Università degli studi di Roma
"La Sapienza"

Dipartimento di Ingegneria Informatica,
Automatica e Gestionale "Antonio Ruberti"

Master Degree Thesis in
Artificial Intelligence and Robotics

Knowledge Based Reasoning and Execution of Complex Behaviors for a Social Robot

Supervisor

Prof. Luca Iocchi

Co-Advisor

Prof. Giuseppe De Giacomo

Student

Fabio Maria Carlucci
1550340

Academic Year 2013/2014

Acknowledgments

Firstly, I would like to thank my supervisor, professor Luca Iocchi, for giving me the possibility of working on such a fascinating subject and for his constant support and advice.

My thanks also go to the whole Lab Ro.Co.Co. for giving me the opportunity of broadening my knowledge and for all the feedback they gave me.

Many of my student colleagues deserve an acknowledgment, but special thanks must go to Fabrizio and Marta for their friendship and for having the patience of helping me out when I was in need.

My family has my full gratitude for their support and warmth during all these years. Without your constant optimism and encouragement none of this would have been possible.

All my closest friends have also earned a great thanks for making my life more joyous and for the patience they sometimes need to deal with me ;)

Dulcis in fundo, an enormous thank you to Angela for her love and for always being there for me. You are a constant of my life and without you, I might have never found this fantastic path.

Contents

Chapter 1

Introduction

This chapter contains an overview of the thesis. The motivation, scope and results of this work are presented here.

From the point of view of robotic development, the last few decades have been extraordinary: fifty years ago we had nothing more than toys and today we send robots into space, sea and land. We have cars that drive by themselves[1] and we perform robotic surgery. Industrial robots in particular have really become dominant, they have started to surpass the numbers of factory workers; we are still not free of "slavery"[2] (or, better said, mindless repetitive jobs) but we are getting there.

One field where science fiction is still well ahead of reality is that of social, service robotics. Robots perform perfectly in structured environment, but

[1]In June 2011, Nevada passed the first law allowing driverless cars in traffic

[2]Aristotle speculated in his Politics (ca. 322 BC, book 1, part 4) that automatons could someday bring about human equality by making possible the abolition of slavery:

There is only one condition in which we can imagine managers not needing subordinates, and masters not needing slaves. This condition would be that each instrument could do its own work, at the word of command or by intelligent anticipation, like the statues of Daedalus or the tripods made by Hephaestus, of which Homer relates that "Of their own motion they entered the conclave of Gods on Olympus", as if a shuttle should weave of itself, and a plectrum should do its own harp playing.

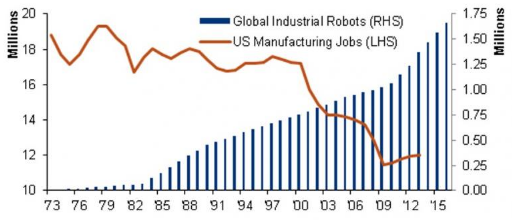


Figure 1.1: Chart from Bank of America Merrill Lynch. Note that the two plots use different scales; this is not a comparison of absolute values but of trends

as soon as we add humans in the equation, the environment stops being deterministic and things do not work so nicely anymore. Humans are highly unpredictable and, as a consequence, so are their social habitats (houses, shopping malls, train stations, ...).

1.1 Context and Motivation

In the past (this is starting to change), there has been a, mostly, strict division between Robotics and AI researchers. Roboticists usually focus on a particular, well defined and structured, application on an actual robot. Their outcome is something that works in the real world but has a specific scope and lacks flexibility.

AI people tend to work on general reasoners and complex planners, abstracting from any physical implementation. Their outcome is something that works extremely well in theory but has a hard time working on actual robots in the real world. This division of labor has excellent reasons to exist and has allowed the respective fields to progress greatly. But if we want to make robots common place in social spaces, it is time to draw from both fields.

The scope of this thesis lies exactly here: to design and develop a robust, reasoning based, robotic system capable of reacting in a smart way to the intricacies of the human world. As a context frame for such a system, we decided to insert it in an active and lively social context: a shopping mall.

1.1.1 COACHES

Figure 1.2: A photo of the Les Rives de L'Orne shopping mall in Caen

This thesis will also contribute to the COACHES (COoperative Autonomous robots in Complex and Humans EnvironmentS) project.

COACHES is an European project (under the FP7 CHIST-ERA call) involving four Universities: University of Caen, Vrije Universiteit Brussel, Sapienza University of Rome and Sabenci University (Istanbul). The main goal of the project is "to design an autonomous multi-robot system evolving in public areas, able to provide multiple services. This system could be deployed in public areas, such as Malls and touristic places, where the environment is complex and overcrowded and requires strong and robust abilities of navigation, recognition and assistance."

A project with such a vast scope is extremely complex and different partners work on different parts of it. Here at Sapienza, we focus on:

- An **explicit and high-level** representation of the knowledge about the environment, the user needs and their social behavior.
  - **Social Conventions**

    - **Semantic representation of the environment** (KB modelling and reasoning)

- Short term Human-Robot interactions

- Safe navigation in a crowded environment

In this context, the contributions from this thesis will focus on the development of the knowledge-based modelling and reasoning techniques to be used in the human-robot interactions .

1.2 Specific goals of this thesis

The main goal of this thesis is to model and develop a **system capable of accomplishing complex tasks by reasoning and acting reliably in a social context**. To do so, the following problems had to be faced:

- Define what is a socially acceptable behavior for a robot when interacting with people; investigate which social rule should a robot follow. Are there universal social rules? What kind of protocol should a robot follow in order to start and maintain an interaction with a human?

- Decide how to model social rules in a way that easily enables the generation of plans compatible with such restrictions. Should these rules be modelled implicitly or explicitly? What kind of formalism allows to generate plans while keeping in mind common sense knowledge and social expectations?

- Decide how to build a system able to handle plan failure in the real world. Rarely things go as smoothly as expected in the planning stage; to be robust, a system must be able to understand why it failed and then recover from it. At what level of complexity is this best handled? At the planner level or at the action level?

The developed system revolves around the successful combination of a reasoning based planner, which has the expressive power to explicitly model common sense and social rules, and the Petri Net Plans formalism that allows the execution of complex and robust behaviors while removing complexity from the reasoner.

The handling of possible failures has been spread between the reasoner, the PNPs and the atomic actions, trying to give responsibility where it is due, to minimize development complexity and to maximize system efficiency.

1.3 Chapter Structure

Chapter One provides a brief introduction to the thesis and overview of the different chapters

Chapter Two presents an overview of the state of the art in the fields of social conventions, human-robot interaction (especially on how and when to start a conversation), planners and in general on projects with a similar scope.

Chapter Three presents an outline of the software and hardware tools used in this work. The ROS framework, the reasoning tools, the Petri Net Plans formalism and the real life robot (its capabilities, sensors and safety considerations) are described in this chapter.

Chapter Four describes the architecture of the whole system. Its layers and interactions and the control loops are presented here. Different kinds of failure and how to recover are here discussed.

Chapter Five describes how the planning based on reasoning works and why we chose this kind of plan generation. Examples of how common sense knowledge and social rules are explicitly modelled are given here.

Chapter Six presents to the reader why and how Petri Net Plans were used in this work. How using PNPs enables the development of fewer atomic actions and how this formalism deals with plan failure is answered here.

Chapter Seven describes what happens at the level of ROS nodes. How fluents are generated from sensor data and how nodes interact. Selected nodes are discussed in more detail if particularly relevant.

Chapter Eight describes the experimental setup used to evaluate the system's performances. Different use cases are presented, highlighting the particularities and requirements of each. The simulated and real life environments are also detailed here. Successes and potential pitfalls of each case are also discussed here.

Chapter Nine presents the conclusion on this present work and proposes future developments.

Chapter 2

Related works and state of the art

This chapter presents an overview of the state of the art.

Overview of the field, benefits and shortcomings of the present work

2.1 Conventions for a Social Robot

When a robot is expected to interact with humans, it is generally believed that, in order to give the appearance of natural and friendly behavior, it should attempt to reproduce socially acceptable human behavior.

2.1.1 Social Distances

One of the most well know studies on Proxemics[1] is that of Hall[1, 2]. His work is quite broad, but what is of more direct use in robotics is his scale of social distances and reciprocal orientation.

According to Hall, humans tend to define boundaries, dividing the space around them in social areas (see figure 2.1). The closest area is that of

[1]Proxemics, the study of how man unconsciously structures microspace - the distance between men in the conduct of daily transactions, the organization of space in his houses and buildings, and ultimately the layout of his towns. [1]

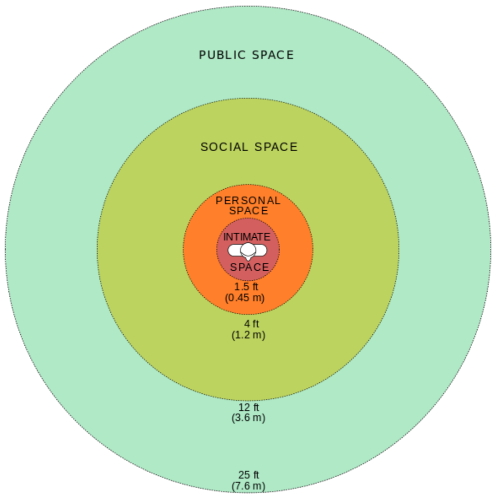


Figure 2.1: Hall's Social Distances

their personal space and only intimate people are allowed to enter; strangers invading that zone would produce a sense of discomfort. In all, four different interpersonal distances where defined:

- **Intimate Space**: reserved for extremely few people. At this range there is an exchange in body heat, touch and smell. (Typical range: 0-0.45m)
- **Personal Space**: reserved for friends. Social exchange happen at arms length (Typical range 0.45 - 1.2m)
- **Social Space**: commonly used for interaction with unfamiliar people. The extra distance allows for a greater feeling of safety. (Typical range: 1.2m-3.6m)
- **Public Space**: usually no social exchange happens at this distance. (Typical range: 3.6m -)

Hall also notes that the specific boundaries can vary very much due to cultural differences, thus the number represent the conventions of a particular culture.

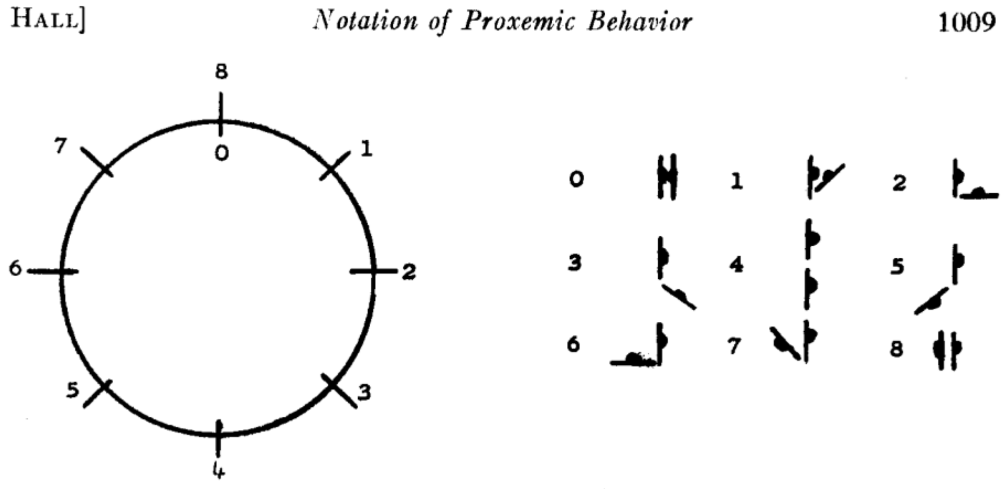


Figure 2.2: Hall's Sociofuga-Sociopetal axis notation code

As can be intuitively grasped, distance is not the only variable to observe to understand a social exchange: eye gaze, relative orientation and body language in general must be considered. As can bee seen in figure 2.2 there are many ways in which two persons can position themselves.

> 0,1,2,4 and 8 are most frequently observed. 0 is for direct communications where the intent of one or both of the participants is to reach the other with maximum intensity.
>
> 2 is more casual and less involved. A subject of common interest is often discussed using this axis. The subjects may shift to 0 or 4, depending on how involved or uninvolved they become.
>
> 4-the shoulder-to-shoulder axis-is one in which two people are normally watching and/or discussing something outside themselves, such as an athletic event or the girls going by on a Saturday afternoon, without necessarily being involved with each other. This is the axis for very informal, transitory communications.
>
> Position 6 is used as a means of disengaging oneself. It is not

quite, but almost as, sociofugal as 8.[1]

2.1.2 Approaching Humans

Most of the work on the field of Human Robot Interaction assumes that it is always the human that approaches the robot and asks him to do things. Luckily, research considering a robot proactively engaging with a human does exist.[3, 4, 5, 6]

In "How to approach Humans?" [6], Satake et al. consider the case of a mobile social robot than can initiate interaction by its own accord. They note that a passive robot might miss the opportunity of an interaction with somebody unaware of its capabilities: a senior citizen might not know what the robot can do, or he might hesitate to ask for help.

They modeled the approach in three steps: selecting of a target, approaching the target and starting a conversation. In the first step they analyzed what markers were to be considered in deciding a suitable target. Firstly they discarded all people presumed to be too busy to interact with the robot (this was estimated from their trajectories) and then they selected the targets whose trajectory they estimated to be able to intercept, considering the robot's speed. As a second step (they call it the "Interaction at public distance") the robot must signal his intentions to the user while still being too far away to speak. This, while not easy, is ideally accomplished by entering his eye of sight during the approach phase and signaling the robot's desire to start a conversation. The third phase is about initiating a conversation: this is done in two steps. The robot stops at a social distance and **faces** the human; if the human stops and looks at the robot this is seen as an acknowledgment and the robot starts the conversation.

At the end of the experiments, they categorized the failures in 4 categories: unreachable, unaware, unsure and rejected.

Unreachable and rejected respectively mean that the robot was unable to reach the human and the human refused to engage in conversation; there is little to do for this failures.

Unaware and unsure are more interesting: unaware categorizes all the cases where the human did not see (or understand the function of) the robot. Unsure was when a human noticed the robot but did not clearly understand how to proceed. This tells us that special care must be taken to make the robot's capabilities clear and visible.

Finke et al.[5] approached a similar problem: how to recognize a human's will to interact and how can the robot encourage the human by using simple orientation movements. It has been shown that a person's interest can be estimated to a certain degree by analyzing the distance from the robot and the movement speed. This study also suggests that the simple orientation cues might not be enough to attract people's attention but the authors also note that this might be due to software and hardware limitations that made the robot react with a delay of nearly 5 seconds.

A 2011 study from Buss et al. [3], highlights the importance of social behavior and multi-modality for a natural interaction with users unfamiliar with robots. Their findings suggest that a robot smiling and gesturing towards the human is more likely to attract his attention.

2.2 Trajectory Detection and Estimation

As seen in the previous section, in order to estimate user's intentions and to reason on whether it is possible to intercept a specific human, we need an accurate evaluation of the user's trajectory.

The previously mentioned work by Finke et al. [5] raises a very important point: trajectory estimation must be fast, a few seconds delay might prove enough to loose the potential user.

Vision based detection approaches can perform well but have the drawback of being computationally intensive; to bring a vision based tracker to real time Mitzel [7] proposes to limit the regions of interest of the image to a small subset, selected by applying a statistical Poisson process model in order to rate the urgency of each ROI. The proposed system reached the state of the art while also guaranteeing over 15 frames per seconds.

Laser based detection is usually fast, but due to the lower representational power of a single slice of distance measurements it can be imprecise. Carballo et al.[8] propose to fuse the readings from two laser scanners at different heights; they note that the added discriminative power and occlusion resistance(it might be possible that one laser is occluded while the other is not) makes the system more robust.

Detection alone is not enough for some applications, we need to estimate the trajectory. Work by Van Gool's group[9] uses a social behavior model to estimate the pedestrian's future motion. They note that more than one reasonable path may be available at any given moment and that deterministic approaches can fail to model this. They propose a stochastic extension to a simulation based model that can account for multiple alternative.

2.3 ASP Reasoning and Social Planning

ASP based planners have been proposed by different authors, to name a few: Lifschitz[10], Balduccini et al.[11], Dix et al.[12].

In the specific field of social robots, Erdem et al. in 2012 [13] have successfully used ASP to model an efficient planner for collaborative house keeping robots. The considered work automatically extracted common sense information from the ConceptNet[14] dataset and exploited it to better perform the given task.

2.4 Social Robots in Shopping Malls

A paper by Kanda et al.[15], proposes an interesting approach to anticipate people's behaviors from their trajectories in the context of a shopping mall. To do so, they use a laser scanner network spread around the mall. This information is then used to select an appropriate target for the robot to approach. Field trials showed promising results.

Subsequent work from Kanda et al.[16], investigated the social impact of a robot in a shopping mall, specifically on rapport building with the customers. Their, semi autonomous, robot provided shopping suggestions and guidance, much like the system considered in this thesis. Their results suggest that a robot can influence the customer's shopping habits.

2.5 Related Projects

There are other projects which have dealt (or are dealing) with similar situations:

URUS (http://www.iri.upc.edu/project/show/59) Ubiquitous networking robotics in urban settings, their objective is *to analyze and test the idea of incorporating a network of robots (robots, intelligent sensors, devices and communications) in order to improve life quality in urban areas.* They act in a social context and interact with humans. Relevant to this work, they produced a number of articles on how to estimate human trajectories in crowds[17]. They have also produced a complete probabilistic framework for human motion prediction in social environments[18]. Of crucial importance, they also analyzed the legal challenges faced to introduce a robot in a social, public place [19].

MOnarCH (http://monarch-fp7.eu/) – Multi-Robot Cognitive Systems Operating in Hospitals. Targets *the development of a novel framework to model mixed human-robot societies, and its demonstration using a network of heterogeneous robots and sensors, in the pediatric area of an oncological hospital. The robots will engage in edutainment activities and the uncertainties introduced by people and robots themselves handled to yield natural interactions.* They have focused on the ethical challenges that such a robot will encounter[20] and highlight the importance of conveying emotions with the robot's appearance and motion[21]. Furthermore they recognize the importance of acting following social rules and model them in a Constraint Network, based on works by Pecora et al.[22].

EUROPA (http://europa.informatik.uni-freiburg.de/) - European Robotic Pedestrian Assistant *is a project funded by the European Commission within FP7. The goal of the EUROPA project is to develop the foundations for service robots designed to autonomously navigate in urban environments outdoors as well as in shopping malls and shops to provide various services to users including guidance, delivery, and transportation.* Relevant to this thesis, they have produced a large volume of research on pedestrian tracking and trajectory estimation[23, 9, 7, and others].

GIRAFF (http://www.giraffplus.eu/) - *GiraffPlus is a complex system which can monitor activities in the home using a network of sensors, both in and around the home as well as on the body. The sensors can measure e.g. blood pressure or detect e.g. whether somebody falls down. Different services, depending on the individual's needs, can be pre-selected and tailored to the requirements of both the older adults and health care professionals. At the heart of the system is a unique telepresence robot, Giraff, which lends its name to the project.* Among other things, they focus on the importance of social interaction [24] and on how to evaluate and improve the likability of a social robot[25].

Chapter 3

Software and Hardware Tools

In this chapter the reader can find a coarse overview of the software and hardware tools used in this thesis.

3.1 ASP

Answer set programming (ASP) is a form of declarative programming oriented towards difficult search problems. As an outgrowth of research on the use of non monotonic reasoning in knowledge representation, it is particularly useful in knowledge-intensive applications. ASP programs consist of rules that look like Prolog rules, but the computational mechanisms used in ASP are different: they are based on the ideas that have led to the creation of fast satisfiability solvers for propositional logic.[26]

In ASP, to solve a a problem instance I, we first encode a non-monotonic logic program P such that the solutions of I are represented by models of P. Then we invoke an ASP solver to find a stable model M of P. At last we extract the solution I from M.

Since most ASP solvers are based on the DPLL algorithm, they, in theory, always terminate (unlike SLDNF resolution employed in Prolog).

ASP is also fully declarative, the developer is not expected to anticipate how the solver will internally interpret the encoded model; the order in which instructions are written does not matter.

Most ASP programs are written in three sections: generate, test and define. Generate instructs the solver how to come up with possible models, test sections are used to discard invalid solutions. The define section, if present, contains auxiliary predicates.

In ASP there are two kinds of negation, negation as failure and strong negation; this allows an easy representation of defaults and an elegant solution to the frame problem.

An informal overview of ASP's syntax will now be given[1] to help the reader follow small code snippets in the rest of this thesis:

```
1 p :- q.
2 cross :- not train.
3 cross :- ~train.
```

These three lines represent traditional, "Prolog-style" rules. Es. if q holds, then p is in the stable model.

Line 2 is an example of negation as failure: it is okay to cross if we know nothing on whether there is a train approaching.

Since this is not a good idea (if we want to cross safely, that is), a better formalization, using strong negation, is given in line 3: it is okay to cross if it is known that the train is *not* approaching.

Informally, what is to the left of **:-** is called *head* and what to the right *body*. If the rule head is missing we are expliciting a constraint:

```
:- drink, drive.
```

Constraints are used to eliminate some stable models; in this case we are removing all models in which we drink and drive.

Must ASP implementations offer many more constructs to simplify the formalization of the problem. Some of the most common are: **aggregates** (count, sum, min, max, average, even and odd), **set generation** capabilities and **cardinality constraints** on the sets.

[1]the excellent paper by Lifschitz[26] was a source of inspiration for this section

3.1.1 Clingo - Potsdam Answer Set Solving Collection

http://potassco.sourceforge.net/

Clingo is both a grounder and a solver for Answer Set Programming. It has been successfully used in many works. [13, 27]

3.2 Petri Net Plans

http://pnp.dis.uniroma1.it/

Petri Net Plans (PNP) is a formalism for high level description of complex plans (i.e., set of actions interacting in a complex way). PNP is useful to program cognitive agents (such as robots, videogame agents, multi-robot/multi-agent systems, etc.).

PNPs are inspired to languages for reasoning about actions, yet they are more expressive than most of them, offering a full fledged set of operators for dealing with non-instantaneous actions, sensing actions, action failures, concurrent actions and cooperation in a multi agent context. PNPs include also choice operators used for non-deterministic execution and for learning in the plan space through a Reinforcement Learning algorithm.

It is easy to show that PNPs are strictly more expressive than Finite State Machines (FSM). Moreover, PNPs allow for automatic plan analysis, which can provide formal guarantees on the performance of the plans. Execution of PNPs is extremely efficient and allows the design of real-time pro-active and reactive behaviors. PNPs have been used in several robotic applications, including soccer (Best Demo Award at AAMAS'08), search and rescue, surveillance, multi-robot cooperation, social robots, etc. [28, 29, 30]

A Petri Net Plan is defined by a tuple **PN = (P,T,F,W,M$_\mathbf{0}$)[31]**

P $= \{p_1, p_2, ..., p_n\}$is a finite set of *places*

T $= \{t_1, t_2, ..., t_n\}$is a finite set of *transitions*

F $\subseteq (P \times T) \cup (T \times P)$ is a set of *edges*

W : $F \Rightarrow \{1, 2, 3, ...\}$is a weight function.

$\mathbf{M_0}$: $P \Rightarrow \{0, 1, 2, ...\}$is the initial marking.

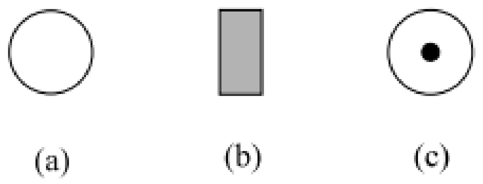


Figure 3.1: (a) an empty Place, (b) a transition, (c) a place with one marking

Places and transitions are handled as in a classical Petri Net: a transition is enabled only if each input place is marked with at least the number of tokens in the edge weight. If a transition fires, the number of tokens equal to the weight on the input edges is removed from the input places and a number of tokens, equal to the weight of the output edges, is added to the output nodes.

Petri Net Plans are defined in terms of **Actions** and **Operators.**

Actions are divided into ordinary actions and sensing actions (see figure 3.2), both of which are non instantaneous. Ordinary actions are deterministic and well defined, with a start an execution and a specific end. Sensing actions are non-deterministic and the outcome depends on the evaluation of certain conditions at run time.

Operators allow the composition of actions in order to model complex behaviors:

Sequences are used to execute a series of actions in order.

Loops are usually used with an Interrupt to repeat an action until certain conditions are met

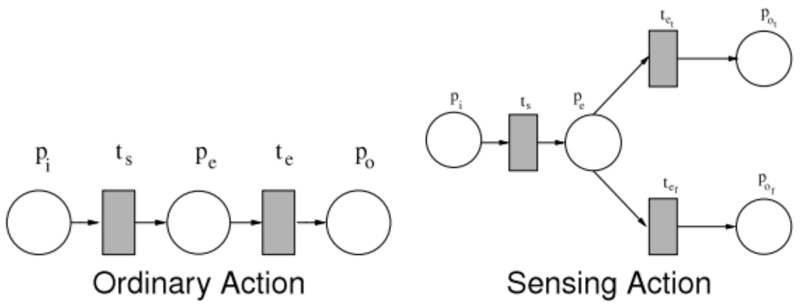


Figure 3.2: Ordinary and sensing actions

Fork\Join operators are used to handle concurrency. A Fork generates multiple threads from one and the Join allows the synchronization of multiple ones into a single thread.

Interrupts are used to change the normal execution flow: while an action is being executed it gets stopped by some external event and the flow will continue on a new branch.

As an example, let us consider figure 3.3:

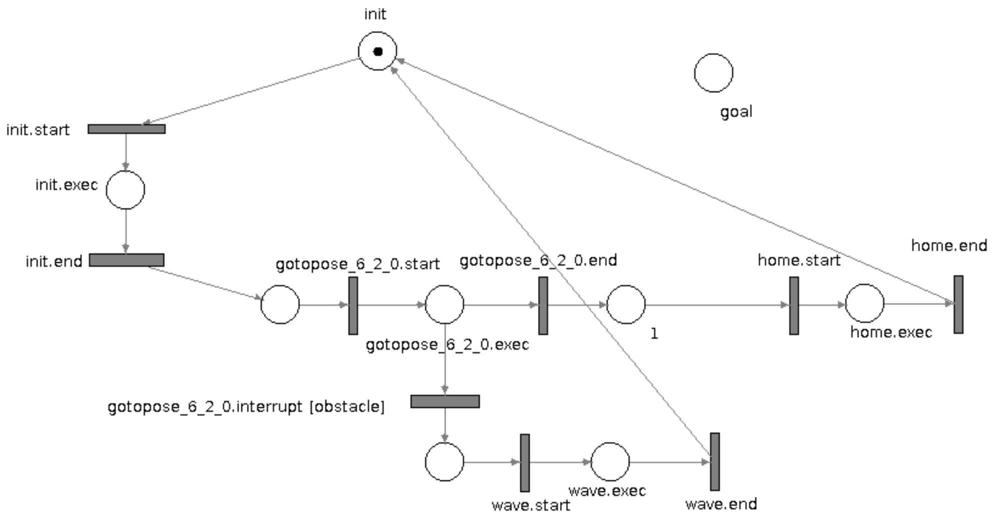


Figure 3.3: Example of Interrupt

In this simple plan, our robot is tasked to move (gotopose) towards the position (6,2). During the execution of this action (when the token will be positioned in Place gotopose_6_2_exec) there is the possibility that an external event (an obstacle in this case) will interrupt the normal action flow and bring the execution to a different branch. It can be seen that using the given operators, it is easy to generate plans corresponding to many complex behaviors using a relatively low number of atomic actions.

Petri Net Plans have been successfully used in many fields; Robotic Soccer and Search and Rescue are two of the most successful applications of this formalism.

3.3 Robot Operating System (ROS)

http://www.ros.org/

Robot Operating System (ROS) is a collection of software frameworks for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages[2].

ROS was originally developed in 2007 by the Stanford Stanford Artificial Intelligence Laboratory; then from 2008 to 2013 it was primarily maintained by Willow Garage. Finally, in February 2013, ROS stewardship transitioned to the Open Source Robotics Foundation.

Before the birth of this Robot Operating system, each development (or researcher) laboratory had to build their own in-house specific solution, most of the time extremely tied to a certain robot. This of course meant that a lot

[2]https://en.wikipedia.org/wiki/Robot_Operating_System

of development time had to get wasted to reinvent the wheel and write code to do mundane tasks that had already been written countless times before - pretty much like what happened in the Computer Systems field before the advent of general purpose operating systems. Furthermore this also slowed done the spreading of new algorithms and ideas since there was no easy way to apply a piece of code written for robot A to robot B.

The base idea behind this Operating System is to add an abstraction layer over the hardware and enable the creation of purpose specific but reusable nodes. As a consequence, a library of nodes already exists to solve mundane (but time consuming, if written from scratch) tasks frequently encountered in the robotics field (ex. mapping, localization, ...)

ROS is currently widely used both by researchers and by private companies.

Stage is a robot simulator included in the ROS suite. It provides a virtual world populated by mobile robots and sensors, along with various objects for the robots to sense and manipulate.

3.4 The DIAGO Robot

Due to the social scope of this work, the chosen robot had to fulfill certain prerequisites regarding the hardware.

Firstly, it had to be almost of average human height; socially interacting with something at knee height is usually considered awkward for humans. Furthermore, a few sensors perform better at face height (a microphone, for example – but also any kind of camera performing face detection) and other, such as a tablet, would be uncomfortable to use if positioned too low.

The robot needs to be able to move (possibly fast when required) in a flat environment; currently, wheels are the state of the art for such task.

In one of the possible use cases, the robot carries the shopping bag for a customer. This means it must also be powerful and big enough for such task.

Keeping in mind the aforementioned requisites, we chose the DIAGO robot as our platform of choice.

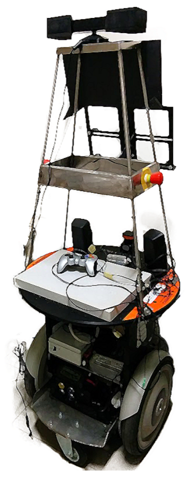


Figure 3.4: DIAGO Social Robot

> DIAGO is a social mobile robot used for experiments in social human-robot interaction, knowledge representation and reasoning, and cognitive robotics.
>
> DIAGO is built on top of a Segway RMP 100 mobile base, equipped with a RoboTorso that contains sensors (2 laser range finders, 2 Kinect RGBD cameras, 1 microphone), actuators (audio speakers), and computational units (one laptop for controlling the mobile platform and the sensors, one tablet for human-robot interaction and speech recognition and synthesis). [32]

3.4.1 Safety

The robot height is 165 cm and it weights around 50kg; such a hefty robot can clearly become dangerous if care is not taken. Safety is handled at different levels, both at the reflexive behavior level and at the high-level reasoning layer.

At the reflexive level, the robot:

- Uses Artificial Potential fields to keep away from the obstacles.
- If the laser stops working, the robot will stop.
- If the Artificial Potential field node fails, the robot will stop.
- If anything is detected nearer than 20cm the robot will halt for then 10 seconds and the recheck.

This, and other, fail-safes make sure that even if the robot has to move in a crowded environment, it can do so safely.

More details on the safety measures implemented in the high-level reasoner will be given in chapter **5**.

Chapter 4

System overview

This and the following chapters (up to the Use Case description) detail the system developed for the thesis as a whole (this chapter) and then observing one layer at a time. Here we describe the different components and their interactions. The different kinds of failures and how to handle them are also analyzed here.

Our system has two main requisites:

- To generate plans to reach goals in accordance to given social rules
- To be able to recover from the very likely failure of the plan and generate a new one if needed

For a certain number of reasons (discussed in depth in chapter 5) we decided to implement the planner using ASP. This allowed us to explicitly represent the social constraints and the common sense reasoning we needed.

Once a plan is obtained, it must be executed and monitored. The actual plan is composed of high level actions; each of these actions is implemented as a PNP which decomposes it into a network of elementary steps (see figure 4.1). It must be kept in mind that the term **action**, depending on context, can hold two meanings in this work: a high level (or ASP or reasoner) action corresponds to an arbitrary complex PNP plan; a low level (or atomic) action corresponds to the actual C++ implementation.

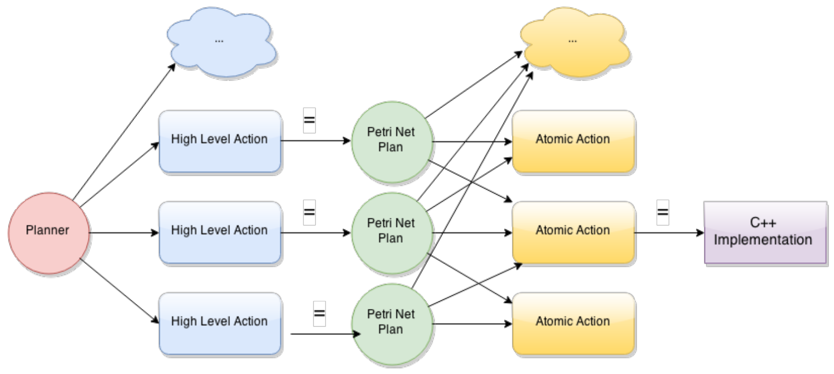


Figure 4.1: Action types

This division of labor, with a reasoning high level planner and complex behaviors modelled in PNP allows to obtain the best from both worlds: flexibility from the reasoner and robustness from the hand made PNPs. Furthermore, we decided to spread the system's resistance to failure on three levels: the reasoner, the PNP and the atomic actions.

4.1 Main Nodes

In accordance with the Single responsibility principle[33], it was decided to build a ROS node for each macro responsibility.

The Reasoner: this node, once it receives a goal, gathers all current sensor fluents, transforms them into ASP syntax and launches Clingo[1](with the static knowledge file) to obtain a plan. If a plan failure arises, it also handles the replanning.

The Plan Monitor: this node has the responsibility of executing and monitoring each macro action in the plan sequence. If something goes unexpectedly it also has to update the reasoner's knowledge base.

[1]The ASP grounder and solver

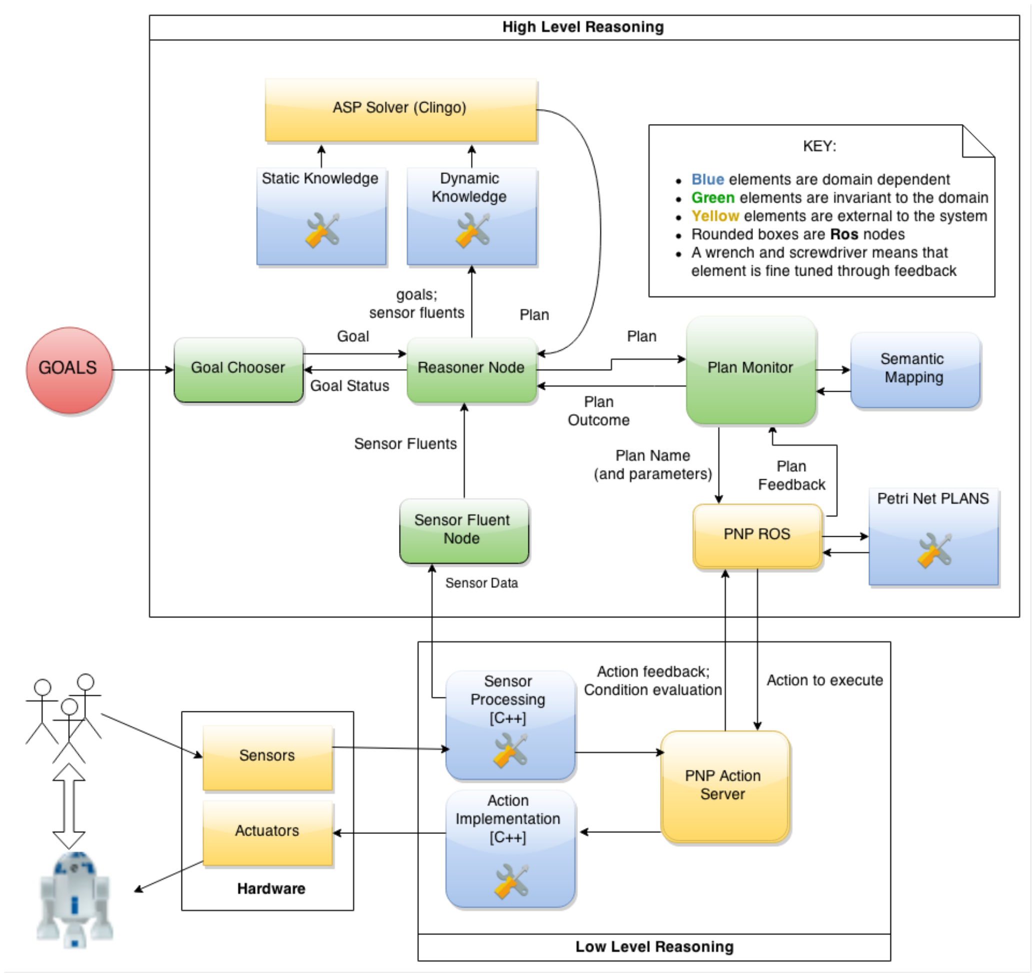


Figure 4.2: The main control architecture.

The PNP Action Server has to execute and monitor the atomic actions of a PNP; it must also evaluate special conditions if present.

The Sensor Fluent (Creator): this node analyses the output of the different sensors available to the robot and produces logical fluents. For example, if the laser range finder detects a point cloud at a certain distance, and a face is detected in a compatible location by the camera, then it can be assumed that a person is present there.

The Goal Chooser: goals given from the outside of the system are sent to this node, each with a certain priority. This node sends them one at a time to the Reasoner and, depending on their outcome and of the priority policy, cycles them.

The Action Implementation: each atomic action has its own server to monitor and control the execution of its own activity.

It must be kept in mind that these nodes do not represent the entirety of the system; for example, the actual planner is not a ROS node but its logic is embedded in a file written in ASP's syntax. All the social rules, all the common sense and *reasoning* of the system is explicitly encoded in that file.

Likewise, the PNP ROS node is part of the PNP Ros library and was not built for this thesis - but it is a part of the system as it interprets the PNPs, interacting with our PNP Action Server, interrupting it to change the flow of the network whenever specific conditions arise.

4.2 Control Overview

A simplified version of the control sequence (shown in figure 4.3) will now be presented.

The whole system is goal oriented, if no goals are defined the robot simply waits idly. As soon as a goal arrives to the **Goal Chooser** node, it is sent to the Reasoner to obtain a plan. If no successful plan is available, the system waits while periodically checking if the newly created sensor fluents allow to reach the goal.

As soon as a plan is found, control passes to the Plan Monitor node. This monitor will execute one high level action at a time by invoking PNP Ros node with the action's name. The PNP Ros node in turn dialogues with the PNP Action Server (PNPAS from now on) by sending the name of the atomic actions that must be executed.

The PNPAS has two tasks: to execute the atomic actions and to signal if any of the interrupt conditions in the PNP are met. If everything goes as expected and the whole plan is executed with success, the plan monitor will inform the reasoner of the success. In turn, the reasoner will notify the goal chooser.

On the other hand, if any of the actions fail, the plan monitor will send the reasoner the (supposed) reason of failure; with this knowledge the reasoner can build a new plan that will, hopefully, work around the encountered problem.

While all that has been described to this point is executing, the sensor's fluent creator constantly processes sensor data and sends logical fluents to the Reasoner node; in this way whenever a replan is in order we are assured that it will consider the most up date data.

4.2.1 Planning model

For efficiency considerations, we decided to execute off line planning. Once a plan is obtained, no replanning is done unless the goal has been reached or failure is encountered. For such a system to work reliably, the implemented action's specifications must be well defined as to match exactly the expected post conditions of the actions in the planner. This key consideration was well kept in mind while modelling failures at the PNP action level.

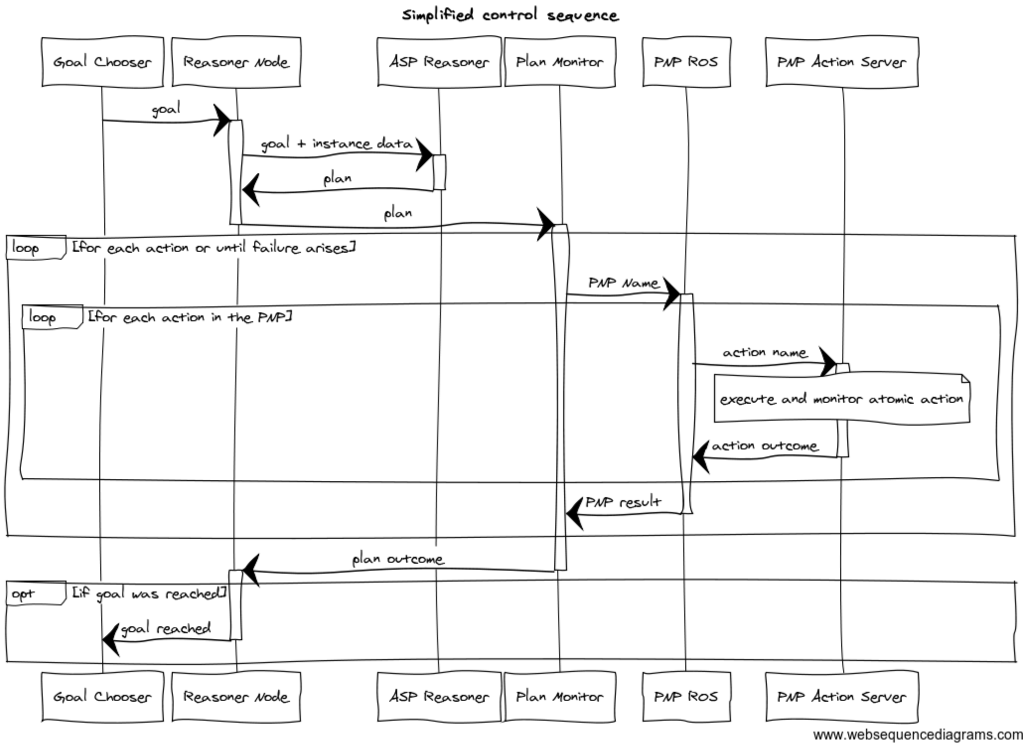


Figure 4.3: A simplified control sequence for the whole system

4.3 Test Driven System Evolution

As can be seen in figure 4.2, some nodes are domain independent - once they meet the functional specifications there is no need to further modify them. They can be easily used again in a project with the same architecture and a different domain.

What needs to change are all the nodes that implement some domain specific behavior: the reasoner model (including the static knowledge), the PetriNet plans and the atomic actions. These elements have been fine tuned by feedback on the considered use cases.

What usually happens in plan based systems, is that, when some failure arises, the developers need to choose where to fix the issue: in the planner or in the action implementation? Most of the time this approach works well, but there are specific cases where the error does not quite fit in any of the two considered layers (more on this in the next section).

In this work, we performed an incremental test-driven development based on three feedback loops (see figure 4.4): a feedback loop on the planner, one loop at the PNP level and one at the action implementation level.

For each considered use case, we firstly made the system output a reasonable optimistic plan. When the robot was performing the optimistic plan with success, we started considering key point were the system had some assumptions and we made sure things would not go as expected. As failure arised, we had the freedom of choosing exactly at what level to fix it (our policy for this is explained in sections 4.4.1 to 4.4.3). As soon as the system started exhibiting successful behavior again, we introduced other elements of controlled chaos. This cycle was then repeated until a reasonable robustness was obtained.

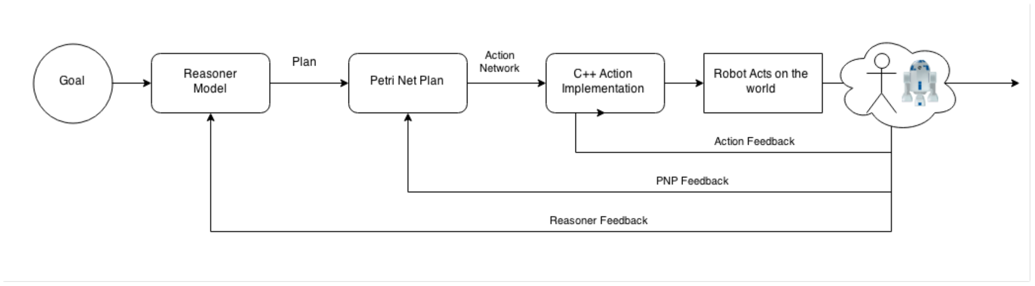


Figure 4.4: Failure based feedback

4.4 Plan Monitoring, Failure and Responsibilities

The most marked difference between theoretical plans and plans used on real life robots is that in the latter case failure is expected. The real world is constantly changing at a rapid pace and any assumption used during the planning phase is likely to stop holding sooner or later. Usually, two different failure types are considered: at plan level and at action level. In practice, this is a continuum more than a binary distinction - as many unexpected events span both levels.

To better discuss this, let us imagine a simple task: the robot must bring a glass of milk to the user. An example of action level failure would be a robot incorrectly grasping the glass of milk: the expected preconditions are there, it makes sense to execute that specific action but for some reason the robot is failing the grasping. At this level, the robot can continue attempting the action until it gets it right and then move on with the rest of the plan. But, if at a certain point, someone takes away the glass of milk from the table then the needed preconditions for the action stop holding and a re-planning is needed (plan level failure).

When something unexpected happens, it must be dealt with: as commonly happens when dealing with complex systems, there are many possible ways to do so. One approach is to attempt to model all possible failures in the plan; this increases the complexity (and thus the needed resources) and risks bogging down the the planner with minute details relevant only to specific conditions and actions. Furthermore, there is no guarantee (on the contrary!) that all possible circumstances have been considered.

A polarly opposite approach is to make only the atomic actions as robust as possible. This has two drawbacks: firstly, if some preconditions stop holding for external intervention (like the glass of milk example) it must still be somehow modeled in the planner or the robot will continue attempting the same senseless action and, secondly, this will lead to a plethora of slightly different, but very specific, actions.

In this work, it was decided to have a three layers system and deal with failure on all different levels; this means the system must have some sort of monitoring on all levels:

4.4.1 Planner level

The planner outputs an optimistic sequence of high level actions; such actions are defined in such a way that problems that can emerge at the action level never become an issue to the planner. Failure for the planner can be defined as some unpredicted intervention from the external world that removes the expected preconditions for an action. The glass can no longer be grasped because it is no longer there, for example.

The reasoner deals with this kind of problem by adding in the KB the fact that the glass is no longer on the table; this means that the next generated plan will try to retrieve the glass from somewhere else, if possible.

4.4.2 PNP level

The PNP layer is what allows us to have a clean high level planner and a relatively low number of hand made atomic actions. Complex behaviors are decomposed into a network of elementary actions with an action flow that can be changed at any given moment by external interrupts. This allows the human developer to model the intricacies of a given situation with a graphical and intuitive high level tool. If a previously unforeseen event makes the plan fail it is a simple matter of adding new interrupts to the network to deal with them.

If the failure arises at a local level, this layers has the task to resolve it without invoking a replan at a higher level. Only if the entire network has become obsolete it should return an action failure.

4.4.3 Action level - C++

For a single, specific case it usually *is* easier to simply make the corresponding action more robust at code level. But, as previously noted, overspecializing will lead each action to perform only a very particular task. Any new functional requirement will require writing and adding a new action.

In this work it was chosen to make a trade-off: the actions are as robust as possible while not loosing generality. As a result, we have few basic atomic actions that are composed into complex and robust behaviors by the Petri Net Plan.

Each atomic action has a very focused and defined goal: for example, to move, a short distance, from A to B or to ask a simple specific question to the user.

As far as monitoring goes, each action thus defined is easy to control. If the robot actually moves from A to B then it is a success, otherwise a failure.

4.5 The Goal Chooser\Manager

Is the node that receives the goals from the external world; it then sends them, together or singularly (see next chapter for more details on multi goal handling) to the reasoner. Goals are broadly categorized in two categories: single instance and recurrent.

Single instance goals are those which represent one time events; once one of these goals is reached there is no reason to accomplish it again. Examples of this kind of goals are: helping a customer bring his bag to the parking lot, giving a tour of the whole shopping mall to a VIP and so on.

Recurrent goals are meant to be executed over and over again. Promoting a daily discount is a fine example: as soon as one customer is been informed, it is a perfectly good idea to repeat the process with a new client.

This distinction of objectives is unknown to the reasoner, it is handled uniquely in this node: a completed single instance action is marked as such and removed from the pool of waiting goals, a completed recurrent goal is marked as such but is sent back to the pending pool.

Furthermore, each goal has an assigned priority to decide which goal must be completed first. Priorities are meant to be set by the robot's managers (the shop owners in our example) at the moment of goal creation.

Chapter 5

The reasoning layer

This chapter describes the high level, knowledge-based, planner built in ASP. Different examples of social aware planning and common sense reasoning are presented.

This is the strategic core of the system. It decides, with broad strokes, what is the best plan to reach a given goal.

5.1 Why a planner based on reasoning?

One of the main requisites of this project was that the robot should be able to plan following social rules and common sense.

For a robot to be social, it means it must behave in accordance to what we humans consider social rules; but these rules are, most of the time, embedded in common sense and tradition. As a consequence, most of these unwritten laws change oftenly in time and geographic distance. This must be kept in mind, as it suggests that social behavior on a robot should be implemented in such a way as to allow for easy modifications.

Many social robots **implicitly** model such constraints in the planning layer. In this way, these principles are hard coded into the robot; this approach, while functional, has some drawbacks.

Firstly, there is no explicit representation of these social rules - only the original developer actually knows how and where they are embedded. Anyone else, not fully familiar with the system would be clueless to their existence.

Secondly, this intrinsic modelling is extremely hard to maintain and update. What happens if new social rules are introduced? Basically the whole system must be rethought.

As noted, manners and social policies commonly vary in time and geographic location; the need of new features can emerge at any given time. To build a planner based on a high level reasoning core is an excellent way to handle small and big variations to plans with the least amount of effort. This approach allows to explicitly model the social rules and the common sense assumptions that a smart system should follow. Simply adding one additional rule can completely change the generated plan, in order to comply with the new restriction.

As a, non socially related, example of plan generation flexibility, let's assume that our robot is currently being asked by a customer how to reach the closest restroom. It checks its knowledge base and finds that the closest restroom (appropriate for the customer's gender) is a certain one, say restroom1. We can safely assume that all restrooms must be cleaned once or more per day and that the work of the cleaning crew is faster if no customers are around. Simply notifying the robot that a given restroom is being cleaned is enough to make him generate a different plan next time he is asked for the location of the closest toilet.

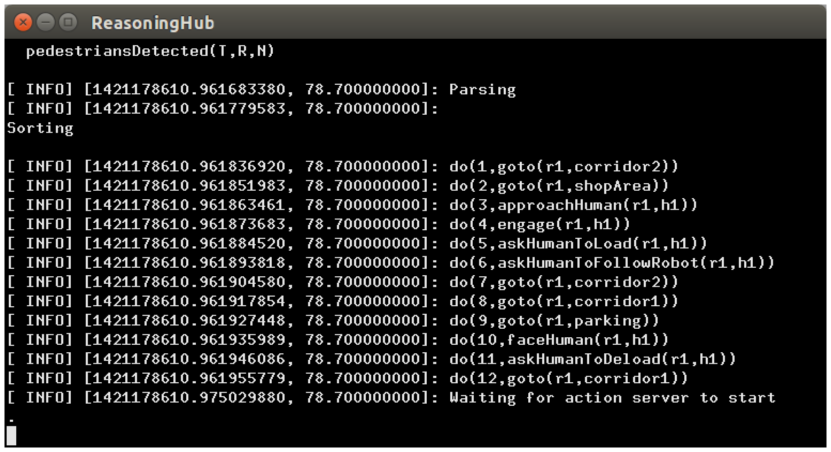
5.1.1 Why ASP?

Different reasoning tools might have been used but Answer Set Programming was chosen as it provided us with a series of advantages[26, 10, 13, 34, 27] in this task:

Expressivity - ASP allows constructs which simplify the formalization of certain concepts. **Aggregates** (count, sum, min, max, average, even and odd) are useful to reason on sequences of fluents or predicates, **cardinality constraints** help to select a variable number of elements from a given set, **defaults and exceptions** have proved an invaluable tool to model simply many basic rules. The presence of both **strong negation** and **negation as failure** allow for a simple solution to the frame problem.

An Intuitive way of modeling - modeling in ASP is relatively straightforward: we describe the world as it is. No need to worry about how ASP works under the hood and\or write statements in a certain order (as in Prolog, for example).

Efficiency - in a similar domain[13] it has been seen that the Clingo reasoner uses less memory and has comparable computation times to a causal reasoner.



```
  pedestriansDetected(T,R,N)

[ INFO] [1421178610.961683380, 78.700000000]: Parsing
[ INFO] [1421178610.961779583, 78.700000000]:
Sorting

[ INFO] [1421178610.961836920, 78.700000000]: do(1,goto(r1,corridor2))
[ INFO] [1421178610.961851983, 78.700000000]: do(2,goto(r1,shopArea))
[ INFO] [1421178610.961863461, 78.700000000]: do(3,approachHuman(r1,h1))
[ INFO] [1421178610.961873683, 78.700000000]: do(4,engage(r1,h1))
[ INFO] [1421178610.961884520, 78.700000000]: do(5,askHumanToLoad(r1,h1))
[ INFO] [1421178610.961893818, 78.700000000]: do(6,askHumanToFollowRobot(r1,h1))
[ INFO] [1421178610.961904580, 78.700000000]: do(7,goto(r1,corridor2))
[ INFO] [1421178610.961917854, 78.700000000]: do(8,goto(r1,corridor1))
[ INFO] [1421178610.961927448, 78.700000000]: do(9,goto(r1,parking))
[ INFO] [1421178610.961935989, 78.700000000]: do(10,faceHuman(r1,h1))
[ INFO] [1421178610.961946086, 78.700000000]: do(11,askHumanToDeload(r1,h1))
[ INFO] [1421178610.961955779, 78.700000000]: do(12,goto(r1,corridor1))
[ INFO] [1421178610.975029880, 78.700000000]: Waiting for action server to start
.
```

Figure 5.1: A sample plan

5.2 Description

This reasoning layer, based on Answer Set Programming, has the responsibility to generate a high level optimistic plan. The ASP solver is invoked on two files: one containing the static knowledge and the other, dynamically created, containing situation specific information (the instance file).

These two component will be described in more detail in the following paragraphs, but as a general overview they can be thus described:

The Static Knowledge module is the constant element of this layer. It holds all stable information on the environment, the available actions, social rules and common sense intuitions.

The Instance Data file is meant to change at each invocation of the reasoner layer. It models the user defined goals and the robot perceptions at that specific moment.

Static Knowledge
Dynamic Knowledge
PLAN

Figure 5.2: The plan is generated by the union of static and dynamic knowledge

5.3 Static Knowledge

This module contains all that is meant to be relatively stable: core social rules, common sense reasoning assumptions, the definition of all the available actions (pre conditions and post conditions) and the stable information on the environment (the topology, the map semantics, ...). It is the *mind* of the robot and it is tuned to a high level of abstraction for many reasons, with efficiency as a first concern.

5.3.1 Common Sense knowledge

Every day we constantly assume a great deal about our environment. For example. we assume that a red traffic light will eventually become green, that someone walking towards us and then stopping right in front has the desire to interact with us (more on this later) and many more. We know that if we keep a bag in our hand and we go to work, the bag will have moved with us (ramification), but if we try the same experience with an ice cube we will arrive at work empty handed.

These speculation are what allows us to perform efficiently; we perform apparent feats of foretelling by acting as if we know what will happen. Sometimes we are wrong, but most of the times our assumptions allows us to behave in a *smart* way.

We spend all our life learning what we call common sense knowledge and our robots do not have this luxury (at this time, automatic learning of common sense knowledge is considered AI-Complete); this means that if we want them to act smartly we need to encode this knowledge somehow. As previously discussed for Social Rules (in section 5.1) it is usually a better idea to model them explicitly.

Some example of domain specific assumptions used in the planner - they mostly seem trivial to us but they really are not for a robot:

Objects on the robot move with the robot: this obvious fact must be modelled somehow if we want the robot to find a plan to carry bags to the parking area.

```
% objects on the robot move with the robot
at(T, B, L) :- robot(R), at(T,R,L), onRobot(T,R,B),
    time(T), object(B).
```

People will follow a robot carrying their bag: this is a reasonable assumption.

```
% humans follow robots with their objects
at(T, H, L) :- robot(R), at(T,R,L), onRobot(T,R,B),
    object(B), hasBag(H,B), time(T).
```

Customers are more likely to buy something they like: imagine the robot has proposed the offer of the day to a customer and he is not interested. The system might replan and propose a random new offer but it would be a matter of pure luck if the customer is interested or not. A better approach would be to know that humans tend be interested in things they like.

```
%% The current offer for a customer is the main one,
    unless he is not interested in it.
currentOffer(H, Offer) :- mainOffer(Offer), human(H),
    not -humanInterested(H, Offer).
%% If the human is not interested in the main offer,
    try something he likes
currentOffer(H, Offer) :- -humanInterested(H, MainO),
    mainOffer(MainO), human(H), offer(Offer), likes(
    H_db, Offer), sameUser(H, H_db).
```

5.3.2 Social rules

5.3.2.1 Rules of engagement

The most simple(and widespread) modality of human-robot interaction (notice the ordering between human and robot) is based on a human actively approaching a passive robot, waiting for human input. A different approach is possible: the robot might start interaction on its own initiative.

This proactive paradigm enables the robot to help people with potential needs. As noticed by [6], some users might not be aware of the presence of the robot and\or might not be knowledgeable about its capabilities!

Work by Finke et al. [5] has investigated how the person's movement and distance from the robot can be used to assess the interest of a person in interacting with the robot.

By combining Hall's work [2, 1] with studies done on the Robot-Human interaction[3, 4, 5, 6] we designed the following protocol for approaching a potential conversation partner.

5.3.2.2 Social Distances and orientation

If safety concerns allow for the robot moving around (more on this in the following paragraphs), we have to decide who to approach. As discussed in other cited works, it is appropriate to select a human that (1) is not in a hurry (this can be roughly estimated by his speed and trajectory) and (2) is close and slow enough to be effectively reached by the robot. Furthermore, whenever choice is available, we can rank potential targets by their trajectories (favoring those who appear to be moving towards the robot) and by their position (favoring the closer ones). (how the robot handles social interaction when standing still for safety reason is explained in paragraph **Social Interaction with Safety Concerns:**).

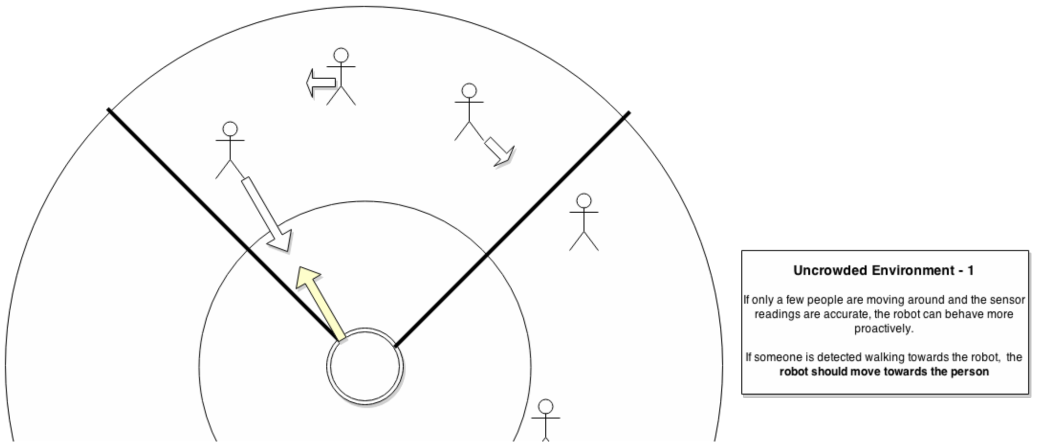

Uncrowded Environment - 1
If only a few people are moving around and the sensor readings are accurate, the robot can behave more proactively.
If someone is detected walking towards the robot, the **robot should move towards the person**

Figure 5.3: If safety concerns allow it, the robot should move towards promising targets

It was felt that this kind of discrimination was better suited for an ad hoc node; this node generates the fluents that tell the robot who is appropriate for an approach. Unsuitable persons are invisible to the reasoner.

As pedestrian tracking and trajectory estimation is still not a solved problem and it was not the main goal of this thesis, the node used in practice (based on a laser scanner) is simplified and can track the distance of only one person.

5.3.2.3 Safety

Given the size of the robot, it is appropriate to make it reason on ***when*** and ***how*** should it be able to move.

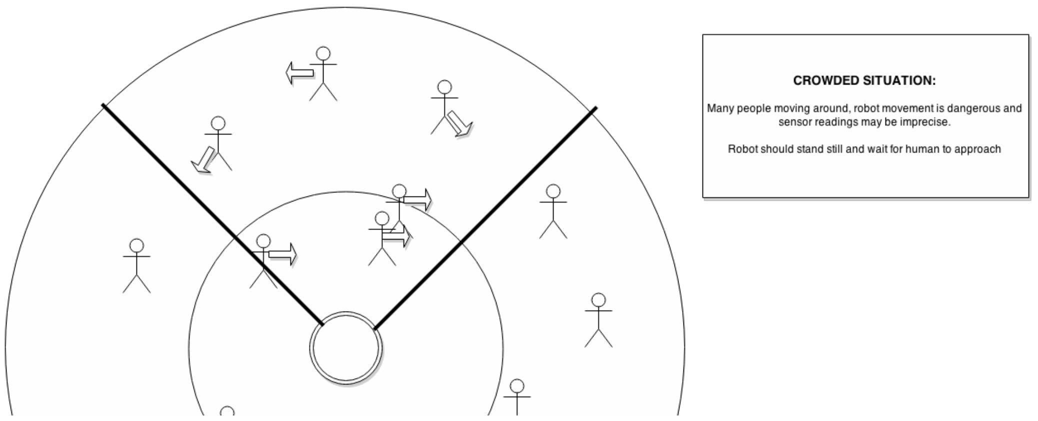

CROWDED SITUATION:
Many people moving around, robot movement is dangerous and sensor readings may be imprecise.
Robot should stand still and wait for human to approach

Figure 5.4: A crowded environment

When When too many people are moving around it (or when the sensors are too noisy to get reliable information) is safe to assume that the robot should not move.

```
%% when more than a certain number of people are detected
   it's crowded
crowded(T) :- detected(T, N), time(T), N>THRESHOLD.
%% it is not possible to move when it's crowded
:- goto(T,R,L), crowded(T), robot(R), location(L), time(T).
```

```
%% it is not safe to move when too much sensor noise is present
:- goto(T,R,L), highSensorNoise(T), robot(R), location(L), time(T).
```

Social Interaction with Safety Concerns: In such conditions, the robot will no longer proactively move towards the human but will still behave actively by initiating a conversation on its own accord if a human is detected in the appropriate circumstances.

If a human is in social distance, facing towards the robot and is standing still for a little amount of time (just to make sure he is not simply passing through) the robot will face him and then start conversation.

```
canEngage(T,R,H) :- isFacing(T,R,H), canSee(T,H,R), inSocialRange(T, R, H), time(T), robot(R), human(H).
% preconditions for engaging
0 {engage(T,R,H): human(H), canEngage(T,R,H)} 1 :- time(T), robot(R).
% effect of turning:
isFacing(T+1, R, L) :- faceHuman(T,R,H), robot(R), human(H), time(T).
```

Simply telling the reasoner these kind of information is enough to make it build a plan to behave as expected.

It must always be kept in mind that sensor readings might be unreliable at times and it is possible that a user approaches the robot without being detected. In such a case the human can call the robot by name or use the available touchpad to automatically start a conversation.

How: How fast the robot can move can be easily encoded in the KB, for example, in relation to the number of detected people:

```
%% if no one is detected go as fast as possible
speed(T, fast) :- detected(T,0), time(T), not exception(T).
```

```
%% go slower as more people are detected
speed(T, normal) :- detected(T,N), 0 < N < FEW_PEOPLE, time
    (T), not exception(T).
speed(T, slow) :- detected(T,N), FEW < N < MANY_PEOPLE,
    time(T), not exception(T).
```

And just as easily we can model exceptions:

```
exception(T) :- inHurry(T), time(T).
speed(T,fast) :- inHurry(T), time(T).
```

5.3.3 Action definition

From the system's point of view, an action is completely defined by its preconditions and postconditions. Preconditions are what must be true at time T, postconditions describe what will hold at time T+1. To model the concept of pre and post we need the concept of time:

```
time(1..T) :- maxMoves(T).
```

This code tells the system that we have from 1 to T time steps available, where T is the number of maximum moves.

Some examples of action definition:

```
0 {goto(T,R,L): location(L), at(T,R,L1), connected(L,
    L1) } 1 :- time(T), robot(R).
```

This particular line is telling ASP to generate, for each time step and each robot, the set of all possible **goto** statements which satisfy the preconditions (the new location must be connected to the one the robot is currently in) and then select one or zero. **If** a goto is generated (another action might be chosen instead) it will trigger the action's postconditions:

```
% effect of moving:
at(T+1, R, L) :- goto(T,R,L), robot(R), location(L),
    time(T).
```

5.3.4 Constraints on the number of concurrent actions

At the level of abstraction at which the planner works, it was decided to allow one single high level action per time step:

```
1 %% LIST OF ACTIONS:
2 do(T, goto(R,L)) :- goto(T,R,L), time(T).
3 do(T, approachHuman(R,H)) :- approachHuman(T,R,H),
      time(T).
4 ... other actions defined here
5 do(T, engage(R,H)):- engage(T, R, H), time(T).
6
7 %% CONCURRENCY COSTRAINTS
8 %% ===================
9 :- do(T, ACTION), do(T, ACTION2), time(T), ACTION!=
     ACTION2.
```

Line number 8 is a headless rule, the ASP way of saying that a certain condition should not happen (two actions executed at the same time).

5.3.5 How the Frame Problem was handled

The frame problem is the problem of formalizing the commonsense law of inertia: Everything is presumed to remain in the state in which it is.[26]

ASP allows the use of both **strong negation** and of **negation as failure**. This enables us to model inertia in an elegant way (here we consider the example of inertia applied to the position of an actor):

```
at(T+1, R, L) :- at(T, R, L), not -at(T+1, R, L),
     location(L), actor(R), time(T).
```

An actor positioned in location L at time T will still be there in the next time step **if it is not know** that he is not there.

While in Situation Calculus the number of required frame axioms in A*F (A actions, F fluents), in ASP we need to model only a frame axiom for each fluent, independently from the number of actions.

Furthermore, choosing a different formalization, this can be reduced to the following fixed statements:

```
1 holds(F,T) :- occurs(A,T), add(A,F).
2 nolds(F,T) :- occurs(A,T), del(A,F).
3 holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
```

If an action occurs at time T and that actions adds the fluent F then F holds at time T; the opposite is true for fluent deletion (line 2). Finally we can say that what holds at time T-1 will hold at time T if it is not know that is does not hold (nolds).

5.3.6 Semantic Map

One of the most basic use cases considered in this project describes a visitor asking the robot for directions. The robot is then expected to guide the user directly, if safety concerns allow him to move, or show him how to reach the destination otherwise. Another, related, use case illustrates the event where a visitor ask where he can find a shop selling a certain item.

In order to successfully resolve both cases, the system needs a semantic map of the environment to know how to reach certain locations and what is available there.

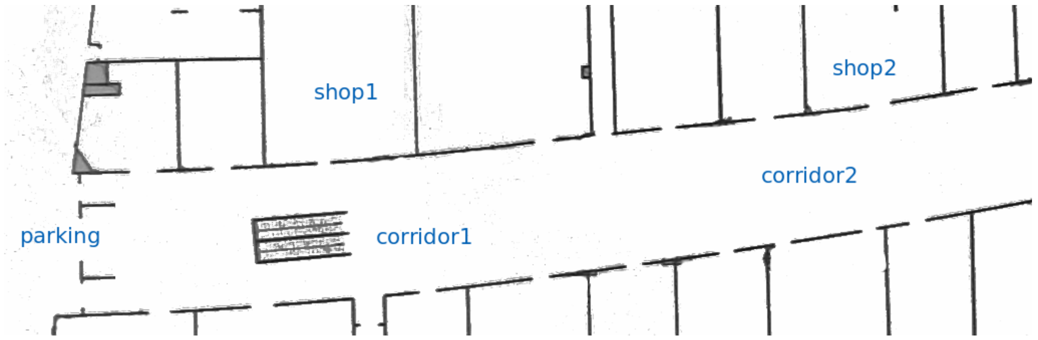


Figure 5.5: Section of the Rives De L'Orne map with some semantic labeling

This information can be modelled in the following way (a simplified example):

```
%% environment knowledge
%% ==========================
%% available locations
shop(shop1).
shop(shop2).
location(X) :- shop(X). % a shop is a location
location(corridor1).
location(corridor2).
location(parking).

%% Map layout
connected(corridor1, corridor2).
connected(corridor1, parking).
connected(corridor1, shop1).
connected(corridor2, shop2).
% connections are bidirectional:
connected(X,Y) :- connected(Y,X), location(X), location(Y).

%% What does the shop sell
sells(shop1, food).
sells(shop1, wine).
sells(shop2, clothes).
```

Such formalization is all is needed to answer different user queries on shop location and content.

5.4 Instance Specific Data

This module is generated on the fly; this is what allows the planner to perceive the actual world and to accomplish user defined goals. It is composed of sensor fluents, which give information on the world, user(managers, shop keepers) defined goals and special rules.

5.4.1 Sensor Fluents

Sensor fluents enable the planner to *perceive* the external world. All relevant sensor reading are transformed into logical fluents of the following form:

```
typeOfSensorReading( time_of_the_sensor_reading,
agent_who_made_the_reading,
value1, value2, ...)
```

Here are some commonly used sensor fluents (time is not considered here for simplicity):

Sensor Fluents	Description
inApproachRange(r,h1)	The human h1 lies in the area the robot r can safely navigate
inSocialRange(r,h1)	The human h1 lies in the robot's social range. This is a prerequisite for starting a conversation.
humanFacingRobot(r,h1)	The human h1 is currently facing the robot
detectedPeople(r, 6)	Robot r sensors detect 6 people around him
noiseLevel(r, 50dB)	Robot r is detecting an acoustic level of 50dB

All of these fluents are used by the reasoning system to come up with a better plan. Is the noise level too high? Then it is unlikely that the human can hear the robot, better output to screen. Is someone in the robot's social range and looking at it? It might be a good time to start a conversation.

5.4.2 Goals

Goals represent the end condition we want to reach with our plan. We can model them in the following straightforward way, simply describing the end result:

```
%% We want the robot to help a customer bring his bag
    to the parking area
goal(T) :- at(T, bag1, parking), at(T,r1, corridor1),
    time(T).
```

We want the customer's bag to be located in the parking area and the robot back to this starting position. How to accomplish this is left to the system.

If more than one goal is given at a certain time, there are three ways to handle them:

1. Add one goal definition for each task - this means the reasoner must find the plan to reach at least one of the goals, but there are no guarantees he will reach them all.

2. Combine all the objectives in one single goal - all goals must be satisfied by the same plan. This might not be always possible.

3. Add one goal at the time to the instance file and keep track of which tasks have been completed in the **Goal Chooser** node.

Solution 2) has at least two drawbacks:

- Given certain sensor fluents, it might be possible to accomplish only one of the goals and then at a later date new events will enable another goal and disable the first. In this case a plan that accomplishes everything will never be found.

- A plan with more steps (as needed to accomplish more goals) is computationally harder to find - for efficiency considerations we have set an upper bound to the maximum plan size.

Strategies 1) and 3) can work together if needed and cater for different needs; both were used in our implementation.

5.4.3 How to get the minimum plan?

The planner is structured in such a way that it needs to know at what time step we want the goal conditions to hold:

```
:- not goal(T), maxMoves(T).
```

If by the last time step (maxMoves) the goal still does not hold there is a failure; this means no plan was found.

This architecture is exploited by the ROS node executing the ASP reasoner to select the plan with the smallest number of steps. The planner is executed at different times while iterating on the number of max moves: if it does not find a plan with 1 move it will attempt to find one with 2 and so on, up to the maximum number of moves.

5.4.4 Special rules

Special rules are meant to be defined by end users (managers, shop keepers) and are used to inform the system that a temporary change to the knowledge base is necessary. For example, during the holidays the shopping center might add a Christmas stand in a certain location and shop keepers want the robot to advertise it. To inform the system of this new place we can formalize:

```
location(christmas_stand).
connected(corridor1, christmas_stand). % how to reach
    the christmas stand
```

And then send the system the goal of advertising the stand.

At other times, a shop might temporarily close. Instead of removing it from the KB we notify the system that it is currently closed but will reopen; in this way the system will still be able to answer specific queries on it if needed.

Chapter 6

Robust action execution: PNPs

In this chapters it is explained why the PNP formalism was chosen and how it helps building a more robust and efficient system

This layer's job is to take a high level task given by the planner and execute and monitor the appropriate atomic actions needed to accomplish it. Thus every action defined by the planner corresponds to a PNP; each PNP is composed of atomic actions and can range in complexity from a single sequence (to simply execute one or more actions without any kind of special logic) to a complex network using forks, joins, interrupts, conditionals and all the other operators available in this formalism.

6.1 Motivation: Advantages of using PNPs

This level deals with the execution of complex behaviors, decomposing them into atomic actions. Failure at this level can exist at different levels of complexity.

Let us imagine that our robot is currently loaded with a customer's bag and must reach the the parking zone. At a first glance this might appear as a simple atomic action; the robot is at point A, it must reach point B. But if we treated it as such, there would be quite a number of possible failures that would have to be considered at the planner level.

We are assuming that the human will follow a robot carrying his bag, but what happens if the human stops, maybe to greet a friend along the way? Should the robot simply continue moving, oblivious to this event? Should the system generate a plan in which the robot stops at every second to make sure the customer is coming along? We could simply model it as a plan failure, but does it make sense to model and replan such a situation if the human is actually only stopping for 10 seconds?

Adding these minute details to the planner can easily cause it to become incredibly complex and slow. Besides, this specific failure is strictly related to this action and adds nothing of value for other situations.

We could create a very specific action, goToParkingWhileCarryingBag-ForHuman, and it would probably work well; but it is easy to see that we would need to write dozens (if not more) of such actions for each specific case. Doing so in a relatively low level language such as C++ is error prone and time consuming.

PNPs allows the definition of complex behavior and failure handling using atomic level actions. To continue the given example, we would model a robust sequence with the Petri Net show in figure 6.1:

The basic action is a goto from A to B, but we have modelled that this action should be interrupted if the user is no longer following the robot. If the stop is only a few minutes long the robot simply waits; if more, then a supervisor is notified of the event (we assume we do not want the robot to spend the whole day waiting).

The point here is that we can make this high level action as complex and robust as we like with a minimal effort and without over complicating the reasoner.

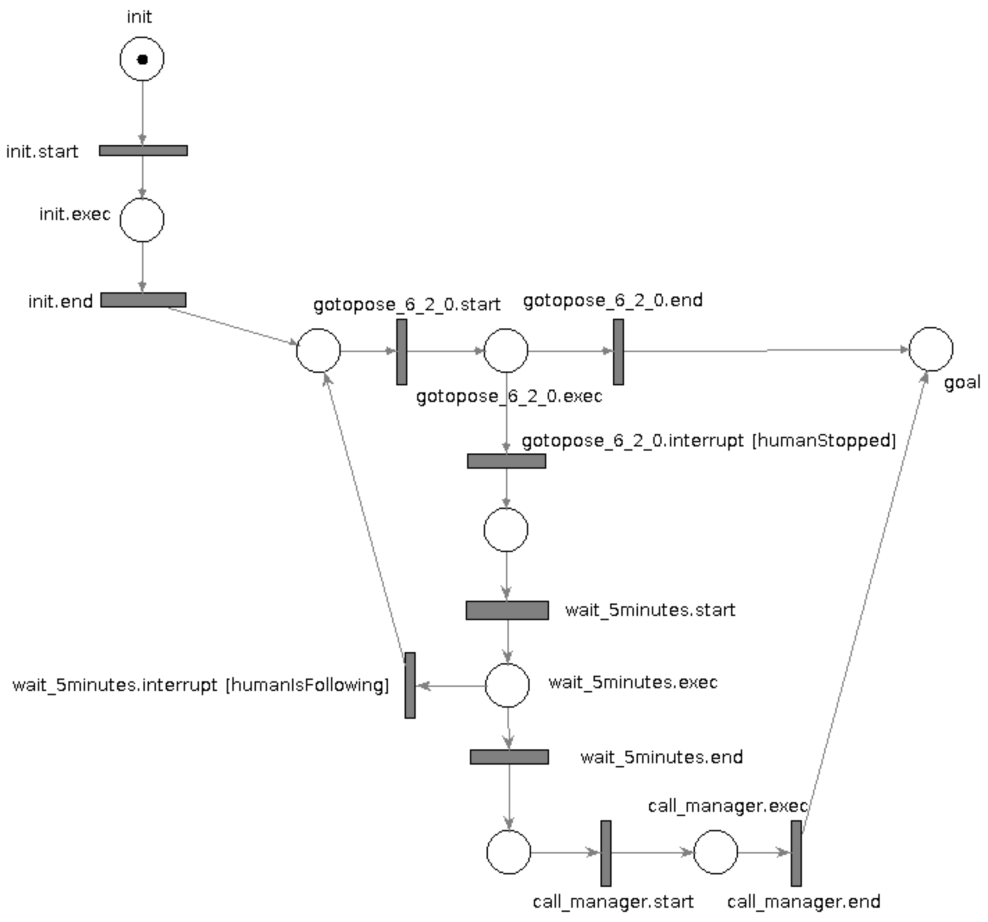


Figure 6.1: A PNP that can handle the carry bag problem

6.2 Improve the multimodality of the experience

At the planner level, we assume that our robot can execute one single action at a time. While this is a reasonable assumption at the level of complex behaviors, when decomposing them at a lower level we might find out that we actually need to execute two or more basic actions.

As an example, let us consider the **GiveDirections** action, which is used to make the robot give instructions on how to reach a specific place. There are at least three ways in which instructions can be given: by voice ("go left"), on the screen (a map) and by pointing with the robotic arm. These three methods are all independent one from the another and executing them at the same is a sensible choice which will probably improve the user's experience.

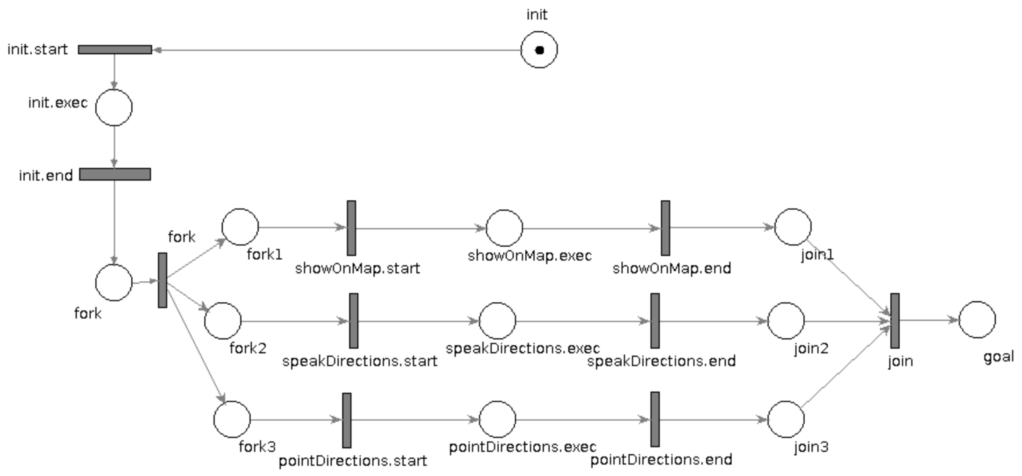


Figure 6.2: A PNP used to execute a task in a multi modal way

6.3 Improve action reliability and replanning insight

When a plan action fails there are two key points to consider: (1) was there any chance that executing a variation of the action it would have succeeded, and if not (2) why did it fail. On one hand we do not want to give away a perfectly valid action sequence for a minor inconvenience and on the other hand if a replan is needed, it must be clear why it failed.

Handling these two points at plan level is extremely hard as a lot of action specific knowledge must be included to understand the reasons of the failure. Using PNPs we can build a network that does what is possible to successfully complete the goal in case of adversity and if it fails it will give feedback to the reasoner so that the next plan will consider the problem and, hopefully, find a way around.

As an example, we can consider the case where the robot is guiding a customer to a specific shop. We might deal with two kinds of possible failures:

- Someone is temporarily blocking the robot's path. We could imagine that the robot asks the human to give way and then proceeds to destination. If the human refuses to move it would be sensible for the planner to come up with a new route (if present). If no such route is available the robot should apologize to the user and give him directions.

- The robot detects that the human is no longer following him and is no where to be seen. In this case the planner will recognize that no plan can work and return a failure to the **Goal Chooser** node.

In figure 6.3, a failure resistant PNP is showed.

Some important elements to notice:

- There are three end places: goal, fail:pathBlocked and fail:lostHuman.

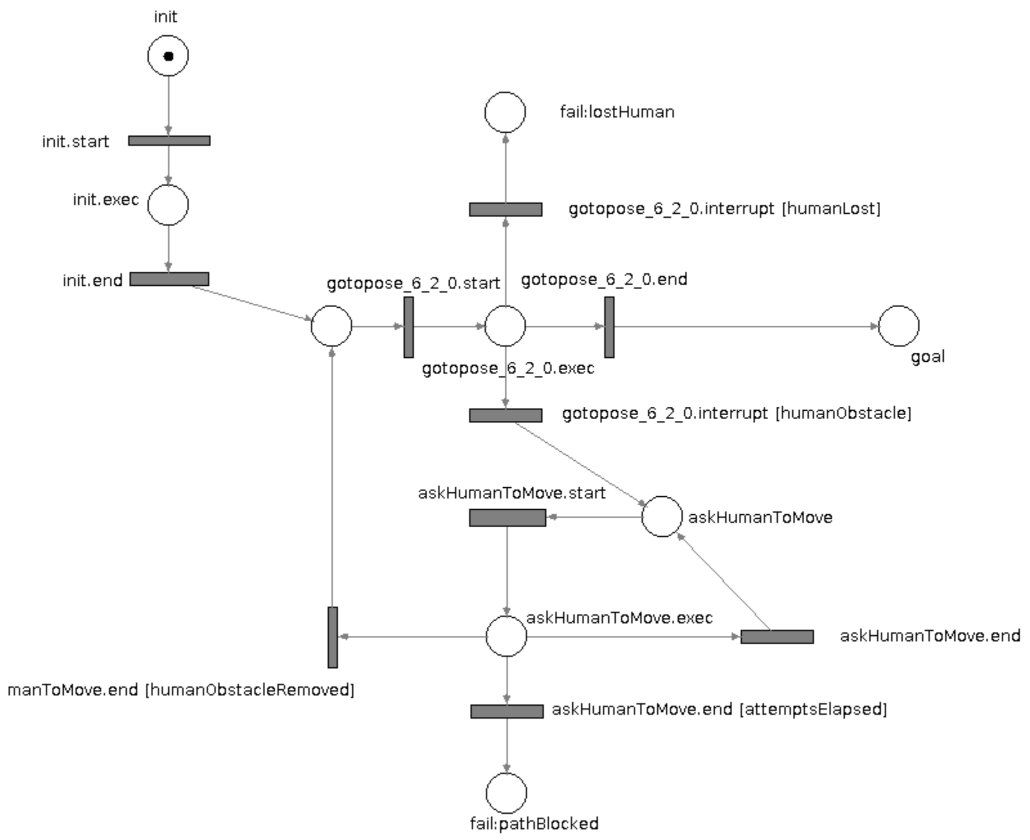


Figure 6.3: A failure resistant plan for guiding a human

Goal is success, the other two places model a specific kind of failure (of course this feedback is sent to the reasoner)

- The PNP is structured in such a way as to attempt resolving the issue before reporting a failure; in this case it will ask the human acting as an obstacle to move. Only after a few failed attempts will the whole PNP report the problem to the reasoner.

Chapter 7

The Ros layer

This chapter provides an overview of the lower level control architecture and describes some of the ROS nodes that were built to support the system.

7.1 Ros nodes and topic interactions

In order to feed the higher level reasoning with information on the world and to execute its plans, a complex low level network of ROS nodes exist. A full description of this layer would not add much to this thesis but some structural points can be highlighted to give some insight.

Topics Are a mechanism provided by ROS to establish a communication between publisher nodes and subscribers where there is no need for an answer. We use this mechanism extensively: to send sensor fluents and goal definitions to the reasoner node, to signal PNP Ros which plans to execute, to broadcast velocity commands and so on.

Services Are used to model request\reply interactions between nodes; usually the kind of request can be replied in a short amount of time. The Reasoner node (which receives all the sensor fluents) exposes a service to allow other low level nodes to query the current knowledge base.

This allows us to reduce redundancy and have only one version of the world.

Actionlib Whenever the request for a task might require too much time, the Services infrastructure might not be ideal as it does not allow any kind of control on the execution of such task. In our system, all atomic actions servers (the nodes that actually execute a specific action) are implemented by extending ROS actionlib servers. In this way we can exploit the existing infrastructure to receive feedback on the status of the action and eventually stop it.

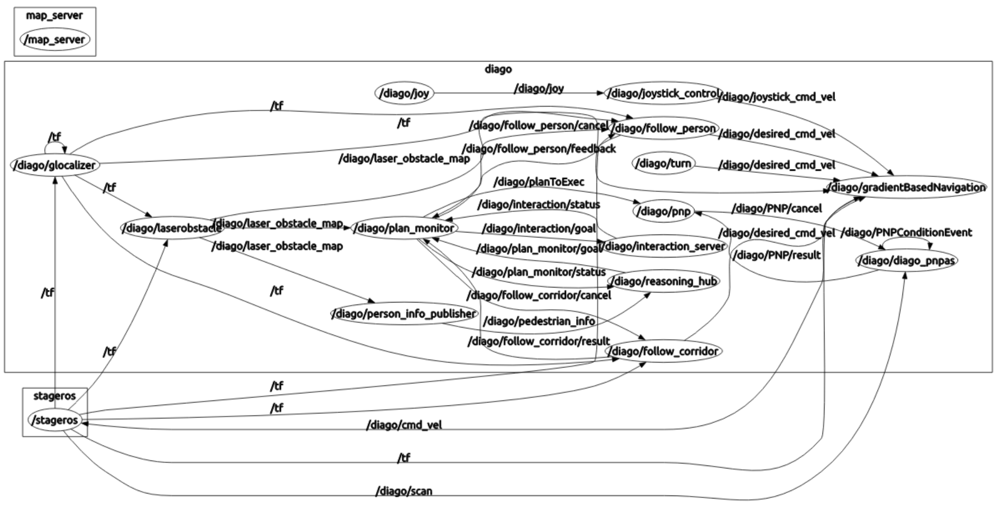


Figure 7.1: A view of the interactions between ROS nodes during a plan execution (for the purpose of visual clarity some nodes were removed from this image)

7.2 Fluents from sensors readings

There are a certain number of sensor readings that can be fused into a logical fluents; as this was not the focus of this thesis it was decided to use only the laser range finder and a microphone as a source of information on the world. It can be seen that with little effort it is possible to add better sensing capabilities by simply improving the nodes generating the fluents.

By using the laser range finder we can reliably localize the robot (thanks to the GLocalizer node); this information can then be transformed into a semantic label. We can also estimate the distance of a human (LaserObstacle node) and use this information to establish his presence in the social or in the public range.

The microphone provides two kind of information: the average noise level and the words spoken by a human. The first in an interesting auxiliary information to know, the second is used for interactions.

7.3 Safe Navigation

As mentioned frequently during this thesis, we hold the belief that safety is of the utmost importance, especially in the case of a social robot moving in a crowded environment. To cover all bases, we considered this at the reasoner level, at the PNP level and at the lowest level of control.

At this level we increased safety in two different ways, directly and indirectly.

Directly: it was decided to route all the velocity outputs generated by the atomic actions towards a node we call **Gradient Based Navigation.**

Indirectly: the maximum linear and turning speed of the robot were tuned to minimize any collision risk.

The Gradient Based Navigation node, written by Marco Imperoli, receives velocity inputs from other Ros nodes and transforms them in accordance with the artificial potential fields generated by the obstacles detected by the laser. Basically, whenever the robot approaches an obstacle it is progressively slowed up to a stop, before having any chance of hitting it. If the robot is far enough from any repulsive field, he will not be affected by it.

With the appropriate parameters, this node strongly improves the navigation safety. We improved this node to handle unexpected failures from other nodes and to receive an emergency stop signal and\or an override from the joystick in the unluckily event that a safety hazard was detected by human supervisors.

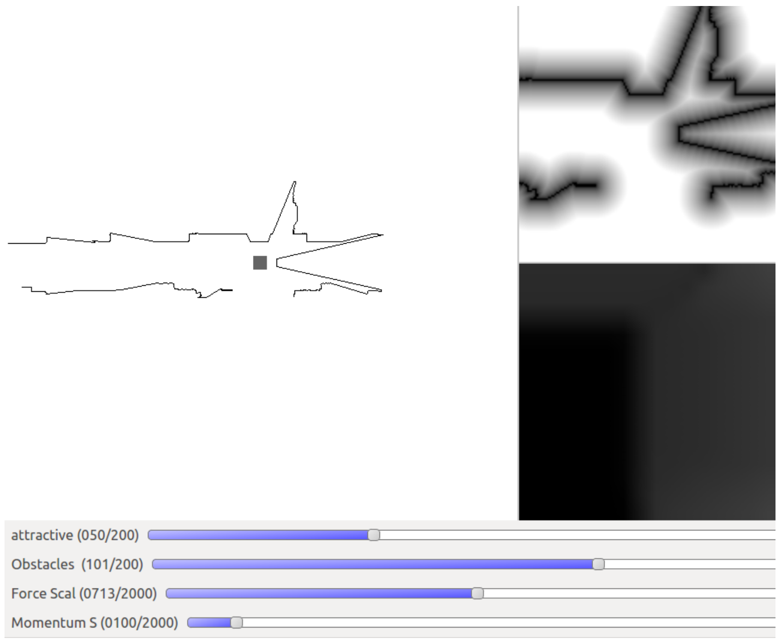

attractive (050/200)
Obstacles (101/200)
Force Scal (0713/2000)
Momentum S (0100/2000)

Figure 7.2: The Gradient Based Navigation GUI

7.4 Social Interaction Server

As part of being social includes interactions, it is not a huge surprise that in all the considered use cases there was some form of communication between the human and the robot. In order to deal with this requirement, a specific action node was written. Since Natural Language Understanding is still an unsolved problem, we decided to structure the interactions in the simplest possible way: the robot asks a binary question using a text to speech engine and then awaits for the human to answer, by voice, yes or no. If the user answers something else, the robot simply remarks to reply with a yes or a no. If no reply is received in a certain interval of time then this actions is considered as failed. More complex dialogues are expressed as a tree of these atomic interactions by using the PNP formalism.

7.5 PNP Action Server and PNP Ros

In order to use the PNP formalism at least two nodes are needed: the PNP Ros node, part of the PNP library, and the PNP Action Server, written for this thesis.

As soon as PNP Ros receives a plan name over a topic, it starts the execution and monitoring of the PNP; whenever needed, it will invoke the PNPAS to make it execute an action or evaluate a condition. In PNPAS we basically execute and monitor the basic action requested by the PNP Ros and control any potential conditions to evaluate.

Chapter 8

Experimental Evaluation

After describing the intricacies of the system in previous chapters, it is time to appraise its performances. This chapter presents an overview of the experimental setup we used to evaluate our work. The different use cases are described and considered, highlighting the focus points of each. Results obtained by the system in each case are discussed, presenting successes and potential shortcomings of our approach.

8.1 Experimental Setup

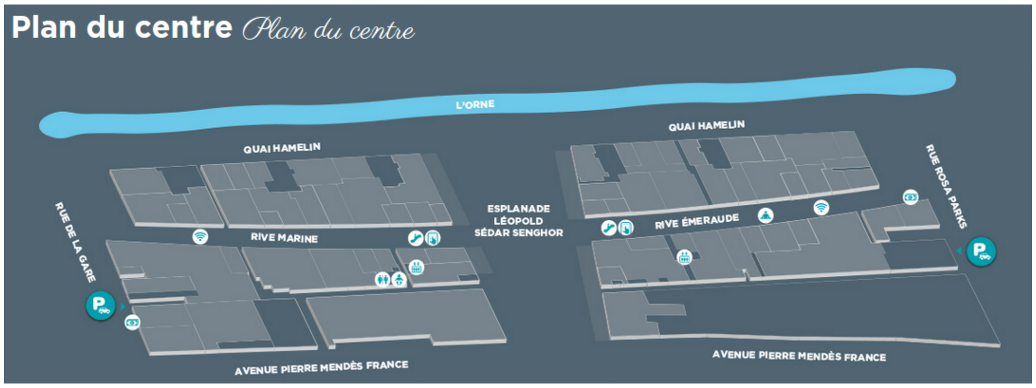


Figure 8.1: Map of Les Rives De L'Orne

The expected habitat for our social robot is a busy shopping mall, specifically that of Les Rives De L'Orne in the french city of Caen. We performed the simulation of all our use cases on a map of this location using Stage to verify the correctness of the plans and of the PNPs. Once we received positive feedback, we repeated the tests on a real life robot to better evaluate the system.

For convenience and safety concerns, we performed the real life experiments in the corridors of Sapienza University. Apart from the obvious differences between simulated and physical environments (sensors readings, unpredictability of the latter system) the setup was the same in both cases with ROS nodes serving as an abstraction layer between the actual robot and the rest of the control architecture.

8.2 Iterative Development of the Use Cases

When dealing with software projects of a certain complexity, it has been proven that an Iterative Development approach performs better than a Waterfall Development one[33]. In this spirit we developed our system in cycles; assessing early prototypes on the following use cases. This allowed us to gather important feedback and improve our system. Each use case was selected to highlight one core principle of this work.

The layered structure chosen in our architecture allowed us to model and ease in the system many key points which were not originally considered. We found that the PNP layer acted as a moldable bridge between the plan and the actions, allowing us to deal with unforseen action pitfalls without adding complexity to the plan or the actions.

To display this evolutionary process we will give two descriptions of the use case: one complete and detailed (what we ended up after many cycles), the other deliberately vague (what we started with). We will then present a **naive** behavior as originally conceived by the developer. Secondly, a plan produced by an **improved model** will be described; this plan arised from feedback received from the execution of the first one. This second plan fixes all the problems which we could modeled in a sensible way in the reasoner - unfortunately, certain issues would overcomplicate the planner if considered. To improve robustness, we consider a third case (**improved model with robust PNP actions**), using exactly the same second plan but this time utilizing a robust version, done in PNP, of the high level actions.

8.3 Use case 1: Customer asks for help carrying his\her bag

The objective of this scenario is to highlight the social awareness of the robot in the completion of a given task. There are many ways to carry a bag for a customer, but only a few might be socially acceptable at a given moment. To make sure, the robot asks to understand the priorities of the customer.

8.3.1 Description

8.3.1.1 Complete

The robot is called by a manager (or by the customer himself) to assist someone in carrying his\her bag. The robot must reach the exit of the shop and approach the dedicated loading area. When in position it looks for the customer and as soon as it establishes contact, it asks the user if he is willing to go to the parking zone. On positive confirmation, it asks the user to load the bag in the appropriate container. Once loaded, the robot will ask the customer if he is in a hurry. If the customer is in a hurry, the robot will proceed at a sustained speed to the parking lot. In the other case the robot will leisurely proceed to the parking lot, proposing intermediate stops to shops with interesting offers. In any case the robot will modulate its speed on the number of detected people, to minimize the risk of collision. If people are blocking the robot in a corridor, it should politely ask them to move. If the human stops following the robot, it should wait for him.

8.3.1.2 Rough description

The robot receives the task to help a customer carry his bag to the parking area. The robot navigates to the position of the customer, interacts with him and asks him to load the bag. It then proceeds to the parking area where it will ask the human to unload.

8.3.2 Naive behavior

A simple plan to accomplish the given task could be:

1. Move to where the customer is waiting
2. Ask the customer to load his bag
3. Ask the customer if he is ready to go
    (a) If he is not, wait for him to be ready
4. Proceed to the parking area
5. Ask the human to unload the bag
6. Return to *home* area

In theory, this sequence of high level actions should accomplish the assigned goal. Apparently we could be satisfied by this behavior, but after a few trials we considered that the robot was not acting in a very social way. He had no requirements to face the human when speaking (the human was likely behind the robot after it guided him to the parking) and did not ask the human's opinion on how to execute their shared action.

This kind of behavior is perfect for a factory robot, where the jobs must be done quickly, but fails to meet human expectations regarding social conduct. When interacting with people in a social way, we need to consider their needs and their wishes.

8.3.3 Improved model

Armed with this knowledge, we modified the reasoner to be mindful of the individual's desires:

1. Move to where the customer is waiting
2. Greet the customer
3. Ask the customer to load his bag
4. Ask the customer if he is ready to go
    (a) If he is not, wait for him to be ready
5. Ask the customer if he desires to follow the robot or set the pace himself
    (a) If he decided to set the pace himself, follow him (next step: 8)
6. Ask the customer if he is in a hurry
    (a) If he is not, the robot will select a longer path, stopping to illustrate special offers along the way
7. Proceed to the parking area
8. Face the human
9. Ask the customer to unload the bag
10. Say goodbye to customer
11. Return to *home* area

After a few tests with this new plan we were happy with the robot's behavior; everything was performed as would be expected by a social robot. This is, unless something went unexpectedly.

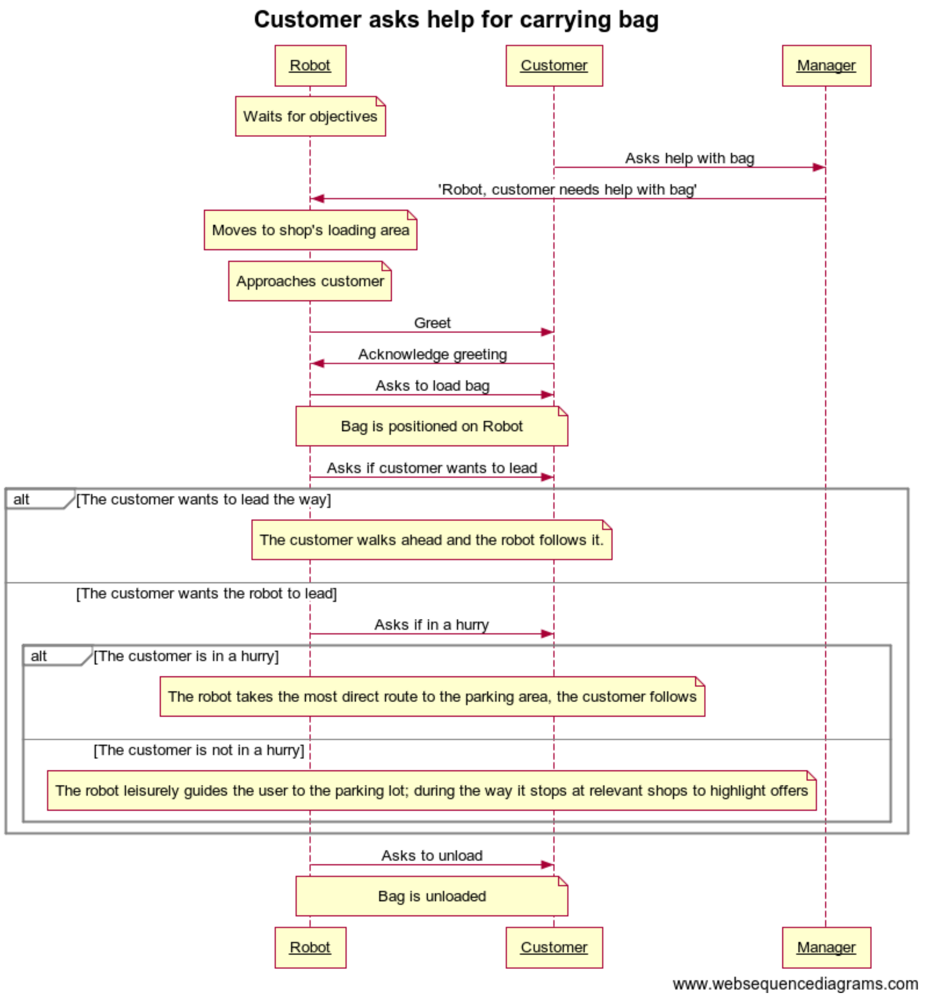

Customer asks help for carrying bag
Robot
Customer
Manager
Waits for objectives
Asks help with bag
'Robot, customer needs help with bag'
Moves to shop's loading area
Approaches customer
Greet
Acknowledge greeting
Asks to load bag
Bag is positioned on Robot
Asks if customer wants to lead
alt
[The customer wants to lead the way]
The customer walks ahead and the robot follows it.
[The customer wants the robot to lead]
Asks if in a hurry
alt
[The customer is in a hurry]
The robot takes the most direct route to the parking area, the customer follows
[The customer is not in a hurry]
The robot leisurely guides the user to the parking lot; during the way it stops at relevant shops to highlight offers
Asks to unload
Bag is unloaded
Robot
Customer
Manager
www.websequencediagrams.com

Figure 8.2: Sequence diagram of the bag carry use case

8.3.4 Improved model and robust PNP actions

In practice, we found there are many ways in which such a plan can go wrong. Here are two we chose to consider as being the most likely to happen:

- While following the robot, the customer encounters somebody he knows and stops. The robot should also stop and wait for him.
- While moving to destination, the robot is blocked by humans standing in the corridor. The robot should ask them to move and return a failure only if for some reason they refuse to do so.

How this was handled is detailed in sections 6.1 and 6.3. After adding these improvements the robot performed the task much more reliably.

8.4 Use case 2: Customer refuses first robot proposal, accepts specific one

This use case highlights a reasoning based recovery from a failure. The first offer is rejected by the customer; before proposing randomly a new offer, the robot tries to understand the user's preferences.

The proactive nature of the robot's behavior (when allowed by security concerns) is also underlined in this use case: when no people are moving around the robot can decide to approach itself a potential customer.

8.4.1 Description

8.4.1.1 Complete

Managers have told the robot that he needs to promote a special offer for a movie. The robot is also aware, in his persistent KB, of the existence of other secondary offers. Since there are not any people moving around the robot starts waiting and scanning. As soon as it sees a suitable user (at the right distance and the right speed), the robot intercepts the customer and asks him if he would be interested in the special offer for the movie. The user is not interested. The robot then asks the user to slide his customer card in order to propose the most appropriate available offer. After reading the card, the robot finds out that the customer is a woman and that she has made many purchases in the personal care department. It reasons that the special offer on the facial cream would be of interest to her. After being informed of the offer, the customer reveals to be interested and asks the robot for directions to reach the specific shop. Since not many people are moving around, the robot can safely guide the customer itself.

8.4.1.2 Rough description

The robot receives the task to propose an offer. As soon as a suitable customer is detected the robot approaches him and informs him of the offer. The human is not interested and the robot proposes something else.

8.4.2 Naive behavior

The naive plan for this task would be

1. Wait for a suitable customer to be detected
2. Approach the customer
3. Ask the customer if he is interested in the daily offer [customer REFUSES]
4. Randomly choose another offer and ask the customer [customer ACCEPTS]
5. Ask customer if directions are needed [customer ACCEPTS guidance offer]
6. Guide customer to destination

8.4.3 Improved model

The key issue with the naive plan is that the second offer the robot proposes is selected randomly, which has no guarantees of working. Knowing that many customers in the mall have a personal shopper card, it makes sense to ask them to ID themselves. Once recognized, the system can leverage their history of purchases to find out what might interest them. In this way the next proposal has a higher probability of being accepted by the customer.

1. Wait for a suitable customer to be detected
2. Approach the customer
3. Ask the customer if he is interested in the daily offer [customer REFUSES]
4. Ask the customer if he could ID himself to receive better suggestions [customer ACCEPTS]
5. Ask the customer if he is interested in specific offer tailored to his needs [customer ACCEPTS]
6. Ask customer if directions are needed [customer ACCEPTS guidance offer]
7. Inform customer that the robot will proceed to desired location
8. Guide customer to destination

8.4.4 Improved model and robust PNP actions

As in the previous case, to ensure a successful execution, we need to pinpoint possible failures and build strong PNPs to take care of them:

- The customer will follow the robot. All previous considerations must also be applied here.

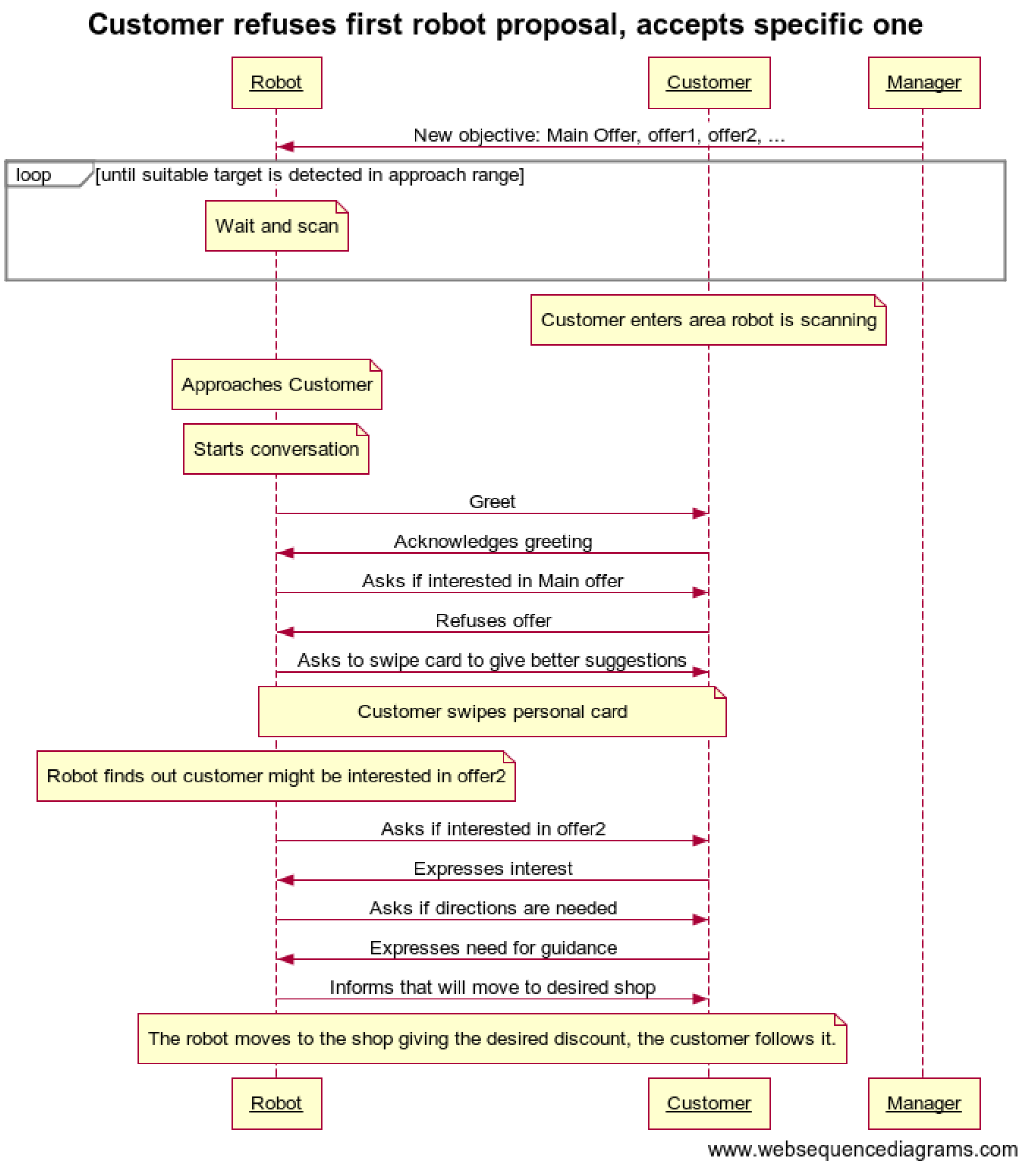

Customer refuses first robot proposal, accepts specific one
Robot
Customer
Manager
New objective: Main Offer, offer1, offer2, ...
loop
[until suitable target is detected in approach range]
Wait and scan
Customer enters area robot is scanning
Approaches Customer
Starts conversation
Greet
Acknowledges greeting
Asks if interested in Main offer
Refuses offer
Asks to swipe card to give better suggestions
Customer swipes personal card
Robot finds out customer might be interested in offer2
Asks if interested in offer2
Expresses interest
Asks if directions are needed
Expresses need for guidance
Informs that will move to desired shop
The robot moves to the shop giving the desired discount, the customer follows it.
Robot
Customer
Manager
www.websequencediagrams.com

Figure 8.3: Sequence diagram of the second use case

- Approaching the customer might prove tricky.

In the first use case the customer was waiting for the robot to arrive so we safely assumed that he would be ready to greet the robot (possibly even walking towards it). In this case, there is the chance that the human selected by the reasoner is unaware of the robot. What could happen is that the robot moves forward while the human, unaware, walks past him and out of the laser scanner (see figure 8.4). At that point, even if the human actually wanted to interact, the robot would be clueless of his existence.

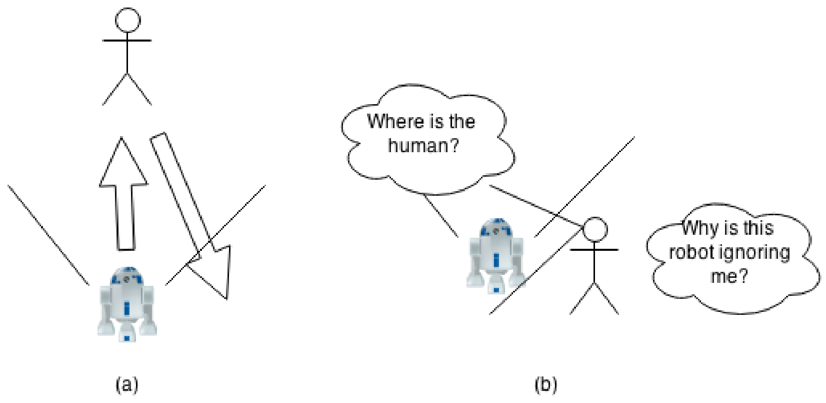


Figure 8.4: (a) robot moves towards human. (b) human walked past robot and out of his sensor range

To defuse this embarrassing situation we can model the following simple PNP (figure 8.5):

The robot should attempt approaching the human; if, for any reason, the human vanishes (that is, he was close and then abruptly disappeared) the robot will turn around to see if it can find him. If it does, good, continue to approach (if needed), otherwise report to the reasoner that the action failed because the customer is no longer there.

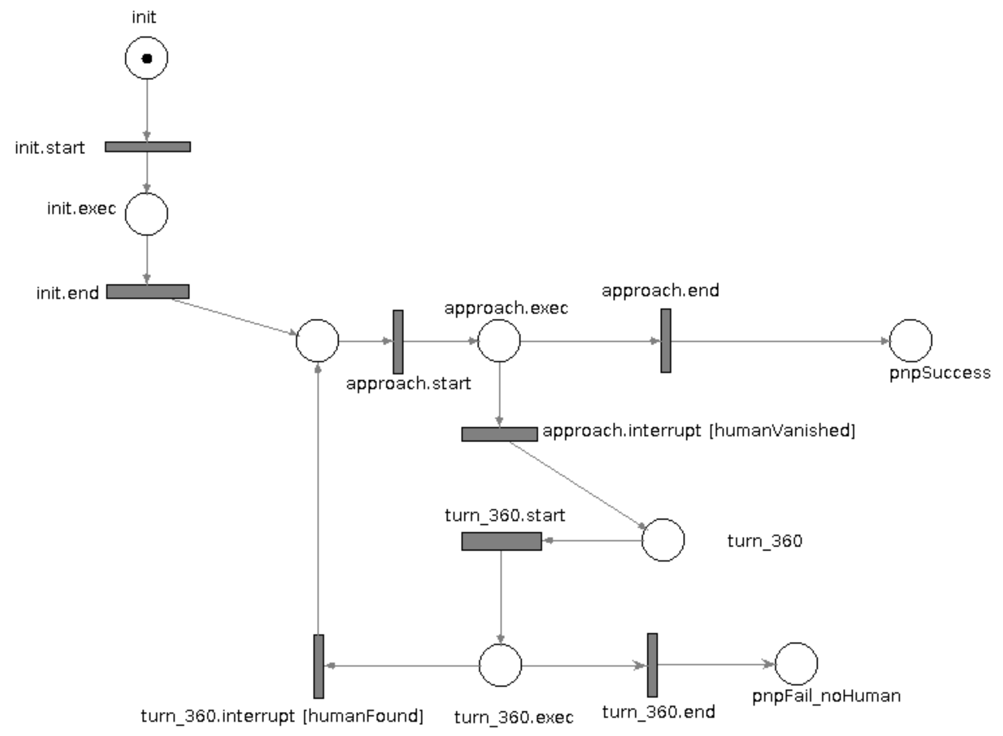


Figure 8.5: A PNP for increasing the robustness of the approach action

8.5 Use cases 3 and 4

The following use cases were considered to evaluate the reasoner's social plan generation capabilities; they were tested in a simulated environment but were not explored in detail for potential pitfalls as for the previous two examples.

8.5.1 Proposing and ice cream to a children

This use case demonstrates the common sense reasoning capabilities of the robot. It is not socially acceptable for the robot to guide a child to a location without its parents permission!

8.5.1.1 Description

Manager from local ice cream shop asks robot to inform children about a special offer. The robot looks around for children. When he sees one, he approaches him and informs him of the offer. The child is passionate about having an ice cream; but before directing him there, the robot asks for the parent's authorization. Once the parents agree to allow the child to follow the robot, the robot will proceed to the ice cream shop.

8.5.1.2 Generated Plan:

1. Wait for a suitable customer to be detected (a child in this case)
2. Approach child
3. Ask child if he is interested in the daily ice cream offer
4. Approach parents
5. Ask permission to guide child to ice cream shop
6. Guide child to destination

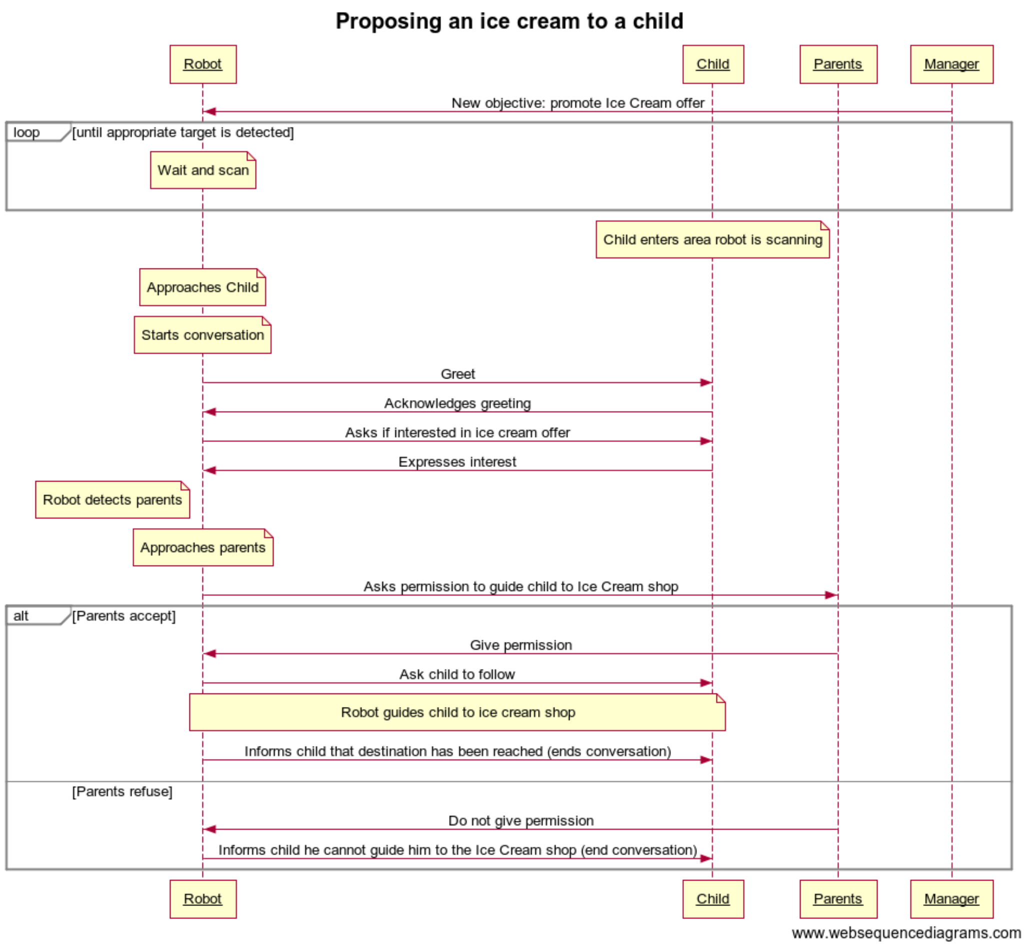

Proposing an ice cream to a child
Robot
Child
Parents
Manager
New objective: promote Ice Cream offer
loop
[until appropriate target is detected]
Wait and scan
Child enters area robot is scanning
Approaches Child
Starts conversation
Greet
Acknowledges greeting
Asks if interested in ice cream offer
Expresses interest
Robot detects parents
Approaches parents
Asks permission to guide child to Ice Cream shop
alt
[Parents accept]
Give permission
Ask child to follow
Robot guides child to ice cream shop
Informs child that destination has been reached (ends conversation)
[Parents refuse]
Do not give permission
Informs child he cannot guide him to the Ice Cream shop (end conversation)
Robot
Child
Parents
Manager
www.websequencediagrams.com

Figure 8.6: Option A: Parents are near the child

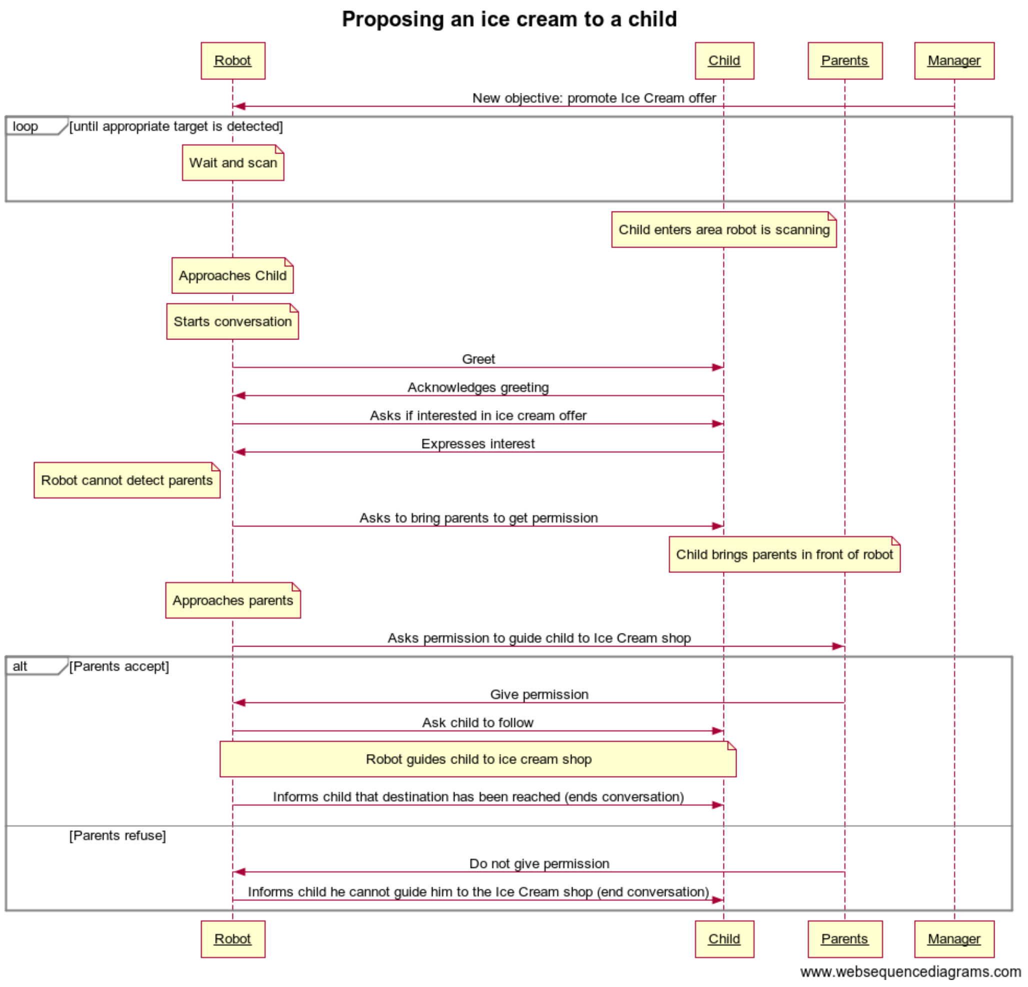

Proposing an ice cream to a child
Robot
Child
Parents
Manager
New objective: promote Ice Cream offer
loop
[until appropriate target is detected]
Wait and scan
Child enters area robot is scanning
Approaches Child
Starts conversation
Greet
Acknowledges greeting
Asks if interested in ice cream offer
Expresses interest
Robot cannot detect parents
Asks to bring parents to get permission
Child brings parents in front of robot
Approaches parents
Asks permission to guide child to Ice Cream shop
alt
[Parents accept]
Give permission
Ask child to follow
Robot guides child to ice cream shop
Informs child that destination has been reached (ends conversation)
[Parents refuse]
Do not give permission
Informs child he cannot guide him to the Ice Cream shop (end conversation)
www.websequencediagrams.com

Figure 8.7: Option B - Parents are not near the child

8.5.2 Customer asks for directions in crowded environment

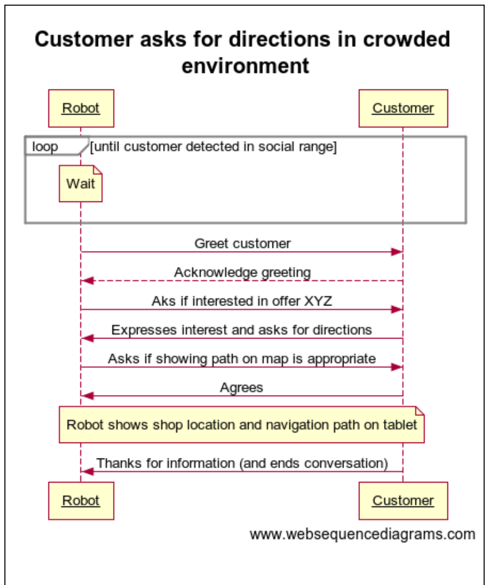
Scope of this use case is to remark that the robot must always be aware of safeness considerations; never must it behave hazardously in presence of humans.

8.5.2.1 Description

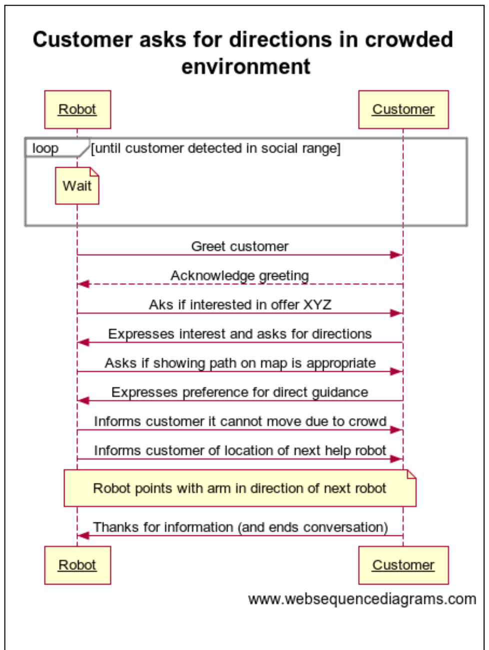
The robot has the objective to inform the customers of a certain few available offers. The shopping mall is very crowded and, for safety considerations, the robot reasons it should not move but rather wait for users to approach. As soon as it sees a user, standing still at the appropriate social distance, looking at it, the robot initiates conversation and proposes the available offer. The user is interested and asks for directions. Since the robot cannot safely move, it shows the desired path on his tablet and informs the user that 20m along the corridor he will find another robot for more specific information, if needed. The human thanks the robot and goes to the next one to receive further directions.

8.5.2.2 Generated Plan:

1. Greet customer [the customer approached the robot]
2. Ask customer if he might be interested in offer XYZ
3. Ask customer if directions are needed
4. Show customer how to reach desired destination


Customer asks for directions in crowded environment
Robot
Customer
loop [until customer detected in social range]
Wait
Greet customer
Acknowledge greeting
Aks if interested in offer XYZ
Expresses interest and asks for directions
Asks if showing path on map is appropriate
Agrees
Robot shows shop location and navigation path on tablet
Thanks for information (and ends conversation)
Robot
Customer
www.websequencediagrams.com

Robot shows path on map


Customer asks for directions in crowded environment
Robot
Customer
loop [until customer detected in social range]
Wait
Greet customer
Acknowledge greeting
Aks if interested in offer XYZ
Expresses interest and asks for directions
Asks if showing path on map is appropriate
Expresses preference for direct guidance
Informs customer it cannot move due to crowd
Informs customer of location of next help robot
Robot points with arm in direction of next robot
Thanks for information (and ends conversation)
Robot
Customer
www.websequencediagrams.com

Robot shows where next helper is

Figure 8.8: Customer asks for directions in crowded environment

Chapter 9

Conclusions

As time goes by and research advances, robots will more and more be part of our every day life. To do so, they will have to behave **reliably** in a **social** context, following our **ever evolving** social rules. Reliably because our world is incredibly complex and prone to challenge the robot in many different way. The robot must also act socially, since that is what people expect from each other; that humanoids behave in a certain manner is part of our common sense assumptions.

The system architecture proposed in this work was developed to satisfy these prerequisites. The high level ASP reasoner enables us to explicitly model the common sense assumption and social rules that the robot is expected to follow. This representation offers the maximum clarity of what are the core robot's laws and allows us to modify and add to them directly, even at run time.

Keeping in mind efficiency concerns and modelling complexities, we decided to set the granularity of our reasoner to a high level and to deal with the unpredictability and complexity of the real world using context and action specific PetriNetPlans. This choice permitted us to add action specific knowledge and reasoning to build more robust behaviors. Furthermore, the use of this formalism allowed to easily model why and how an action failed and give better feedback to our reasoner for the replanning phase.

As reliability was one of the key concerns in this work, the monitoring and recovery functionalities were spread between all layers. If our robot encounters a human obstacle while moving, it will first try to get around it (low level action controller) then ask him to move (PNP level) and finally find a different route (replanning); only if all three layers fail the robot will abort the goal.

Safety, other fundamental issue, was likewise treated redundantly: the reasoner will decide if to move, and how fast, depending on the number of detected people. In any case, the low level navigation, based on artificial potential fields will further mitigate any chances of collision.

Experimental evaluation on the considered use cases showed us that this system is indeed capable of generating a suitable robotic performance, able to meet our expectations on social behavior and reliability.

Overall, it appears that the proposed architecture succeeds in filling the gap between high level planning and low level action execution by using a formal language capable of connecting the semantics to the actions.

9.1 Future developments

All the social norms and common reasoning assumptions were modeled by hand in this work; while this was reasonable and allowed us to focus on other specifics of the system, we are aware that this is (beyond being time consuming) vulnerable to definition errors on the part of the human writing the rules. An interesting next step would be to automatically extract such information from an existing common sense database (as ConceptNet[14]). This has partly been done by Erdem et al.[13] and others and it would be thought-provoking to see how it would perform in a different domain.

On a related matter, all the special event rules are meant to be requested and defined by the shop and mall manager; since the encoding is done in ASP some kind of expert supervision is needed to transform their natural language requests in appropriate code. The development of an interface to automatically translate the manager's desires into syntactically valid rules would be an extremely useful addition to the system.

In our work, part of the action robustness was obtained by hand tuning of the PNP following the failure feedback. As this is a time consuming process, we propose to implement, in future, a mechanism to partly automate this process and help the human developer with some kind of improvement suggestion generated by the system itself.

As a long term development, it would be interesting to apply our architecture on different domains to test the modular structure of our system.

Likewise, we would like to understand if the proposed methodology can be effectively extended to use different formalisms for the three layers we divided the system in.

Bibliography

[1] Edward T. Hall and et al. Proxemics [and comments and replies]. *Current anthropology*, 1968.

[2] E. T. Hall. *The Hidden Dimension.* Anchor Books,, 1966.

[3] Martin Buss, Daniel Carton, Barbara Gonsior, Kolja Kuehnlenz, Christian Landsiedel, Nikos Mitsou, Roderick de Nijs, Jakub Zlotowski, Stefan Sosnowski, Ewald Strasser, Manfred Tscheligi, Astrid Weiss, and Dirk Wollherr. Towards proactive human-robot interaction in human environments. *Cognitive Infocommunications (CogInfoCom), 2011 2nd International Conference on*, 2011.

[4] Jens Kessler, Andrea Scheidig, and Horst-Michael Gross. Approaching a person in a socially acceptable manner using expanding random trees. *Proceedings of the 5th European Conference on Mobile Robots, ECMR 2011*, 2011.

[5] Marcus Finke, Kheng Lee Koay, and Kerstin Dautenhahn. Hey, i'm over here! how can a robot attract people's attention? *2005 IEEE International Workshop on Robots and Human Interactive Communication*, 2005.

[6] S. Satake, D. F. Glas T. Kanda, Michita Imai, Hiroshi Ishiguro, and Norihiro Hagita. How to approach humans? strategies for social robots to initiate interaction. *Human-Robot Interaction (HRI), 2009 4th ACM/IEEE International Conference on*, 2009.

[7] Patrick Sudowe Dennis Mitzel and Bastian Leibe. Real-time multi-person tracking with time-constrained detection. In *Proceedings of the British Machine Vision Conference*, 2011.

[8] Alexander Carballo, Akihisa Ohya, and Shin'ichi Yuta. Multiple people detection from a mobile robot using double layered laser range finders. In *ICRA Workshop*, 2009.

[9] Stefano Pellegrini, Andreas Ess, Marko Tanaskovic, and Luc Van Gool. Wrong turn-no dead end: a stochastic pedestrian motion model. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, 2010.

[10] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 2002.

[11] Marcello Balduccini, Micchael Gelfond, R. Watson, and M. Nogueira. The USA-advisor: A case study in answer set planning. In *Logic Programming and Nonmotonic Reasoning*, volume 2173 of *Lecture Notes in Computer Science*, pages 439–442. Springer Berlin Heidelberg, 2001.

[12] Jürgen Dix, Ugur Kuter, and Dana Nau. Planning in answer set programming using ordered task decomposition. *KI 2003: Advances in Artificial Intelligence*, 2003.

[13] Esra Erdem, Erdi Aker, and Volkan Patoglu. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intel Serv Robotics*, 2012.

[14] Hugo Liu and Push Singh. Conceptnet - a practical commonsense reasoning tool-kit. *BT technology journal*, 2004.

[15] Takayuki Kanda, Dylan F Glas, Masahiro Shiomi, Hiroshi Ishiguro, and Norihiro Hagita. Who will be the customer?: a social robot that anticipates people's behavior from their trajectories. In *Proceedings of the 10th international conference on Ubiquitous computing*, 2008.

[16] Takayuki Kanda, Masahiro Shiomi, Zenta Miyashita, Hiroshi Ishiguro, and Norihiro Hagita. An affective guide robot in a shopping mall. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, 2009.

[17] Gonzalo Ferrer and Alberto Sanfeliu. Bayesian human motion intentionality prediction in urban environments. *Pattern Recognition Letters*, 2014.

[18] Gonzalo Ferrer and Alberto Sanfeliu. Behavior estimation for a complete framework for human motion prediction in crowded environments. In *Ferrer, Gonzalo and Sanfeliu, Alberto*, 2014.

[19] Alberto Sanfeliu Cortés, Albert Punsola, Yuji Yoshimura, María Rosa Llácer, María Dolors Gramunt, et al. Analysis of the legal challenges required for the deployment of network robot systems in European urban areas. Institute of Electrical and Electronics Engineers, 2009.

[20] ISABEL FERREIRA and JOÃO SEQUEIRA. When children interact with robots: Ethics in the monarch project. In *17th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2014.

[21] Maria Braga and João Sequeira. Recognizing expressive motion by robots. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, 2014.

[22] Federico Pecora, Marcello Cirillo, Francesca Dell'Osa, Jonas Ullberg, and Alessandro Saffiotti. A constraint-based approach for proactive, context-aware human support. *Journal of Ambient Intelligence and Smart Environments*, 2012.

[23] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012.

[24] S Coradeschi, A Cesta, G Cortellessa, L Coraci, J Gonzalez, L Karlsson, F Furfari, A Loutfi, A Orlandini, F Palumbo, et al. Giraffplus: Combining social interaction and long term monitoring for promoting independent living. In *Human System Interaction (HSI), 2013 The 6th International Conference on*, 2013.

[25] Susanne Frennert, Britt Östlund, and Håkan Eftring. Would granny let an assistive robot into her home? In *Social Robotics*. Springer, 2012.

[26] Vladimir Lifschitz. What is answer set programming? *Association for the Advancement of Artificial Intelligence - aaai.org*, 2008.

[27] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 2012.

[28] Vittorio Amos Ziparo and Luca Iocchi. Petri net plans. *In Proc. of Fourth International Workshop on Modelling of Objects, Components, and Agents*, 2006.

[29] V.A. Ziparo, Iocchi, Luca Nardi, Pedro Lima, and P. Palama. Petri net plans - a framework for collaboration and coordination in multi-robot systems. *Autonomous Agents and Multi-Agent Systems*, 23(3):344–383, 2011.

[30] V. L. Iocchi D. Nardi P. Palamara Ziparo and H. Costelha. Pnp: A formal model for representation and execution of multi-robot plans. *In Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems*, 2008.

[31] V. Ziparo, L. Iocchi, and D.Nardi. Petri net plans: A formal model for representation and execution of plans. http://www.dis.uniroma1.it/ iocchi/Teaching/rp/PNPROS/2-PNP-multi.pdf.

[32] Luca Iocchi. Diago sapienza social robot. https://sites.google.com/a/dis.uniroma1.it/diago/home, 2014.

[33] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices.* Prentice Hall, 2002.

[34] G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 2011.

List of Figures