Linking an HTN Planner to a Universal Robot Controller for High-Level Activity Control

Dane Alan Alexander

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2006

Abstract

In this thesis we investigated the possibilities for developing a generic high-level robot controller, using an architecture consisting of a HTN planner (provided by the I-X framework) generating and sending actions to a robot via a generic controller (provided by URBI) in order to perform some complex task that could not be executed by the robot directly. The complex task that was chosen was to have our robot navigate from one zone to another in a generic office environment, with zones representing rooms and open spaces.

In order to implement a generic controller we first focused on identifying the robot's capabilities as represented by the commands that URBI provides. We modelled these commands in such a way that I-X would be able to reason about and create plans from them. This resulted in a library of primitive I-X actions implemented in the I-X LISP notation such that there existed a one to one mapping to URBI commands. These actions then provide a foundation for building higher-level methods & tasks. The communication and monitoring of commands is handled by the robot controller that was designed and implemented in this project. It was also the job of this controller to update I-X's knowledge of the state of the world.

This system gave us the opportunity to evaluate the approach, by doing some limited experiments involving the generation of plans in I-X and executing them on a particular simulated robot in environments that were tailored specifically for a complex task.

Our first hypothesis was that I-X and URBI together can provide a generic high-level controller for robots. However, from the results of the experiments and general considerations of I-X and URBI as components in this architecture, the system built during this project provided evidence only for a weaker hypothesis, namely that I-X and URBI together provide a high-level controller for a **specific** robot in a **specific** environment.

Acknowledgements

I would first like to express my sincere gratitude to my supervisors Dr. Gerhard Wickler and Dr. Stephen Potter for their time, patience and guidance throughout the course of this thesis. Without their help and expertise, this thesis would have not come as far as it has today.

I would also like to thank my family especially my parents and my uncles for providing me this opportunity to accomplish my dreams.

My deepest gratitude to the love of my life, Lisa, for supporting me in everything that I do. Without your moral support and your “You Can Do It!” attitude I would not have come this far.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dane Alan Alexander*)

Table of Contents

Chapter 1

Introduction

For many years in the field of robot control, research has focused mainly on the low level capabilities of the robot, such as basic path finding, obstacle avoidance and different visual techniques for identification etc. Only recently have scientists begun to focus on robots that will be able to reason about their environment at a higher cognitive level and autonomously perform complex tasks by more sophisticated means. One area in this type of research involves planning. A planning system that would aid in the decomposition of high-level commands into primitive actions and transmitting those actions to a robot via some generic controller would lead to many potential applications as outlined below. When we say "high-level task" we refer to giving the robot a set of commands one might give another person e.g. telling someone to go and retrieve some documents from the manager's office. Such commands would normally not correspond to actions a robot can perform directly, i.e. without decomposition. Such a system would be a large step forward from existing methods.

1.1 Motivation: Possible Applications

Security and surveillance is an area that can be explored. For example, if you are abroad on a vacation and your robot is at home permanently on standby and communicating via WIFI to a dedicated server, you can log onto the server with the intention of checking the surveillance plans generated by a sophisticated planner which are being sent to your robot. After all tasks have been achieved, the robot can send back the results of the tasks.

Such a system also has the potential for controlling more than one robot, again due to the fact that the planner is not necessarily on-board the robot. This makes it easier

to centralise information coming from each robot so the planner has a good idea about what coordinating plans it should produce in order to accomplish some task. This idea could be usefully implemented on robots that search for mines for example. The more robots are available, the more coverage of an area can be achieved. Such a system could be employed on a multi-agent level with a centralised planner.

Another potential application involves emergency response robotic units, deployed when earthquakes strike, for example. These robots would have the ability to go to areas where no humans can safely venture and search for survivors. However, in this scenario the robots themselves would need to have on-board planning capabilities just in case something disrupts their communication. They can save the plans that were sent by the central task-setting controller and still carry out their mission. Upon identifying a victim a unit would attempt to get in contact with the central controller to report the problem. If this fails then it could try to find an area where communication can be regained.

1.2 Hypothesis

The aim of this project is to attempt to create a generic, high-level robot controller. By "generic high-level controller" we mean:

- It will be generic in the sense that the primitive actions are not specific to one type of robot acting in one type of environment.
- Control of the robot will be at a high-level in the sense that tasks can be set at a level more abstract than the primitive actions provided by the robot.

A generic controller for a robot is already provided by URBI [7]. The URBI framework provides a fairly low level of control and access to a robot's joints, sensors, etc which can be manipulated by sending URBI commands. The I-X framework [32] provides a way of modelling and reasoning with abstract activities. Higher-level tasks can be invoked from an I-X Process Panel, which are then decomposed into primitives by the I-X HTN planner, transformed into URBI commands, and sent to the actual robot. (The robot that was to be used for this project is the Sony AIBO ERS-7M3 [4].)

Our hypothesis is therefore that I-X and URBI together can provide a high-level generic controller for robots.

1.3 Thesis Structure

The structure of this thesis is as follows. In chapter 2 we go over the important background material that is necessary to give a clear understanding of the rest of this thesis. In chapter 3 we describe and discuss the design and implementation of the architecture that was created to control a generic robot on a higher level. In chapter 4 we discuss the experiments that were used to evaluate the system as a whole and provide an analysis of the results obtained concluding with a general discussion of our findings and how far they support our hypothesis. In chapter 5 we go over the work that was done, highlighting the main design decisions and concluding with a discussion about the extent to which the developed system supports our research hypothesis, i.e. whether our controller can be considered generic and high-level.

Chapter 2

Background

In this chapter we discuss related work that has been done in generic controllers and with planning in robotics. Then we give an overview of some of the components necessary for achieving the goals of this project. We highlight the important aspects of the project by doing a review of known HTN systems. After which we describe other important parts of the project such as an overview of HTN planning and its application in robotics. We discuss what is URBI and what is I-X and its important parts. We describe the robot that was used and why it was used.

2.1 Hierarchical Task Network (HTN) Planning Overview

This section gives a brief conceptual overview of HTN planning. We also discuss the advantages that HTN planning has over other classical planning techniques.

Hierarchical Task Network (HTN) planning is an alternative to classical planning [16] which uses "task decomposition" for providing plans in solving complex problems. We use the characterisation of tasks from [12] which is "activities we need to plan i.e. things that need to be accomplished". A task network is a set of tasks interconnected through the constraints within them. A hierarchical task network is a set of task networks whose initial or root task describes the problem, which is viewed as a very high-level description of what is to be done, for example "building a house" [16].

Planning in the HTN uses task decomposition which is the refining of higher-level tasks into an ordered set of lower level tasks. The decomposition of higher-level tasks continues until only primitive actions remain.

Figure 2.1 gives an example of an actual HTN method. The task "Build (?house)" describes the overall problem. This task can be decomposed into a set of other lower

```
Method Build (?house)
        Precondition: (and (own land) (have money))
        Effects: (built ?house)
        Applicability: (single-family-home ?house)
        Expansion:
                    S1: Build-Foundation(?house)
                    S2: Build-Frame(?house)
                    S3: Build-Roof(?house)
                    S4: Build-Walls(?house)
                    S5: Build-Interior(?house)
                    S6: Decorate(?house)
        Orderings: S1<S2, S2<S3,S2<S4, S3<S5, S5<S6
        Links:
                    S1 causes (foundations laid) for S2
                    S2 causes (frame erected for S3 and S4
                    S3 causes (roof built) for S5
                    S4 causes (walls built for S5)
                    S5 causes (interior done) for S6
```

Figure 2.1: Example HTN plan for building a house(from [31])

level tasks as shown by the “Expansion” in the figure 2.1. In this example the tasks have an ordering which needs to be fulfilled.

Hierarchical Task Network (HTN) were originally designed in order to close the gap between other planning techniques such as Strips-style planning, and operations-research techniques for project management and scheduling [30]. One important difference between HTN and classical style planning is the way in which a planning problem is defined. HTN planners attempt to perform a set of tasks, whilst classical style planning tries to reach to some “goal state” by performing a sequence of actions. In classical planning a search algorithm is responsible for finding the best actions to perform in order to reach some goal state. However, if the set of goal conditions is very large these searching algorithms take a very long time in coming up with solutions. HTN has the advantage over classical planning in the sense that for it to reach a goal state there is a predefined set of methods that achieve this on some hierarchical level.

Some applications of HTN planning can be found in gaming, multiagent team behaviour [24], robot navigation [9], motion planning and search and rescue tactics etc. Nonlin which was created by Tate in 1976 [29] is one of the first AI planners to use the HTN planning architecture. It was also famous for its early role as a basis for NASA’s Jet Propulsion Laboratory Deviser [35] which was their first planner used in spacecraft mission sequencing. This illustrates the great practical relevance of HTN planning.

2.2 HTN Planning in Robotics

This section reviews some robotic systems that implement the HTN Control architecture. We then conclude with a discussion on how the goals of this project differ in comparison with these robotic systems.

In [21] Mastrogiovanni et al. expand the HTN architecture claiming that the current state of HTN planning does not handle re-planning well for service type robots. These service type robots are required to work in dynamic environments where static plans have the potential to fail. This is because in a dynamic environment (for example in a busy shopping centre), new variables such as changes to the environment can be introduced which may have not been considered at the time of plan generation for some service type activity. They claim that HTN planning can lead to sub-optimal plans by not focusing on the long term plan in highly dynamic environments. To fix this problem they created a more simple yet effective method of calling the HTN al-

gorithm to run on sub-tasks where applicable. Their system which comprises of two interacting subsystems are made up of a 1) Knowledge representation system and 2) an HTN planner (see [21] for details of these systems) which are handled by agents using a multi-agent system. Basically, these two systems work together which constantly update and repair plans. However, their system was only applied to one service type robot. Also, the environments that they used for testing were static and changes to the environment were directly made by the robot instead through other external sources. However, the experiments did show that compared to non-HTN planning techniques HTN planning performed faster in finding solutions for complex activities.

For the AAAI Mobile Robot challenge 2005, researches at the Université de Sherbrooke proposed their own built U2S robot [5] with on-board HTN planner. The main goal for the U2S robot was to navigate through the conference, register itself, have a basic conversation with others and aid in human driven tasks [8]. The HTN planner that was used was an improvement on the SHOP2 algorithm [23] with added features such as time constraints and added post-processed planning [11] for improved flexibility which was crucial in plan repair. The planning utility was only specially designed for the purpose of the conference and for the robot. Integrating such a system into other environments is still under investigation.

These robotic systems apart from their HTN planning capability have a some things in common which are:

- Planners are situated on-board the robots.
- Robots were designed for specific environments.
- Planning architecture was quite specific for the type of robots.

2.3 I-X

This section describes a brief overview of the I-X architecture highlighting the I-X tools that were used in the design of our architecture.

According to [3] “I-X provides a new systems integration architecture which can be used to create agents or non-agent systems whose design is based on the O-Plan [31] agent architecture”. Within I-X the tools that we used are the I-X HTN planner I-Plan, I-X Process Panels and the ability to define our own I-X activity handlers, whose defintions are discussed more in detail below.

2.3.1 I-X Process Panels & I-Plan

I-X process panels are one of many tools used in the I-X architecture [32]. According to [32] ”I-X Process Panels are used to support users who are carrying out processes and responding to events in a cooperative working environment”. I-X Process Panels also are the principal interface by which users access I-X’s tools. These panels are flexible in that they can be made to communicate with other panels and a wide variety of other services. I-Plan [32] is one of the core tools available in I-X and which provides HTN planning for task decomposition. I-X has been applied to:

- Search & Rescue Coordination
- Help Desk Support
- Unmanned Autonomous Vehicle Command
- Responding to Simulated Oil Spill Emergency

The I-X Process Panels provide an interface to I-Plan; therefore, to exploit the I-X planning facilities in this project, the task becomes one of communicating the appropriate commands between I-X panels and the robot. This will form the basis/foundation of our generic robotic controller.

2.3.2 I-X Activity Handlers

An activity handler in terms of the I-X architecture and this project can be seen as a dispatcher. This dispatcher assigns activities to different agents and these agents handle the activities which report to the I-X-Process Panel whether or not the activities were carried out successfully. The agents also have the option of sending back extra information to the I-X Process Panel about the completion or failure of that activity.

2.3.3 I-X <I-N-C-A> Ontology

<I-N-C-A> stands for *Issues*,*Nodes*, *Constraints* and *Annotations* which is the ontology used in the I-X architecture that describes the product of a synthesis task [32]. The synthesis task pertaining to this project can be seen as synthesizing a course of action. However, this project focuses only on Nodes and Constraints because there is no utility for Issues and Annotations at the moment.

- Nodes: This high-level task that needs to be decomposed into primitive actions or the primitive actions themselves that are used in instructing the robot.
- Constraints: This represents the I-X agent's knowledge of the world[1].

2.4 Generic Robotic Controllers

In this section we discuss the concept for generic controllers and then later on describe the Universal Robotic Body Interface (URBI).

The area of generic robotic controllers is considered a relatively new field in robotics. For years robotics scientists have depended on control architectures very specific to the type of robot that they were doing research on. However maintaining different control architectures for unique robots became an extremely challenging task. Therefore having one control architecture for a multitude of different robots became an attractive area of research.

When we think about a generic controller we think about controllers that are capable of controlling any type of robot, or one that would provide an interface that would be able to easily program control in these robots. One such robotic generic controller is the Universal Robotic Body Interface (URBI) [7], the brain child of Dr. Jean-Christophe Baille. URBI provides us with the functionality and the interface to help achieve the main objectives of this project.

2.4.1 URBI

URBI is a piece of ongoing research which attempts to make a standard for controlling robots of any type. The number of robots (be they for entertainment or commercial purposes) that are developed each month is rising. The control architectures for such robots are unique and complex. For a robotics scientists it is hard trying to keep up with such developments. URBI is trying to address this problem.

URBI was developed with simplicity in mind which addresses the complexity problem faced by other control architectures. Compared to other generic controllers such as Tekkotsu [33] and Player/Stage [34] [7] raise the argument that URBI is a lot easier to use. This is an attractive quality to URBI as this would help decrease development time in robotic programming and control.

[1] Information about the world is sent by the robot for this project.

For this project URBI provides us with the three main things that are crucial and these are:

1. A scripting language for programming robotic control.
2. A TCP/IP client/server control architecture so we can communicate actions to our robot via some medium that supports TCP/IP.
3. An URBI library called libUrbi so we can extend the capabilities of our robot.

URBI also provides libraries collectively called "libUrbi" for different languages such as Java, Matlab and C++. Here Java is used as the main programming language to extend the robots capabilities. This is chosen because the I-X architecture is also written in Java[2], therefore this eases the problem of having to switch between different languages.

URBI is also based on a client/server side type architecture. The server side handles the low level control of the robot such as the robots joints and sensors while the client side is used to control the robot from a higher-level, that is, the sending of higher-level actions such as a walk method etc. "Devices" are essentially what make up the robot [7]. These devices in terms of URBI is anything that can be controlled, such as the robot's motors and sensors. At a higher-level we can control these devices by using the URBI scripting language that URBI provides. The client/server architecture allows for communication of commands in two ways:

1. The server can be accessed remotely for commands to be sent on port 54000.
2. The server can be activated on the robot thereby allowing an IPC style of communication between the server and the client.

This form of remote-operation is advantageous to us which is the reason why URBI is critical to the project. It provides a communication channel to the robot through a medium which is already available to us. With other generic controllers such as Tekkotsu [33] it is more complex to implement this form of communication.

So far, URBI has been tested on fairly small robots such as the Sony AIBO ERS-7 [4], HRP-2 Humanoid [20] and also in game controllers. URBI has not been implemented in any industrial scale robots. This may be due to the fact that these robots cost

[2]The current version of the Java libUrbi is 0.9.1 which has not yet caught up with the functionality of its C++ counterpart (1.0 beta). However, this version of the Java libUrbi still possesses the main functionality that would help us achieve our objectives.

millions of dollars to run, and since URBI is still relatively new, it still has a long way to go before it can be operated on this level where reliability, efficiency and safety is of major concern.

We now give a brief overview of the general URBI/Client server architecture and "URBI Tags", which, when used together provide the foundations of the robot controller that will be described in the next chapter.

2.4.2 URBI Server/Client Architecture and URBI Tags

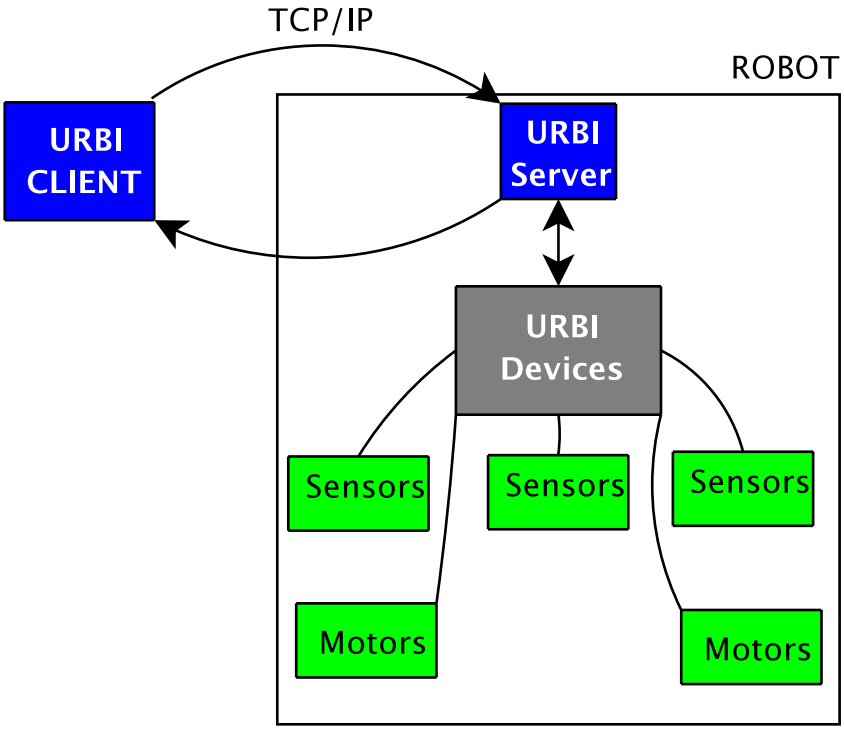

Figure 2.2: The URBI Client/Server Architecture

As you can see from figure 2.2 an URBI server is situated on the robot itself. The URBI server is in charge of all the sensors and motors which have been translated into the form of URBI devices. Due to the client/server architecture it is possible for a client to log onto the URBI server by specifying an IP address and the port number of the URBI server. Upon connecting to an URBI server it then becomes possible to send to the robot URBI commands.

There are different ways of structuring an action that is to be sent by the client. URBI provides the option of tagging URBI commands. An URBI tag is a user-defined word which is attached to an URBI action or a set of actions that are to be monitored

by the URBI server. When a user sends an untagged URBI action to the server, that action is executed by the server but the server does not respond to the user with the result of that action, i.e. whether it started or finished. With tagging it is possible to receive such information. Tagging also comes with parameters that allow the client to get different forms of reports. These options are:

- *begin*: The server alerts the client that the tagged commands have started.
- *report*: The server alerts the client that the tagged commands have not only started but also when they finish.
- *end*: The server alerts the client that the tagged commands have finished.

The structure of a tag follows the form:

```
<URBI TAG>+<Reporting Options>:[URBI_Commands]
```

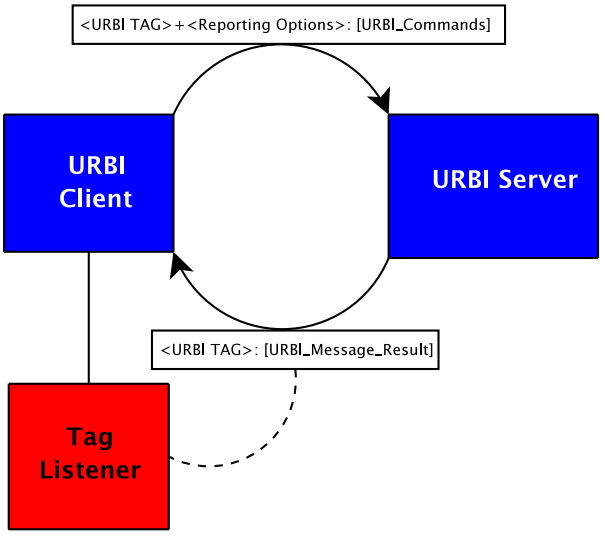

Figure 2.3: Using the Listener with URBI Tags. The dotted lines shows and indirect relationship between the tag and a registered URBI listener. The solid line indicates that the Listener is part of the URBI Client while the arrows represent the flow of data between the client and the server.

URBI tags are important because they provide us with a way of using event-style programming. Depending on the tag and the type of tagging option that we choose, when we send tagged actions it is possible for the client to then see the result of that tagged action. This concept is expanded further by doing something about tags that

are reported back to the client. URBI provides a package “libUrbi” which essentially provides the interface for creating listeners for a tag. A listener in this sense is a type of interface which looks out for certain tags that appear on the client. Within these listeners we can embed our own code which can react appropriately when encountering a particular tag. Figure 2.3 shows this use of tags.

2.5 Robotic Simulators and WEBOTS

In this section we discuss robotic simulators in general and discuss why these robotic simulators came about. We discuss the WEBOTS robotic simulator environment especially why it was used for this project. We also highlight the advantages and disadvantages to using WEBOTS as a robotic simulator.

Robotic simulators are used by robotics scientist to either test a robot in an environment which is not readily available or to test some important component of a robot whose physical parts are hard or expensive to develop. Therefore, simulators are a good research tool for trying out different things without using up the physical resources of the real world. Before robotic simulators robotics scientists would have spent long hours trying to test an important component of a system and or recording the results of how that component acted in a real environment [3]. Not only would they want to test the components themselves but also the intelligent controllers that operated these components. To fix the problem of wasting development time and money they started to develop simulators for testing out virtual components which mimicked the functionality of the real robots and ran their experiments in these virtual worlds. However, it is important to state that simulations can never replace the real testing and only gave an approximate level of performance when testing out these robots.

The first robotic simulators back in the early 1980s had the look and feel of a two-dimensional world. With the advent of better graphics and faster computers during the 1990s up until the 21st century, robotics scientists took the opportunity to expand their simplistic two-dimensional world and made it into a full blown three-dimensional world equipped with better physics [4]. This gave scientists the opportunity to develop robotic simulators which were crisp and clear in graphics and gave the user a more realistic experience when working with these virtual worlds. Some simulators even

[3] Not to mention actually getting the materials to make the components and then adding them to the robot.

[4] The extra dimension gave the scientists the opportunity to expand the physical environments and hence the physics of that environment.

have the important task of aiding before and after medical surgery such as [19] and [10].

2.5.1 WEBOTS

WEBOTS is the result of an ambitious project by Cyberbotics Ltd. At first their goal was to expand the Kephera Simulation environment [22] but then they came to realise that the architecture that they were using was almost good enough to be expand past their original objectives and designed WEBOTS to provide functionality for building and controlling components and even for designing environments for the robot.

With development time in mind the scientists at Cyberbotics also made it easy for transporting code from the virtual robot to the real robot [5]. When developing environments provided is a Virtual Reality Modelling Language (VRML 2.0) that comes with the package. This is a useful feature as one can use any other three-dimensional modelling tool and export VRML component to the WEBOTS environment. This gives one the ability to design complex environments in other programs and exporting them into WEBOTS.

The WEBOTS simulation environment is used in this project primarily because this is the only robotic simulation environment that access was provided to. After some considerable use of the program the only disadvantage that was found in WEBOTS was that it was a bit buggy and even hindered the performance of the robots as well. However the advantage to this system was that it was fairly easy to manipulate (for example the ‘environment’) making changes on the fly.

2.6 Sony AIBO ERS-7M3

In this section we introduce the Sony AIBO ERS-7M3 (as seen in figure 2.4, the robot that was to be used for this project. We give a technical specification of the AIBO robot and what features URBI provides for the AIBO.

The Sony AIBO ERS-7M3 is an entertainment pet-style robot which is part of a generation of robots created by Sony whose robotic origins started in the late 1990s [14]. The reason for creating these robots was basically to explore a new form of robot. Robots until then were primarily for conventional purposes and had to be reliable with zero tolerance for error. These entertainment style robots did not have to have zero-

[5] This naturally would decrease the development time in production.

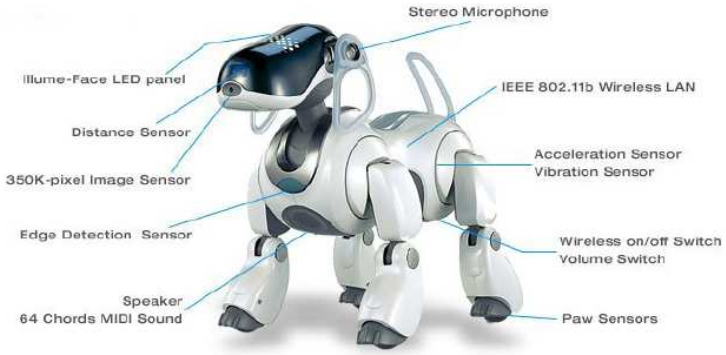

Figure 2.4: AIBO ERS-7M3 (Picture from [14])

tolerance and did not need to be efficient 100% of the time, instead, they needed to display human-like action, so mistakes by the robot were allowed to happen . However, this does not mean that the AIBO lacks sophistication, as a matter of fact the AIBO robot is at that level which allows it to be as human-like as possible, which makes it a very complicated system.

The AIBO with its special Sony software on-board is capable of:

- Recognising Faces
- Listening for and understanding certain pre-built commands
- Learning new commands.
- Performing complex movements such as dancing
- Evolving into “mature” stages depending on the level of interactivity that humans have with the AIBO.
- Recognising its “toys” such as its bone and its pink ball.

This list is nowhere near exhaustive.

2.6.1 AIBO in WEBOTS

A picture of the AIBO in WEBOTS is given in fig 2.5.

There were a couple of difference between the AIBO in the real world and the AIBO in the WEBOTS simulation environment. The AIBO’s battery power in the real world depleted after extensive use of the robot. In the simulation world this was not

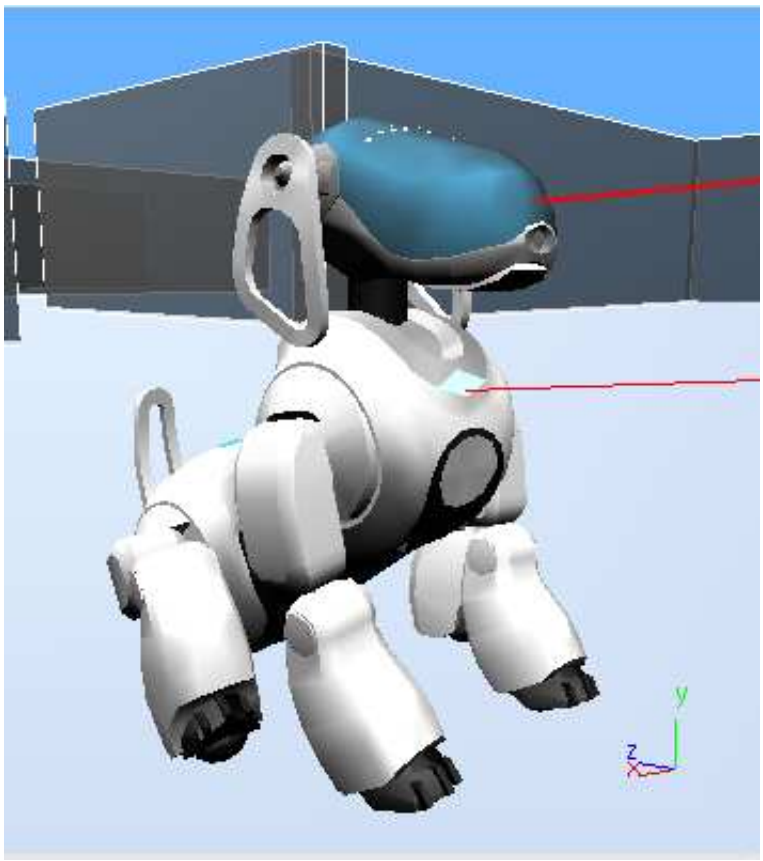

Figure 2.5: AIBO ERS-7M3

much of a problem as the battery power was always 100 %. The other difference is due to the fact that the AIBO robot in the real world used WIFI communication while the AIBO simulator was on the same localhost and was able to receive commands. However, these were very small differences and would not hinder the development aspect of the project.

2.6.2 AIBO and URBI

As previously discussed in subsection 2.4.1 URBI was implemented for the AIBO. With URBI as its robotic controller the AIBO lacks the autonomous intelligence and control sophistication provided by the on-board Sony program. The AIBO is now a drone which cannot really do anything complex besides basic movement as provided by the URBI system. This therefore limits the AIBO's capabilities. The AIBO can no longer use its sophisticated face recognition and it cannot hear and recognise voices. URBI AIBO only has the ability to recognise pink objects, and perform basic movement! However with libUrbi 0.9.1 and the URBI specification language it is possible to give the robot basic perceptions, such as the ability to recognise other objects.

URBI also provides a walk device which handles the movement for the AIBO. The AIBO, when given certain URBI commands, can therefore move left, right, and walk forwards or backwards. It is also possible to manipulate other joint devices through the URBI scripting language such as getting a picture from the camera [6].

2.7 Chapter Summary

This chapter looked at the HTN Planning Architecture and its involvement in other systems. We also looked at the different important components of the project such as URBI, I-X and WEBOTS and discussed its strengths and its weaknesses. The next chapter focuses on the design and implementation of the project including more in depth detail on how the previously mentioned topics came together.

[6] Full colour pictures.

Chapter 3

Design and Implementation

The motivation behind the design and implementation was to develop a working model of a generic high-level controller which provides generic HTN planning capability. This was carried out based on an understanding of the I-X architecture, with its underlying <I-N-C-A> ontology and the URBI framework, as was described in chapter 2. What will follow in this chapter is a description of the major design and implementation decisions that took place throughout the design and implementation phase.

In section 3.1 we introduce the design of the overall system and discuss in brief the major components and the communication between them. In section 3.2 we discuss the process of modelling the predicates, primitive actions and higher-level actions. In section 3.3 we discuss the job of our activity handler. In section 3.4 we go through in detail the design of the generic controller and its vital functions. In section 3.5 we discuss the simulation environment used for testing the planning capabilities of our system executed by our robot. And lastly in section 3.6 we discuss in detail the sensing capabilities of the robot within its environment and the methods we used to enhance the robot's capabilities. As a result of this work we will have the working model of a generic high-level controller that allows us to conduct our experiments and evaluate our hypothesis.

3.1 The Project Architecture

We describe in brief the overall view of the design architecture, discussing in summary its major components and their relationship with each other.

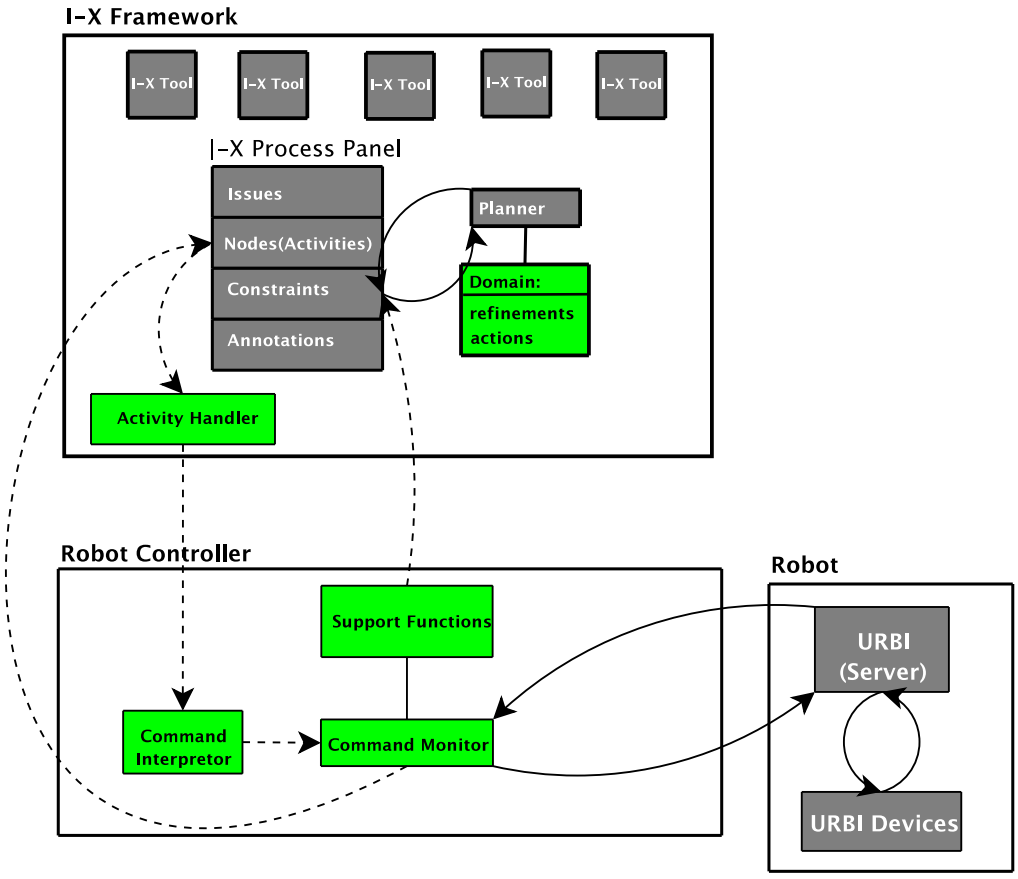

Figure 3.1: The Architectural Design of the project.

From figure 3.1 the green boxes signify the components that were created for this project while the grey boxes represent those components that were already developed. The arrows represent the flow of data between the components and they are of two types. For each activity that is sent from the panel to the controller the dotted line arrows represent one and only one occurrence of data flow between the components while the solid line has more than one occurrence in the direction of the arrow between components.

It is important to have an understanding of this architecture as the different component parts are discussed in detail throughout the rest of this chapter. This architecture is shown in figure 3.1. Here is a brief overview of these components.

We use the I-X Process Panel as an agent which essentially keeps information about the world in the form of a set of world-state constraints. We set an activity by adding

it as a node onto the I-X Process Panel. The planner (which continually monitors changes to the panel) compares the activity against the available refinements defined in the domain. If a matching refinement is found, and its conditions are met by the current world state constraints, then an appropriate activity handler is made available to the user. If the refinement corresponds to a high-level action (see section 3.2.4) then the activity is handled by decomposition into lower level activities (which are then added to the panel). If the refinement corresponds to a primitive action (see section 3.2.2), then the activity is handled by sending it to the "Command Interpretor". The "Command Interpretor" decodes the I-X activity into an URBI formatted command which is then sent to the "Command Monitor". The "Command Monitor" will set the status of the corresponding I-X activity to "executing" and then send the command to the URBI server. The URBI server will then execute that command and respond to the "Command Monitor" that it has finished executing. The "Command Monitor" will then update the status of that I-X command reporting the result of the execution [1]. Depending on the command that is being sent, the URBI server will invoke certain support functions that also may change or update the world constraints within the I-X Process Panel.

3.2 Planning Domain

This section details the planning domain aspect of our project and in particular, the constraint types used to describe our world states, and the refinements of our high-level actions.

Definitions for the terms used here are:

- Constraint Type: A predicate which describes the relationship between its arguments[16].
- Primitive Actions: Actions which need not be decomposed into lower sets of atomic actions and can be directly performed by the robot.
- High Level Actions (Complex Tasks): Higher-level, more abstract tasks that must be broken down into a set of primitive or lower-level (complex) actions. Higher level actions are abstract in the sense that they refer to the overall task, or some abstract part of it. For example, the complex task "robot-gotoZoneD" (see

[1] We only test in this case whether the execution of the command was finished and if it has, then that command execution was a success

section 3.2.4) decomposes into first finding a way of getting to Zone D then and navigating to Zone D.

The planning that has been done throughout the project is relatively simple from a conceptual point of view but was difficult to implement. It was difficult primarily due to the fact that our robot's abilities/interface are limited. For instance there is no way of getting odometer readings through the URBI interface which would have helped in calculating the distance travelled by the robot. Another option was the use of a map model, i.e a topographic map of the area, which would have also needed to have been designed, but without any way of telling how far the robot travelled, keeping track of the robot would seem a daunting task and the focus would have shifted away from the motivation of the project. We wanted to focus primarily on planning and communicating these plans to the robots, so programming new abilities in the robot was a secondary concern. When it did come to enhancing the robot's capability the focus was more on sensing the environment which is detailed in section 3.6.

3.2.1 Constraint Types

In the I-X formalism, world state constraints take the form of *predicate-term=value*, for example *on(block1 block2) = true* describes the fact that "block1" is on "block2" is true. Values can either be boolean, longs or symbols. Values that are symbols are case sensitive, therefore a symbol value such as "Banana" is different from "banana". For our purposes, the constraint types that we work with are:

- Posture ?robot [2]: There are two types of values constituting the range for this constraint type: 1). standing 2). sitting.
- Obstructed ?robot: This simple predicate is constantly updated by URBI describing whether the robot has been obstructed by a wall or another object, and its range is a boolean value. Most of our motion actions have this as their precondition, which indicates that the robot needs open space to navigate.
- Inzone ?robot: This predicate describes which zone the robot is currently in. Zones are written for example "ZoneA", "ZoneB" ... up to "ZoneE". Zones are described more in detail in section 3.5

[2] ?robot is a variable which can be bound to some value. In this case you would expect a robot name to be bound to the variable.

- Battery ?robot: This predicate describes the battery level of the robot. In our case the battery level is always 100% in the simulation.

Following an investigation of the capabilities of URBI on AIBO, it became apparent that the robot's behaviour is governed to a large extent by its current state and immediate environment. However, it has only a very limited perception of its state and environment. The representation of this perception is modelled as world-state constraints so that our I-X agent will essentially contain some knowledge of the robot's world. Essentially the I-X Process Panel would know what are the available actions to be sent to the robot at that particular time with the help of the planner. For a simple example let's look at the case where the robot is standing up and we want it to sit down. The robot itself does not know that it is standing up but the I-X Process Panel knows that it is. This is because, on starting the I-X application the predicate term "Posture Robot1[3]" is set to the value "standing". This is a constraint at start up time for the robot: it is assumed that the robot is initially standing. This constraint is a precondition to an action which makes the robot sit. This sit action would have not been available if the value of the "Posture Robot1" was set to "sitting".

For this project the usage of 'effect' predicates differs from the way one might typically use them within I-X. In I-X, if you want to build an action that has an effect on the world then typically you specify the effect in the I-X description of that action. Then, whenever you execute that action from the I-X Process Panel the effect of that action is asserted and seen in the constraints area of the I-X Process Panel. However, for an action that is to be executed on a robot, it is not always satisfactory to simply assert the effects of that action; rather, some feedback is required from the robot to confirm that the action has been successfully completed and to state what its effects were. For a case such as this we have to program into the robot the ability to tell the I-X Process Panel that there was an effect of this action. Details of how this is done is described in section 3.4. A failed action would obviously result in the process panel not being updated.

3.2.2 Primitive Actions

For us to have basic control over our robot we need to first look at all available primitive URBI actions. Most of these URBI actions are basic motion commands such as:

[3]In this case, Robot1 is bounded to the variable ?robot.

- robot.turn(seconds)[4]. This is a function which takes time as an argument, i.e. the robot turns for that number of seconds, a positive number for clockwise rotation or a negative number for an anti-clockwise rotation.
- robot.swalk(steps): This is a walk function which takes the number of steps as its argument i.e., the robot walks forward for that number of steps.
- robot.stand(): Makes the robot stand.
- robot.sit(): Makes the robot sit.

We modelled these actions as possible activity refinements in the I-X formalism. One example of a primitive URBI action modelled in the I-X formalism is the *robot.swalk(steps)* and the *robot.stand()* functions which are shown figures 3.2 and 3.3 in the I-X LISP formalism. The majority of our URBI primitive actions are encoded in this way.

```
(refinement robot-move-by-walking (robot-move ?robot ?steps)
   (variables ?robot ?steps)
   (constraints
        (world-state condition (Posture ?robot) = standing)
        (world-state condition (Obstructed ?robot) = false))
   (annotations
        (comments = "Primitive robot Movement in Steps")))
```

Figure 3.2: robot.swalk(steps) which was designed in the I-X LISP formalism

As you can see in fig 3.2 the LISP style notation is pretty straightforward. Let us look at a brief description of the parts that make up this primitive action refinement. URBI actions in the I-X formalism are the basic components that make up a plan.

3.2.3 I-X Lisp Notation For the Primitive Action Domain

Because of the lack of documentation it was necessary to derive this description from examples generated using I-X.

From fig 3.2 a description of its parts is described here:

[4]Another version of this robot.turn(seconds) is robot.sturn(steps) which uses the number of steps to turn in a clockwise fashion if the number is positive or anti-clockwise if the number is negative. Since robot.turn(seconds) uses time it has greater resolution than robot.sturn(steps) and therefore is more precise, which is why it is favoured out of the two.

```
(refinement robot-standup (robot-standup ?robot)
   (variables ?robot)
   (constraints
        (world-state condition (Posture ?robot) = sitting)
        (world-state effect (Posture ?robot) = standing))
   (annotations
        (comments ="Makes the named Robot Stand UP")))
```

Figure 3.3: robot.stand() which was designed in the I-X LISP formalism

- refinement: One way to look at the refinement is as an option for action, but this option is available only if certain conditions are met. Every refinement has a unique name ("robot-move-by-walking" in fig 3.2) and unique pattern ("robot-move ?robot ?steps" in fig 3.2) which is some activity name followed by any number of parameters. This pattern is used to match the activities currently shown on the I-X Process Panel.

- variables: Arguments that follow this keyword are basically the variables that are used throughout this action. The variables for our primitive action "robot-move" are *?robot* and *?steps* where ?robot refers to the name of the robot that you would like to send this action to and ?steps refer to the number of steps the robot should take.

- constraints: There are two types of constraints on an action. The first is called a "world-state condition" and the second is called a "world-state effect". A "world-state condition" is a statement that can describe particular attributes about the world which must be true at the given point that we wish to apply the action [5]. If there are one or more of these "world-state conditions" then it is necessary for the conjunction of all these statements to be true before this action can be considered. A "world-state effect" is the immediate consequence of an action which modifies the agent's[6] perspective of the world. The words that follow the constraints are the propositions as described in section 3.2.1

- annotations: [32] describes annotations as being "additional human-centric information or design decision rationale to the description of the artefact"; the

[5] This is the same concept as a "precondition"

[6] In this case the Agent is represented by the I-X Process Panel.

artefact in this case is our primitive action “robot-move”.

Other primitive actions that we have developed in the I-X formalism are:

- (robot-searchZone ?robot ?zone): This action, given appropriate values for ?robot and ?zone is sent to the URBI interface for that ?robot which searches for the zone. As of now this primitive action requires no preconditions, but the effect of this action is asserted by URBI. This action utilises vision programming for searching for zones. See section 3.6.

- (robot-inzone ?robot ?zone): This primitive action is sent to the URBI interface to check which zone the robot is currently in, and also requires no precondition. The effect is asserted by URBI

- (robot-align ?robot): This action aligns the robot and it is also used to reset joint positions in the robot because during motion it accumulates positioning errors in its joints. If the total error is able to throw off the walking patterns of the robot this actually forces a major reset in the joint positions which can throw the intended direction of the robot way off. The precondition to using this action requires the posture of the robot to be standing and there are no effects that can be asserted by this action.

Most of the primitive actions descriptions are in a form which does not correspond directly to the names of the URBI actions. The advantage of having it this way is that its in a more human readable form. The disadvantage lies in the fact that we need to interpret each action that is sent so that URBI essentially understands and executes the action.

3.2.4 High Level Actions

From the previous sections 3.2.1 and 3.2.2 we use this knowledge to design our higher-level actions. The way that our higher-level actions are represented in the I-X LISP formalism is a little different from the way our primitive actions are represented. For high-level actions there are two more features that we need to add to our action definition for it to be considered a higher-level task. But first let us look at an example of our higher-level tasks “robot-gotoZoneD”. See figure 3.4 below.

The two new features are:

```
(refinement robot-gotoZoneD(robot-gotoZoneD ?robot)
   (variables ?robot)
   (nodes
     (1 (robot-turn ?robot -19s))
     (2 (robot-align ?robot))
     (3 (robot-move ?robot 30))
     (4 (robot-turn ?robot 5s))
     (5 (robot-align ?robot))
     (6 (robot-move ?robot 18))
     (7 (robot-turn ?robot 12s))
     (8 (robot-align ?robot))
     (9 (robot-move ?robot 17))
     (10(robot-turn ?robot 15s))
     (11(robot-align ?robot))
     (12(robot-turn ?robot 2s))
     (13(robot-move ?robot 2))
     (14(robot-stand ?robot))
     (15(robot-inzoneOCR ?robot)))
   (orderings
      (1 2) (2 3) (3 4) (4 5) ( 5 6) (6 7)
      (7 8) (8 9) (9 10) (11 12)
      (12 13) (13 14) (14 15))
   (constraints
      (world-state condition (Inzone ?robot) = "ZoneA"))
   (annotations
       (comments ="Robot goes to zone B from Zone A")))
```

Figure 3.4: Higher Level Task “robot-gotoZoneD” in I-X LISP formalism

- nodes: This higher-level task is expanded by the Activity Handler into a set of actions called nodes. Notice the nodes in figure 3.4 are also numbered, which play an important role in the orderings.

- orderings: All nodes within a higher-level task can have a full or partial temporal ordering. Most of our higher-level actions have a temporal ordering just like in figure 3.4.

Note that many of the node activities in 3.4 have precise numerical parameters indicating the time and direction a robot turns and the number of steps it takes, etc. It was necessary to determine these values empirically for the given environments.

There are two sets of high-level actions; those for the "Labelled Zones Environment" (which will be described in section 3.5.1 and those for the "Coloured Zones Environment (Which will be described in section 3.5.2). For the OCR environment our high-level actions are:

- robot-gotoZoneD ?robot: This high-level task decomposes into a subset of actions which were heuristically found for getting a robot to "Zone D". The precondition to this action is that the robot should be in "Zone A".

- robot-exitZoneD ?robot: This high-level task decomposes into a subset of actions which were heuristically found for getting a robot from "Zone D" to "Zone A". The precondition to this action is that the robot should be in "Zone D"

- robot-gotoZoneB ?robot: This design for this action follows the same format for "robot-gotoZoneD" with special heuristic actions for getting the robot to "Zone B". The precondition is that the robot should be in "Zone A".

- robot-exitZoneB ?robot: Again this action is similar to "robot-exitZoneD" in terms of how it was designed instead the robot exits "Zone B" to go to "Zone A". The precondition to this action is that the robot should be in "Zone B"

There are two main high-level tasks for the "Coloured Zones Environment". These are:

- searchForZoneA ?robot: This high-level task decomposes into a subset of actions which were heuristically found to get the robot to find "Zone A" from any zone. By "find" we mean to "locate visually". You can use this high-level task in any Zone (even in "Zone A"). There are no preconditions to using this action.

- gotoZone ?robot ?zone: This high-level task decomposes into a series of primitive actions which instruct the robot to go to the Zone represented by ?zone. The precondition to using this action is that the robot should be in “Zone A”.

Effects of all of these actions are asserted by the support functions that we have created. See section 3.4.3 for more details.

3.3 Activity Handler

This section describes the purpose of our activity handler and why it is such a critical component to this system. Explanation is aided via a diagram of an example activity being handled.

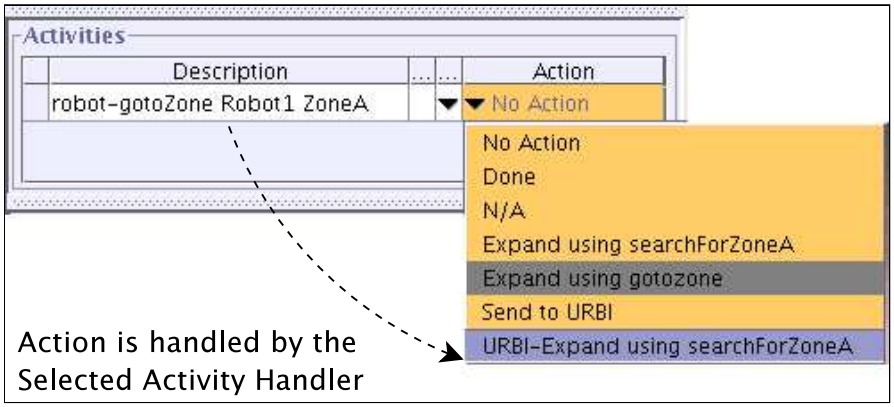

Figure 3.5: Figure shows a snapshot of an action being handled by the selected Activity Handler “URBI-Expand” (in purple), in the Activity Area of the I-X Process Panel. The dotted arrow shows the relationship that that activity is carried out by the handler.

The name of our activity handler is ”URBI-Expand” which primarily has two functions:

- When this activity handler comes into contact with a higher-level action, our HTN planner first checks to see if there are any available refinements for the user to use. The user will then select the refinement and that higher-level activity will then be decomposed into a lower set of primitive or higher-level actions. In figure 3.5 the refinement available for the higher-level action “robot-gotoZone” is “searchForZoneA”.

- When this activity handler comes into contact with a primitive action then that action's "Description" is dispatched to the "Command Interpreter" of the "Robot Controller".

Because of the plug-in capabilities of I-X and the activity handler it would have been difficult to send actions to the robot controller otherwise.

3.4 The Robot Controller

In this section we discuss one of the key aspects of the project, the robot controller and its primary components.

The Robot Controller is made up of the following:

1. Command Interpreter
2. Command Monitor
3. Support Functions

3.4.1 Command Interpretor

The main job of the command interpretor is to turn the I-X action description into its respective URBI command with parameters. However, when an URBI command is selected an URBI Tag (see section 2.4.2) is created in order for keeping track of that command. The URBI command and the URBI tag is then sent to the "Command Monitor".[7] All URBI tags have the "+end" option attached to them (see section 2.4.2 for more details on tagging options).

3.4.2 Command Monitor

The command monitor is responsible for a number of things. These are:

1. To deploy URBI commands to the URBI Server.
2. To use the URBI Tags created by the command interpretor and monitor the execution status (if that command has finished or not) of the URBI commands.

[7]The naming conventions for the URBI Tag was so that they almost matched the name of the I-X action. For example if the primitive action "robot-move Robot1 23" was sent to the command monitor then the URBI tag representative for that action would be "robot_move".

3. To update the status of that action on the I-X Process Panel i.e. the action is painted green if that action is currently being executed and blue if that action has finished.

4. Preparing the Support Functions.

3.4.3 Support Functions

The support functions are special functions which were created in order to enhance the robot's ability to perceive certain things about the environment and about the robot itself. These functions also have the added responsibility of relaying this information to the I-X Process Panel in the form of "world-state constraints" (see figure 3.1). They are also represented in the I-X formalism as primitive actions so when sent to the robot controller can carry out its primary function. Below is a list of the support functions that were created:

- ColourZone: This function when activated searches for a given zone by using colour recognition and is activated by the URBI tag "zone".

- InOCRZone: This function like the "ColourZone" uses OCR recognition to identify the current zone that the robot is in. It is activated by the URBI tag "OCR";

- InColourZone: This function when activated checks which zone the robot is in by using colour recognition. This class is activated by the URBI tag "inzone".

- RobotBatteryCmd: This function is responsible for constantly updating the battery information on the I-X Process Panel. It is activated by the URBI tag "battery".

- RobotObstructionCmd: This function is responsible for constantly checking to see if the Robot is obstructed and updates the I-X Process Panel accordingly. It is activated by the URBI tag "obstruction".

- RobotNoObstructionCmd: Like the "RobotObstructionCmd" class this function constantly checks to see if the Robot is not obstructed. It is activated by the URBI tag "noObstruction".

3.5 The Test Environment

In order to test our system a test environment was needed. It was decided that we model the CISA/AIAI (Centre for Intelligent Systems and their Applications/Artificial Intelligent and Applications Institute) room layout as the test bed because it is an environment which includes interesting features such as a wide open corridor and adjacent offices. The environment offers us challenging issues in navigating to other rooms because these rooms may not be visible from all points.

A relaxed assumption was made about the size of the rooms, corridor spacings and open areas. The model was first designed in Blender 3D [2], a free open source 3D development program. The VRML code from Blender was then ported into the WEBOTS simulation environment and was used as an overlay to build walls and a ground floor. The final model is shown in figure 3.6. However, from the many discussions on the limited basic perceptions that our robot possessed it was felt that this final model would have been a great challenge for the robot to navigate with these perceptive abilities as was discussed in section 2.6. Therefore two simplified modifications to the environment were developed and was deemed suitable for evaluation purposes. The environments that were created are split into zones. A zone can be classified as an open space or a room. All zones are shown and labelled in figure 3.7 and figure 3.8.

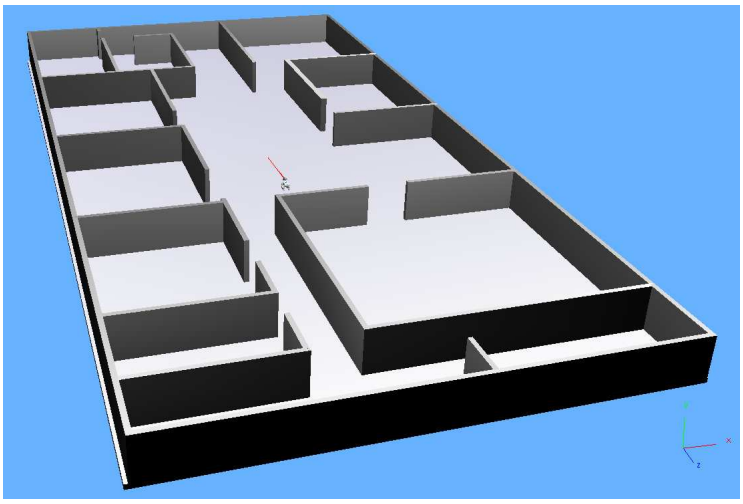

Figure 3.6: The Model of the Test Environment in WEBOTS.

3.5.1 Labelled Zones Environment

To address the problem of our robot's limited navigation capabilities we needed an environment that would aid its motion to the different zones.

The first environment is a simplified version of the overall area, where a number of different rooms were blocked. In this environment signs with a unique letter on each one indicate different zones. These signs were also strategically placed in different rooms. This placement was to ensure that upon identifying a sign the robot will not see other signs in its view and it will essentially know where it is and which way it was facing, which gave us the opportunity to design fairly simple heuristic refinements (such as that shown in figure 3.4)for getting from one zone to the next.

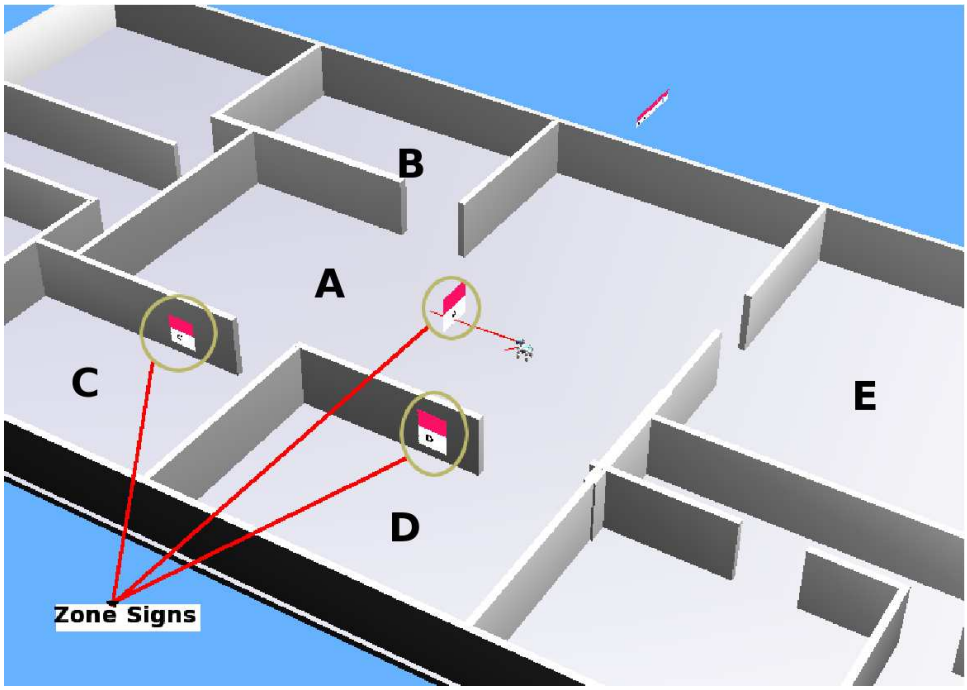

Figure 3.7: The Modified Environment With Signs. There are five zones that are accessible to the robot from Zone A - Zone E.

Details of using OCR in this way are given in section 3.6.1.

3.5.2 Coloured Zones Environment

For this environment the same rooms were blocked off as in the "Labelled Zones" environment. However, the difference here is that each zone is colour coded. See figure 3.8.

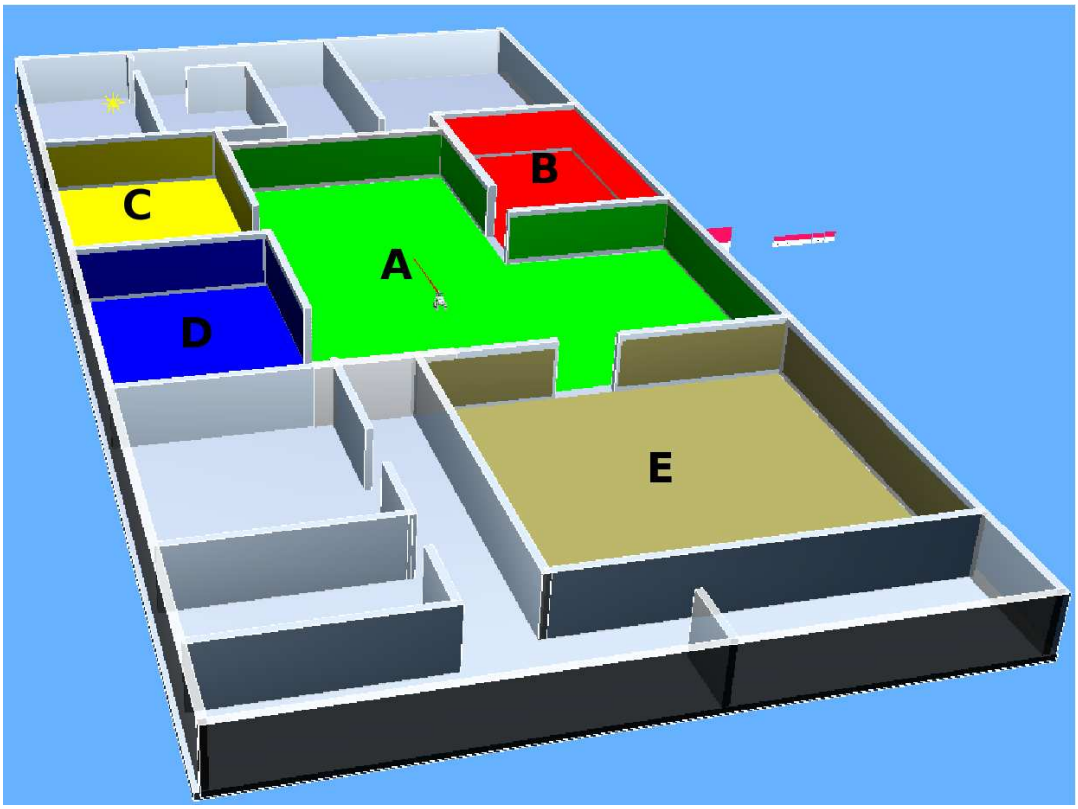

Figure 3.8: The Multi-Coloured world. Each zone stands for a colour. Zone A for Green, Zone B for Red, Zone C for Yellow, Zone D for Blue and Zone E for Dark Khaki.

In order to navigate within this environment the robot would need to know which zone it was currently located. This was done with the aid of colour recognition which is detailed in section 3.6.

Although these two environments are simplified versions of the real environment, the modifications did not stray too far from the main idea which was to keep the environment challenging enough to locate and navigate to different zones. Details of how well our planning representation was suited for the environment and the performance of the robot dealing with such a representation is discussed in detail in Chapter 4.

3.6 Sensing the Environment

This section describes the two different ways that the robot uses to sense its environment. We first describe the colour recognition techniques that were employed, and then go on to discuss the Optical Character Recognition (OCR) technique that was used as another form of "Zone" identification. Both these techniques work on the images that are received by the camera. The camera has a resolution of 208x160 (as default). Because the AIBO robot receives relatively low resolution images this has the potential advantage of faster visual processing.

3.6.1 Using Optical Character Recognition (OCR)

We programmed within the robot the ability to read simple signs. This was done using OCR recognition as provided by an external Java package called the Asprise Java SDK Package[8][1]. Precise details of how the Asprise package provides the functionality for recognising characters is out of the scope of this project,however, we can discuss from a higher-level of abstraction the process of getting OCR results.

Applying the OCR recognition was quite simple. All that was needed to be supplied was the robot's current image, which possibly contained the zone signs (and, hence, the letter displayed on these) and the Asprise recognition function handled the rest. After this, the final task was to verify if the text recognised (if any) matched the current target zone letter. Here is the algorithm doing this:

[8]Normally OCR recognition packages are extremely expensive such as the Tasman SDK package (http://www.tasman.co.uk) which costs about 3000 pounds just for a user license. The Asprise Java package came with a demo license which was free to use. However included in this demo version, extra output was embedded when getting results from the recognition process. I therefore had to find a way of filtering the extra output.

```
Let Image = From Camera
Let TextResult = Get the result of Image when passed to
                 Asprise Text Recognition Function

For The Number of Zones
  {
    IF (TextResult == String Representation of Zone)
 Exit recognition process and return result
  }

Exit Recognition Process and return failure.
```

There were limitations to this approach. For instance the robot needs to be in line with the sign to identify the text correctly. This affected the types of refinements that were developed from moving from one zone to the other using this approach. An analysis of this approach is detailed in chapter 4 section 4.1.1.

3.6.2 Using Colour Identification

For colour identification two methods were employed. One of those methods uses the Euclidean distance between the reference colour that we are searching for and the colour the camera is receiving. Each colour represented a zone which was talked about in section 3.5. The algorithm for identifying a Colour:

```
Image1 = Get Image from Camera
Let count_pixel_accepted = 0

For each pixel in Image1
{
    Let D = Euclidean distance between the
    pixel and the reference colour

  if D < Pixel_Accepted_Distance_Threshold
  {
```

```
            let count_pixel_accepted = count_pixel_accepted + 1
  }

  if count_accepted > Zone_Threshold
  {
  then zone found
  } else {
         zone not found
  }
}
```

Although the algorithm worked it was hard to choose a suitable threshold for a pixel to be accepted in the colour range that we were looking for. However, we looked for the simpler solution which was heuristically coding the ranges for each Red Green Blue Channels.

```
Image1 = Get Image From Camera
Let Zone = Zone to Search For
Let count_pixel_accepted = 0

For Each pixel in Image1
{
   if (pixel In Zone ’Red’ Channel Range
       AND pixel in Zone ’Green’ Channel Range
       AND pixel in Zone ‘Blue’ Channel Range)
   {
       count_pixel_accepted = count_pixel_accepted + 1
   }

   if count_accepted > Zone_Threshold
  {
  then zone found
  } else {
         zone not found
  }
```

```
}
```

This was most effective for the number of colours that we wanted to identify. The next step was to find a representation for these actions in the planning domain.

3.7 Chapter Summary

In this chapter we have described at the design and implementation aspect of the project. We first described in summary the architecture that was developed to provide a base for the reader to logically flow through the rest of the chapter. We then discussed the design of our planning domain and provided answers to the following questions.

1. What were the predicates and how were they designed?
2. What were the primitive actions that were developed and in what notation did they take?
3. What were the high-level actions that were built up as a result from our primitive actions?

We then focused on the robot controller by first describing the way URBI clients tag their messages and how the URBI server handles these tagged messages. From there we looked at utilising this utility to update the I-X Process Panel's knowledge of the robot's world and also the status of sent actions. As a result of this work we now have a working model of a generic high-level controller that allows us to conduct our experiments and evaluate our hypothesis.

In the next chapter we describe the experimental procedures for evaluating our system and give a detailed analysis of the results of our experiments. We critically review the architecture that was designed and also comment on any improvements that we may need to make to the planning model and/or the robot controller.

Chapter 4

Evaluation & Analysis

Our primary goal was to provide evidence in support of our hypothesis which was that "I-X and URBI together can provide a generic controller for Robots". In this chapter we discuss the experiments that may provide evidence. The nature of this hypothesis, and the complexity of the system architecture and robot environment, meant that defining the best way to evaluate the system proved to be a very difficult task. Our experimental approach focuses on having our robot perform the higher-level actions that were discussed in subsection 3.2.4. Since our higher-level actions primarily focused on navigation, one option was to place the robot in situations where it depended on the planning capability of the I-X HTN planner to provide a plan for getting the robot from one zone to the next. While these experiments were unlikely to provide conclusive results, it was hoped that an analysis of failed experiments, and of the system as a whole, would provide insights into the adequacy of the developed architecture. Unfortunately we were not able to test the implemented architecture using the real AIBO robot because this would also have required altering the real environment to match the simulated environments (see fig 3.8); instead, the experiments were performed in the WEBOTS simulation environment.

For the evaluation, two sets of experiments were performed. For each set of experiments a description of the experiment is given along with the results of these and at setup an analysis of the results. We then continue to give a general analysis of I-X and URBI architecture and the extent to which it supports our hypothesis.

4.1 Labelled Zones Experiment

We first discuss the experimental setup and give the results. An analysis of the results that were obtained from the experiments is then given.

In this experiment we test the high-level navigation actions for the "Labelled Zones" environment. The environment for this experiment has only three zones which are *Zone A*, *Zone B* and *Zone D* and the four high-level actions that have been implemented are *robot-gotoZoneB,robot-exitZoneB,robot-gotoZoneD* and *robot-exitZoneD*. The precondition for the robot to move to another zone is that it must be in *Zone A* (which is also the initial state of the robot). For each experiment we choose a maximum number of four high-level actions which are decomposed by the I-X HTN planner and sent to the robot. This number was chosen to limit the run time of the experiments since each action took eight to ten minutes to complete. Successful runs are based on the fact that the robot can perform all actions and identify the zones which it is in correctly. Overall failure will be the result of

1. Failing to identify the zone that it is currently in.
2. Failing to complete a goal of an action i.e. the Robot did not reach its intended target even though all the actions were completed.

Four runs of this experiment were completed and their respective results are recorded in table 4.1.

Run #	Actions to Complete	Completed Actions	Result
1	robot-gotoZoneD, robot-exitZoneD, robot-gotoZoneB, robot-exitZoneB	robot-gotoZoneD, robot-exitZoneD, robot-gotoZoneB, robot-exitZoneB,	Success
2	robot-gotoZoneB, robot-exitZoneB, robot-gotoZoneB, robot-exitZoneB	robot-gotoZoneB, robot-exitZoneB, robot-gotoZoneB, robot-exitZoneB,	Success
3	robot-gotoZoneD, robot-exitZoneD, robot-gotoZoneD, robot-exitZoneD	robot-gotoZoneD, robot-exitZoneD, robot-gotoZoneD	Fail
4	robot-gotoZoneB, robot-exitZoneB, robot-gotoZoneD	robot-gotoZoneB, robot-exitZoneB, robot-gotoZoneD	Success

Table 4.1: Table shows the results from 4 experimental test runs.

4.1.1 Analysis of the Labelled Zone Experiment Results

From the results from table 4.1 there was only one test run that failed. This was because the robot could not identify the letter on the sign using our support functions. This in turn was because, after performing the necessary actions in getting to “Zone D” the robot was placed in a slightly unusual position where it could not read the sign due to the accumulation of the errors in the joints as a result of a bad heuristic in our refinement for travelling to “Zone D”.

Note that many of the node activities pertaining to this environment have precise numerical parameters indicating the time and direction a robot turns and the number of steps (see figure 3.4 for example) it takes, etc. Because of this and the time it actually took to run the experiments a more generalised environment and more generalised actions needed to be defined and experimented on.

4.2 Coloured Zones Experiment

We first discuss the experimental procedures of this set of experiments and then give the achieved results. An analysis is then given of these results.

In this experiment the immediate goal was to record the success and failure of the robot navigating to different random zones using the higher-level tasks *robot-searchForZoneA* and *robot-gotoZone*[1].

Using the grid map in figure 4.1 a number was randomly generated between 1-58 in order to determine an approximate starting position for our robot. Another number was picked between 1-5 which represented the end goal zone that the robot was going to navigate to, with 1 = Zone A, 2 = Zone B ... etc. If the robot’s starting square was located in the target zone then another target zone was selected.

The experiment was run fifteen times[2]. The starting position and the intended target zone was recorded for each experiment run. The actions that the HTN planner selected for deciding how to get to a zone were also recorded. A run was successful if the robot reached its intended target, unsuccessful otherwise. Table 4.2 summarises the results obtained from doing a test run of 15 runs.

[1] refer to section 3.2.4 for a review of these two high-level actions.

[2] The same explanation applies for this experiment as it did for Labelled Zones Experiment

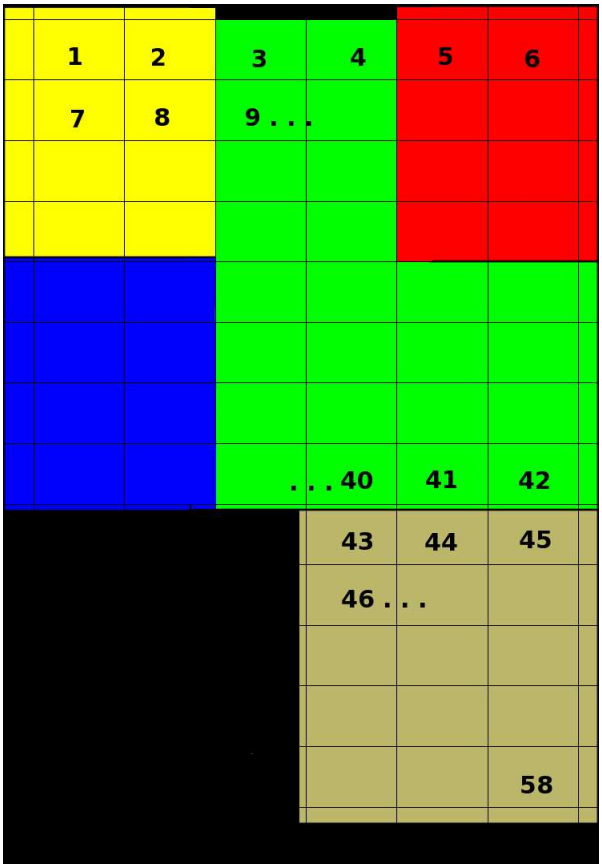

Figure 4.1: The numbered grid. For each test run a number is chosen at random between 1-58. The robot is then placed in the approximate position of that square.

4.2.1 Analysis of Coloured Zone Experiment Results

From the results from table 4.2 it can be seen that most of the test runs resulted in success. For the first two runs, failure was due to the fact that the AIBO simply could not visually locate [3] the target zones and so the support functions did not assert this fact on the I-X Process Panel. Thus, no further actions were applicable and planning failed. For test runs #5, #12 and #15 the robot was placed in a position where it identified the target zone but failed to reach it. This was a result of a poor heuristic in our "ColourZone" support function (See section 3.4.3) which calculated the distance in terms of steps depending on how much of the zone colour that the robot identified. For run #11 the robot was placed very close to the wall but could still identify the connecting "Zone A". However, when approaching Zone A the wall disturbed the direction heading of the robot and so it never entered "Zone A" and could not reach its intended target "Zone

[3] The identification is handled by the "ColourZone" see section 3.4.3.

Run #	Position on Grid	In Zone	Target Zone	Result
1	5	Zone B	Zone D	failure
2	30	Zone A	Zone B	failure
3	16	Zone A	Zone B	success
4	34	Zone A	Zone E	success
5	49	Zone E	Zone B	failure
6	38	Zone D	Zone C	success
7	43	Zone E	Zone D	success
8	43	Zone E	Zone B	success
9	17	Zone B	Zone D	success
10	20	Zone C	Zone A	success
11	45	Zone E	Zone D	failure
12	5	Zone B	Zone E	failure
13	18	Zone B	Zone C	success
14	25	Zone D	Zone B	failure
15	35	Zone A	Zone E	success

Table 4.2: Table shows the results of the 15 experiments for navigating in the colour zone.

D".

In conclusion of these results, success was possible only if the robot was placed in a position where it was able to identify the connecting Zone A and also had a clear view of other zones from Zone A. In a real world scenario this idea of a coloured world would be more difficult in identifying colours since the lighting conditions may change. Since our colour environment is one that has no lighting fluctuations our robot was able to identify all the colours all the time.

Both types of experiments provide us with evidence that I-X and URBI together are capable of providing robot control at a higher-level. The question of whether these experiments provided any evidence on the genericity of the system is answered in section 4.3.3.

4.3 General Discussion

In this section we take a step back to look at the influence of the architectural design on the results from our experiments. We first discuss the relevance of I-X as a framework for this application, then we focus on URBI as a generic robot controller. We then discuss how well the two work together and proceed to answer the question of whether this combination of I-X and URBI has the potential to encompass the generic environment and robots.

4.3.1 I-X as a Robot Planner

Some of the flaws within I-X that have been noted in the course of this work:

- The inability for the I-X planner to process I-X primitive or complex actions' effects because the support functions do this.
- I-X does not provide any utility for recovery from plan failure
- I-X does not do conditional planning; For example If the robot fails to reach its intended target, have the robot search again.
- I-X definition of a primitive action is epistemologically inadequate.

The I-X framework is used primarily as a planning tool to able to come up with complete plans, but at the moment it provides no support for plan execution and monitoring. The true power of I-X lies in its planning capability in conjunction with the $< I - N - C - A >$ ontology. If the planner does not know all of the true world-state conditions and effects required by a primitive or high-level action then the planner will not be able to plan effectively. For our project (to recap) the refinements available to the planner model only partially the conditions and especially the effects of the corresponding actions. Our support functions assert most of the effects of their counterpart I-X actions which may be necessary for subsequent actions. This is necessary if the refinements are actually going to be enacted, since only when the robot has completed the action can the effects be said to be true, but means they are inadequate for planning purposes! Therefore I-X is perhaps not capable of being fully integrated into any robot controller. Other architectures such as ATLANTIS and ALFA [15] are integrated within their robots, therefore they can fully observe the effects of all actions(high-level or primitive) and are able to plan efficiently.

The I-X architecture provides no utility for plan failure. Once a plan is prepared by the I-X Planner then that plan is "static" and will not change regardless of what goes on in the environment. If I-X was to incorporate conditional planning within the I-X Planner then this would increase our chances of fulfilling all high-level plans and possibly improve the success rate of our experiments in finding their intended target zones.

I-X's definition of a primitive action is epistemologically inadequate. If we look back at our discussion of the I-X LISP notation (see section 3.2.3) where our primitive actions are defined to be refinements but with no nodes or orderings of those nodes, intuitively we know that a primitive action cannot be a refinement. Nevertheless, this was felt to be necessary to provide an activity 'placeholder' in plans and corresponding to the actual URBI primitive(s), which could not be represented in the refinements but only in the code invoked when the activity is handled.

4.3.2 URBI as a Generic Robot Controller

Some of the noted flaws within URBI are:

- Reduced Functionality
- Non-generic parameters
- Potential Communication Problems

Robots normally have the ability to perform a multitude of actions. As we have discussed before in section 2.6 the AIBO with its built-in Sony operating system allows for the robot to perform quite sophisticated actions. We noted that the URBI interface provides access to only a subset of this functionality. For instance, instead of being able to automatically identify the shape and colour of its toys (Pink Bone and Pink Ball) the AIBO with URBI can only identify the colour pink.

Coming back to the actions that URBI provides us, some of those actions are non-generic. For instance, the URBI action "robot.swalk" uses the number of steps as an argument. What would happen if this was applied to a wheeled robot or a flying robot? This function is specific to legged-robots. Other functions such as "robot.turn" uses time as its argument which is quite ambiguous if you were to apply the same function to a different robot. A better argument would be to turn in degrees which would be the same regardless of what type of robot it is.

In section 3.4.3 we discussed the communication of commands between the "Command Monitor" and the URBI Server. The flaw lies within the communication. A successful transaction of commands between these two components solely depends on the nature of communication. Because the robot, its simulation environment, our robot controller and I-X reside on the same computer the communication link was efficient and reliable. If these actions were to be sent through a different communication medium such as a "Wireless Fidelity" (WIFI) connection which is part of the real AIBO then the probability of the robot not being able to perform actions sent remotely by some URBI client will be considerably higher since for instance:

1. Our Robot is in constant motion and so signal strength from our wireless server will vary.

2. The WIFI connection may not be as reliable the moment we would like to connect to the URBI server.

4.3.3 I-X and URBI: A Generic High-Level Robotic Controller?

Let us say for instance that we do some manipulation on our environment to help answer the question. For our "Labelled Zones" environment we build bigger rooms and we add more signs. With our current architecture configuration the refinements that we have designed for navigating in this environment would fail and so will the previous experiments that we have done. This is because when designing our refinements we took the previous environment configuration into consideration and became very specific as to the number of steps it should take when getting from one zone to another. Scalability is a non-generic issue in the sense that when the scale changes we will have to modify the actions in our action domain to suit the environment.

For our "Coloured Zone" environment let us say for instance that we add more rooms and hence more colours. The colours that were chosen for our previous environment were such that the robot can easily tell them apart. However, the more colours we have the more precise our algorithm for identifying colours would have to be and hence more room for error since some colours will be very similar to others. So again, the environment was specifically made to accommodate the limitations of URBI.

In the end our environments held the key for our navigation and were specific to our development of our action refinements. With this in mind and the flaws that we noted about I-X and URBI, I will conclude by saying that there was not enough

evidence provided from experiments and analysis to give any support to our hypothesis. Instead, another weaker hypothesis is put forward as a result and this is that I-X and URBI together can provide a robotic controller capable of performing some high-level tasks for a specific robot in a specific environment as shown by our implementation. However, no claim about genericness can be made, mostly because no such claim is true for URBI.

4.4 Chapter Summary

In this chapter we described the experiments that we used to evaluate our system. We then gave an analysis of the results from those experiments and how they contributed to our hypothesis. After which we gave a general discussion on the architecture itself, critically analysing the short falls of our system, and with the results of our experiments concluded that not enough evidence was provided in support of our original hypothesis.

Chapter 5

Conclusion

In this chapter we provide a brief overview of the work done in this project highlighting and enforcing main design decisions that were made. We then conclude by restating the hypothesis which was that "I-X and URBI together can provide a generic controller for Robots", and a brief discussion about our findings and why sufficient evidence could not be provided to support it.

In this thesis we investigated the development of an architecture that would involve a generic HTN planner (provided by the I-X framework) sending actions to a robot via a generic controller (provided by URBI) in order to perform some complex task that cannot be executed by the robot directly. Our primary complex task having our robot navigate from one zone to another using a generic office environment, with zones representing rooms and open spaces. In order to do this we first focused on identifying the robot's capabilities which are available actions that URBI provided. We modelled them in a way that I-X would be able to reason about and create plans from. This led to a small library of primitive I-X actions implemented in the I-X LISP notation such that resulted in a one to one mapping to URBI commands. The communication and monitoring of commands were handled by the robot controller that we designed and implemented. It was also the job of the robot controller to update I-X's knowledge of the world.

However, these commands were not enough because the robot still lacked the functionality to know where it was in the world and without this knowledge, navigation within the environment cannot be achieved. Therefore, it was decided that the set of URBI commands to be enhanced and so the ability to identify colours and read letters were created for our robot. This also meant that we would have to alter the environment to suit the new abilities of our robot. Therefore, the environment was redesigned into

two different ways where one environment was coloured such that each zone had its own colour, and the other was amended with signs with letters where each zone had a sign with a different letter to represent that zone. This also meant that we updated the I-X action domain where the robot's new abilities were represented as primitive actions. It gave us the opportunity to evaluate our system by doing some limited experiments for each environment by testing the navigational abilities of the robot.

The results of the experiments for the labelled environments showed that the robot completed all higher-level tasks for 3 out of 4 the test runs which provided evidence that showed the system was able to decompose and execute higher-level task. Failure in the other 1 out of 4 of the test runs was due to a a very strict heuristic high-level refinement which did not account for minor errors in navigation and the robot failed to identify the zone which it was in. The results for the experiments in the coloured zone showed that on 9 out of 15 of the test runs the system was able to locate its target zone, which again provides evidence that the robot is able to decompose and execute higher-level tasks. Again, the other 6 out of 15 of the test runs were failures as a result of insufficient heuristics for finding the distance between zones or for visually locating target zones. However, this empirical testing still did not provide enough evidence to support our hypothesis.

Our hypothesis was that I-X and URBI together can provide a generic high-level controller for robots. The main reason for the lack of support of evidence for our hypothesis was due to the fact that from the beginning we modelled our actions and environments to suit the limitations of URBI and also we did not test our action domain on other robots. This therefore does not support the genericness aspect of the hypothesis that I-X and URBI together can be applied to any environment and to any robot. From the results of the experiments and general discussions of I-X and URBI as important components to this architecture this only provided evidence for a weaker hypothesis, namely that I-X and URBI together provide a high-level controller for a **specific** robot in a **specific** environment.

Bibliography

[1] Asprise Optical Character Recognition Package. http://asprise.com/home/.

[2] Blender 3D Modelling Program. http://www.blender.org.

[3] I-X Architecture. http://www.aiai.ed.ac.uk/project/ix/architecture.html.

[4] Sony Aibo ERS7. http://www.sony.net/Products/aibo/.

[5] U2S Robot. http://u2s.gel.usherbrooke.ca/accueil.html.

[6] Urbiforge. http://www.urbiforge.com.

[7] J-C Baillie. URBI: Towards a universal robotic body interface. *2004 4th IEEE-RAS International Conference on Humanoid Robots*, 1:33 – 51, 2004.

[8] E. Beaudry, F. Kabanza, and F. Michaud. Planning for a mobile robot to attend a conference. Technical report, University of Sherbrooke, 2004.

[9] T. Belker, T. Hammel, and J. Hertzberg. Learning to optimize mobile robot navigation. *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, pages 4136–4141, 2003.

[10] N. Constantinos, B. Jaramaz, and D. Anthony.

[11] Minh Binh Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)*, 20:155–194, 2003.

[12] K. Erol, J. Hendler, and D. Nau. Semantics for HTN planning. Technical report, University of Maryland, College Park, March 1994.

[13] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. *Proceedings of the National Conference on Artificial Intelligence*, 2:1123 – 1128, 1994.

[14] M. Fujita. Robot entertainment for digital creature's era. *30th International Symposium on Robotics. Celebrating the 30th Anniversary toward the Next Millennium*, pages 287 – 93, 1999.

[15] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. *AAAI-92. Proceedings Tenth National Conference on Artificial Intelligence*, pages 809 – 15, 1992.

[16] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*, chapter 11. Morgan Koffman Publishers, 2004.

[17] R.P. Goldman, K.Z. Haigh, D.J. Musliner, and M.J.S. Pelican. Macbeth: a multi-agent constraint-based planner [autonomous agent tactical planner]. *21st Digital Avionics Systems Conference. Proceedings (Cat. No.02CH37325)*, vol.2:7 – 3, 2002.

[18] A.K. Jonsson, P.H. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in interplanetary space: theory and practice. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 177 – 86, 2000.

[19] T. Kanade, D. Gioia, D. Anthony, O. Ghattas, B. Jaramaz, M. Blackwell, L. Kallivokas, F. Morgan, S. Shah, and D. Simon. Simulation, planning, and execution of computer-assisted surgery. In *Proceedings of the NSF Grand Challenges Workshop*, March 1996.

[20] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot hrp-2. *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, Vol.2:1083 – 90, 2004.

[21] F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria. A system for hierarchical planning in service mobile robotics. Technical report, Laboratorium DST, University of Genova, Italy, 2004.

[22] O. Michel. Webots: symbiosis between virtual and real mobile robots. *Virtual Worlds. First International Conference, VW'98. Proceedings*, pages 254 – 63, 1998.

[23] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[24] O. Obst, A. Maas, and J. Boedecker. HTN planning for flexible coordination of multiagent team behavior. Technical report, AI Research Group, University of Koblenz, 2003.

[25] M. Piaggio and A. Sgorbissa. Real-time motion planning in autonomous vehicles: A hybrid approach. In Evelina Lamma and Paola Mello, editors, *AI*IA*, volume 1792 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 1999.

[26] M.E. Pollack. Evaluating planners, plans, and planning agents. *SIGART Bull.*, 6(1):4–7, 1995.

[27] M.E. Pollack. The uses of plans. *Artificial Intelligence*, 57(1):43 – 68, Sept. 1992.

[28] M. Ridsdale. Create Advanced AI Mission Control Algorithms For Use In a Simulated Environment. Master's thesis, University of Edinburgh, 2004.

[29] A. Tate. Project Planning Using a Hierarchic Non-linear Planner. Technical Report 25, Department of Artificial Intelligence, University of Edinburgh, 1976.

[30] A. Tate. Generating project networks. In *Proceedings 5th IJCAI*, pages 888–893, 1977.

[31] A. Tate and K. Currie. O-plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[32] A. Tate, J. Dalton, and S. Potter. Intelligible messaging - activity-oriented instant messaging. Technical report, University of Edinburgh, Artificial Intelligence Applications Institute, 2003.

[33] David S. Touretzky and Ethan J. Tira-Thompson. Tekkotsu: A framework for aibo cognitive robotics. *Proceedings of the National Conference on Artificial Intelligence*, 4:1741 – 1742, 2005.

[34] R.T. Vaughan, B.P. Gerkey, and A. Howard. On device abstractions for portable, reusable robot code. *Proceedings 2003 IEEE/RSJ International Conference on*

Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), vol.3:2421 – 7, 2003.

[35] S Vere. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5, 1981.

[36] T. Wagner and K. Hübner. An egocentric qualitative spatial knowledge representation based on ordering information for physical robot navigation. In *RoboCup*, pages 134–149, 2004.