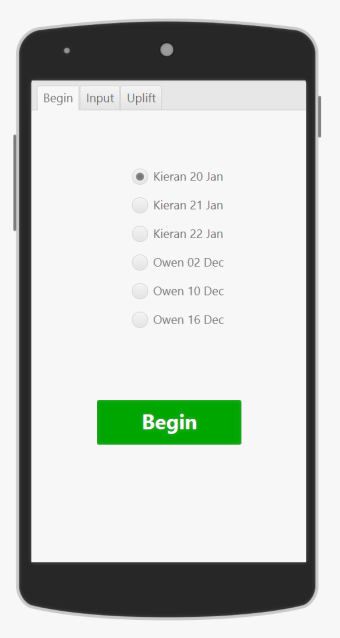


FIGURE 4.24: The *NAbsDesktop* home screen.

simulation of delivery. An interface is also provided which shows the user's current status. This takes the form of an event from the user's *Google Calendar*. If the user had no event scheduled at the time of the notification delivery, then the user attributes will be unknown.

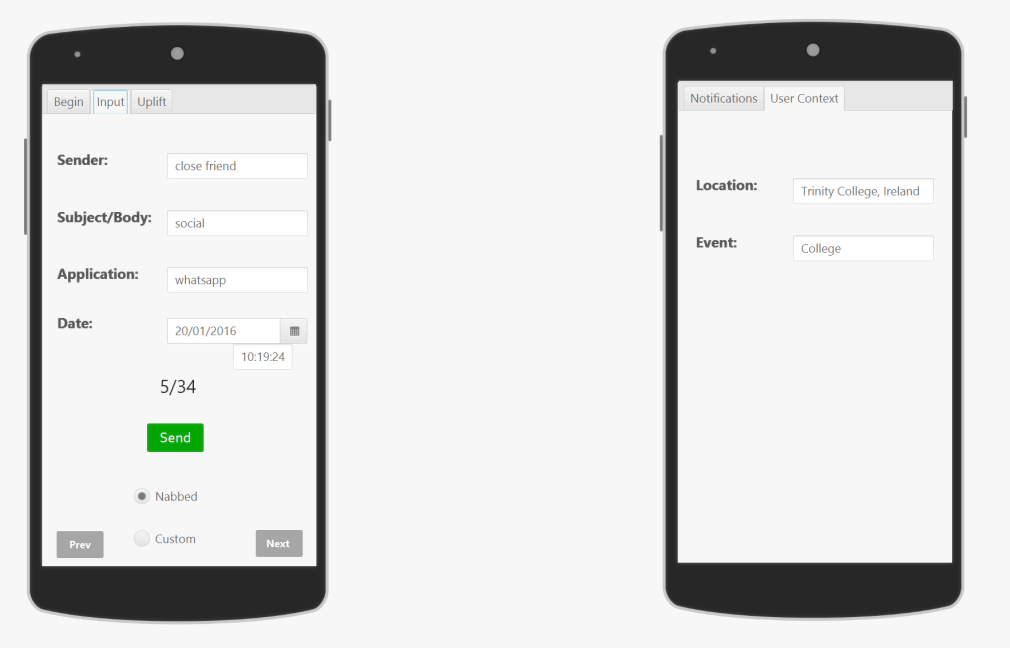
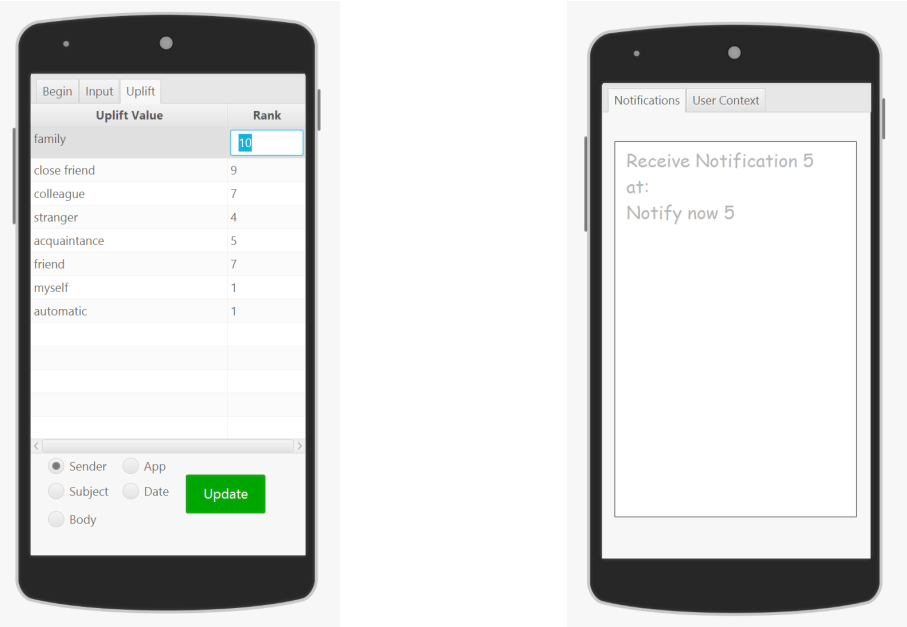


FIGURE 4.25: The *NAbsDesktop* simulation screen.

Figure 4.26a depicts the ranking screen of the application where a user can update the rankings of the notification uplift terminology which is used throughout the inference process. Lastly, figure 4.26 illustrates the notification delivery screen where the result of the NMS appears via a message detailing when the notification would arrive.

4.4 Summary

Throughout this chapter the implementation of the various components contributing to the NMS, were discussed. This included the development of a number of applications involved in the data capture, data preparation and concept evaluation of the project. The development of these applications were structured by the requirements set out in Chapter 3 - design. The following is a brief summary of the application implementation:



(A) The rankings uplift screen. (B) The notification delivery screen.

FIGURE 4.26: The *NAbsDesktop* application.

1. NAbsMobile

   As discussed, this application was first to be implemented and was for the purposes of data capture. This mobile application was developed, using *Android*, for installation on the phones of two volunteers of the project. Data was gathered on a local *SQLite* database within the phone and extracted via USB. Privacy and security were two key aspects of the implementation.

2. NAbsUplift

   This application was implemented for the purposes of enabling the volunteers to uplift their own data without sensitive information being divulged. As described, it was developed as a desktop application using *Java* and *JavaFx*. An *SQLite* database was created to store the uplift terminology and *Apache POI* was used to convert the data in the exported *SQLite* database to an *Excel* spreadsheet for uplift. Key aspects surrounding this application were once again privacy and security, but also scalability.

3. NAbsDesktop

   The final application was the NMS, which integrated the prepared data from the previous two applications, in order to simulate incoming notifications, and attempt to manage them on behalf of the user by delivering them at the most contextually relevant time. It was developed using *Java* and *JavaFx* once again, and also implemented a *SQLite* database with which to persist the data. User data was integrated with the system by querying a RESTful API using the

Google Client library. *Apache POI* was used to import the *Excel* spreadsheet of uplifted notification data provided by the user. In order to model and manage both the notifications and the user, an info-bead model framework was created, which included a *Mamdani* FIS (implemented using the *jFuzzyLite* library).

Chapter 5

Evaluation

5.1 NAbsMobile

The *NAbsMobile* application was the proposed implementation of the first stage of the *Notification Management Pipeline* (figure 1.1), "Intercept incoming notifications". This was the data capture stage of the project and it was an important step, as the NMS required real users' notification data in order to evaluate how it would perform under real-world conditions. The *NAbsMobile* application was installed on the device of both volunteers (the author and the supervisor) for a period of approximately 10 weeks (68 days) from November 30th 2015 to February 5th 2016. The following sections are the evaluation of various aspects of the project.

5.1.1 Method

The best method of evaluating the *NAbsMobile* application is through evaluating the resulting data-sets (obtained from the two volunteers) which were acquired through the application. To accomplish this, there are a number of key data quality metrics relevant at this stage of the project which are considered [Pipino, Lee, and Wang, 2002].

1. *Accessibility*

   The *accessibility* of the data is the degree to which data is available and easily accessible. This was an important design aspect of the application, as the data had to be easily and quickly extracted from the device for the uplift stage of the project. For the uplift to be successful the user needed to have a clear idea of the context surrounding the notifications. Hence, the turnaround on getting the data from the *NAbsMobile* application into the *NAbsUplift* application was important. As discussed in the previous chapter, there is an interface through which a user can view the accumulated data-set at any time on their mobile device. The data itself however, can't easily be manipulated or edited while on the mobile phone - hence there is also functionality to export the data to a desktop device through a USB connection.

2. *Believability*

   The *believability* of the data is the extent to which the data is trustworthy. In this case, the application was designed to extract the data from the notification and log it exactly as it appeared, with no interaction from the user. This is an important dimension to measure, as the trustworthiness of the data is critical to the performance of the following two applications involved in the project. If the data is not to be trusted by the users', then the uplift process could be compromised by the user deciding a notification-attribute value is wrong, and thus uplifting it incorrectly. Also, as inferences are being later made on the notification data captured by the *NAbsMobile* application, if it is considered untrustworthy, then the NMS can't hope to satisfy a user by recommending a delivery time for a notification.

3. *Completeness*

   The *completeness* of the data-set is the degree to which data is not missing. There were five key attributes that the *NAbsMobile* application attempted to log from a users incoming notification: the sender, the subject, the body (of the message), the application and the date/time the notification was delivered to the user. *Completeness* is an important metric to track, as it ensures the *NAbsMobile* application is logging and storing the data as per the functionality entails. However, there is some flexibility allowed in the result of analyzing *completeness*, as some notification types simply lack data. For example, some emails are sent without a subject line because the sender forgot to assign one while sending the notification. This is not an error in the quality of the data, but a characteristic of the nature of the notification data. Hence, a low *completeness* value for the data-set may be interpreted as acceptable in this case.

The above three data quality dimensions will be used to assess both data-sets acquired from the *NAbsDesktop* application, in order to ascertain its performance at capturing a user's real world notifications.

5.1.2 Procedure

The procedure for carrying out the data quality assessments outlined above, involves applying the following functions [Pipino, Lee, and Wang, 2002] to each obtained data-set:

1. Accessibility

$$max(0, 1 - \frac{t_d - t_r}{t_e - t_r}) \tag{5.1}$$

   - t_d = the time the data was delivered to the user.
   - t_r = the time the data was requested by the user.

- t_e = the time from which the data is no longer useful (expired).

2. Believability

$$min(T_{ds}, T_c, T_e) \tag{5.2}$$

- T_{ds} = a trustworthiness rating (between 0 and 1) associated with the data source. In this case, the data source being the *NAbsMobile* application.
- T_c = a trustworthiness rating (between 0 and 1) associated with a common standard. In this case the common standard is the *Android* system which normally sends the users their notifications.
- T_e = a trustworthiness rating based on experience. In this case, a rating based on the users own experience with other applications which offer a similar service to *NAbsMobile* (gives the user a notification which originated from another application e.g. *Chrome* browser notifications).

Note: ratings were given by the two volunteers to whom the data belonged.

3. Completeness

$$1 - \frac{cells_{empty}}{cells_{total}} \tag{5.3}$$

- $cells_{empty}$ = the number of empty cells in the data-set.
- $cells_{total}$ = the number of total cells in the data-set.

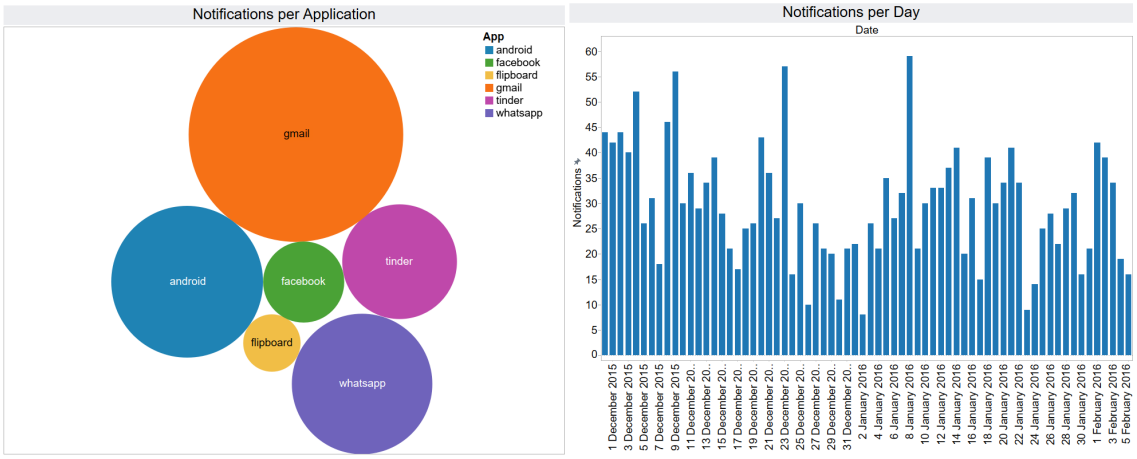
5.1.3 Results & Observations

The resulting output from the *NAbsUplift* application were two data-sets, one from each volunteer within the project:

1. Author's data-set

   This data-set had a total of 1982 notifications over the period of 68 days. The average number of notifications per day therefore amounts to 29.1, which translates to approximately 29 interruptions to the user throughout a single day, which is quite significant.

   Through studying the data-set which was procured through *NAbsMobile*, trends surrounding notification delivery can be derived for a particular user. For example, figure 5.1a illustrates the applications through which the user receives notifications, with the most popular applications being most apparent as the largest "bubble". Analysis into the application usage of the user can help detect which application is used on a regular basis, and which applications could be reserved for specific purposes. For example, in the context of notification delivery,



(A) Notifications by applications. (B) Notifications by day.

FIGURE 5.1: Analysis of which applications were most popular for delivery of notifications and a breakdown of the number of notifications per day over the 68 day period.

if a notification is sent through an application that is used solely for "family" purposes, then this individual application could receive a higher ranking than others. In this manner, notifications can be understood to a greater degree and delivered to the user more accurately. Similarly with 5.1b, specific types of days could be categorized, depending on the number of notifications being delivered, and the load could be distributed more evenly within the context of the user's schedule. Insights such as that also illustrated in 5.2, which gives a breakdown of the type of notification subjects which are delivered per hour, give not only an increasingly granular view of notifications but also of the user themselves. Hence the derived user model can also be improved using this data. In the scope of this project, analysis on the data-sets was not implemented, as it shifts focus from the performance of the info-bead model and fuzzy inference framework. However, it is recommended that this data analysis should be used as input to the system to replace the manual ranking system currently in place. More will be discussed on this matter in chapter 6.

The following are the results of the data quality metrics discussed in the previous section:

(a) Accessibility

The results from the *accessibility* metric are good, due to the fact that t_e is quite large, compared to the time required to access the data ("access the data", in this case, meaning exporting the *SQLite* database from *NAbsMobile* to the computer and browse/edit it using *NAbsUplift* or an *SQLite* browser). t_e, in this case, is a flexible value, as the notification data will only begin to become less useful once the user has forgotten the context

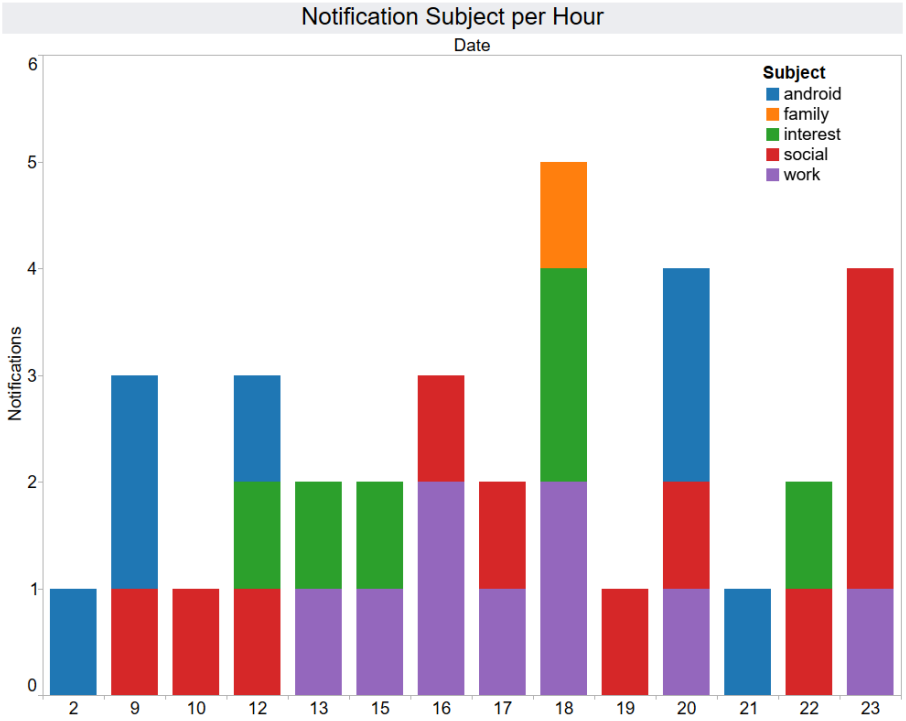


FIGURE 5.2: Analysis of the notification subject breakdown for a particular day (January 20th).

which surrounds it. Hence, while a low value ranging from a few days to a week would be recommended, it is not strictly required. The value for t_d was, experimentally, found to be approximately 4.3 minutes. This essentially means that the fraction in equation 5.1 will tend toward 0 and the equation will evaluate to an approximate *accessibility* of maximum value 1.

(b) Believability

The result of the *believability* metric for this data-set is 0.6. This is the minimum value found of the three believability variables from equation 5.2 for this data-set rated by the author: T_{ds} = 0.8, T_c = 0.9, T_e = 0.6. This value translates to the data source being 60% trustworthy, which means there is room for improvement on the transparency of the application. Potentially this value indicates that, if a user is skeptical with regard to the competency of a system to collect their notification data without error, a user may not trust the full management of their notifications to a NMS.

(c) Completeness

The result of *completeness* for this data-set is 0.99 (of a maximum value of 1). For equation 5.3 it was found that $cell_{empty}$ = 98 and $cell_{total}$ = 9910. The 98 empty cells were all due to an error in the *Sender* column of the data-set. It was found that some, but not all, of the *Facebook Messenger* notifications

were being logged without a sender. This is simply a fault between how some notifications are being captured by the *NAbsMobile* application. The value of completeness for the data-set in total is quite good and ensures that the uplift process has maximum potential of being correct.

2. Supervisor's data-set

   This data-set had a total of 1192 notifications over the period of 68 days. The average number of notifications per day therefore amounts to 17.5. It must be noted that for this user, not all social media user accounts had push notifications active during the 68 day period. Hence, this would not be a complete picture of the number of notifications this user receives on a daily basis. Nonetheless, 17 distractions throughout a day is a significant amount. The accessibility of this data-set is identical to the previous data-set. The following are the results of the remaining metrics:

   (a) Believability

   The result of the *believability* metric for this data-set, is 0.5. This is the minimum value found of the three *believability* variables from equation 5.2 for this data-set rated by the supervisor: T_{ds} = 0.6, T_c = 0.8, T_e = 0.5. This value translates to the data source being 50% trustworthy, which is a further 10% less than the previous data-set, again highlighting the emphasis which needs to be placed on further transparency throughout the NMS, and also perhaps empowering the user to have a greater input into the delivery mechanism. There is an element of this already designed in the *NAbsDesktop* application, as the user must enter a ranking of the uplifted terms. However, this could be expanded.

   (b) Completeness

   The result of *completeness* for this data-set is 0.99 (of a maximum value of 1). For equation 5.3 it was found that $cell_{empty}$ = 61 and $cell_{total}$ = 5960. The 61 empty cells were made up of 60 errors in the *Sender* column and 1 error in the *Subject* column. Again the errors in the *Sender* column can be accounted for by the fault in *NAbsMobile* discussed previously, and the error in the *Subject* column is simply due to an email sent without a subject line (which is not a data quality issue as it is a characteristic of this particular notification). The value of *completeness* for the data-set in total, is again, quite good for this data-set.

This concludes the evaluation of the *NAbsMobile* application. The application functioned as per required and the resulting quality of the obtained data-sets were acceptable for use in the NMS. The insights into user behavior and the nature of

notification delivery gained from analyzing these data-sets, was also invaluable, and will become useful in future work on this project.

5.2 NAbsUplift

The *NAbsUplift* application is used for preparing the notification data for input into the NMS. This application ensured that user data remained secure, while also enabling the uplift process to occur quickly and with minimum effort for the user. Another key aspect of this application was to also maintain the uplift terminology, and gather a users ranking on the uplift terms, which is pivotal for the inference mechanism of the NMS. Subsequently, the output of this application was two data-sets, which were to be input to the NMS.

There is less to evaluate with this application, as its functionality is less verbose than that of *NAbsMobile*. The main purpose behind this application is to protect the privacy of the users' data, while extracting it from the *SQLite* database output from *NAbsMobile* and prepare it for manual uplift by the user (and enable the project to easily scale to a greater number of test users if necessary, while maintaining an acceptable level of data privacy).

The application succeeded in these tasks as the notification data is extracted and placed in an easily editable form, an *Excel* spreadsheet, for the test users to uplift. A key aspect of this application was also the management of the uplifted terms. The term set, having being stored in an *SQLite* database, was easily accessible via the interface of the application.

In conclusion, the *NAbsUplift* application was functional, coherent and added increased privacy to the users' data, by minimizing the exposure of the sensitive data to others, and enabling the test user to carry out their own manual data uplift.

5.3 NAbsDesktop

NAbsDesktop is the application which encapsulates the info-bead model and fuzzy inference framework, whose performance with regard to inferring contextual delivery times (given the users notification input data) is the main interest of this project. The following sections discuss the results of the NMS and a number of observations which were made regarding these results.

5.3.1 Method

The method of evaluating the performance of the NMS, was to select a number of interesting days from both the author and supervisor's data-sets, and compare the simulated results from the NMS with the expected results created by the two volunteers. This required the following steps:

- Using the *Google Calendar* schedule of both volunteers, select a number of characteristically different days for evaluating the NMS. This includes days which were well cataloged in *Google Calendar*. as well as days which were closer to empty and increasingly vague.
- Both volunteers assign a value to each notification before it is simulated within the NMS. This value represents the expected delivery method they believe best matches the context of their notification. The assigned value is limited to a number within the range of 1 - 5, each figure representing a delivery method as such:
  1. Deliver the notification immediately.
  2. Deliver the notification at the next break in the user's schedule.
  3. Deliver the notification at the user's next free period (nothing scheduled for a period greater than 30 minutes).
  4. Deliver the notification at the next contextually relevant event.
  5. Deliver the notification at the next contextually relevant event (reserved generally for notification's of least importance).
- Each notification is simulated using the NMS and the result is recorded and compared to the expected value. Any discrepancy between the expected and simulated result is then evaluated by the volunteer and either accepted as an error or categorized as a permissible error. A permissible error is one which does not match the expected result, but also one which the user will accept in the context of the notification. Through this comparison two new metrics can be calculated:
  1. Strict Correctness Ratio

$$\frac{correct_{total}}{notifications_{total}} \times 100 \tag{5.4}$$

     This is the percentage of correct inferences made by the NMS over the total number of notifications. This value illustrates the effectiveness of the system to correctly categorize the notifications in a manner such that replicates the user themselves. Naturally, the higher the SCR percentage, the more intelligent the system.
  2. Permissible Correctness Ratio

$$\frac{(correct_{total} + permissible_{total})}{notifications_{total}} \times 100 \tag{5.5}$$

     This is the percentage of the sum of correct deliveries and permissible errors over the total number of notifications. This is a slightly more flexible

Date	Correct Delivery	Permissible Errors	Undeniable Errors	Total Notif.	S.C. Ratio	P.C. Ratio
Jan 20th	21	9	4	34	**61.7%**	**88.2%**
Jan 21st	26	5	10	41	**63.4%**	**75.6%**
Jan 22nd	26	5	3	34	**76.4%**	**91.1%**
Jan 25th	18	2	5	25	**72%**	**80%**
Jan 26th	20	5	3	28	**71.4%**	**89.2%**
Jan 27th	20	0	2	22	**90%**	**90%**
Jan 28th	19	4	6	29	**65.5%**	**79.3%**
Jan 29th	26	4	2	32	**81.2%**	**93.7%**
Avg:	22	4.3	4.4	30.6	**72.7%**	**85.8%**

TABLE 5.1: Simulation results of select days from author's data-set

Date	Correct Delivery	Undeniable Errors	Total Notif.	S.C. Ratio
Dec 2nd	4	23	27	**14.8%**
Dec 10th	4	31	35	**11.4%**
Dec 16th	7	11	18	**38.8%**
Avg:	5	21.7	26.7	**21.7%**

TABLE 5.2: Simulation results of select days from supervisor data-set

metric and is used to judge whether the proposed NMS would satisfy users in a real world environment, on the publication of this report. Naturally, it is the objective to strive to increase the SCR. However, the PCR is a good early indicator of acceptable notification delivery results.

5.3.2 Results & Observations

The results of the implementation of the above method is illustrated in tables 5.1 and 5.2. An extract of the underlying data from which these tables were derived, can be found in appendix B for the author's results, and appendix C for the supervisor's results. *Google Calendar* extracts for both individuals, can also be found in the respective appendices.

Immediately clear from these results is that the system seems to be biased towards the author. There are a number of reasons why the results achieved by the author in table 5.1 are superior, the first of which is the scheduling of the user's calendar. The design of the NMS is, presently, very dependent on the population of a user's schedule in their *Google Calendar*. Through comparing figures B.4 and C.2 in appendix B and C respectively, it is clear that the calendar of the author is populated to a greater extent as opposed to that of the supervisor's. As there is a greater

amount of information available to the system pertaining to the user's location, activity, and who they are with in the author's calendar, it enables the system to make increasingly accurate inferences of whether or not the notifications should be sent to the user at the particular time the notification is sent.

For instance, a notification was sent to the author at 9:53am on the 21st of January (figure B.2 of appendix B). The notification was from a close friend and the subject matter was that of a social nature. Checking the author's calendar it can be seen that this notification was sent during a college lecture. Naturally, as the notification is not contextually relevant to the author's current activity of "work", the notification can wait. The period of time before the notification is delivered is dependent on the importance of the person who sent it, the application through which it was sent, and also the context of the remaining events in the users schedule. In this case, a notification sent by a close friend will be delivered at the break of the user's current activity - hence the value of 2 being assigned as the delivery method. If there was no activity scheduled for the user at this point in time, the notification would have been immediately delivered to the user, as the system assumes if nothing is currently scheduled, then the user is free and available (**Note:** this is assumed only for notifications with a delivery method of 3 and below). For example, on the 20th of January, the user was sent a social notification through *Tinder* at 12:51pm. This notification is correctly immediately delivered to the user as there is nothing scheduled for the author between 12:30pm and 2pm. The NMS assumes the user is free and available to see the notification. Had this notification been sent during the author's lecture, it would be expected that the notification would not reach the user immediately. Hence, it would have a delivery method of 1 or 2.

Comparing the above scenario with a contextually similar scenario from the supervisor's result set, it can be seen that the lack of detail in the calendar greatly diminishes the effect of the NMS. For example, the supervisor received a number of notifications between 4pm and 5:30pm, which were expected to be delivered at the user's next break in their current activity or their next free period (greater than 30 minutes). The corresponding schedule of the user, as perceived in the supervisor's calendar however, is that they are completely free and available for all notifications whose delivery methods would have been classed as 2 or 3 if there had been an event scheduled. Consequently, all these notifications are sent to the user immediately. This highlights the necessity for the NMS having a very broad and granular level of data available to it in order to perform well. Naturally, if the system had access to other data sources which could enable the system to verify the user's current activity (such as location data), then those errors associated with the incorrect classifications of notification deliveries, as those given in the example above, could be avoided.

Another reason for the bias of results towards the author over the supervisor, may be the formulation and combination of the membership functions and ranking system

used within the FIS. As the author is the developer of the system, there could be a bias in the heuristic logic used to develop the membership functions/knowledge base. Further experimentation is needed to explore the results of variation in membership functions and their effects on the results of the NMS for a greater test bed of users.

Aside from the bias in the system, the NMS seems to manage the author's notifications in a consistently accurate manner, with an overall SCR average of 72.7%. This is quite a promising figure for the system, as it illustrates that the info-bead model and fuzzy inference combination can be quite competent at managing real-world notifications. Furthermore, the system has a PCR average of 85.8%, which is high enough to tempt implementing the system within a prototype for immediate use. Figure 5.3 also demonstrates that the difference between total and correct notifications remains consistent, no matter the flux in the number of notifications received, meaning that the system is dependable at its current level of performance.

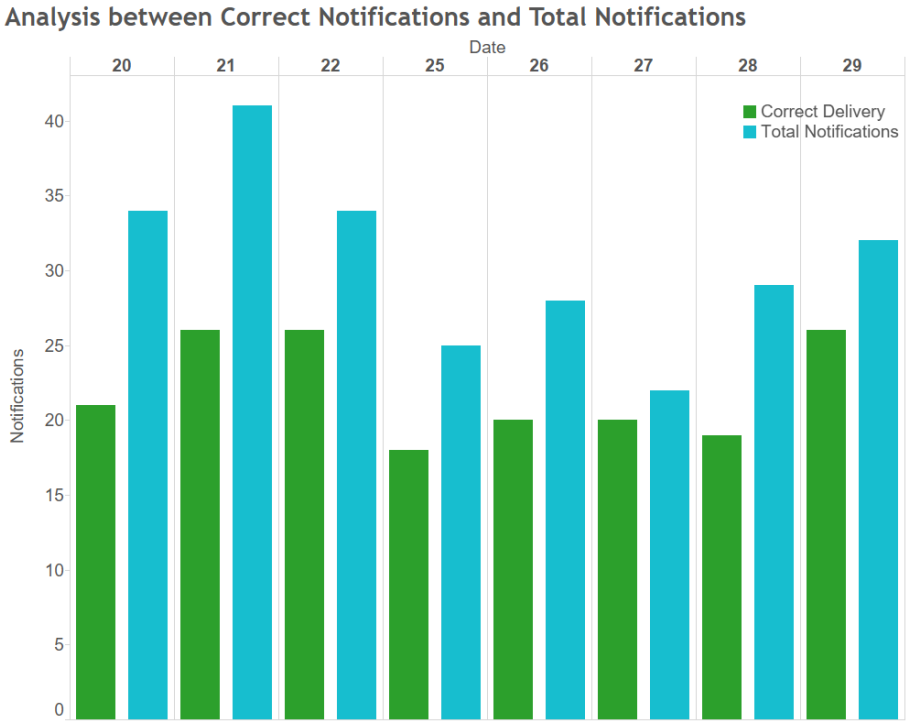


FIGURE 5.3: Comparison between total notifications and correctly classified notifications over a number of evaluated days.

By delving deeper into the analysis of particularly interesting days of the author's result set of notifications, a number of characteristics of the system can be highlighted (and used for focusing future work). For example, in figure B.1 in appendix B, there are 4 undeniable errors made by the system while classifying notification deliveries.

- The first of these is a work notification delivered at 1:23pm. It was expected to deliver this notification later at a contextually relevant time, as opposed to immediately, as simulated. The reason for the simulated value is most likely due to

there being nothing scheduled for the author at the particular time of delivery. However, as the sender is automatic (which is ranked low by the author, appendix A) and the subject isn't contextually relevant, then it would be expected that the notification should be assigned a less important delivery method than 1. Most likely, as the user has a scheduled event which is contextually relevant to "work" quite soon, this notification was marked as relevant in the context of time. This incorrect classification, highlights the subtleties through which the system is attempting to navigate and the fact that the system does struggle with outlying scenarios. A work notification, from an application which is ranked highly and sent at a time contextually relevant to a work related event, is usually deliverable immediately, as simulated in the example. However it is the edge case whereby a notification fulfills the criteria but still shouldn't be sent, that needs to be explored and improved upon in the future.

- The second undeniable error made, was a social notification sent from a friend and delivered at 4:52pm. This corresponds to being delivered during the author's college lecture. As the notification is social and it was delivered via *Facebook*, it is safe to assume that the notification should not interrupt the user. However, in this case, it was set a delivery method of 1. The "college" event is connected with the sender of the notification, as the author's friends are also in attendance. However, it would be assumed that the subject of "social" would mean the notification should wait for the break in schedule. The reason behind this notification slipping through, could be the events scheduled later in the day, two being social events ("badminton" and "drinks"). Hence, their additional contextual relevancy (while minor, as these events are later in the day) could have been enough to tip the balance of when the notification was delivered. This particular scenario highlights the necessity for greater granularity in data. If the sender was known to the system on a personal level, then individual people could be distinguished and a greater level of control achieved. For example, the individual friend sending the message, may not be linked to the social events later in the day - hence there would be a decreased contextual relevancy contributing to the delivery method for the social notification. Using the broad uplift terms such as "friend" and "social" has been an aspect of this project's attempt at keeping sensitive data private. However, it is clear that with the increase in privacy, there also comes the price of a decrease in performance. Future work will need to explore the relationship between these two variables, in order to maximize the potential of the NMS.

- The third undeniable error was another social notification sent by a friend through *Whatsapp* at 7:27pm. This notification was delivered while the author was studying, and as such, shouldn't have been immediately delivered, but

delivered instead at the break between the scheduled events. Again, as this notification is classed as social and as it is nearing contextual relevancy (the event of "drinks" beginning just an hour later), it could have been mistakenly given a high importance value. As discussed previously, more information regarding the user and the notification, would help classify notifications which arise in this type of scenario better.

- The final error was a work notification being delivered at 11:35pm. This notification was neither contextually relevant (as its subject was that of "work") nor important (as its sender value is "automatic"), yet it was delivered immediately. This again highlights the dependency of the system on the data it is provided with. There are no events scheduled in the author's calendar for this time. Hence, the system assumes the user is free and available. While it might be expected that if the user is free during the day that this email would be perfectly suited to being delivered (as the user has nothing else scheduled anyway), late at night when the user is at home, a "work" notification is not especially relevant or desired. This highlights an interesting point. The uplift rankings, as implemented in this project, remain fixed for the user throughout the day. This is not completely accurate, as a user's perception of what is important may change throughout the day. For example, work might have a high importance ranking during the hours of 9am to 5pm, but a much lower value throughout the rest of the day. Future work will therefore be necessary on creating a system of dynamically classifying the ranking list of the user, in order to model the users in an increasingly accurate fashion.

Overall the performance of the NMS was quite good when tested on a number of days which were varying in complexity and context. It is clear from the comparison between the author and supervisor's results, that future work needs to be directed towards expanding the generality of the system, to enable it to accurately manage notifications on behalf of a broader spectrum of users. The results and observations above, do prove the concept of the info-bead model and fuzzy inference system. Although there is clearly bias in the system described in this project, implementing a form of analysis on past user data and integrating it with the fuzzy membership functions and ranking system, would most likely yield results as seen above by the author.

5.4 Summary

In this chapter, the results of all the various aspects of the project were discussed. This spanned from the data-sets produced by the *NAbsMobile* application, to the privacy encapsulated by the *NAbsUplift* application, and finally to the contextual delivery simulations output by the *NAbsDesktop* application.

A number of observations regarding performance and future work were also discussed with regard to to the results of these applications. This included metrics on the quality of the data-sets created, as well as metrics on the performance of the NMS managing real-world notifications. A number of scenarios within the result set of the *NAbsDesktop* application were also discussed, as well as the logic surrounding the decision making process. All observable discrepancies within the system were also analyzed.

Chapter 6

Conclusion

The main goal of this project was to develop a system which would manage a user's notifications by delivering them in a contextually relevant manner. The proposed solution was to implement an info-bead modeling approach and fuzzy inference system to act as an intelligent framework between incoming notifications and the user. A number of objectives were outlined at the beginning of the project to structure the development of the info-bead model and fuzzy inference framework. These objectives were also created to ensure the feasibility of the solution was properly evaluated while the overall goal of the project was achieved. The objectives, as stated in Chapter 1, and conclusions on completion are as follows:

1. Gather real-world notification data

   It was necessary to accumulate a data-set of real-world notifications in order to test the feasibility of the proposed info-bead model and fuzzy inference system design. To ascertain whether the design could perform in practice, real notification data needed to be captured, stored and be accessible to the NMS. For this purpose, *NAbsMobile*, an *Android* application, was developed to log the details of incoming notifications. The mobile application performed well in practice and was able to accumulate sufficient notification data from two volunteers for the purposes of testing the NMS. The application also adequately met ethical expectations as it ensured the gathered data was secure at all times. A criticism of *NAbsMobile* did surface while evaluating the application however. The *believability* between the user and the resulting data-set produced by the application was moderate at best. User's didn't seem to fully trust the application to competently capture all their notifications and log them accurately in the data-set. From this finding it is clear that user's may not completely wish to relinquish control of their incoming notifications to an autonomous agent. It is suggested that a hybrid model comprised of automation and a certain level of user input is the solution to allow the process of capturing notifications and delivering them to a user become, to a greater degree, more transparent. For example, the user input could include permitting the user to match delivery methods to particular categories of notifications. A recommendation for future work on gathering

notification data would be to log additional data with the notification such as the users location and the time lapse between the notification being delivered and read. The greater amount of information available to the NMS, the greater accuracy with which the contextual delivery can be inferred. This will also lend insights into the behavior of the user and enable increasingly accurate user models to be implemented. For the purposes of this project, the functionality for intercepting notifications was built as a separate application so as to simplify and focus development of the info-bead model framework and FIS. Future work on the NMS should concentrate on developing a NMS within a mobile device, combining the functionality expressed by the three separate applications discussed in this paper: *NAbsMobile*, *NAbsUplift* and *NAbsDesktop*.

2. Harvest user data from various sources

   To accurately infer when to deliver a notification, an accurate and dynamic user model needed to be created and constantly updated with sources of information pertaining to the user's current state. For this purpose a number of data sources, in addition to the notification data-set, had to be integrated with the info-bead model framework. In the scope of this project a user's *Google Calender* was harvested in order to deduce their current activity. This was implemented by sending requests to the *Google Calendar* API. The quality of the information received back from the *Calendar* API was completely dependent on the user keeping their online schedule up to date and enriched with data which the NMS could use for contextually matching notifications. Once the user's *Calendar* was relatively well populated the NMS was able to sufficiently manage user notifications. The user's themselves were also used as data sources. Uplifted terms were given rankings and this data was used within the inference mechanism to determine the importance of notification attributes. The ranking data had one significant flaw in that it wasn't dynamically updated to reflect the users current state at every point of notification delivery. The rankings were taken as general importance values for the uplift terms and, as such, reduced the accuracy of the decisions chosen by the NMS. For example, "work" notifications during the day could be important but during out-of-office hours the importance value could, potentially, have dropped. Future work on the NMS should include automatically inferring the uplift term rankings through analysis of a user's social media connections and previous notification habits.

3. Develop a user/notification model

   This objective involved modeling the notification and user using the info-bead modeling approach. Various attributes of the notification were modeled using info-beads, such as the *Sender* info-bead and the *Subject* info-bead for example. The user, in contrast, wasn't modeled through info-beads in the scope

of this project. Instead, the user's *Google Calendar* was used to provide the notification-attribute info-beads with contextual information pertaining to the user, such as their current activity. This was a design choice made early in the project in order to simplify development to enable concept be evaluated. Future work on the project should include the separation of the notification and user models in order to provide a cleaner definition of both entities. This will enable the user model to be reusable across any number of other applications, not necessarily in the domain of notification management. Similarly it will enable the notification model to be reused for purposes other than just managerial. The implemented info-bead model was nonetheless effective and, as a generic info-bead model library was created during this project, future work on adapting the created model should progress more quickly.

4. Develop an inference mechanism to contextually deliver notifications

   A *Mamdani* Fuzzy Inference System was implemented as the inference mechanism of choice for the info-beads within this project. The FIS is the intelligent part of the info-bead which infers an attribute value, in this case an importance value pertaining to a notification attribute, such as "Sender Importance". The implemented FIS used the ranking data provided by the user and a value related to an event's timeliness, obtained from the user's *Google Calendar*, as inputs to the system. The FIS performed well for one volunteer, the author, and less so for the second volunteer, the supervisor. It was reasoned that the membership functions mapping the ranking values to heuristic variables would need to be dynamic (as opposed to fixed, as they are in this project) and adapt to individual users. This could be achieved by analysing the behavior of a user and their notifications. The FIS was competent at managing user notifications for the author hence the concept of combining the info-bead model with a FIS was an adequate solution to the problems surrounding notification delivery. Future work involving the combination of fuzzy inference, as demonstrated by this project, and machine learning, so as to analyse patterns in user/notification behavior, should be the next step of investigation in the area of notification management.

5. Scale the developed framework to ascertain effectiveness

   This objective was achieved only in part as the framework, while tested using a second data-set, requires further testing on a much larger scale to better evaluate its performance over a wider range of users. The performance of the NMS on the second data-set was poor compared to that of the first. Reasons for the drop in performance were, as discussed in Chapter 5, possible bias built into the system in the form of fixed membership functions, static rankings failing to capture the users dynamic importance contexts throughout the day and a lack

of data in the users *Google Calendar*. The former two reasons are aspects of the NMS which can be improved upon through additional development and research. The latter reason can only potentially be minimized through searching for additional data sources on the user, but in reality the NMS's performance will always be restricted by the granularity of data provided by the user.

The majority of the objectives described above were fully accomplished throughout the project and a NMS was built, tested and evaluated using real-world notification data thus verifying the concept of an info-bead model and FIS as a solution to contextual notification management.

Appendix A

Notification Uplift

Sender		Application		Subject		Body		Date	
Family	10	Facebook	8	Social	5	Social	5	Ocassion	8
Close Friend	9	Gmail	8	Interest	4	Interest	4	Holiday	3
Colleague	7	Tinder	4	Work	8	Work	8	Not Significant	1
Stranger	4	Android	2	Android	2	Android	2		
Acquaintance	5	Flipboard	3	Family	10	Family	10		
Friend	7	Whatsapp	8						
Myself	1	LinkedIn	6						
Automatic	1	Viber	7						
		Skype	9						
		Twitter	5						
		Android SMS	2						
		Calendar	5						
		Run Keeper	4						

FIGURE A.1: Author's uplift term importance ranking.

Sender		Application		Subject		Body		Date	
Family	10	Facebook	10	Social	8	Social	8	Ocassion	5
Close Friend	8	Gmail	9	Interest	9	Interest	9	Holiday	5
Colleague	6	Tinder	1	Work	6	Work	6	Not Significant	5
Stranger	1	Android	3	Android	2	Android	2		
Acquaintance	3	Flipboard	2	Family	10	Family	10		
Friend	7	Whatsapp	10						
Myself	10	LinkedIn	3						
Automatic	2	Viber	6						
		Skype	7						
		Twitter	8						
		Android SMS	2						
		Calendar	9						
		Run Keeper	7						

FIGURE A.2: Supervisor's uplift term importance ranking.

Appendix B

Results - Author

Expected Result	Simulated Result	Correct	Permissible Errors	Undeniable Errors	Sender	App	Subject	Body	Date
5	5	1			automatic	android	android	android	20/01/16 02:36:41
5	5	1			automatic	android	android	android	20/01/16 09:48:29
3	4	0	1		automatic	tinder	social	social	20/01/16 09:49:19
5	5	1			automatic	android	android	android	20/01/16 09:49:35
3	2	0	1		close friend	whatsapp	social	social	20/01/16 10:19:25
5	5	1			automatic	android	android	android	20/01/16 12:50:11
1	1	1			colleague	gmail	interest	interest	20/01/16 12:50:58
1	1	1			friend	tinder	social	social	20/01/16 12:51:40
1	1	1			close friend	whatsapp	interest	interest	20/01/16 13:05:29
4	1	0		1	automatic	gmail	work	work	20/01/16 13:23:55
4	5	0	1		automatic	flipboard	interest	interest	20/01/16 15:25:18
2	3	0	1		automatic	gmail	work	work	20/01/16 15:41:58
3	3	1			automatic	gmail	work	work	20/01/16 16:25:26
3	1	0		1	friend	facebook	social	social	20/01/16 16:52:16
4	3	0	1		automatic	gmail	work	work	20/01/16 16:56:36
1	1	1			friend	facebook	social	social	20/01/16 17:18:33
1	1	1			automatic	gmail	work	work	20/01/16 17:54:38
2	2	1			colleague	gmail	work	work	20/01/16 18:08:42
4	3	0	1		automatic	gmail	interest	interest	20/01/16 18:11:17
2	2	1			colleague	gmail	work	work	20/01/16 18:12:00
3	2	0	1		colleague	gmail	interest	interest	20/01/16 18:29:05
1	1	1			family	gmail	family	family	20/01/16 18:49:59
2	1	0		1	friend	whatsapp	social	social	20/01/16 19:27:21
5	5	1			automatic	android	android	android	20/01/16 20:01:06
5	5	1			automatic	android	android	android	20/01/16 20:06:14
2	2	1			colleague	gmail	work	work	20/01/16 20:15:29
2	1	0	1		close friend	whatsapp	social	social	20/01/16 20:27:08
5	5	1			automatic	android	android	android	20/01/16 21:38:47
3	3	1			automatic	gmail	interest	interest	20/01/16 22:41:46
1	3	0	1		automatic	tinder	social	social	20/01/16 22:43:52
1	1	1			friend	whatsapp	social	social	20/01/16 23:07:54
1	1	1			friend	tinder	social	social	20/01/16 23:18:06
4	1	0		1	automatic	gmail	work	work	20/01/16 23:35:22
1	1	1			friend	whatsapp	social	social	20/01/16 23:35:46

FIGURE B.1: Results for Jan 20th.

Expected Result	Simulated Result	Correct	Permissible Errors	Undeniable Errors	Sender	App	Subject	Body	Date
4	1	0		1	automatic	gmail	work	work	21/01/16 00:38:58
1	1	1			friend	whatsapp	social	social	21/01/16 00:43:55
1	1	1			friend	whatsapp	social	social	21/01/16 01:10:33
5	5	1			automatic	android	android	android	21/01/16 02:14:43
5	5	1			automatic	android	android	android	21/01/16 02:43:05
5	5	1			automatic	android	android	android	21/01/16 02:43:40
5	5	1			automatic	android	android	android	21/01/16 06:55:40
3	4	0	1		automatic	tinder	social	social	21/01/16 07:20:30
1	1	1			colleague	gmail	work	work	21/01/16 09:26:11
5	5	1			automatic	android	android	android	21/01/16 09:32:06
5	5	1			automatic	android	android	android	21/01/16 09:32:23
2	2	1			close friend	whatsapp	social	social	21/01/16 09:53:51
1	1	1			friend	gmail	work	work	21/01/16 10:21:44
1	1	1			colleague	gmail	work	work	21/01/16 10:31:30
2	2	1			friend	whatsapp	social	android	21/01/16 10:52:04
3	1	0		1	colleague	gmail	work	work	21/01/16 11:07:29
5	5	1			automatic	android	android	android	21/01/16 11:10:03
3	2	0	1		friend	whatsapp	social	social	21/01/16 11:32:51
3	2	0	1		friend	facebook	interest	interest	21/01/16 11:50:14
2	2	1			friend	whatsapp	social	social	21/01/16 12:00:58
3	5	0		1	automatic	android	interest	interest	21/01/16 12:12:59
3	4	0	1		automatic	tinder	social	social	21/01/16 12:14:00
2	1	0		1	friend	whatsapp	social	social	21/01/16 18:26:42
2	2	1			friend	facebook	interest	interest	21/01/16 18:27:20
4	5	0	1		automatic	flipboard	interest	interest	21/01/16 18:27:21
1	1	1			friend	gmail	work	work	21/01/16 18:27:31
2	1	0		1	friend	whatsapp	social	social	21/01/16 18:31:06
2	1	0		1	friend	whatsapp	social	social	21/01/16 18:34:17
2	1	0		1	friend	whatsapp	social	social	21/01/16 18:36:05
1	2	0		1	friend	facebook	interest	interest	21/01/16 19:29:22
1	2	0		1	friend	facebook	interest	interest	21/01/16 19:49:14
3	3	1			automatic	gmail	work	work	21/01/16 19:49:39
1	2	0		1	friend	facebook	interest	interest	21/01/16 19:53:22
4	4	1			automatic	tinder	social	social	21/01/16 22:30:19
1	1	1			friend	whatsapp	social	social	21/01/16 22:58:52
5	5	1			automatic	android	android	android	21/01/16 23:30:06
5	5	1			automatic	android	android	android	21/01/16 23:30:56
5	5	1			automatic	android	android	android	21/01/16 23:31:49
5	5	1			automatic	android	android	android	21/01/16 23:32:39
5	5	1			automatic	android	android	android	21/01/16 23:33:27
5	5	1			automatic	android	android	android	21/01/16 23:35:00

FIGURE B.2: Results for Jan 21st.

Expected Result	Simulated Result	Correct	Permissible Errors	Undeniable Errors	Sender	App	Subject	Body	Date
3	4	0	1		automatic	tinder	social	social	22/01/16 00:13:09
5	5	1			automatic	android	android	android	22/01/16 01:46:55
3	4	0	1		automatic	tinder	social	social	22/01/16 02:26:58
3	1	0		1	automatic	gmail	interest	interest	22/01/16 03:21:01
1	1	1			colleague	gmail	work	work	22/01/16 10:11:34
1	1	1			colleague	gmail	work	work	22/01/16 10:44:06
1	1	1			colleague	gmail	work	work	22/01/16 10:54:44
5	5	1			automatic	android	android	android	22/01/16 11:11:08
5	5	1			automatic	android	android	android	22/01/16 11:13:06
5	5	1			automatic	android	android	android	22/01/16 11:17:11
3	3	1			automatic	gmail	interest	interest	22/01/16 11:41:02
3	4	0	1		automatic	tinder	social	social	22/01/16 11:59:17
1	1	1			close friend	whatsapp	work	work	22/01/16 12:02:04
3	2	0	1		colleague	gmail	interest	interest	22/01/16 12:21:39
1	1	1			close friend	whatsapp	work	work	22/01/16 12:27:58
1	1	1			close friend	whatsapp	work	work	22/01/16 12:29:15
1	1	1			close friend	whatsapp	work	work	22/01/16 12:47:05
1	1	1			automatic	gmail	interest	interest	22/01/16 12:50:21
1	1	1			close friend	whatsapp	work	work	22/01/16 12:53:52
2	2	1			colleague	gmail	social	social	22/01/16 13:05:25
2	1	0		1	colleague	gmail	work	work	22/01/16 13:08:33
2	1	0		1	colleague	gmail	work	work	22/01/16 13:09:22
1	1	1			automatic	gmail	work	work	22/01/16 14:54:43
1	1	1			colleague	gmail	work	work	22/01/16 15:11:48
1	1	1			colleague	flipboard	interest	interest	22/01/16 15:26:00
1	1	1			close friend	gmail	work	work	22/01/16 15:36:52
1	1	1			colleague	gmail	work	work	22/01/16 16:15:25
1	1	1			colleague	gmail	work	work	22/01/16 16:36:32
1	1	1			colleague	gmail	work	work	22/01/16 16:36:34
1	1	1			colleague	gmail	work	work	22/01/16 16:44:08
3	4	0	1		automatic	tinder	social	social	22/01/16 17:11:03
2	2	1			friend	facebook	social	social	22/01/16 20:33:28
2	2	1			friend	facebook	social	social	22/01/16 21:58:55
5	5	1			automatic	android	android	android	22/01/16 22:57:26

FIGURE B.3: Results for Jan 22nd.

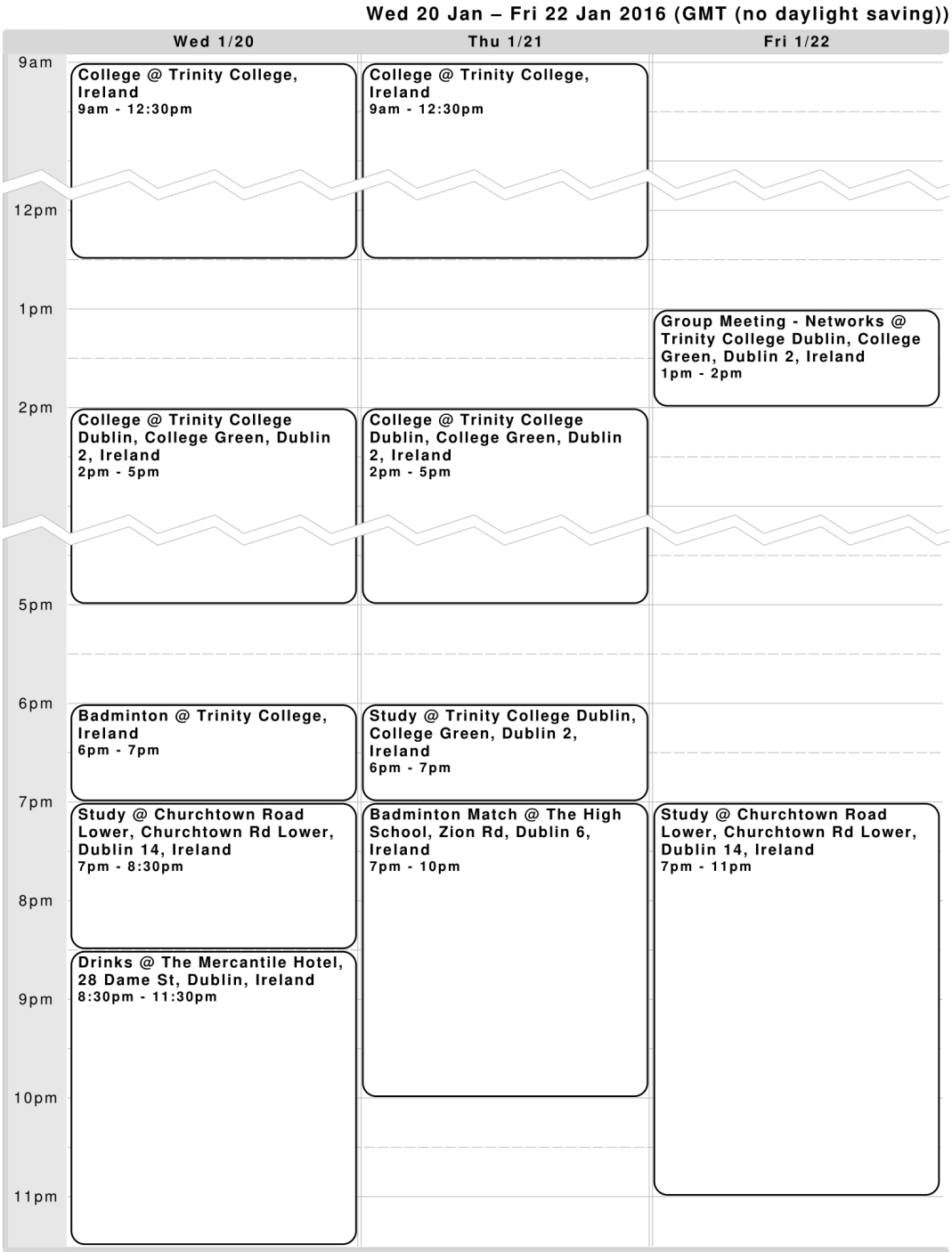


FIGURE B.4: *Google Calendar* of author for Jan 20th - Jan 22nd.

Appendix C

Results - Supervisor

Expected Result	Simulated Result	Correct	Sender	App	Subject	Body	Date
2	1	0	family	whatsapp	family	family	02/12/15 09:57:08
3	1	0	family	whatsapp	family	family	02/12/15 09:58:53
3	1	0	family	gmail	social	social	02/12/15 10:08:11
3	1	0	close friend	gmail	social	social	02/12/15 11:05:59
3	1	0	close friend	gmail	social	social	02/12/15 11:08:59
3	2	0	stranger	gmail	work	work	02/12/15 13:19:03
3	2	0	stranger	gmail	work	work	02/12/15 13:20:20
3	1	0	automatic	twitter	work	work	02/12/15 14:02:19
3	1	0	friend	gmail	interest	interest	02/12/15 14:02:29
3	1	0	colleague	gmail	work	work	02/12/15 16:15:47
1	1	1	acquaintance	androidSMS	family	family	02/12/15 16:21:22
2	1	0	friend	gmail	interest	interest	02/12/15 16:36:16
3	1	0	colleague	gmail	work	work	02/12/15 16:41:43
3	1	0	automatic	twitter	social	social	02/12/15 16:46:02
2	1	0	friend	gmail	interest	interest	02/12/15 17:30:59
1	1	1	automatic	androidSMS	family	family	02/12/15 18:05:44
1	1	1	automatic	calendar	work	work	02/12/15 18:50:04
2	1	0	colleague	androidSMS	work	work	02/12/15 19:08:09
1	1	1	colleague	androidSMS	work	work	02/12/15 19:10:03
2	1	0	friend	gmail	interest	interest	02/12/15 19:45:44
3	1	0	colleague	gmail	interest	interest	02/12/15 19:49:25
2	1	0	friend	gmail	work	work	02/12/15 19:50:39
2	1	0	friend	facebook	interest	interest	02/12/15 21:22:30
4	5	0	automatic	android	android	android	02/12/15 22:01:16
4	5	0	automatic	android	android	android	02/12/15 22:01:48
4	5	0	automatic	android	android	android	02/12/15 22:02:43
4	5	0	automatic	android	android	android	02/12/15 22:04:42

FIGURE C.1: Results for Dec 2nd.

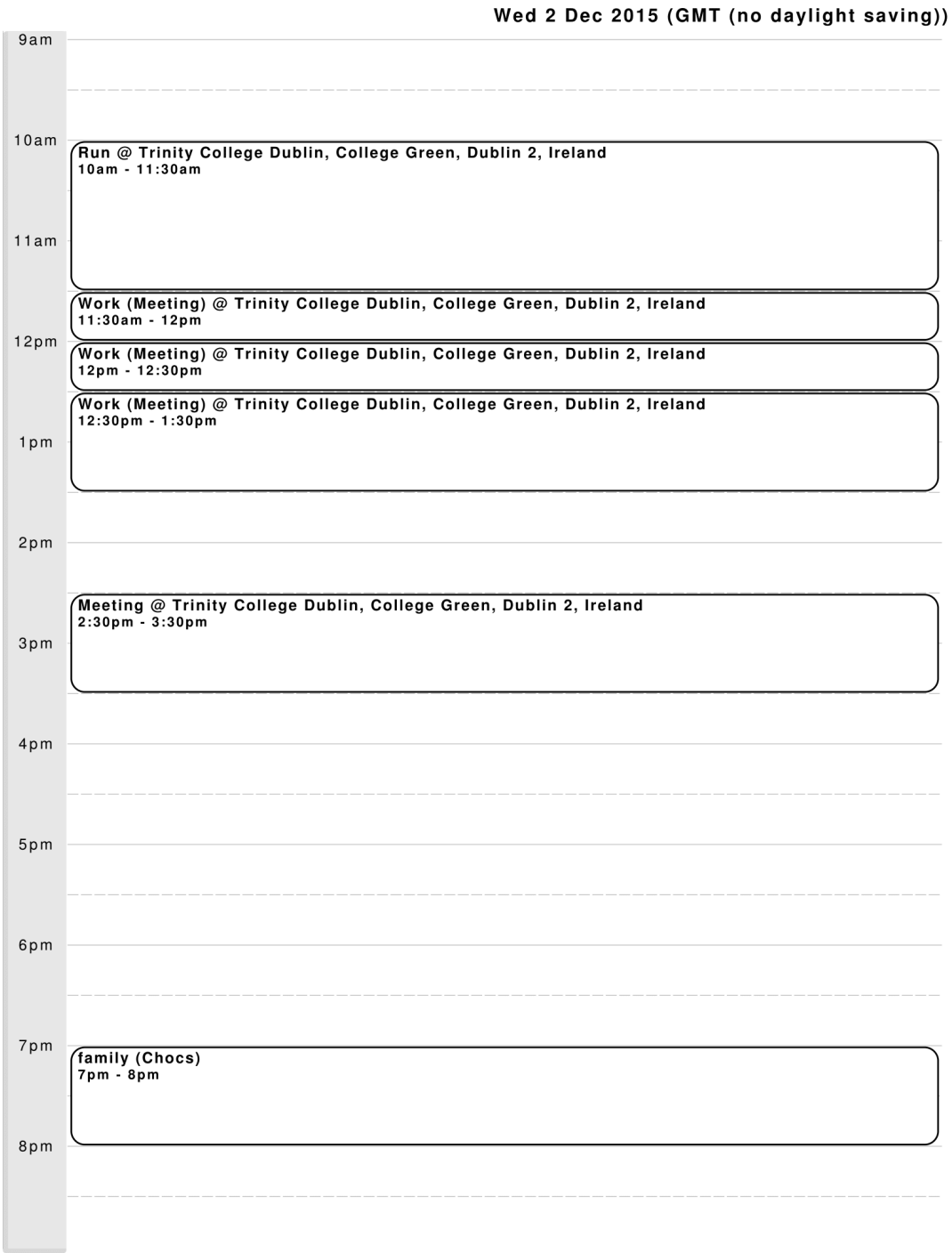


FIGURE C.2: *Google Calendar* of supervisor for December 2nd.

Appendix D

Fuzzy Inference System

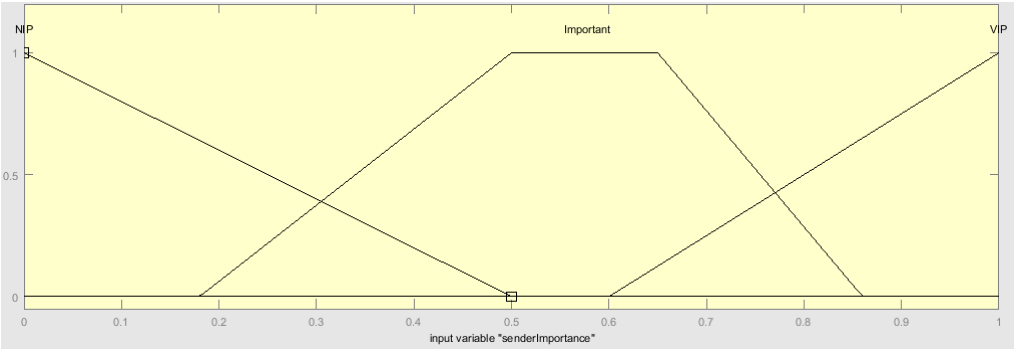


FIGURE D.1: The *senderImportance* membership function of the FIS in the *Sender* info-bead.

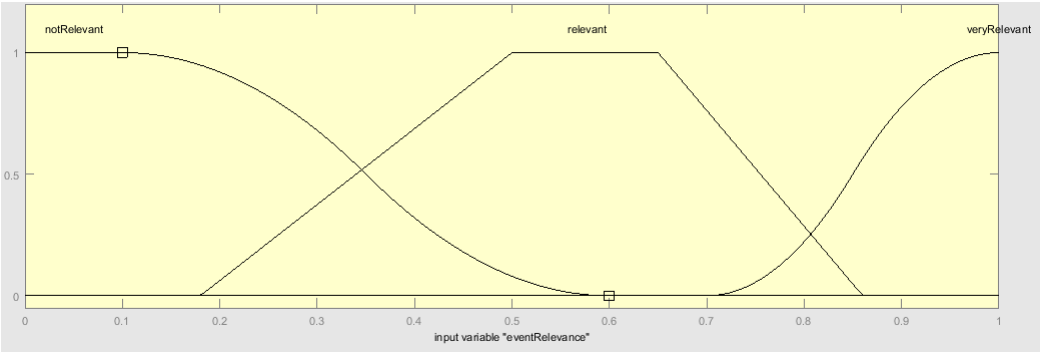


FIGURE D.2: The *eventRelevance* membership function of the FIS in the *Sender* info-bead.

1. If (senderImportance is NIP) and (eventRelevance is notRelevant) then (senderContext is low) (1)
2. If (senderImportance is Important) and (eventRelevance is notRelevant) then (senderContext is medium) (1)
3. If (senderImportance is VIP) and (eventRelevance is notRelevant) then (senderContext is medium) (1)
4. If (senderImportance is NIP) and (eventRelevance is relevant) then (senderContext is low) (1)
5. If (senderImportance is Important) and (eventRelevance is relevant) then (senderContext is medium) (1)
6. If (senderImportance is VIP) and (eventRelevance is relevant) then (senderContext is high) (1)
7. If (senderImportance is NIP) and (eventRelevance is veryRelevant) then (senderContext is medium) (1)
8. If (senderImportance is Important) and (eventRelevance is veryRelevant) then (senderContext is high) (1)
9. If (senderImportance is VIP) and (eventRelevance is veryRelevant) then (senderContext is high) (1)

FIGURE D.3: The knowledge base of the FIS in the *Sender* info-bead.

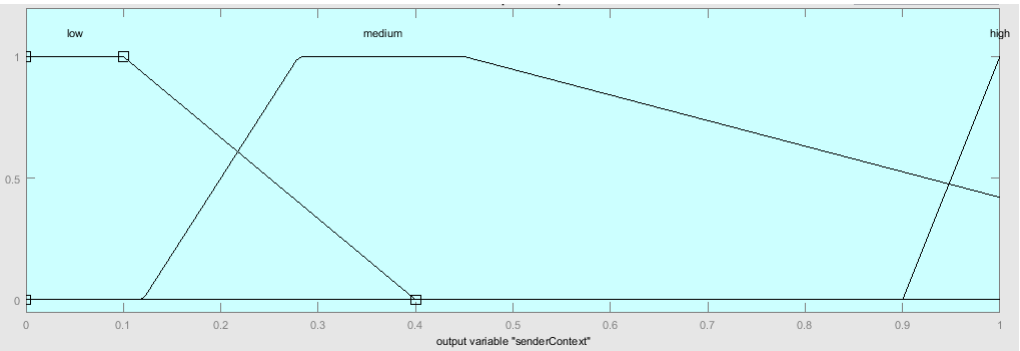


FIGURE D.4: The output membership function of the FIS in the *Sender* info-bead.

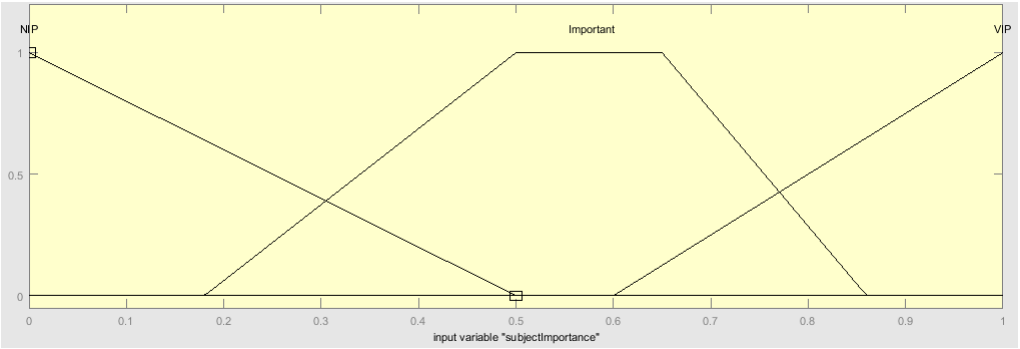


FIGURE D.5: The *subjectImportance* membership function of the FIS in the *Subject* info-bead.

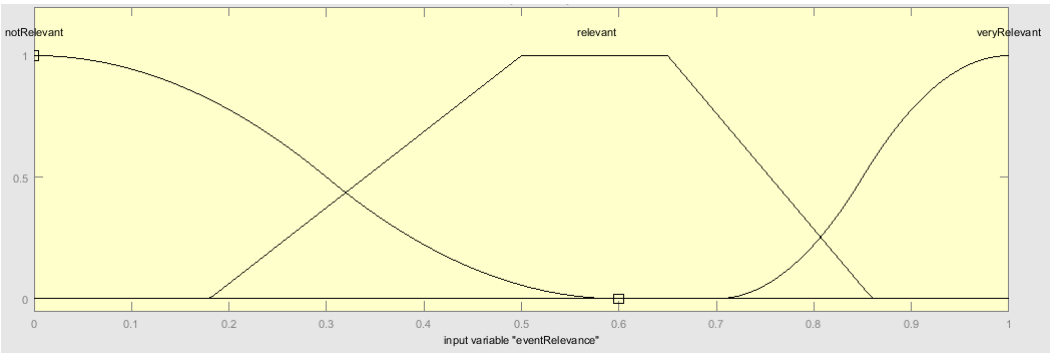


FIGURE D.6: The *eventRelevance* membership function of the FIS in the *Subject* info-bead.

1. If (subjectImportance is NIP) and (eventRelevance is notRelevant) then (subjectContext is low) (1)
2. If (subjectImportance is Important) and (eventRelevance is notRelevant) then (subjectContext is medium) (1)
3. If (subjectImportance is VIP) and (eventRelevance is notRelevant) then (subjectContext is medium) (1)
4. If (subjectImportance is NIP) and (eventRelevance is relevant) then (subjectContext is medium) (1)
5. If (subjectImportance is Important) and (eventRelevance is relevant) then (subjectContext is high) (1)
6. If (subjectImportance is VIP) and (eventRelevance is relevant) then (subjectContext is high) (1)
7. If (subjectImportance is NIP) and (eventRelevance is veryRelevant) then (subjectContext is medium) (1)
8. If (subjectImportance is Important) and (eventRelevance is veryRelevant) then (subjectContext is high) (1)
9. If (subjectImportance is VIP) and (eventRelevance is veryRelevant) then (subjectContext is high) (1)

FIGURE D.7: The knowledge base of the FIS in the *Subject* info-bead.

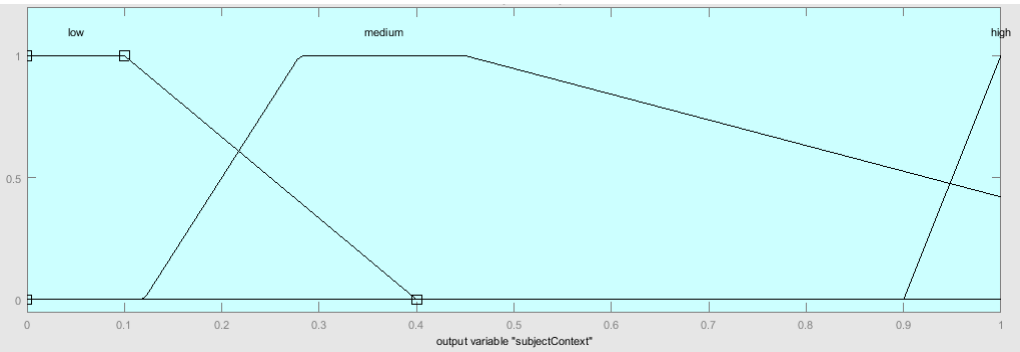


FIGURE D.8: The output membership function of the FIS in the *Subject* info-bead.

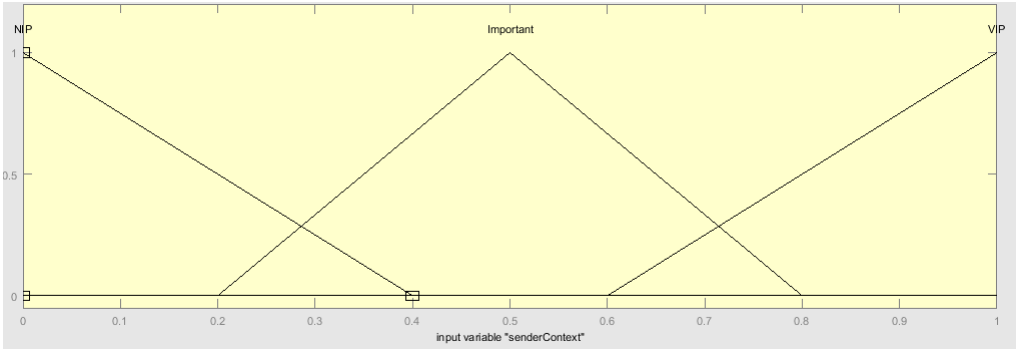


FIGURE D.9: The *senderContext* membership function of the FIS in the *Alert* info-bead.

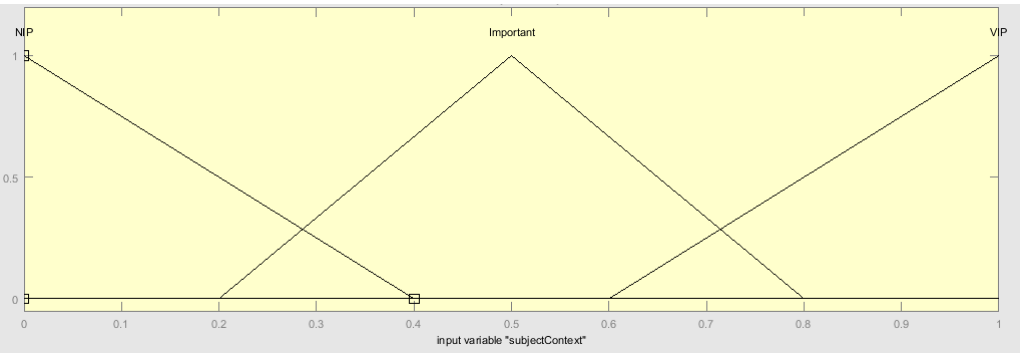


FIGURE D.10: The *subjectContext* membership function of the FIS in the *Alert* info-bead.

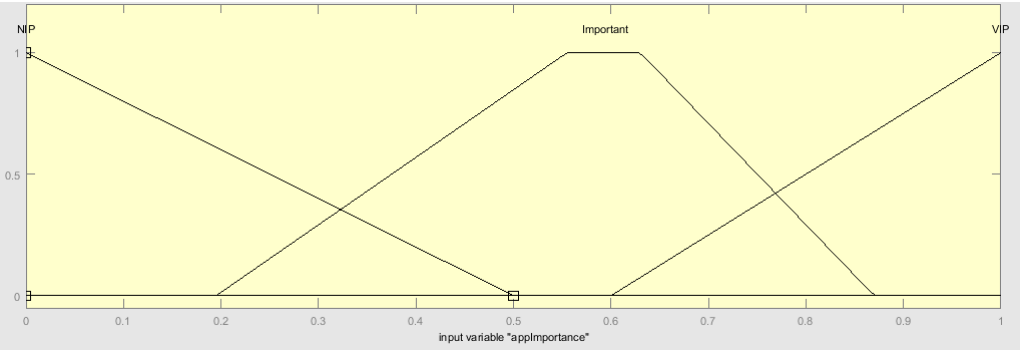


FIGURE D.11: The *appImportance* membership function of the FIS in the *Alert* info-bead.

1. If (senderContext is NIP) and (subjectContext is NIP) and (appImportance is NIP) then (contextDelivery is much_later) (1)
2. If (senderContext is NIP) and (subjectContext is NIP) and (appImportance is Important) then (contextDelivery is much_later) (1)
3. If (senderContext is NIP) and (subjectContext is NIP) and (appImportance is VIP) then (contextDelivery is later) (1)
4. If (senderContext is NIP) and (subjectContext is Important) and (appImportance is NIP) then (contextDelivery is much_later) (1)
5. If (senderContext is NIP) and (subjectContext is Important) and (appImportance is Important) then (contextDelivery is later) (1)
6. If (senderContext is NIP) and (subjectContext is Important) and (appImportance is VIP) then (contextDelivery is soon) (1)
7. If (senderContext is NIP) and (subjectContext is VIP) and (appImportance is NIP) then (contextDelivery is soon) (1)
8. If (senderContext is NIP) and (subjectContext is VIP) and (appImportance is Important) then (contextDelivery is soon) (1)
9. If (senderContext is NIP) and (subjectContext is VIP) and (appImportance is VIP) then (contextDelivery is verysoon) (1)
10. If (senderContext is Important) and (subjectContext is NIP) and (appImportance is NIP) then (contextDelivery is later) (1)
11. If (senderContext is Important) and (subjectContext is NIP) and (appImportance is Important) then (contextDelivery is soon) (1)
12. If (senderContext is Important) and (subjectContext is NIP) and (appImportance is VIP) then (contextDelivery is soon) (1)
13. If (senderContext is Important) and (subjectContext is Important) and (appImportance is NIP) then (contextDelivery is soon) (1)
14. If (senderContext is Important) and (subjectContext is Important) and (appImportance is Important) then (contextDelivery is verysoo
15. If (senderContext is Important) and (subjectContext is Important) and (appImportance is VIP) then (contextDelivery is verysoon) (1)
16. If (senderContext is Important) and (subjectContext is VIP) and (appImportance is NIP) then (contextDelivery is soon) (1)
17. If (senderContext is Important) and (subjectContext is VIP) and (appImportance is Important) then (contextDelivery is now) (1)
18. If (senderContext is Important) and (subjectContext is VIP) and (appImportance is VIP) then (contextDelivery is now) (1)
19. If (senderContext is VIP) and (subjectContext is NIP) and (appImportance is NIP) then (contextDelivery is verysoon) (1)
20. If (senderContext is VIP) and (subjectContext is NIP) and (appImportance is Important) then (contextDelivery is verysoon) (1)
21. If (senderContext is VIP) and (subjectContext is NIP) and (appImportance is VIP) then (contextDelivery is verysoon) (1)
22. If (senderContext is VIP) and (subjectContext is Important) and (appImportance is NIP) then (contextDelivery is verysoon) (1)
23. If (senderContext is VIP) and (subjectContext is Important) and (appImportance is Important) then (contextDelivery is verysoon) (1)
24. If (senderContext is VIP) and (subjectContext is Important) and (appImportance is VIP) then (contextDelivery is verysoon) (1)
25. If (senderContext is VIP) and (subjectContext is VIP) and (appImportance is NIP) then (contextDelivery is now) (1)
26. If (senderContext is VIP) and (subjectContext is VIP) and (appImportance is Important) then (contextDelivery is now) (1)
27. If (senderContext is VIP) and (subjectContext is VIP) and (appImportance is VIP) then (contextDelivery is now) (1)

FIGURE D.12: The knowledge base of the FIS in the *Alert* info-bead.

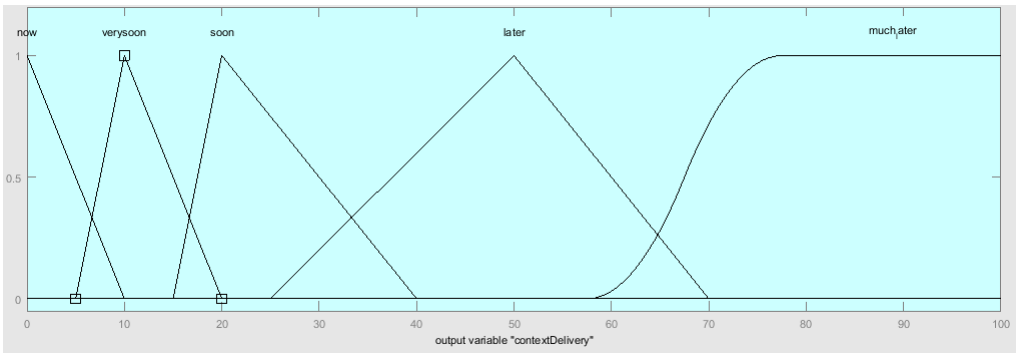


FIGURE D.13: The output membership function of the FIS in the *Alert* info-bead.

Bibliography

Brar, Ajay and Judy Kay (2004). *Privacy and security in ubiquitous personalized applications*. School of Information Technologies, University of Sydney.

Brusilovsky, Peter, Sergey Sosnovsky, and Olena Shcherbinina (2005). "User Modeling in a Distributed e-Learning Architecture". In: *Proceedings of the 10th International Conference on User Modeling*. UM'05. Edinburgh, UK: Springer-Verlag, pp. 387–391. ISBN: 3-540-27885-0, 978-3-540-27885-6. DOI: 10.1007/11527886_50. URL: http://dx.doi.org/10.1007/11527886_50.

Dim, Eyal and Tsvi Kuflik (2014). "Automatic Detection of Social Behavior of Museum Visitor Pairs". In: *ACM Trans. Interact. Intell. Syst.* 4.4, 17:1–17:30. ISSN: 2160-6455. DOI: 10.1145/2662869. URL: http://doi.acm.org/10.1145/2662869.

Dim, Eyal, Tsvi Kuflik, and Iris Reinhartz-Berger. "User Modeling Criteria and the Info-bead User Modeling Approach". In:

— (2015). "When User Modeling Intersects Software Engineering: The Info-bead User Modeling Approach". In: *User Modeling and User-Adapted Interaction* 25.3, pp. 189–229. ISSN: 0924-1868. DOI: 10.1007/s11257-015-9159-1. URL: http://dx.doi.org/10.1007/s11257-015-9159-1.

Fink, Josef and Alfred Kobsa (2000). "A Review and Analysis of Commercial User Modeling Servers for Personalization on the World Wide Web". In: *User Modeling and User-Adapted Interaction* 10.2, pp. 209–249. ISSN: 1573-1391. DOI: 10.1023/A:1026597308943. URL: http://dx.doi.org/10.1023/A:1026597308943.

Fischer, Joel E et al. (2013). "Understanding mobile notification management in collocated groups". In: *ECSCW 2013: Proceedings of the 13th European Conference on Computer Supported Cooperative Work, 21-25 September 2013, Paphos, Cyprus*. Springer, pp. 21–44.

Gartner's Hype Cycles for 2015. https://www.gartner.com/doc/3111522?ref=unauthreader&srcId=1-3478922254. Accessed: 2016-05-18.

Iancu, Ion and Claudiu-Ionut Popirlan (2010). "Mamdani Fuzzy Logic Controller with Mobile Agents for Matching". In: *Proceedings of the 11th WSEAS International Conference on Nural Networks and 11th WSEAS International Conference on Evolutionary Computing and 11th WSEAS International Conference on Fuzzy Systems*. NN'10/EC'10/FS'10. Iasi, Romania: World Scientific, Engineering Academy, and Society (WSEAS), pp. 117–122. ISBN: 978-960-474-195-3. URL: http://dl.acm.org/citation.cfm?id=1863431.1863451.

Kay, Judy, Bob Kummerfeld, and Piers Lauder (2002). "Adaptive Hypermedia and Adaptive Web-Based Systems: Second International Conference, AH 2002 Málaga, Spain, May 29–31, 2002 Proceedings". In: ed. by Paul De Bra, Peter Brusilovsky, and Ricardo Conejo. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Personis: A Server for User Models, pp. 203–212. ISBN: 978-3-540-47952-9. DOI: 10.1007/3-540-47952-X_22. URL: http://dx.doi.org/10.1007/3-540-47952-X_22.

Knijnenburg, Bart P et al. (2012). "Explaining the user experience of recommender systems". In: *User Modeling and User-Adapted Interaction* 22.4-5, pp. 441–504.

Kobsa, Alfred (2001). "Generic User Modeling Systems". In: *User Modeling and User-Adapted Interaction* 11.1, pp. 49–63. ISSN: 1573-1391. DOI: 10.1023/A:1011187500863. URL: http://dx.doi.org/10.1023/A:1011187500863.

— (2007). "The Adaptive Web: Methods and Strategies of Web Personalization". In: ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Generic User Modeling Systems, pp. 136–154. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_4. URL: http://dx.doi.org/10.1007/978-3-540-72079-9_4.

Kopecky, Dieter, Klaus-Peter Adlassnig, and Andrea Rappelsberger. "Patient specific adaptation of medical knowledge in an extended diagnostic and therapeutic consultation system". In:

Kuflik, Tsvi, Yevgeni Mumblat, and Eyal Dim (2015). "Enabling Mobile User Modeling: Infrastructure for Personalization in Ubiquitous Computing". In: *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*. MOBILESoft '15. Florence, Italy: IEEE Press, pp. 48–51. ISBN: 978-1-4799-1934-5. URL: http://dl.acm.org/citation.cfm?id=2825041.2825049.

Murugesan, S. (2007). "Understanding Web 2.0". In: *IT Professional* 9.4, pp. 34–41. ISSN: 1520-9202. DOI: 10.1109/MITP.2007.78.

Phuong, Nguyen Hoang and Vladik Kreinovich (2001). "Fuzzy logic and its applications in medicine". In: *International journal of medical informatics* 62.2, pp. 165–173.

Pielot, Martin, Karen Church, and Rodrigo de Oliveira (2014). "An in-situ study of mobile phone notifications". In: *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*. ACM, pp. 233–242.

Pipino, Leo L, Yang W Lee, and Richard Y Wang (2002). "Data quality assessment". In: *Communications of the ACM* 45.4, pp. 211–218.

Rada-Vilela, Juan (2014). *fuzzylite: a fuzzy logic control library*. URL: http://www.fuzzylite.com.

Russell, Stuart J. and Peter Norvig (2003). *Artificial Intelligence: A Modern Approach*. 2nd ed. Pearson Education. ISBN: 0137903952.

Sageder, B et al. (1997). "The knowledge model of MedFrame/CADIAG-IV". In: *Studies in health technology and informatics* 43 Pt B, 629—633. ISSN: 0926-9630. URL: http://europepmc.org/abstract/MED/10179742.

Saleh, Ahmed Abou Elfetouh, Sherif Ebrahim Barakat, and Ahmed Awad Ebrahim Awad (2011). "A fuzzy decision support system for management of breast cancer". In: *IJACSA Editorial*.

Silver, D. avid et al. (2016). "- Mastering the game of Go with deep neural networks and tree search". In: - 529.- 7587, pp. —489.

Vassileva, Julita, Gordon Mccalla, and Jim Greer (2003). "Multi-Agent Multi-User Modeling in I-Help". In: *User Modeling and User-Adapted Interaction* 13.1-2, pp. 179–210. ISSN: 0924-1868. DOI: 10.1023/A:1024072706526. URL: http://dx.doi.org/10.1023/A:1024072706526.

Vetterlein, Thomas and Agata Ciabattoni (2010). "On the (Fuzzy) Logical Content of CADIAG-2". In: *Fuzzy Sets Syst.* 161.14, pp. 1941–1958. ISSN: 0165-0114. DOI: 10.1016/j.fss.2009.09.011. URL: http://dx.doi.org/10.1016/j.fss.2009.09.011.

Wang, Yang and Yanyan Chen (2014). "A comparison of Mamdani and Sugeno fuzzy inference systems for traffic flow prediction". In: *Journal of Computers* 9.1, pp. 12–21.

Wang, Yang and Alfred Kobsa (2007). "User Modeling 2007: 11th International Conference, UM 2007, Corfu, Greece, July 25-29, 2007. Proceedings". In: ed. by Cristina Conati, Kathleen McCoy, and Georgios Paliouras. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Respecting Users' Individual Privacy Constraints in Web Personalization, pp. 157–166. ISBN: 978-3-540-73078-1. DOI: 10.1007/978-3-540-73078-1_19. URL: http://dx.doi.org/10.1007/978-3-540-73078-1_19.

Wang, Yang et al. (2006). "PLA-based Runtime Dynamism in Support of Privacy-Enhanced Web Personalization". In: *Proceedings of the 10th International on Software Product Line Conference*. SPLC '06. Washington, DC, USA: IEEE Computer Society, pp. 151–162. ISBN: 0-7695-2599-7. URL: http://dl.acm.org/citation.cfm?id=1158337.1158689.

Yimam, Dawit and Alfred Kobsa (2000). "Centralization vs. decentralization issues in internet-based knowledge management systems: experiences from expert recommender systems". In: *The Workshop on Internet-scale Software Technologies July 13-14, 2000, Irvine, Ca*.

Yimam-Seid, Dawit and Alfred Kobsa (2003). "Expert-Finding Systems for Organizations: Problem and Domain Analysis and the DEMOIR Approach". In: *Journal of*

Organizational Computing and Electronic Commerce 13.1, pp. 1–24. URL: http://dx.doi.org/10.1207/S15327744JOCE1301_1.

Zadeh, L.A. (1983). "The role of fuzzy logic in the management of uncertainty in expert systems". In: *Fuzzy Sets and Systems* 11.1, pp. 199 –227. ISSN: 0165-0114. DOI: http://dx.doi.org/10.1016/S0165-0114(83)80081-5. URL: http://www.sciencedirect.com/science/article/pii/S0165011483800815.

Zadeh, Lotfi A (1973). "Outline of a new approach to the analysis of complex systems and decision processes". In: *Systems, Man and Cybernetics, IEEE Transactions on* 1, pp. 28–44.