Vasco Pedro dos Anjos e Santos

DSAAR: Distributed Software Architecture for Autonomous Robots

Lisboa
2008

UNIVERSIDADE NOVA DE LISBOA

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Electrotécnica

DSAAR: Distributed Software Architecture for Autonomous Robots

Vasco Pedro dos Anjos e Santos

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Electrotécnica.

Orientador: Prof. Luís Correia

Elemento de ligação à FCT/UNL: Prof. José Barata

Lisboa
2008

Acknowledgements

This dissertation has been supported by the Portuguese SME company IntRoSys, S.A.[1].

First of all, I would like to express my gratitude to my dissertation supervisor, Prof. Luís Correia, to my colleague Pedro Santana, for all the valuable help, comments, advises, and support. My sincere appreciation to my colleagues at IntRoSys for all the relevant comments and support, namely: Carlos Cândido, Paulo Santos, Mário Salgueiro, Luis Almeida and João Lisboa. Also thanks to Prof. José Barata for the opportunity and motivation.

I cannot forget my family, especially my parents, Maria Madalena Macieira dos Anjos and Victor Manuel dos Santos, for helping me become who I am today. I would also like to thank my friends for their support and enriching experiences. Most of all those who endured my journeys through *Styx*, *Acheron* and other of *Charon's* sailing locations, you know who you are. As *Albert Camus* said:

Don't walk behind me, I may not lead.
Don't walk in front of me, I may not follow.
Just walk beside me and be my friend.

[1] www.introsys.eu

What is left when honour is lost?

Publilius Syrus

Resumo

Esta dissertação apresenta uma arquitectura computacional desenhada de forma a permitir o rápido desenvolvimento e prototipagem de sistemas multi-robôs, denominada de Distributed Software Architecture for Autonomous Robots (DSAAR). Os blocos de construção da arquitectura DSAAR permitem ao engenheiros focar-se no modelo comportamental dos robôs e do colectivo. Esta arquitectura é de especial interesse nos domínios onde vários humanos, robôs, e agentes computacionais têm de interagir de forma contínua. Deste modo a rápida prototipagem e a reutilização são de maior importância. A arquitectura DSAAR tenta lidar com estes requesitos de forma a alcançar uma solução para o problema de n-humanos e m-robôs com um cenário de boas práticas de desenho e de ferramentas de desenvolvimento.

Esta dissertação também se irá focar ao nível das interacções Humano-Robô, especialmente no domínio da teleoperação, onde o julgamento humano faz parte integral do processo, fortemente influenciado pelos dados de telemetria recebidos do ambiente remoto. Logo a velocidade a que os comandos são enviados e os dados de telemetria são recebidos é de crucial importância. Utilizando a arquitectura DSAAR é proposta uma aproximação de teleoperação, desenhada de forma a provendiciar a todas as entitades presentes na rede a noção de realidade partilhada. Nesta aproximação todas as entidades são fontes de conhecimento, numa abordagem semelhante à do *distributed blackboard*. Esta solução foi desenhada de forma a fornecer uma resposta em tempo real, assim como uma percepção o mais completa possível do ambiente circundante ao robô. Os resultados experimentais obtidos com o robô físico sugerem que o sistema é capaz de garantir uma interacção próxima entre os utilizadores e o robô.

Abstract

This dissertation presents a software architecture called the Distributed Software Architecture for Autonomous Robots (DSAAR), which is designed to provide the fast development and prototyping of multi-robot systems. The DSAAR building blocks allow engineers to focus on the behavioural model of robots and collectives. This architecture is of special interest in domains where several human, robot, and software agents have to interact continuously. Thus, fast prototyping and reusability is a must. DSAAR tries to cope with these requirements towards an advanced solution to the n-humans and m-robots problem with a set of design good practices and development tools.

This dissertation will also focus on Human-Robot Interaction, mainly on the subject of teleoperation. In teleoperation human judgement is an integral part of the process, heavily influenced by the telemetry data received from the remote environment. So the speed in which commands are given and the telemetry data is received, is of crucial importance. Using the DSAAR architecture a teleoperation approach is proposed. This approach was designed to provide all entities present in the network a shared reality, where every entity is an information source in an approach similar to the distributed blackboard. This solution was designed to accomplish a real time response, as well as, the completest perception of the robots' surroundings. Experimental results obtained with the physical robot suggest that the system is able to guarantee a close interaction between users and robot.

Keywords: Multi-Agent Systems, mobile robots, software architectures, distributed architectures, teleoperation, human-robot awareness.

Symbols and Notations

Symbol	Description
ACL	Agent Communication Language
AI	Artificial Intelligence
API	Application Programming Interface
BDI	Belief Desire and Intention
DARPA	Defense Advanced Research Projects Agency
DSAAR	Distributed Software Architecture for Autonomous Robots
GPS	Global Positioning System
HID	Human Development Index
HRI	Human-Robot Interaction
IPC	Inter-Process Communications
JADE	Java Agent DEvelopment Framework
KB	Knowledge Base
MAS-IM	Multi-Agent System Interaction Mechanism
ms	milisecond
PE	Physical Entity
PDL	Process Description Language
SME	Small Medium Enterprise
USAR	Urban Search And Rescue

Contents

List of Figures

Chapter 1

Introduction

The motivation for the development of the Distributed Software Architecture for Autonomous Robots (DSAAR) is the current hand craft approach to robotics. Nowadays there is a lack of architectures that gather both computational and control models within the same framework. Usually scientific papers discard all implementation details, focusing solely on the control models; robots are usually one-of-a-kind gadgets that are developed as the scientific contributions require.

However, as robot technology matured, it becomes highly relevant to develop tools focusing on the fast development and prototyping of multi-robot systems; only then mobile robots will be able to deal with fast market development, customisation, and reusability requirements. In this line of thought, the DSAAR architecture intends to provide the engineer and/or researcher with a set of design models and implementation tools to respond to these requirements.

DSAAR provides a computational model (e.g. object-oriented, agent-based) without committing to any control paradigm (e.g. behaviour-based, deliberative). In the future it is expected to have some tools to instantiate DSAAR according to a set of given control models. Furthermore, it is shown that the computational model fits easily with the most relevant control models applied in mobile robots. It will also be shown that the architecture scales according to the available computational requirements. In fact, the separation between computational and control models allows the reusability of both models independently, making both contributions

valuable by themselves.

Other architectures, like CLARAty [Nesnas et al., 2006], Mission Lab [Endo et al., 2002] and MARIE [Côté et al., 2006] were not designed using the multi-agent paradigm, which make them less adequate for multi-robot systems as DSAAR intends to be. Moreover, usually architectures commit to some control models. Conversely, DSAAR is implemented using the multi-agent paradigm and intends to be generic enough to handle any control model, like Behaviour-Based ones (e.g. [Brooks, 1986, Arkin, 1989, Santana and Correia, 2005] and Hybrid ones (e.g. [Konolige et al., 1997, Philippsen and Siegwart, 2003]).

DSAAR architecture is based on Linux inter-process communication (IPC) mechanisms and multi-agent Jade [Bellifemine et al., 2003, Website, 2006] software platform. The former allows interoperability between independent processes running within the same machine, whereas the latter extend this concept to multi-agents that can spread through all over the internet. The success measure of this architecture should be specified in terms of prototyping speed and code reusability. It was selected Linux and Jade because both can be used in commercial platforms even though they have open-source community support. A discussion was started about how much should the DSAAR commit to this open-source approach. Nevertheless, both Linux and Jade have LGPL licenses that allows proprietary software development.

DSAAR multi-agent approach also provides a study on Human-Robot interaction, mainly on the teleoperation subject. In teleoperation, as in any system involving humans, there is a clear need to map concepts between different representations. If one considers a shop floor the human notion of a specific movement, has to be mapped to the robots' own representation of the world. It could be imposed a shared vision of reality, a common ontology, however the low-level notion of reality can not be described the same way as high-level notions are. Teleoperation poses the same problem, since the robots' notion of its' posture or telemetry has to be mapped into human readable information. When considering a system with heterogeneous entities, a common ontology becomes necessary in order to provide a common language to the system. Thus allowing the ability for the several entities to understand each other, when used.

Some multi-agent teleoperation approaches [Nourbakhsh et al., 2005] have considered the

payload on the network excessive, due to telemetry data and teleoperation commands, thus placing them on a separate channel. This approach, does not feel the need of having the telemetry data being represented through the hole network, rather choosing to have seldom points of direct mapping. In this work an approach that considers the telemetry data and teleoperation commands an integral part of the multi-agent systems' network is purposed. As to allow a shared representation of all teleoperation orders and telemetry data.

This system was designed based on a collaborative network, it is assumed that every entity present in the network will work to the best outcome possible without any competition with other entities. This assumption allows for a simplification on the negotiation mechanisms, since there is no need to have elaborated trust mechanisms or partner selection mechanisms.

Another important issue in teleoperation is the perception the user has from the environment where the robot is, as well as, the robots' posture are of crucial importance, as is real time vehicle control. It is reasonable to say that if the operator has not a real perception of the robots' beliefs, e.g. posture or position, the operator can not decide where to lead the robot. The operator must also be able to perform manoeuvres in real time fashion, since any delay may render that manoeuvre useless.

1.1 Problem Statement

As previously stated, the goal of this dissertation is to develop an architecture that copes with the problem of working with heterogeneous systems. This problem has two main subjects, the robotic platform development problem and the human-robot interaction problem.

This first problem can be seen in various forms, since when developing a robotic platform developers have to focus in several modules, e.g. the obstacle detection module; the planner module; the locomotion module; etc., and when the several modules are ready there is still the 'wiring' of those modules. Some architectures have purposed an approach to this problem *clarity, etc.*, however these approaches restricted the developers control method approach. These methods have a predefined approach which forces the developers to follow strict rules while

developing their own modules and not only the outputs. This becomes a problem for instance when developing the locomotion module, vehicles with different kinematics have to follow a similar structure which considering a track or a wheeled vehicle may difficult the developers work [Nayar, 2007].

Regarding the human-robot interaction problem, there are several important issues worth discussing, such as, the multi-user situation where the addition and removal of entities (e.g. supervisors, operators, robots, etc.) becomes a problem. The case study of this thesis is teleoperation. In teleoperation, as in many system involving humans and machines, there is a clear need to map concepts between different representations. This can be done by using a shared ontology to provide a common language among team members, thus allowing the ability for the entities to understand each other. For instance, robot's sensor data must be mapped to a human readable set of symbols. Conversely, the human desired robot heading must be mapped into robot understandable symbols. The operator must also be able to perform manoeuvres in real time fashion, since any delay may render that manoeuvre useless, or even dangerous. This limits the type and amount of information to be mapped between human and robot. Some multi-agent teleoperation approaches [Nourbakhsh et al., 2005] have considered the load of telemetry and command messages on the network excessive thus placing them on a dedicated channel.

Current software architectures are mainly focused multi-robot platform compliance and others consider Human-Robot Interaction only in the case of mostly autonomous robots (e.g. on the execution of plans). Architectures such as MARIE [Côté et al., 2006] focus the HRI on the case of a human interacting directly with the robot (e.g. the robot works as a greeter). Other architectures such as the NASA Ames' Mobile Agents Architecture (MAA) [Sierhuis et al., 2005] focus solely on the coordination of tasks and on information coordination and retreavel. Finally architectures such as CLARAty [Nesnas et al., 2006] focus on the computational model of the robot and its' modularity with several robotic platforms. All of the afore mentioned topics will be further explained in chapter 2. Currently there is no architecture that aims to support the coordination of planning and control with the support of modularity and multi-robot platform support. Therefore, one of the motivations for this dissertation is to find an architecture that can

comply with the requirements presented above.

1.2 Solution Prospect

The DSAAR architecture development is part of the AMI-02 project, which aims the development of a multi-robot system for humanitarian demining (refer to [Santana et al., 2005a] for a more detailed information about the project).

There are a set of characteristics about the humanitarian demining domain that makes it suitable for multi-agent approaches (see [Santana and Barata, 2005] for a study on this issue). In particular:

- Landmines are widespread worldwide and usually located in most of terrains, which requires that robots have to operate in highly heterogeneous environments. Usually this problem is tackled by the exploitation of heterogeneous robots specialised into a set of sensors and terrains. This means that the system's configuration is highly knowledge intensive.

- Landmine detection is a highly complex process, which requires a lot of expertise. This means that one may have many experts, operators, and machines operating in parallel though in cooperation. Thus, the system is distributed by nature.

- The small and closed nature of demining market requires companies to develop products with high potential for technology transfer [Santana et al., 2005b]. Only then it is possible to guarantee return of investment. This means that the system must be fairly generalisable, reusable, and henceforth scalable.

All these characteristics lead to the definition of the multi-agent architecture illustrated in figure 1.1.

The architecture can be seen as composed of three layers, in which the *physical agents layer* refers to the actual robots. The *software agents layer* is responsible for product management (i.e. landmine detection sensors output and higher abstract data), configure/manage the whole

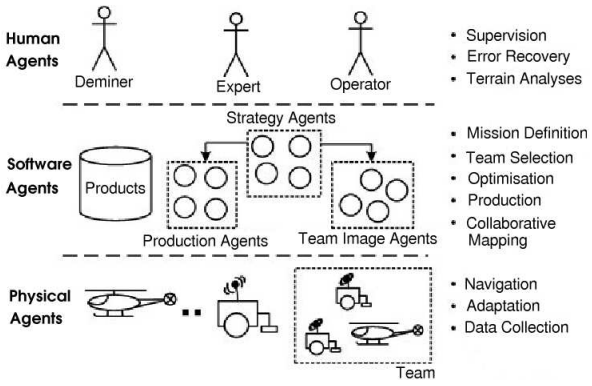


Figure 1.1: Multi-agent system architecture [Santana et al., 2005a]

system, and to create images of robot's world perspective so as to allow operators to observe what is happening in the field. Finally, the *human agents layer* is populated with key personnel involved in the operations, which interact with the system (i.e. software and physical agents) in timely fashion.

Robots in the *physical agents layer* must be endowed with a certain degree of autonomy, so as to guarantee at least safe navigation capabilities and payload sensory information management. Software agents may be distributed over an arbitrary set of machines (e.g. PDAs), which can be dedicated for complex sensor fusion, human-machine interfaces, data logging, etc. Then, these agents communicate with each robot via an unified language (i.e. an ontology). The architecture must be flexible enough to allow the addition and removal of robots and software agents as required. Hence, there is an actual need for distributed multi-agent systems support, along with a need for hardware tightly coupled processes. Moreover, there is the need for an architectural basis that allow developers to scale the system as new robots or robot types are required, users access to the system, etc. Hence, there is a clear need for the system to be evolvable.

In reality DSAAR has been designed so that there is no privileged point to look to and out of the system. The *physical agents*, the *software agents* and the *human agents* are all considered *entities* of the system. A *entity* refers to any physical device that exhibits some autonomous behaviour. For instance, a *entity* can be either a human interacting through a laptop, a robot, or even any other device that could be seen and endowed with some autonomy. An example is

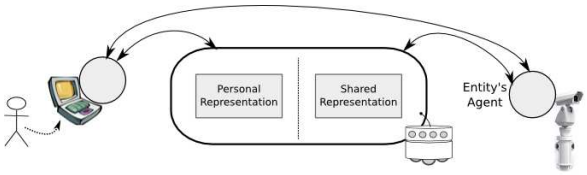


Figure 1.2: DSAAR General Overview

a video camera which can be seen as an autonomous agent requesting services of the robotic platform for displacing itself. *Entities* that represent a physical being can also be called of emphphysical entity. Each entity interacts mainly through an *agent*, a general representation of the system is depicted in figure 1.2.

In this architecture each entity is considered a knowledge source, fostering a shared representation of each entities beliefs, desires and intentions (BDIs), resembling the distributed blackboard architecture [Nolle et al., 2001]. This approach enables a shared representation of all teleoperation orders and telemetry data through the system. Considering that each entity has its own perception on the reality, a shared representation must be imposed, in order to allow a common way of interaction, i.e. an ontological commitment [Grosz and Kraus, 1999], [Cohen and Levesque, 1991], [Grosz and Kraus, 1996] (see Figure 1.2).

Contrary to other approaches, such as [Nourbakhsh et al., 2005], that have a dedicated channel through which all teleoperation commands and telemetry messages are exchanged, in the DSAAR architecture all the teleoperation related information flows in the MAS network. Working with a shared network approach forced the information exchange to be optimised.

Hence, the goal of the DSAAR architecture is to provide the fast development and prototyping of multi-robot systems, as well as, a real time HRI awareness.

1.3 Dissertation Outline

This dissertation is organised as follows:

Chapter 2 reviews the state-of-the-art on human-robot interaction, specifically on teleoperation, and on software architectures.

Chapter 3 presents the DSAAR architecture.

Chapter 4 proposes an approach to the teleoperation problem using the DSAAR architecture

Chapter 5 presents an instantiation of the DSAAR architecture to the problem of Human-Robot interaction in all-terrain environments.

Chapter 6 presents the field trials done with the physical robot in order to test this presented approach.

Chapter 7 aggregates a set of conclusions, main contributions of this dissertation, and further research opportunities on the subject.

1.4 Further Readings

Some of the concepts covered in this dissertation have been published in [Santana et al., 2006b], [Santos et al., 2007], [Santana et al., 2008b] and [Santos et al., 2008]. The developed work is being applied to control an all-terrain robot in the context of a Multi-Robot System for landmine detection,which is being developed by the SME Portuguese company IntRoSys, S.A.. The interested reader may refer to other work developed by this and other team members that compose the referred project [Santana et al., 2006a], [Cândido et al., 2008] and [Santana et al., 2008a].

Chapter 2

State-of-the-Art

This chapter surveys the state-of-the-art on human-robot interaction awareness, on software architectures and reusable software.

Software architectures and reusable software provide insight into the main problems while developing generic software which is meant to be used in different robots. Whereas human-robot interaction awareness provides a closer look to the problems that humans have while interacting with the physical robots.

2.1 Human-Robot Interaction Awareness

Regarding Human-Robot Interaction (HRI) awareness semantics there is a lack of standards on the terms used to describe the several types of awareness, as it is described in the work of Drury et al. [Drury et al., 2003]. In that work Drury et al. expose several contradictions in the definition of the terms used, e.g. awareness is defined as "an understanding of the activities of others, which provides a context for your own activities." by Dourish et al. [Dourish and Belloti,] and as "given two participants p_1 and p_2 who are collaborating via a synchronous collaborative application, awareness is the understanding that p_1 has of the identity and activities of p_2." by Drury [Drury, 2001]. In their work Drury et al. [Drury et al., 2003] purpose a framework in order to classify HRI awareness regarding the n humans and m robots scenario.

HRI awareness (general case): Given n humans and m robots working together on a synchronous task, HRI awareness consists of five components:

Human-Robot the understanding that the humans have of the locations, identities, activities, status and surroundings of the robots. Further, the understanding of the certainty with which humans know the aforementioned information.

Human-Human the understanding that the humans have of the locations, identities and activities of their fellow human collaborators.

Robot-Human the robots' knowledge of the humans' commands needed to direct activities and any human delineated constraints that may require command non-compliance or a modified course of action.

Robot-Robot the knowledge that the robots have of the commands given to them, if any, by other robots, the tactical plans of the other robots, and the robot-to-robot coordination necessary to dynamically reallocate tasks among robots if necessary.

Humans' overall mission awareness the humans' understanding of the overall goals of the joint human-robot activities and the measurement of the moment-by-moment progress obtained against the goals. In recent work [Drury et al., 2007] this concept is simply called ***overall mission awareness*.**

2.1.1 Human-Robot Interaction Awareness

In the field of human-robot interaction awareness the teleoperation scenario will be studied. In teleoperation, as in many system involving humans and machines, there is a clear need to map concepts between different representations. When considering a system with heterogeneous entities (e.g. robots and humans), a shared ontology becomes necessary in order to provide a common language. Thus allowing the ability for the different entities to understand each other.

In teleoperation, if the operator has not a real perception of the robots' internal state (e.g. posture or position), the operator can not decide where to lead the robot. The operator must also

be able to perform manoeuvres in real time fashion, since any delay may render that manoeuvre useless, or even dangerous. This limits the type and amount of information mapping. From the humans point of view there are several ways of providing perceptual clues (e.g. maps, video, etc.). In some work the robots' posture has been included in the video feed [Wang et al., 2004], also the ability of providing a camera with autonomous control has proved to improve the operators' awareness [Joeseph Manojlovich and Gennari, 2003]. The joint use of video and maps has been proved useful by the work of Nielsen and Goodrich [Nielsen and Goodrich, 2006], where several combinations have been tested.

Taking these factors into account Dury et al. [Drury et al., 2007] suggest a series of categories that should be taken into account.

Location Awareness was defined as map-based concept: orientation with respect to landmarks. If an operator was unsure of his or her location, this constituted negative location awareness.

Activity Awareness pertained to an understanding of the progress the robot was making toward s completing its mission, and was especially pertinent in cases where the robot was working autonomously. The human needed to know what the robot was doing at least so that he or she understood its part of the mission.

Surroundings Awareness pertained to obstacle avoidance: someone could be quite aware of where the robot was on a map but still run into obstacles. An operator was credited with having positive surroundings awareness if he or she knew that they would hit an obstacle if they continued along their current path.

Status Awareness pertained to understanding the health (e.g. battery level) and mode of the robot, plus what the robot was capable of doing in that mode, at any given moment.

2.2 Software Architectures and Reusable Software

This section presents some of the most relevant software architectures, namely the MARIE architecture, NASA's CLARAty architecture, NASA's mobile agents architecture and Retsina architecture applied to mobile robots.

2.2.1 MARIE Architecture

MARIE [Côté et al., 2006] is a middleware framework oriented toward s developing and integrating new and existing software for robotic systems. MARIE is designed according to three main software requirements: (1) Reuse available solutions, (2) Support multiple sets of concepts and abstractions, (3) Support a wide range of communication mechanisms and robotics standards. In order to accomplish this, MARIE is based on three design choices:

Component Mediation Approach uses a Mediator Interoperability Layer (MIL), this creates a centralized control unit (the Mediator) that interacts with each colleague (component) independently, and coordinates global interactions between colleagues to realize the desired system. In the MIL, components can interact together using a common language. Using this approach each component can have its own communication protocol and mechanism as long as the MIL supports it. This is done, in order to, exploit the diversity of communication protocols and overcome the absence of standards in robotic software systems. It also promotes loose coupling between components by replacing a many-to-may with a one-to-many interaction model.

Layered Architecture defines different levels of abstraction into the global middleware framework. Three abstraction layers are used to reduce the knowledge, expertise and time required to use the overall system. At the lower level, the *Core Layer* consists of tools for communication, data handling, distribute computing and low-level operating system functions. The *Component Layer* specifies and implements useful frameworks to add components and to support domain-specific concepts. The *Application Layer* contains

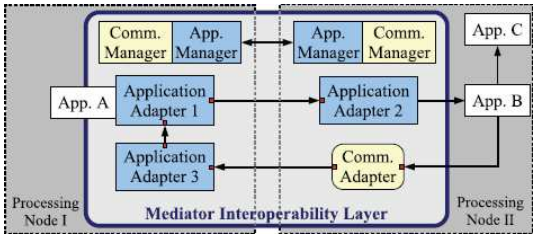


Figure 2.1: MARIE's application design framework [Côté et al., 2006]

useful tools to build and manage integrated applications using available components, to craft robotic systems.

Communication Protocol Abstraction eases components interoperability and reusability by avoiding fixing the communication protocol during the component design phase. This choice should be made as late as possible, depending of which components need to be interconnected together (e.g. at the integration phase or even at runtime). Therefore, a communication abstraction framework, called port, is provided for communication protocols and component's interconnections.

A generic framework of MARIE's architecture can be seen in figure 2.1, there one can see how different applications interact through the MIL. It also shows how when an application which uses a communication protocol which is not supported by MARIE, is integrated through the Communication Adapter.

Discussion This architecture presents an interesting approach to this problem, since it allows for the use of parallel computing and the use of several communication methods for the modules interactions. This approach also supports the addition of single communication interfaces, that work as a translator for a module to be integrated with MARIE's architecture. However this architecture creates a centralized module, the MIL, that can cause the collapse of all the communications, instead of a single communication drop out. Finally this architecture does not consider high level HRI interactions, focusing mainly on the robot's control frame.

2.2.2 Claraty Architecture

CLARAty [Nesnas et al., 2003], [Nesnas et al., 2006] is a domain-specific robotic architecture designed with four main objectives: (i) to reduce the need to develop custom robotic infrastructure for every research effort, (ii) to simplify the integration of new technologies onto existing systems, (iii) to tightly couple declarative and procedural-based algorithms, and (iv) to operate a number of heterogeneous rovers with different physical capabilities and hardware architectures.

The *CLARAty* architecture has two distinct layers: the Functional Layer and the Decision Layer. The Functional Layer uses an object-oriented system decomposition and employs a number of known design patterns [Gamma et al., 1995] to achieve reusable and extendible components. These components define an interface and provide basic system functionality that can be adapted to a variety of real or simulated robots. It provides both low- and mid-level autonomy capabilities. The Decision Layer couples the planning and execution system It globally reasons about the intended goals, system resources, and state of the system and its environment. The Decision Layer uses a declarative-based model while the Functional Layer uses a procedural-based model. Since the Functional Layer for predicted resource usage, state updates, and model information.

The Decision Layer accesses the Functional Layer at various levels of granularity (figure 2.2). The architecture allows for overlap in the functionality of both layers. This intentional overlap allows users to elaborate the declarative model to lower levels of granularity. However it also allows the Functional Layer to build higher level abstractions (e.g. navigator) that provide mid-level autonomy capabilities. In the latter case, the Decision Layer serves as a monitor to the execution of the Functional Layer behaviour, which can be interrupted and pre-empted depending on mission priorities and constraints.

The Functional Layer The Functional Layer includes a number of generic frameworks centred on various robotic-related disciplines. The Functional Layer provides the system's low- and mid-level autonomy capabilities. The Functional Layer has four main features:

It provides a system level decomposition with various levels of abstractions.

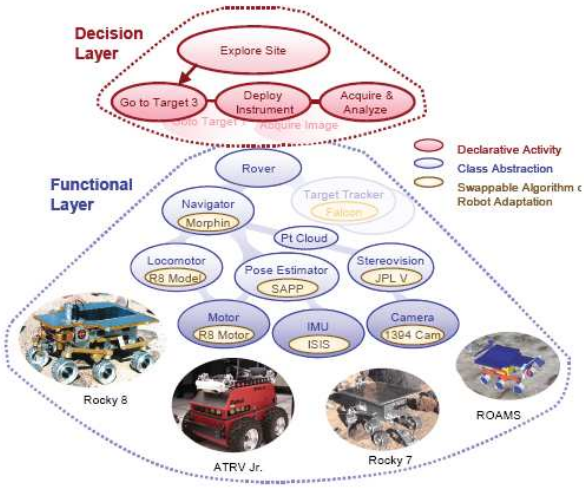


Figure 2.2: The CLARAty Architecture [Nesnas et al., 2006]

The Functional Layer separates algorithmic capabilities from system capabilities.

It separates the behavioural definitions and interactions of the system from the implementation.

the Functional Layer provides flexible runtime models. The runtime model is part of the abstraction model, of which, one part is associated with the generic functionality and the other with the adaptation.

The Decision Layer The Decision Layer is a global engine that reasons about system resources and mission constraints. It includes general planners, executives schedulers, activity databases, and rover and planner specific heuristics. The Decision Layer plans, schedules, and executes activity plans. It also monitors the execution modifying the sequence of activities dynamically when necessary. The goal of a generic Decision Layer is to have a unified representation of activities and interfaces.

The Decision Layer interacts with the Functional Layer using a client-server model. The Decision Layer queries the Functional Layer about availability of the system resources in order to predict the resource usage of a given operation. The Decision Layer sends commands to the Functional Layer at various levels of granularity. The Decision Layer can utilize encapsulated Functional Layer capabilities with relatively high-levels commands, or access lower-level

capabilities and combine them in ways not provided by the Functional Layer.

Discussion Claraty is a domain applied architecture, space exploration, and so it does not take into account issues like real-time human-robot interaction (e.g. teleoperation). Also this architecture forces a very strict design structure, so it narrows the developers design choice (i.e. it difficults the implementation of a biological approach). This approach is designed for an autonomous robot, so it does not take into account any interactions with other entities within the system.

2.2.3 Retsina Architecture applied to mobile robots

This approach integrates a Multi-Agent Systems architecture, RETSINA [Sycara et al., 2003], in a multi-robot scenario [Nourbakhsh et al., 2005]. RETSINA categorizes agents into four types based on their function: (1) *Interface agents*, which facilitate user interaction; (2) *Task agents*, that seek to accomplish user goals; (3) *Middle agents*, which provide infrastructure for dynamic runtime discovery of robots and agents that can perform a given task; (4) *(*Information agents), that can access external information sources, such as disaster site blueprints.

When extending the Multi-agent System (MAS) infrastructure to teams that include physical agents two challenges were identified. First, each robot needed to have *social awareness*, the robot was required to have knowledge of the MAS infrastructure and how to use that infrastructure in order to function as a team member. Although robots may vary in morphology and capability, they all have a reasoning layer consistent with task agents in the RETSINA system. To this effect, a robot architecture that transforms a physical robot into a robot agent was developed. The robot agent architecture (see figure 2.3) extends the three-tier architecture, with each higher layer enforcing a functional abstraction on the layer below and each lower layer enforcing a functional abstraction on the layer below and each lower layer decreasing the look-ahead horizon while increasing detail. The agent layer contains high-level reasoning and RETSINA communication modules, including the necessary social awareness for interaction. The executive layer is responsible for plan execution and monitoring deviations. The control

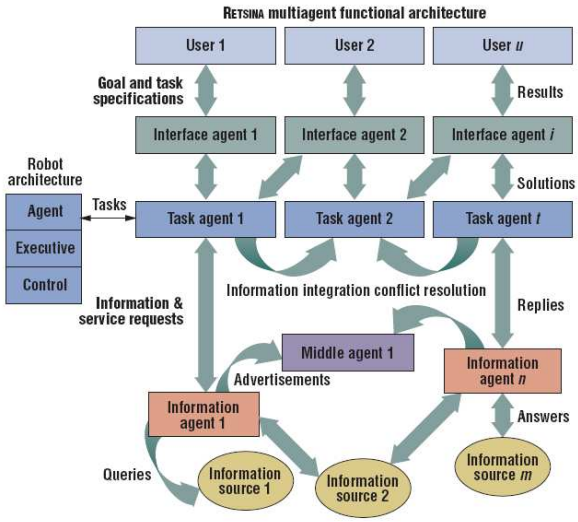


Figure 2.3: The robot agent architecture enables robot integration in the RETSINA multi-agent system functional architecture by transforming physical robots into embodied task agents. [Nourbakhsh et al., 2005]

layer encapsulates the physical robot behaviour implementation. This representation combines the functional abstraction of standard three tier architectures with a high-level semantic abstraction that transforms the robot into a robot agent suitable for conventional software agent coordination and cooperation.

A second challenge in extending MAS to physical agents arises from communication challenges. Physical agents share the same high-level communication requirements as software agents but must also communicate information about their state and the environment state. This low-level communication has high frequency and low latency requirements, yet because the robots are mobile and must communicate wirelessly, the available bandwidth is significantly lower for multi robot systems than multi software-agent systems. An MAS usually has one standard for communication between all agents: an agent communication language. ACLs incur overhead that makes them impractical or infeasible for the transfer of low-level data such as video, audio, sensory, or telemetry data. In response to these opposing needs for high- and low-level communication, a two-tiered communication hierarchy was developed, allowing additional, more efficient lines for low-level communication.

Discussion This approach is focused on HRI, where the MAS is used to provide an interaction pool for the several 'agents' present on the network. However this approach considers the load occupied by ACLs due to telemetry data and teleoperation commands on the network excessive, thus placing them on a separate channel. Meaning that only high-level interactions are contemplated with this approach, such as plan sharing.

2.2.4 NASA's Mobile Agents Architecture

NASA Ames' Mobile Agents Architecture (MAA) [Sierhuis et al., 2005] is a distributed agent-based architecture, which integrates diverse mobile entities in a wide-area wireless system for lunar and planetary surface operations. The MAA is a distributed agent architecture developed in Brahms. Brahms is a multi-agent rule-based BDI language developed at NYNEX S&T, IRL and NASA Ames. The Brahms environment consists of a language definition and compiler, a graphical development environment (the Composer) and a Brahms virtual machine (the BVM), running on top of the Java virtual machine, to load and execute Brahms agents.

Brahms agents are rule-based BDI-like agents. However, Brahms does not use a goal-directed approach, but rather an approach we refer to as activity-based. Brahms agents are both deliberative and reactive. Each Brahms agent has a separate subsumption-based inference engine [Brooks, 1991]. Brahms agents execute multiple activities at different levels at the same time. At each belief-event change (creation or changing of beliefs), situated-action rules (i.e. workframes) and production rules (thoughtframes), at every active activity-level, are evaluated.

Figure 2.4 shows the Brahms agents of and the external systems integrated with the MAA. There are three types of mobile entities that are currently being supported, the Extra-vehicular activity (EVA) astronaut, the EVA Robotic Assistant (ERA) robot and the HabCom crew member in the habitat. Each entity has a support computer with a Brahms model running within a local Brahms VM: the Space Suit Brahms VM (one for each astronaut), the ERA Brahms VM and the HabCom Brahms VM. Each Brahms model has a number of agents that support the mobile entity. Each agent can communicate with remote agents via KAoS through the MEX wireless network.

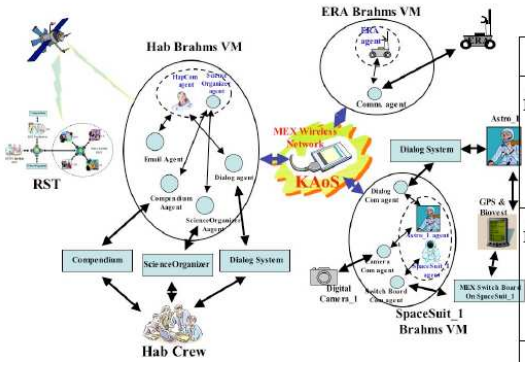


Figure 2.4: NASA's Mobile Agents Architecture: Mobile Agents entities and wireless network [Sierhuis et al., 2005]

Discussion This architecture is designed for high-level interactions, in order to coordinate off world planetary explorations. The Brahms agents are responsible for the interactions between the several entities in the system, however due to the intention of this architecture it does not take into account real time interactions. Also interactions are very specified, which reduces the ability of entities interaction with each other (e.g. the EVA astronaut can only interact with the HabCom through the EVA robot).

2.3 Pitfalls of the Current State-of-the-Art

As demonstrated in the state-of-the-art section, the current software architectures fail in guaranteeing both fast prototyping, multi-robot platform support and human-robot interaction. Solutions either focus on multi-platform compliance and fast prototyping or on Human-Robot Interaction (HRI). Most architectures that focus on HRI tend to ignore the teleoperation scenario or simply choose to make ad-hoc solutions. However the state of the art on HRI has proved that most applications on this field still rely on teleoperation. Hence, there is a lack of an architecture that groups both multi-robot platform support and HRI, also there lacks an approach that considers the teleoperation scenario as an integral part of HRI and not an external function of the system.

Chapter 3

DSAAR Architecture

In this chapter the Distributed Software Architecture for Autonomous Robots(DSAAR) will be presented. Afterwards a general overview of the architecture will be presented. Finally the the physical entity (robot) module will be presented. For a description on the enabling technologies please refer to appendix A.

3.1 DSAAR: an architecture for Human-Robot teams

The coordination problem between *human teams* and *robot teams* has been one of the main concerns in Urban Search And Rescue (USAR), due to communication difficulties and the heterogeneity of the several agents involved in this process, as it shown by [Burke and Murphy, 2004], [Nourbakhsh et al., 2005].

The Distributed Software Architecture for Autonomous Robots (DSAAR) architecture , is presented as a solution to this problem. DSAAR is implemented under the multi-agent paradigm and intends to be generic enough to handle any control model. Although this architecture has been designed to support robot teams, this could not yet be tested as the preparation of the robotic team is still an ongoing activity.

DSAAR architectures' social awareness is supported by the multi-agent Java DEvelopment Framework (JADE) software platform, which allows interoperability between the robot teams

and the human teams. JADE is a Multi Agent System platform, which complies with the FIPA specifications. By being compliant with FIPA standard, JADEs' communication language is based in Agent Communication Language (ACL).

DSAAR has been designed so that there is no privileged point to look to and out of the system. For instance, a robot can be wired to a camera in order to synchronize a certain behaviour (e.g. something blocking the robots' movements), yet the camera can also be interacting with a human user who is using it for teleoperation. The several units that compose the system have been called *physical entities*. A *physical entity* refers to any physical device that exhibits some autonomous behaviour. For instance, a *physical entity* can be either a human interacting through a laptop, a robot, or even any other device that could be seen and endowed with some autonomy. An example is a video camera which can be seen as an autonomous agent requesting services of the robotic platform for displacing itself. Each entity is composed by *interaction components* and *process components*, as it is shown in figure 3.1. In this figure it can be seen two different kinds of components, some are native to the machine (pink ones) and others are in a java virtual machine in the JADE platform (purple). These components can be from data feeders up to agents, depending on their complexity level. Agents can either be found at the *interaction components* or *process components* layers, some implemented in JADE others in a native language.

Process components refer to the those responsible for the direct control of a physical entity's resources. These components are for exclusive use of the entity and are not used in any interaction mechanism.

Interaction components refer directly to the Human-Robot teams problem. It describes all the possible interactions between the several entities requiring a computational backbone. This goes to say that models which do not need direct communication, are not, and do not need, to be represented.

Interactions are not strict to social ones. Physical and hormonal metaphors can also be used. Interactions can even be tighter, such as wired analog force feedback. Thus, the system can not be perceived as strictly logic.

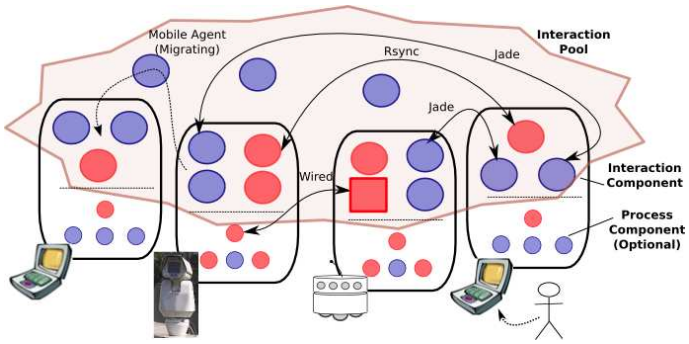


Figure 3.1: DSAAR Detailed Overview

To transfer large amounts of data, such as the mission full log, tools for incremental synchronisation, such as *rsync* or *ftp*, of the files located in the robot to a remote logging machine are being employed. This way the data flow in the network is minimised. Since it allows direct communication between entities and removes any extra unneeded processing, a direct video link over IP for video streaming is preferred to any other option. The transfer of agents is also considered as a kind of interaction. It allows migrations of agents for the necessary entities, enabling an increased efficiency in resource management. This can be seen in figure 3.1.

3.1.1 Physical Constraints

Sometimes physical entities although performing decoupled actions have some physical constraints that do not allow to break them apart. For instance, let us imagine the case of a robot and a camera mounted on top of the former. Although the tasks performed by both can be uncorrelated, the power supply and controlling board are the same. figure 3.2 depicts how the robot supports the cameras' interaction components, resembling the host-parasite relation. The camera is called parasite since it adds a computational overhead into the robots' system. By clearly specifying how physical resources and software components are related, it is then easier to assess the system's robustness. For instance, the failure of the robot's on-board computer, in this case, will affect the camera as well. Therefore, a causal network representing the system makes explicit how all components, software and hardware, and inter-related. This is of special relevance in a domain where devices tend to fail often due to the harsh nature of environments,

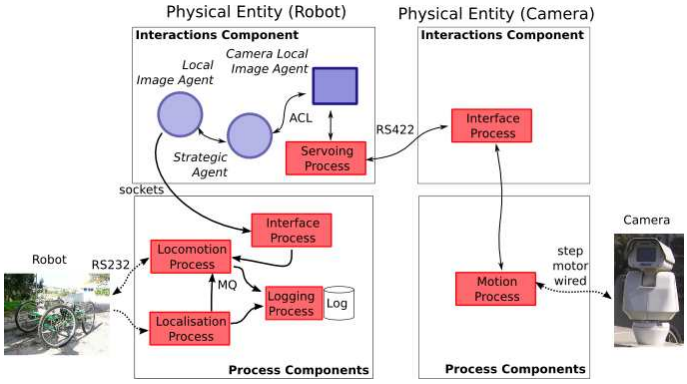


Figure 3.2: Physical Constraints

which require a careful reliability analyses in order to avoid single point of failures.

3.1.2 Multi-User Support

Considering previous work in the Human-Robot teams issues, the advised ratio for robot control is of 2:1 [Burke and Murphy, 2004]. Another recommendation is if a third user is added to the equation, this user should focus on a specific issue. To this effect some tools were developed. Let us considered the example given in figure 3.3. In this figure one can see the *operator 1* teleoperating the robot with help of a tool (represented as a square in the operators entities); and another one controlling the camera using the same tool, yet in a different mode and in a different laptop. Thus, each operator is allowed to focus on a specific task. Then, a third operator, like a landmine expert that is only focused in finding mines, can log into the system via a remote machine for data analyses without disturbing the team-mates.

3.1.3 Robots' Computational Model

Figure 3.4 illustrates the two main software layers on-board each robot. The *interaction layer* refers to the social ability and configurability of the system, whereas the *process layer* controls everything related to robot navigation and hardware resources management.

Hence, the merge of these two systems endows the designer with a complete development system to implement high performing robots with high scalability.

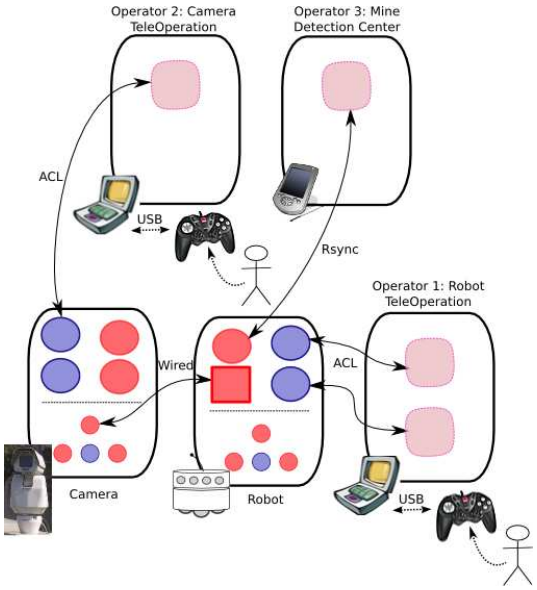


Figure 3.3: multi-user System

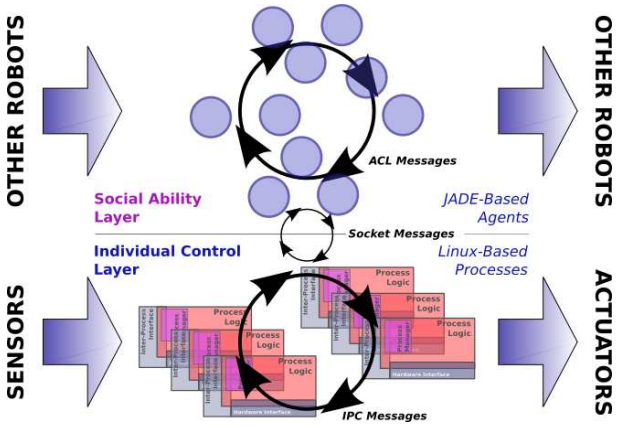


Figure 3.4: On-Board Computational Model.

3.1.4 Social Layer

The *social layer* is based on the Jade multi-agent platform, which handles inter-agent communication in a transparent way, whether agents are in the same or in different machines. Communication among agents follows a well specified ontology. DSAAR defines for this layer the following roles:

- Implementation of collective behaviours, i.e. behaviours requiring interactions between different robots;
- Interface between robot control software and middle layer agents;
- Fast prototyping of robot control behaviours; notice that Java may induce considerable latency in cycle times and so control behaviours in Jade have to be considered in a very careful and temporary way.

As previously referred, agents can be located in any machine, can even be moved from one machine to another. This allows a fast reconfiguration and adaptability of the system. For instance, agents can be moved to the machine as the behavioural model requires. This also allows to have agents distributed as computational resources become available, distributing the load. In addition, agents can be launched in different machines (e.g. in a PDA or in a remote internet connection) and connected to a specific robot on the fly.

Hence, Jade makes DSAAR a highly versatile architecture. However, Java agents can not implement robot control algorithms. This is delegated to the *process layer*. In fact, a sockets-based API (Application Programming Interface) has been developed to allow the interaction between Jade agents and Linux processes, which are running at the *process layer*. This API is responsible for translating Java objects into C structures and vice-versa.

3.1.5 Process Layer

At the *process layer* the system developer is endowed with a set of APIs and design guidelines to implement linux-based processes. With this API, the developer can implement processes

with one or more of the following roles, depending on the control model:

- **Sensor Process.** A process responsible for fetching sensory data, pre-process it, and send the results to other process(es) via an IPC (Inter-Process Communication) mechanism (e.g. Shared Memory, Message Queues). In other words, the output message of such a process is a *percept*, which can be piled up in a Message Queue or stored in a shared memory associated to a time stamp.

- **Actuator Process.** A process responsible for acting in actuators according to incoming IPC messages, produced by other processes;

- **Behaviour Process.** A process responsible for achieving a particular goal of the robot according to incoming data from other processes (e.g. sensory or other *behaviours*) and forwarding data to other processes (e.g. actuators). This is responsible for the implementation of the robot's control logic. In particular a behaviour process, can be a reactive computational unit, a deliberative one, or whatever. The granularity of the semantics associated to each process is left to the control model and no assumptions are made at the computational model level. Nevertheless, as it is possible to see, the provided computational semantics is already driven by well established control model concepts, like behaviours and precepts.

- **Interface Process.** A process responsible for interfacing different systems, like Jade agents and Linux-Based processes. These processes are really important to integrate legacy systems, ad-hoc developments, and even software implemented in different hardware (e.g. interfacing a two PCs through a serial port), which may be useful to fulfil real time constraints.

Figure 3.5 illustrates the main set of modules composing a process. The following sections describe each of these blocks in more detail.

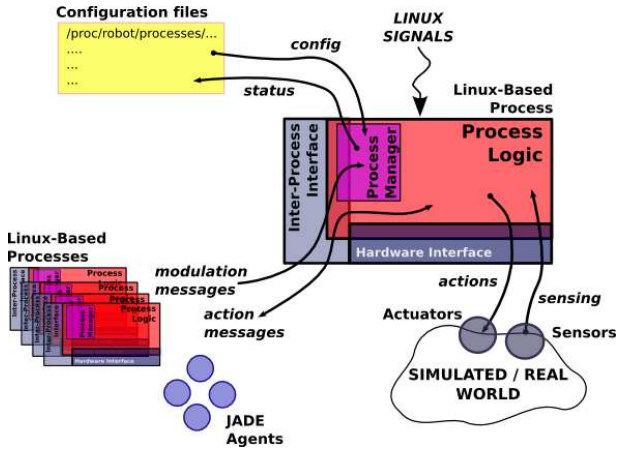


Figure 3.5: Modules composing a process.

Process Logic

This module is responsible for implementing all the logic corresponding to the process behaviour. That is to say that the process logic may a sensory-motor module, a deliberative module, a behaviour coordination node, a sensory input signal processing algorithm, etc.

Therefore, the process logic semantic definition is to be instantiated as specified by the control model, whereas the computational model is concerned about the way modules interact among themselves. For instance, the process logic which is mainly concerned with control and navigation aspects will have to interact with data provided by hardware, configuration files, and other processes. These interfaces are well defined, supported by an API, and overviewed below. The internal structure of the process logic is not covered in this paper.

Inter-Process Interface

Processes interact among themselves via Inter-Process Communication (IPC) Linux mechanisms. The Inter-Process Interface is responsible for abstracting all possible mechanisms under the same API. The IPC clients (message receivers) and servers (message senders) are specified in a configuration file. Hence, processes do not need to know who are the clients and servers at compilation time, they only have to know the communication semantics, i.e. message structures. Then, according to the mission at hand the configuration file is changed and processes redirect their outputs and inputs accordingly. Moreover, since messages are well defined one

can change the processes generating messages as long as they respect the common semantics.

Currently message semantics is specified by C structures. In the future it is planned to full specify messages as an OWL (Web Ontology Language) [OWL Website, 2006] ontology, which allows to use tools like Protégé OWL plug-in [Protege OWL Website, 2006] for their proper manipulation. As it will be shown below, in some IPC mechanisms this would be easy to implement, such as in those string based. However, others require a well C structure specification, which would require a way of translating OWL specifications into C structures and respective compilation.

The following mechanisms are considered in DSAAR:

- **Shared Memory.** Of special interest for large data blocks transfer. The absence of a built-in ”event” generating mechanism, requires from processes to poll for new messages. Linux semaphore mechanisms can be employed to ease this interaction;

- **Message Queues.** Messages are described as C structures, discarding the need for string parsings, as in the previous and subsequent cases. Messages trigger events and so processes do not need further synchronisation or polling mechanisms. It is possible to send messages to different clients by specifying the type of the message. However, there is a limited number of messages that can be flowing simultaneously.

- **FIFO.** Similar to Linux pipes, i.e. special files where processes can write and read messages. This mechanism also requires parsing and special care with messages with different sizes.

- **Sockets.** Sockets are of special use to interact with Java processes. Drawbacks related to string based communication.

Currently, a generic message is defined as a C structure which can then be directly sent to a Message Queue or translated into a string and then sent to a FIFO, a Socket or whatever. In addition, a set of functions to manipulate the C structure are provided in order to build up a complete API. Process developers can then make use of this API to send/receive messages

to/from other processes via any IPC mechanism in a transparent way. As explained earlier the goal is to have OWL message specifications, which will try to automate this process; still, some studies have to be carried out in terms of computational overhead.

All messages have a time stamp associated, which is an absolute time in micro-seconds, corresponding to the event associated to the message (e.g. the time of a sensor acquisition process), and not to the sending or receiving time of the message. This allows a proper manipulation of incoming messages, which is of special interest for sensor fusion.

The goal of the architecture is to allow the developer to focus on the process logic. To do that, the designer only has to take into account that the process will require a set of inputs and outputs, through which messages flow. Thus, both the sender and the receiver must only know how messages are built.

Let us now detail the currently implemented messages API. For each message flowing in the system that has to be two corresponding $xxx_{msg}.h$ and $xxx_{msg}.c$ files. These files publish the following two major entities:

- A C structure whose fields specify the message body (e.g. heading), the time-stamp of the message, and in some cases the identification of the target client. This structure is to be filled in by the sender and read by the receiver.

- Two functions to convert a string into a structure and vice-versa.

Message senders and receivers only have to include these files so as to abstract the messages' transmission medium. The medium through which a message is sent is not set in the referred files nor in the process logic. In fact the transmission medium (e.g. message queues) is set along the message recipient in a configuration file (see above explanations about this issue).

Figure 3.6 illustrates the internal structure of the implemented API. As it is possible to depict from the figure, the structural API (i.e. $ipc.c$) handles messages without being aware of their structure (i.e. semantics). This API handles $void*$ messages. Then, the message customised API (i.e. $xxx_{msg}.c$) provides the process with message semantics via the entities presented

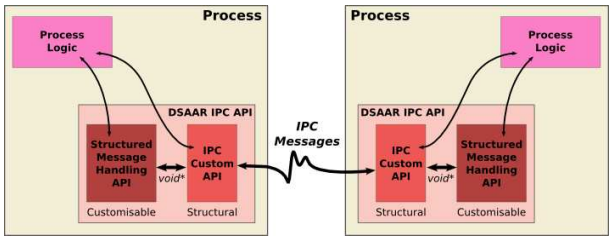


Figure 3.6: DSAAR message API structure.

above. The control logic can interact with both. Preferentially, if a customised message API exists, then it should be used.

The structural API (i.e. $ipc.c$) is mainly composed of functions to manage IPC shared resources (e.g. creation) and to send and receive messages. These latter features depend on the IPC type (e.g. shared memory, message queue), the client identification, the IPC resource Key (i.e. unique ID), and the message itself. If the type is message queue then the message is the structure specifying it; otherwise the message is a string describing it, which has been translated by a customised API (i.e. $xxx_{msg}.c$).

Process Manager

The *process manager* is thread internal to each process. The process manager is responsible for making the process compliant with DSAAR. Other processes can send dedicated IPC messages to the process manager (i.e. $modulate$ type messages) to:

- **Allocate memory.** This message asks the process manager to allocate and register all required IPC resources.

- **Start/Stop/Resume the process.** Self explanatory. This is different from sending a Linux signal to perform these operations. In this case the process logic can be stopped while other activities still running, like logging.

- **Change cycle time.** Changing cycle time is different from changing priority. When one changes priorities via the $renice$ command the process runs slowly than others. Cycle

time refers to the time spent without doing nothing before starting a new cycle, independent of the processing ”speed”.

- **Inhibit IPC input messages.** Inhibition of messages can be used for both debugging and as a part of the behavioural model.

- **Suppress IPC output messages.** Suppression of messages can be used for both debugging and as a part of the behavioural model.

Thus, the process manager, among other functions, it is responsible for maintaining an internal structure with the info received by the $modulate$ messages. Then, the process logic shall comply with these modulation signals, like sleeping after a cycle so as to cope with the cycle time requirements.

This modulation mechanism is generic enough to allow:

- Synchronisation of several processes according to a general intent;

- Modulation of the activity of each process according to some events (either exterioceptive or proprioceptive).

Notice that $modulate$ type messages are received and sent through the IPC DSAAR API, as any other message; hence, a process output can easily be configured to modulate other processes, which is a very important feature. It is common in robotic architectures to have processes modulated according to the current world and internal state. For instance, computational load of certain perceptual processes can be reduced according to some heuristics about the environment.

The process manager also provides the process logic with some API calls in order to read the configuration files, which describe who are the IPC clients and servers the process has to interact with.

General Procedure

Basically, the current procedure of developing, configuring, and launching a network of processes is roughly as follows:

1. The developer implements all the processes and corresponding IPC specific code (e.g. message structures);
2. The $/var/robot/$ tree is updated to handle these new type of processes;
3. Interactions among processes (e.g. who receives and sends messages) are specified in an $interactions$ file according to the robot's goal and behavioural model;
4. Specific process configurations are set in $config$ files according to the robot's goal and behavioural model;
5. A script starts all processes according to $interactions$ file, i.e. by taking into account their priorities;
6. The $processmanager$ reads all configuration and interaction files so as to allow the process logic to operate conveniently;
7. The manager also finds the nice and PID info and update the corresponding files;
8. A central process sends a $modulate$ message to all processes requesting them to allocate all IPC resources;
9. The same process asks then all processes to start operations;
10. Processes interact among themselves via IPC messaging as their control logic requires;
11. Processes generate status information which is set in $status$ and log files;
12. An external process can monitor the log files and signal (e.g. using $renice$ command) processes by inspecting their PID in the corresponding files.

In short, the $/var/robot/$ file tree endows the engineer and all processes/agents within the system with a common way of configuring and monitoring the system. In fact, the middle layer also configures and monitor lower layer processes via this mechanism.

Chapter 4

Human-Robot Interactions under DSAAR

In this chapter it is presented the approach taken regarding to HRI. Here a more detailed explanation is given on how each entity interacts with each other and also with itself. In order to test this approach two tools were designed: the teleoperation tool and the 2D map, whose functionalities will be described in chapter 5. In this chapter the way in which these tools work is explained. Exposing the form in which the human (e.g. operator) interacts with the robot and all the necessary steps for it to happen.

Contrary to other approaches [Nourbakhsh et al., 2005], this thesis proposes an approach where teleoperation commands and telemetry messages have no dedicated channel. Having all teleoperation related information in the main MAS network, lead to an approach where all this information is shared on the network. In this multi-agent system network each entity is considered a knowledge source, fostering a shared representation of each entities beliefs, desires and intentions (BDIs), resembling the distributed blackboard architecture [Nolle et al., 2001]. This approach enables a shared representation of all teleoperation orders and telemetry data through the system.

Having all teleoperation information flowing through the MAS network causes a higher load on the network, forcing the information exchange to be optimised. A study on this subject is also shown in chapter 6, as to see if this shared reality approach promotes an improved operators situation awareness, as well as, of other entities added to the system.

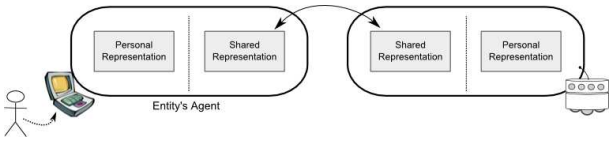


Figure 4.1: Generic System Overview

Section 4.1 presents the teleoperation model, exposing the different components and entities present in the system, design approaches and assumptions. Then, the computational model, where the different interactions semantics used by the agents during their interaction are explained, is presented in section 4.2.

4.1 System Overview

Considering that each entity has its own perception on the reality, a shared representation must be imposed, in order to allow a common way of interaction, i.e. an ontological commitment [Grosz and Kraus, 1999], [Cohen and Levesque, 1991], [Grosz and Kraus, 1996] (see figure 4.1).

This dual reality representation is implemented as part of the architecture described in the previous chapter. These two reality representations are needed in a similar way as humans use it, when humans interact with each other they use language, while interacting with themselves hormones and other stimuli may be used. This allows the entity to have real-time responses concerning itself and lesser time dependent responses while interacting with others. The proposed system was designed based on a network inhabited by cooperative agents, i.e. all contribute for the global good. This assumption allows for a simplification on the negotiation mechanisms, since there is no need to have elaborated trust mechanisms or partner selection mechanisms. It is also assumed that when an entity subscribes a service it has all the required abilities to accomplish that service.

Considering the teleoperation scenario, there is a clear dependence of time. This means that every information to be shared must be time tagged. This works similarly to the concept of blackboard, meaning that each knowledge source, entity, updates its perception of reality within

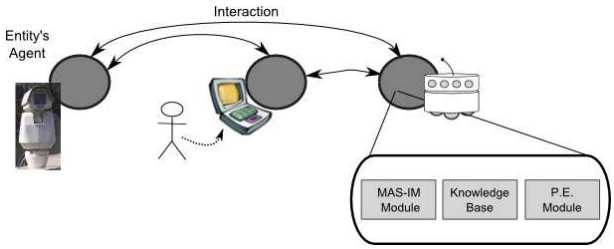


Figure 4.2: System Overview

the community in a time dependent fashion. The shared representation update is conditional on the type of information. This idea will described later in this section and in detail on section 4.2.1.

Each robot, human, or any autonomous mechanism (e.g. camera) is represented by an agent. The entities' agent makes the bridge between the entities' self perception and the shared reality network. Figure 4.2 depicts an instantiated systems' overview, where several *entities* can be seen interacting amongst themselves through their associated agents. An Agent Communication Language (ACL) method is used to support agents interactions. Each *entity* supporting agent is composed by three main components: (1) the *Multi-Agent System Interaction Mechanism* (MAS-IM), (2) the *Knowledge Base* and (3) the *Physical Entity interface* (P.E. interface).

The MAS-IM is the component that enables the agent to interact with other agents in the multi-agent community. In order to exploit its middleware services (e.g. yellow pages service). This module is built over the JADE platform. The Knowledge Base (KB) aggregates the knowledge of the entity, and partial shared knowledge about the other entities which the agent is interacting with. The physical entity module abstracts the physical entity, i.e. providing an access to its control system and telemetry data in the robots' case.

4.2 Interaction Semantics

When different entities with different perceptions of the reality interact, it is necessary to provide not only a common language, but also, strict interaction rules and semantics. Like in any conversation, it is important to not only know the language, as well as, knowing the rules in

which the actors are engaging. For instance, it is not considered polite to interrupt conversations or cross talking. Surely it is important to know how to interact with others, however it is also important to know how to interact with ourselves (e.g. what stimuli to give our body as to accomplish a desired action).

Considering these concepts, interactions were tackled in two separate ways. One relates to the way the agent interacts with other agents (Inter-Agent Interactions); the other refers to the way the agent interacts with the physical entity's control structure (Intra-Agent Interactions). Inter-Agent interactions have a common structure to all who share the network, therefore are reusable and stereotyped. Intra-Agent interactions, on the other hand, are tightly coupled with the entity in question, e.g. the operator interacts with its' agent through a joystick or GUI. This implies that the interface responsible for the latter kind of interactions (Physical Entity Module) is not reusable, and it is mostly native to the entity.

4.2.1 Inter-Agent Interactions

As mentioned before, this type of interactions are common to all entities. In order to create a common language to every entity in the system it was necessary to create a supporting ontology. The next logical step was to define the conditions and rules for these interactions, where two approaches were taken. First, a *publish/subscribe* method for the registration and subscription of the services and information each entity provides was used. This means that, if an operator wishes to teleoperate a robot, it has first to search for a robot that has published a teleoperation service and subscribe it. This method is done taking advantage of JADEs' yellow page service. Secondly, similar in spirit to the distributed blackboard architecture, the operator has to do is to update its desires of movement in the remote robot's knowledge base and the latter will comply with that desire.

These interactions are done through the MAS-IM module using the KB. The Knowledge Base was designed as to create a enable the build up of a shared mental state. This means that when two entities interact, each creates a knowledge base that refers to the other entity's beliefs, desires and intentions (BDI). It is important to note that this remote knowledge base

is not necessarily a full representation of the entity's BDIs, just the BDIs necessary to the interactions. This information is defined in the ontology.

For instance a robot is available to be teleoperated, it publishes the teleoperation service. When the operator finds that robot, the operator subscribes that service. Then each of the physical entities create an empty knowledge base of the remote entity. From this time on each entity may choose whether to update its' BDI or simply to reply them when a query is made, as it is shown in figure 4.3.

From an engineering point of view, Agent Communication Language (ACL) messages may induce a considerable network load when transferring large pieces of data. To this effect two design paths were followed. On one hand there is sparse information (e.g. sensor information), although high in quantity it has very little detail associated. Sensors are subject to false positives and negatives, thus meaning that sometimes the right events are not being communicated to the rest of the network. On the other hand, there is detailed information. This information is not generated as fast as sparse one, however it has more detail. For instance a robot may be constructing a map where it checks the events (e.g. obstacles) from different angles. This provides a way of verifying if the obstacle is still there or not, or if even existed. Other entities have no other way of knowing of the false positives, since the robot only updates new obstacles, it does not update removed ones. Other entities can only synchronise by accessing detailed information and discarding previous sparse information. Meaning that detailed information must be requested with a certain regularity, in order to, provide an improved view of the remote environment. However it can not be done in short time intervals, as it would create great latency in the network. A test on this trade off can be seen in the following chapter.

When considering the information transfer method, two paths were followed. On the sparse information case, queries, replies or updates are done through ACL messages. Messages are filled with knowledge based information. This occurs in the content feeder module, part of the MAS-IM module, as shown in figure 4.3. This image depictcs how an operator interacts with a robot, in which the operator asks for telemetry updates (i.e. robots' beliefs) and updates his desired heading (i.e. the operators' intentions). On the other end the MAS-IM module extracts

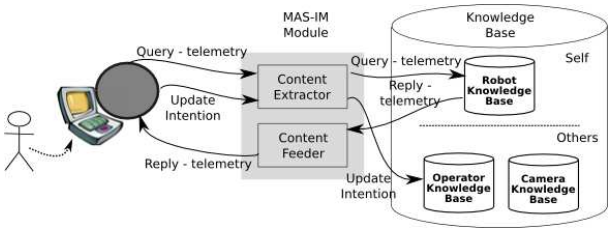


Figure 4.3: Inter-Agent Interaction.

the knowledge base information through the content extractor module updating the information in the knowledge base, also seen in figure 4.3. When considering detailed information, protocols designed for larger data transfer are used, such as the FTP protocol. This information exchange is also allowed by the publish and subscribe wrapper is implemented, where the requesting entity subscribes to the information by reading the protocol definitions and target information.

4.2.2 Intra-Agent Interactions

Intra-Agent interactions refer to the interactions between the entity's agent and the entities internal mechanisms, e.g. the robots' agent consulting the log files containing the telemetry data. These interactions are restricted to the type of entity they were designed for, meaning that they are native to a specific entity. Considering the significant differences between a robots' control system and an operator control system, these kind of interactions usually have to be native to the type of entity considered. Not only because the interfaces are different, in the operators' case a Human Interface Device (HID) (e.g. gamepad) is the obvious interface, while in the robots' case a socket or file interface is more suited.

Intra-Agent interactions are performed by the Knowledge Base and the Physical Entity (P.E.) module, in a similar way that the human brain assimilates the external information and then sends the information to the several muscles, or how the brain receives information from the muscles of fatigue or fitness. There is a daemon that regularly updates the Knowledge Base self beliefs. This daemon activates the Information Encoder on the P.E. module and generates Knowledge Base suitable information. The P.E. module checks the log file system in order to retrieve the most recent information (e.g. telemetry data) and transforming it into KB informa-

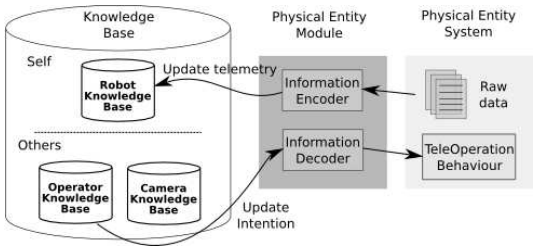


Figure 4.4: Intra-Agent Interaction.

tion, as to update the its' beliefs.

Considering that this work does not refer to an autonomous robot, the robot has not the desire of moving or accomplish anything. However with this representation, by having the other Knowledge Base (e.g. operator), the operator is able to provide the robots' agent with a 'will' of movement. This is done through a daemon that works as an hormone that navigates through the nerves in order to send the desired information, when the operators remote Knowledge base has its BDIs updated the daemon transforms the KB information into low-level information, using the Information Decoder on the P.E. module, and sends it to the robots' control system. All of this is depicted in figure 4.4. Here one can see how the robots' KB interacts with the robots' system. The KB updates the operator desire (other KB) and retrieves from its' system the update of its self beliefs (i.e. telemetry data).

Chapter 5

Situation Awareness Prototype

This chapter presents an instantiation of the presented architecture and teleoperation model. First, in section 5.1 the physical robot, Ares, is presented, as well as, a presentation of the whole system. Afterwards in section 5.2 a mission awareness tool instantiation is presented. Finally, in section 5.3 it is shown some tools designed for improved Human-Robot Interaction awareness

5.1 Global Perspective

The hostile environmental conditions and strict requirements dictated by tasks like search & rescue or humanitarian demining make the development of a service robot a challenging task. This thesis addresses the case of a robot for unstructured all-terrain environments that, due to the previously mentioned consequences, must be sustainable and disposable. This requires the robot to exhibit high mobility, ground adaptability, reduced size, to be low energy demanding, and to be affordable [Santana et al., 2007].

5.1.1 Mechanical Structure

Traction is a very important factor on an all-terrain robot. The robot must be able to adapt itself to the unevenness of the terrains in which it will be deployed, and at the same time, be agile

enough to dodge obstacles. Another important factor is the sensitivity of the terrain in some domains. One example is the easy landmine triggering caused by the robot when revolving the ground while moving, or the collapse of a fragile structure in a search & rescue mission. Therefore, the robot must be able to produce smooth trajectories, avoiding slippage as much as possible , and therefore ground perturbation.. To tackle all of these issues, the Ares robot, which is a four wheels vehicle with independent steering was developed [Cruz et al., 2005].

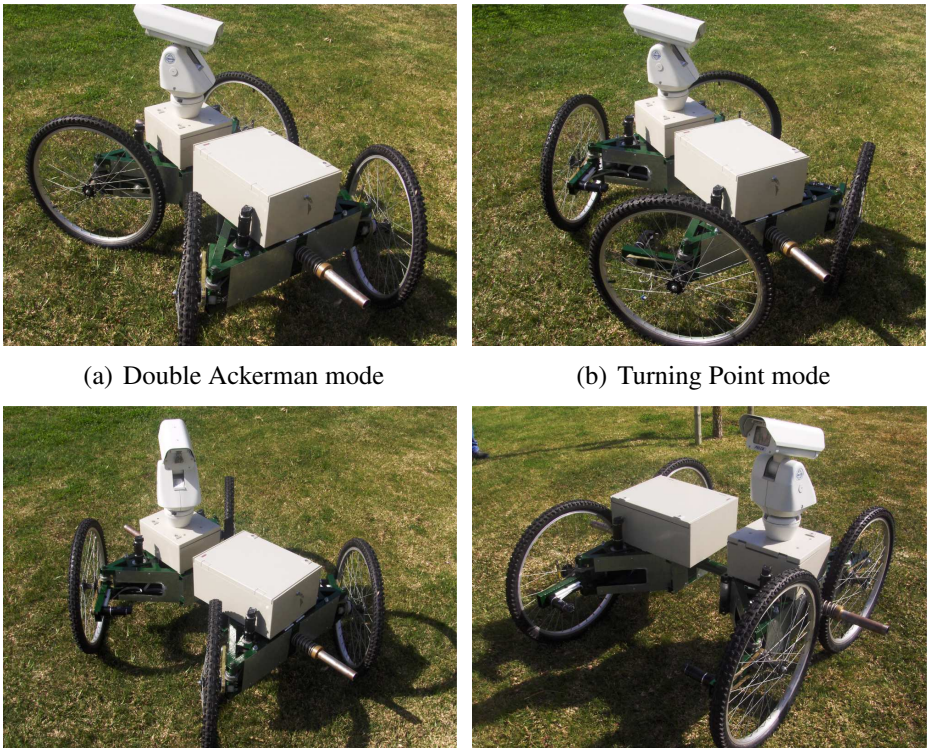
This section presents the refinements to the vehicle and its hardware components. The degrees of freedom Ares possesses allow it to displace in several modes (figure 5.1):

- Double Ackerman (figure 5.1(a)),
- Turning Point (figure 5.1(b)),
- Omnidirectional Steering (figure 5.1(c)) and
- Lateral Mode (figure 5.1(d)).

The steering and traction mechanisms are extremely simple, yet robust, allowing easy and fast substitution in case of excessive mechanical wear and/or damage.

To comply with the unevenness of the terrains where this robot will be deployed, a passive spinal axle, which allows the front and rear wheels to rotate around it was introduced (see figure 5.2). Apart from that, the tires can be easily replaced in order to better comply with the needs of a given terrain. These two characteristics allied to the 40 cm height to the ground provide the Ares robot with a great flexibility, reducing at the same time the perceptional requirements for terrain trespassability assessments.

The current Ares robot instantiation is depicted in figure 5.3, where it can be seen the workload and payload. In this picture it can be seen the Ares robot and its current instantiation. The Ares robot uses a stereo vision camera for obstacle detection and avoidance. Also it uses a GPS, visual odometry and wheel odometry for localization. Considering the workload, Ares has a stereo vision camera which allows for obstacle detection and visual odemetry. For localisation it uses a combined sensor fusion of GPS, visual odometry and wheel odometry (combined with



(a) Double Ackerman mode (b) Turning Point mode

(c) Omnidirectional mode (d) Lateral mode

Figure 5.1: The Ares Robot prototype in Double Ackerman, Turning Point, Omnidirectional, and Lateral Displacement modes.

Figure 5.2: Torsion around Ares' spinal axle

Figure 5.3: Ares Robot.

a compass). On the payload point of view, it supports a teleoperation camera which is considered a separate entity due to its ability to perform tasks simultaneous to the robots' operations. This characteristic will be further explained in the next section.

5.2 Mission Awareness

In this section it is described an approach which provides mission awareness. This approach has been integrated in the DSAAR architecture, for a further description of this work please refer to [Santana et al., 2008b].

Having integrated this tool into the DSAAR architecture, it allowed an improved mission awareness since there is a front end that allows the supervisor to see in which part of the mission the system is currently in. This also allows for a full mission comprehension. All of this is depicted in figure 5.4. This figure shows a template of a surveillance mission where four entities are considered. The camera is performing autonomous activities while the robot is performing a series of tracks and go to gps point. In this mission it also considered an operator for teleoperation purposes and a supervisor to decide which tracks should be followed.

There is also a map tool that had initially been designed for HRI interaction, however the addition of several functionalities granted the use of a mission tool. In mission support the map tool (see figure 5.5) provides the operator or supervisor the ability to define a track for which

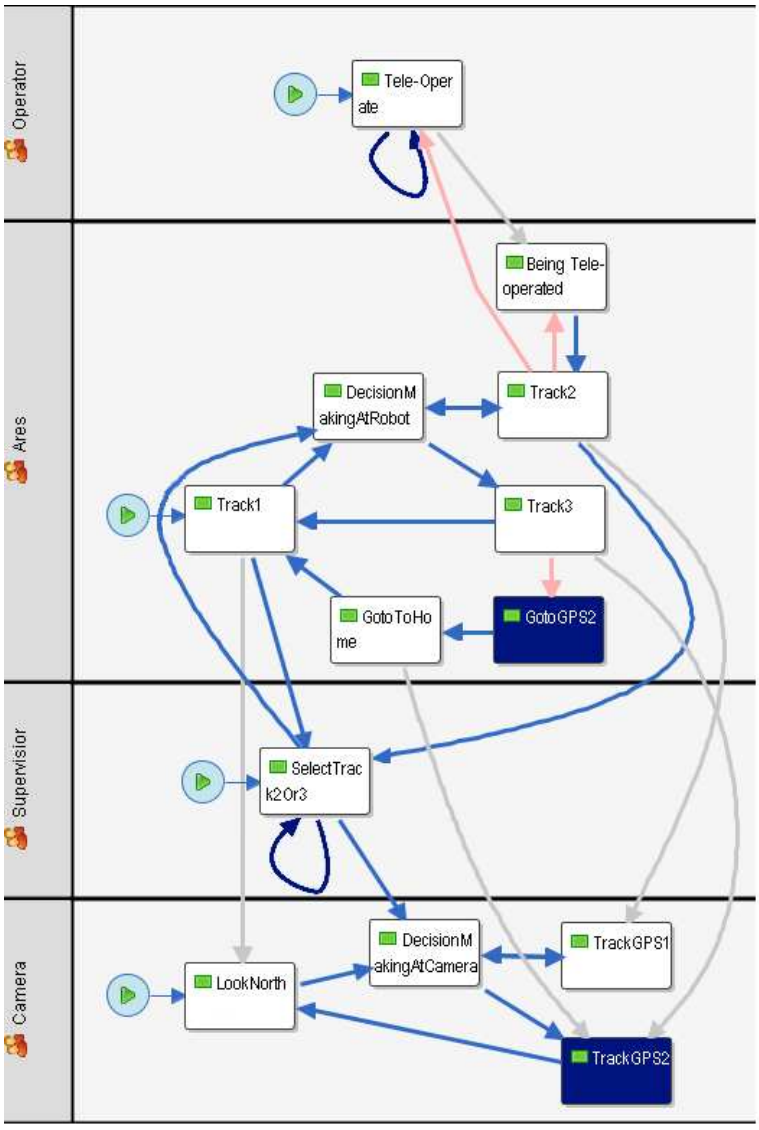


Figure 5.4: Mission Template

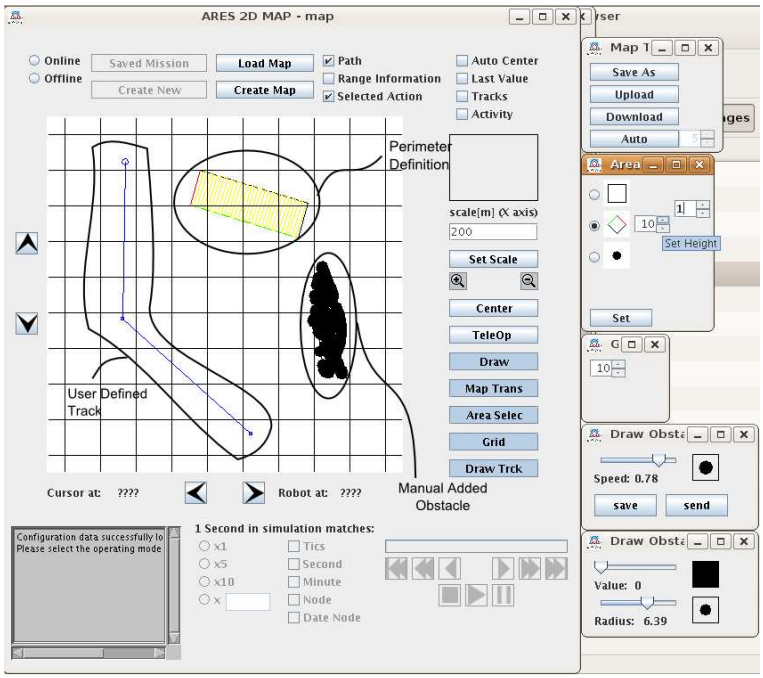


Figure 5.5: 2D Map Tool in mission support

the robot to follow or simply choose a previously defined one; a gps point for the robot to go; it allows for the addition of obstacles, which means if the supervisor does not intend for the robot to go to a certain area, he can add a fake obstacle in that zone; also the supervisor or operator can also define areas for the robot to patrol (e.g. perform a zig-zag through a determined perimeter).

5.3 Human-Robot interaction awareness

In teleoperation perception is a clue element in its process, to this effect two tools were designed. One which shows the operator the robots' and cameras' posture, as well as, the desired commands and another which is a 2D map representation of the world with several functionalities.

The teleoperation tool, is the tool that allows the operator to control the robot and teleoperation through a logitech gamepad[1]. As seen in figure 5.6 the operator may see the robots' posture, current wheel speed and internal status (e.g. battery level), this tool also allows the user to check the desired speed he is trying to send to the robot. Regarding the camera control

[1] www.logitech.com

and information the operator receives the pan, tilt and zoom information, as well as, the operator may choose between several control modes such as follow heading; auto-pilot, where the camera follows the robots' movements while driving; speed mode, where the operator sends a rotation speed of pan and/or tilt; and position mode, which allows the operator to point click in any place shown in the robots' representation in the teleoperation tool. For further description on these modes please refer to [Cândido et al., 2008].

In order to test some of these concepts, some tools were designed to allow a better human/robot interaction. In this section two tools are described: (1) the teleoperation tool and (2) the 2D map tool. Both these tools have been developed using the Jade platform, and so they are supported by Jade Agents. To complement these two tools the operator is also provided with a video link of a teleoperation camera mounted on the robot.

5.3.1 TeleOperation Tool

This tool allows the user to control the robot, a camera or both by using a generic USB gamepad. Via a gamepad the operator can select the different operating modes of the robot. Another feature is the fine tuning mode, which allows the user to control the robot in a smaller range of speeds, providing the operator with an enhanced sensibility. To make the control easier, an axis lock was made available, which means that the operator can lock either the direction axis or the speed axis while controlling the robot; this functionality is also available in the camera, but in this case for the pan and tilt variables.

In this interface, the operator can monitor some telemetry supplied by the robot with ACL messages. Some examples are position, pitch, twist, heading, wheel angle, wheel speed, and battery status. The operator can also select whether commands are sent to the robot in a continuous or step by step way. The operator can also use the robots' autonomy and control it in the modulated mode, where the operator gives a desired heading and speed and the robot uses its' obstacle avoidance behaviour in order to divert from obstacles that may be in its way. All these features can be seen in figure 5.6.

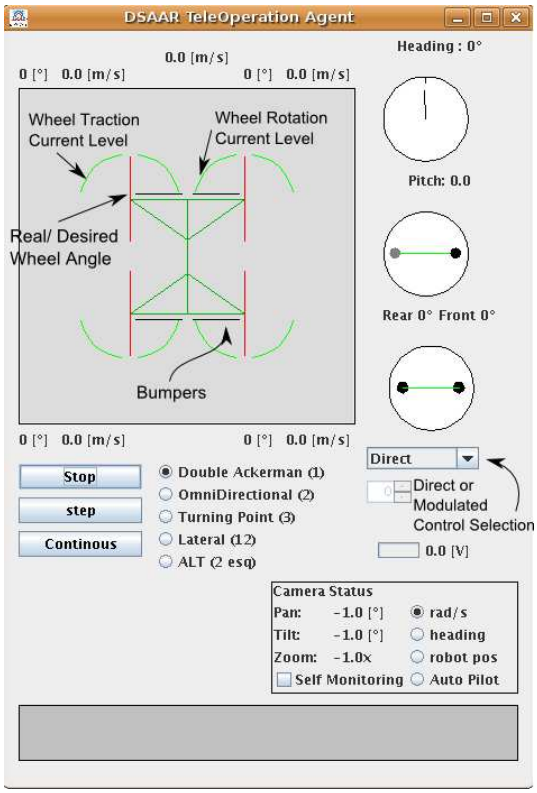


Figure 5.6: Snapshot of the TeleOperation Tool

5.3.2 2D Map Tool

The user can launch a Teleoperation tool that receives its information from the 2D Map, so that the user may see what is happening in a certain position at a specific time.

In offline mode, the user can select between creating a new *Mission Object* from previous mission logs, using a tool created for that effect (see figure 5.8), or by loading a previously saved *Mission Object*. In this mode the user can navigate in the mission with the accuracy down to the millisecond, either in real-time or in a delayed or faster pace. This feature is essential for debugging and failure analyses.

The Online mode allows the user to see what is actually happening in the current mission. Its procedures are similar to the Teleoperation tool were message swapping between JADE agents in the tool and robot are carried out. This agent also requests an amount of sensor information so it can process it and build obstacles the map.

The 2D map tool, is a tool that allows an operator to have a global perspective of the environ-

ment where the robot is conducting its operations. This tool may also be used by a supervisor in order to check the operation progress, since it does not need to have a direct influence in the mission procedure. This tool allows for all the commons features of a regular map, such as, path tracking; obstacles identification; map navigation; scale adjusting; and a adjustable grid, which allows the operator to have a greater notion of the travelled spaces. The 2D map tool also supplies a tool that allows for the user to supply a map where a mission has taken or is taking place and use it as background given a set of pre-requisited information (see figure 5.9). Finally this tool allows for users to watch a previous mission and study the operators' behaviour during the course of that mission. This is done by downloading the mission logs from the robot using the detailed information interaction exchange, then the map launches a process that emulates the robots' agent behaviour and so allowing the user to toggle back and forth through that mission in a variety of time scales. Most of these features may be seen in figure 5.7. Here one can see all the navigation buttons, the zoom buttons, and most of the map functionalities. Also it can be seen a part of a mission where the dark grey and black areas are obstacles and the light grey areas are undiscovered areas. The robot is depicted in red (red square) and on the right box the current robot heading is shown with the red line symbolizing the current heading.

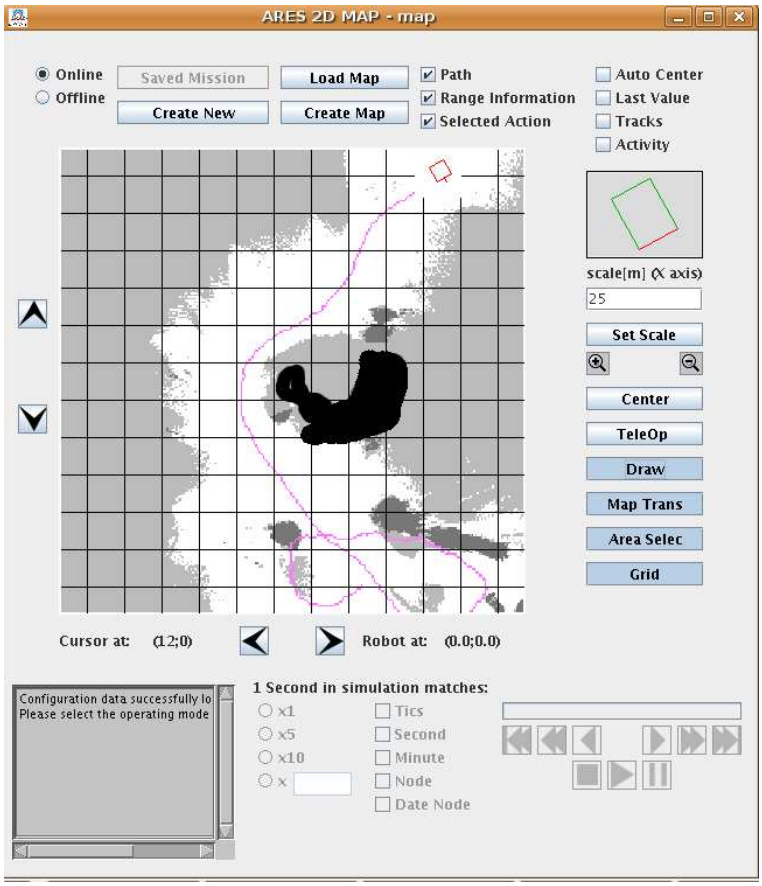


Figure 5.7: Snapshot of the 2D Map Tool.

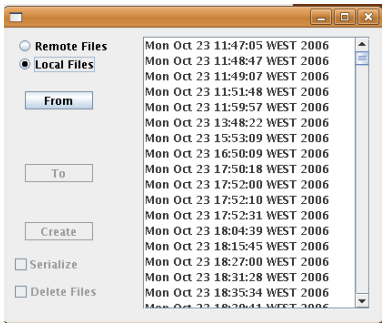


Figure 5.8: Snapshot of the Log Builder Tool. The user may choose between one or more dates and build a mission log during that period of time.

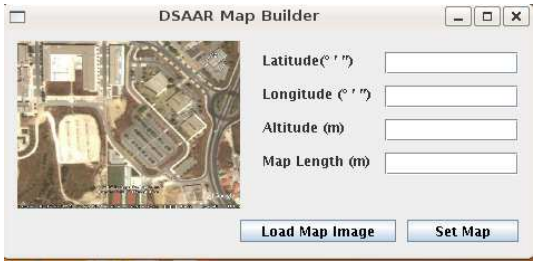


Figure 5.9: Snapshot of the Map BuilderTool. Using this tool the user may insert a map of a determined region and by supplying its gps location and scale the operator may use it in a mission.

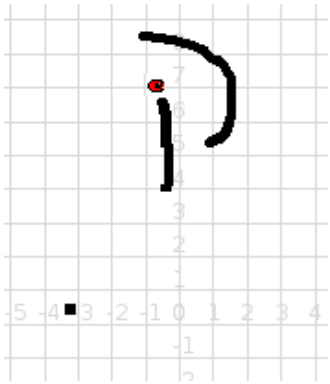
Chapter 6

Experimental Trials and Results

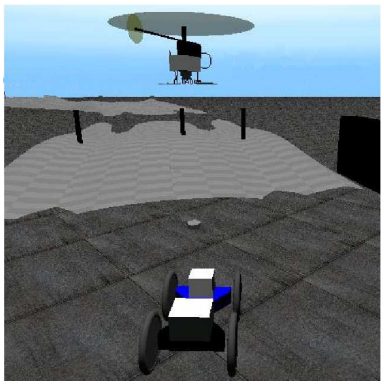
In this chapter it is presented the preformed tests made in order to validate the proposed architecture, as well as, the teleoperation model.Also there is an analysis of the trials results. First, in section 6.1 it is shown the several platforms where this architecture has been tested. Then in section 6.2, an outdoor test is presented where the operators' human-robot interaction awareness is tested.

6.1 Multi-Platform Test

In order to validate the multi-platform characteristic of the DSAAR architecture , a series of experiments were carried out. Initial tests were made using the player/stage/gazebo tool [Gerkey et al., 2003] and with the RWI B12 indoor robot (figure 6.1(a) to 6.1(c) illustrates both platforms). However there was a need to bring it outdoors, in order to test not only communications as well as the navigation difficulties in a more demanding environment. Forced this model to be tested in the Ares outdoor robot [Santana et al., 2008a] (see figure 6.1(d)) which has already been presented in the previous chapter.



(a) Player/Stage



(b) Player/Gazebo [Ribeiro, 2007]

(c) RWI B12 robot

(d) Ares Robot

Figure 6.1: Platforms in which the DSAAR architecture has been tested

6.1.1 Analysis

The initial tests with the player/stage/gazebo tool, allowed for a test of the control APIs and analysis on a wired network. It proven to be a simple task to integrate the DSAAR architecture with player/stage through a set of API with the control system and the sensor output. This test also allowed to verify that the teleoperation commands and telemetry data had a high network load, which lead to a optimisation of this fact. The integration of the RWI B12 robot was also a simple fact of designing the APIs to the current control system and was intended to test the communication via a wireless network. This test proved to grant a reactive control and a good telemetry data flow. Finally the DSAAR architecture was implemented in the final prototype the Ares robot, the results of the HRI awareness are presented in the following section.

6.2 Field Trials

In order to test the ability of the proposed approach to handle scenarios with demanding network load, an experiment was designed that needed both sparse and detailed information. A supervisor was also added in order to take into account the notion of shared reality. The supervisor was added to the network in different time intervals to verify the perception of the reality. Taking into account the setup presented in the previous section, an outdoor environment was considered.

The experimental setup is composed by a teleoperation camera, a robot and an operator. The robot and the camera may be seen in figure 6.1(d). Although placed together, their characteristics were self sufficient to be considered different entities. When searching for specific targets or considering the scenario of autonomous navigation, the teleoperation camera may be used without interfering with the robots' reality.

The operator interacts with the system through a control centre depicted in figure 6.2. This system is composed of a video screen and a laptop with a gamepad. The video screen displays a direct video feed from the teleoperation camera, this feed is considered detailed information so the service is subscribed and then with the necessary description the video server is contacted and the link is formed. The laptop provides two tools from which the operator is able to get a perception of what is happening with the robot and its surroundings.

In this trial the operator updates ones desires and intentions through the teleoperation tool, and requests the robots' beliefs through the teleoperation and map tool. The operator request both sparse and detailed information.

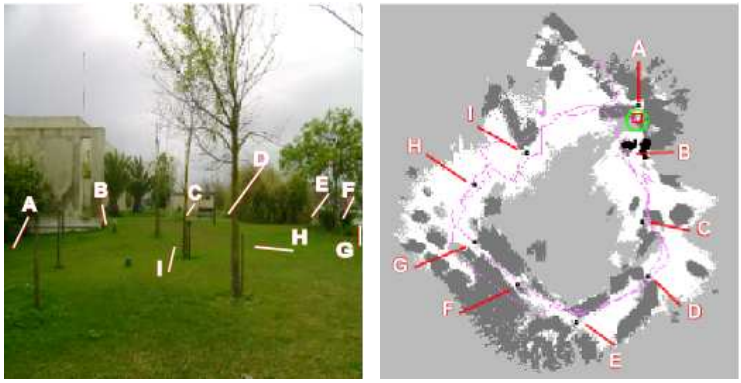
6.2.1 Experimental Setup

In this experiment the operator has to move the robot through a circuit formed by 9 points, from point A to point I. In this course the operator faces several obstacles which have to be avoided, as well as, navigation through narrow areas. The goal of this test is to find the best relation between the amount of detailed information that may be asked from the robot, without losing real time

Figure 6.2: Control Center

control, as well as, the minimum detailed information necessary for competent teleoperation. The test course can be seen in figure 6.3, where the real test site and a robots' representation made previously to the test of the site are shown. In this figure it can be seen which points the user has to pass during the test. In figure 6.4 it can be seen some of the most difficult challenges of this course, for instance in figure 6.4(a) the operator had to take the robot through an area whose width was few centimetres wider than the robots' width. In point C where the operator is faced with an S turn within an area with great movement limitation, (see figure 6.4(b)). In point D the operators' greatest challenge was to avoid hitting any of the concrete pins depicted in figure 6.4(c). The last of the operators' challenges was in the section composed of the points E, F and G, where cars were parked and the operator could not hit them. This course section ended with a right turn which could easily be overlooked, (see figure 6.4(d)). The operator was placed near point H turned with his back to the testing course. During the several runs a supervisor was seldomly added to the network in order to monitor the status of the run.

In this test the operator performed several runs where the detailed information, e.g. the map, was requested from the robot in several time intervals which varied form 1 to 10 seconds, while still receiving sparse information, e.g. telemetry data, and sending orders. Also important to refer is that sparse information had a 300 ms transfer rate and teleoperation orders had a 100 ms transfer rate. In the end the more relevant runs were repeated in order for a better understanding of the results.



(a) Test field picture

(b) Robots' map of the test field overview. (37x39 mxm)

Figure 6.3: Test field overview

Figure 6.4: Test course main challenges'

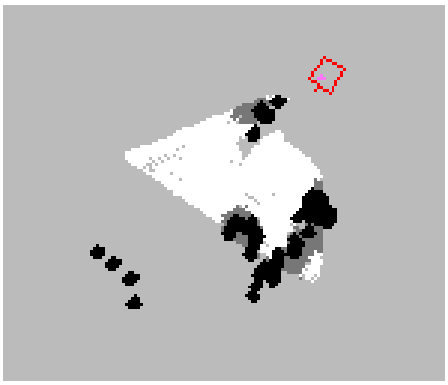


Figure 6.5: Initial run (15x13 mxm).

6.2.2 Results and Analysis

During the course of this test it was found that detailed information, i.e. robots' map, made control unreliable under the 5 seconds transfer rate. This fact got worse with distance of the robot and with the size of mapped area. At the start of each run the operator had access to the robots' map at that time, which is depicted in figure 6.5. Here one can see in white, open terrain; in dark grey obstacle from the robots' map; unknown area in light gray; the black spots are telemetry obstacles; and the robot is depicted by the red rectangle. After that, the rate at which the robots' map is refreshed is according to the run. Here it will be analysed the 5 seconds and 10 seconds run. All runs with a 5 seconds or larger refresh rates were successfully completed. The ones under 4 seconds were interrupted, as a result of the operator not being able to have full robot control and there was danger when approaching the cars near the points E, F and G.

An example of a complete run may be seen in figure 6.6. Here one can see in white, open terrain; in dark grey obstacle from the robots' map; unknown area in light gray; the black spots are telemetry obstacles; and the robot is depicted by the red rectangle. This run was made with a 5 seconds refresh rate. During this run the operator was able to keep control during the whole trial, with few misleading s from false positives from telemetry data. One example of misleading telemetry information may be seen in figure 6.7, in which telemetry data gave the operator the notion that the robot would not be able to pass through point D, and forced the operator to wait for the map to be refreshed in order to proceed. One may see this influence by

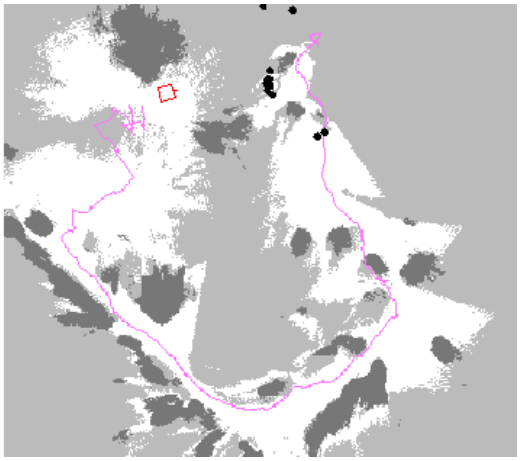


Figure 6.6: Result of complete 5 seconds run (32x28 mxm).

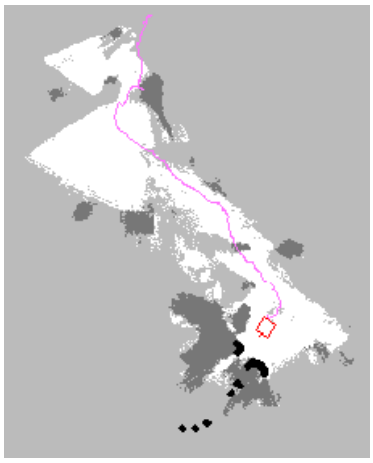


Figure 6.7: Situation awareness in a 5 seconds run near point D (23x28 mxm).

comparing with figure 6.6 in the same area near point D. The problem of misleading telemetry information increased as the refresh rate was smaller, which caused a performance drop. In several situations the operator was forced to wait several times, in order to proceed, this may be seen in figure 6.8. In this situation the wait for detailed information, and sensor false positives gave the operator the impression of a dead end. This can be seen by comparing with figure 6.6 near point

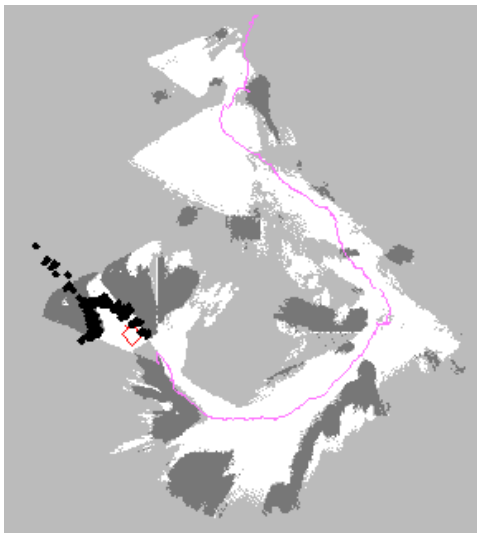


Figure 6.8: Situation awareness in a 10 seconds run near point G (29x32 mxm)

Chapter 7

Conclusions and Future Work

This chapter sums up this dissertation providing a set of conclusions and contributions, and pointing out some directions for future work as well.

7.1 Conclusions

In this dissertation a software architecture for fast development and prototyping of multi-robot systems, as well as, a real time HRI awareness. The proposed model makes no commitment to any control paradigm, leaving that decision for the researcher. DSAAR HRI approach and tools proved to provide the user with a good HRI awareness and robot control.

The DSAAR architecture, aims to be a flexible, scalable, and distributed multi-agent architecture for the development of multi-robot systems. It has been shown that it is possible to implement such a system based on open source software, which increases the interest for research and development projects. In addition, the licenses of the selected software packages allow the blending with proprietary developments. This is of extreme importance if one intends to allow the architecture to be applied in commercial products. Jade endows the system with social ability, making interactions among robots and humans transparent. In addition, Protégé endows the system with the ability to manage and share OWL ontologies, for now this is done solely at the MAS level. This allows experts, users, and machines to interact among themselves

in a convenient way. Finally, Linux-based processes with a custom API allow control system designers to focus on the behavioural model of the system. Some system specific processes and agents are used to build up a robot operating system. Thus, the control system designer is provided with a set of tools to speed up prototyping and incremental development.

A new approach for the problem of Human-Robot teams was proposed. The architecture was validated in the real world, with a robust and disposable robot. Multiple users have successfully been added to the system with no significant extra network latency nor robot's on-board computer overload. The multi-agent based teleoperation system proposed, has been was validated both in simulation and the real world, under a teleoperation scenario. The Multi-Agent System, allows the addition and removal of several entities in a seamless way. When performing the experiment, this situation was tested with the addition and removal of the supervisor. This approach fosters the concept of shared reality, by having recently added entities with need of being updated with other's state information. The field trials shown a clear trade off between sparse information and detailed information. When the operator, asked for too much detailed information, the ability for controlling the robot was clearly decreased. However results have shown that there is no need in having detailed information in short time frames, as sparse information can easily complement that gap. During the field trials, a trade off was found at the 300 ms cycle for defined previously for sparse information and with a 5 seconds cycle for detailed information. This combination provided the operator with a comfortable 100 ms reaction control. Using this transfer rate the operator was able to have real time control, along with a proper situation awareness.

7.2 Future Work

Below, some research opportunities based on this dissertation are mentioned:

Regarding the DSAAR architecture and the adaptation of control modules there are plenty of opportunities for future work. Further IPC mechanisms can be integrated, such as FIFOs' that have been included in the designed but not fully implemented. Also there is an opportunity

to interface with further control mechanisms already widely used (e.g. several planner modules). Looking into the MAS system, there can be an effort on transforming the network into a pure peer-2-peer network, since the JADE platform allows it. Finally the architecture can be packaged and distributed to increase its usage, requirements and contributors.

On the HRI awareness field, there can be some study on ways to improve the shared mental state of all entities present in the network. Also there can be a study to determine necessities of each entity, in order to provide the necessary information was intuitively as possible. Finally, search new ways to improve the network transfer rate while decreasing it's load, by studying the detailed information exchage.

Bibliography

[Arkin, 1989] Arkin, R. C. (1989). Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112.

[Bellifemine et al., 2003] Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. (2003). Jade: a white paper website. *Exp (online: http://exp.telecomitalialab.com)*, 3(3).

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., and Rimassa, G. (1999). JADE – a FIPA-compliant agent framework. In *Proc. of PAAM'99*, pages 97–108, London.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

[Brooks, 1991] Brooks, R. A. (1991). Intelligence without representation. Number 47 in Artificial Intelligence, pages 139–159.

[Burke and Murphy, 2004] Burke, J. and Murphy, R. (2004). Human-robot interaction in usar technical search: Two heads are better than one. In *Proceedings of the IEEE RO-MAN 13th International Workshop on Robot and Human Interactive Communication*, pages 307–312, Okayama, Japan.

[Cândido et al., 2008] Cândido, C., Santana, P., Correia, L., and Barata, J. (2008). Shared control of a pan-tilt camera on an all-terrain mobile robot. In *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008) (to appear)*, Hamburg, Germany.

[Cohen and Levesque, 1991] Cohen, P. R. and Levesque, H. J. (1991). Teamwork. 25:487–512.

[Côté et al., 2006] Côté, C., Brosseau, Y., Létourneau, D., Raïevsky, C., and cois Michaud, F. (2006). Robotic software integration using marie. *International Journal of Advance Robotics Systems (ARS-journal)*, 3(1):55–60.

[Cruz et al., 2005] Cruz, H., Lisboa, J., Santana, P., Maltez, R., Barata, J., , and Flores, L. (2005). Two sustainable and compliant robots for humanitarian demining. In *Proceedings of the IARP International Workshop on Robotics and Mechanical Assistance in Humanitarian Demining (HUDEM2005)*, pages 64–69, Tokyo, Japan.

[Dourish and Belloti,] Dourish, P. and Belloti, V. Awarness and coordination in shared workspaces. In *Proceedings CSCW '92*.

[Drury et al., 2003] Drury, J., Scholtz, J., and Yanco, H. (2003). Awareness in human-robot interactions. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 912–918.

[Drury, 2001] Drury, J. L. (2001). *Extending usability inspection evaluation techniques for synchronous collaborative computing applications*. PhD thesis, Computer Science Dept, University of Massachusetts Lowell.

[Drury et al., 2007] Drury, J. L., Keyes, B., and Yanco, H. A. (2007). Lassoing hri: analyzing situation awareness in map-centric and video-centric interfaces. In *HRI*, pages 279–286.

[Endo et al., 2002] Endo, Y., MacKenzie, D. C., and Arkin, R. C. (2002). Usability evaluation of high-level user assistance for robot mission specification. Technical report, Mobile Robot Laboratory at Georgia Tech.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.

[Gerkey et al., 2003] Gerkey, B. P., Vaughan, R. T., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal.

[Grosz and Kraus, 1996] Grosz, B. and Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357.

[Grosz and Kraus, 1999] Grosz, B. and Kraus, S. (1999). The evolution of sharedplans. In *Foundations and Theories of Rational Agency*, pages 227–262. Springer.

[Joeseph Manojlovich and Gennari, 2003] Joeseph Manojlovich, M. L. and Gennari, J. (2003). Camera control and decoupled motion for teleoperation. In *Systems, Man and Cybernetics, 2003. IEEE International*, pages 1339–1344. IEEE Press.

[Konolige et al., 1997] Konolige, K., Myers, K., Saffiotti, A., and Ruspini, E. (1997). The Saphira architecture: a design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:215–235.

[Nayar, 2007] Nayar, H. D. N. (2007). Re-usable kinematic models and algorithms for manipulators and vehicles. In *Proceedings of the IEEE Intelligent Robots and Systems (IROS)*, pages 833–838.

[Nesnas et al., 2006] Nesnas, I. A., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., Madison, R., Guineau, J., McHenry, M., Shu, I.-H., and Apfelbaum, D. (2006). Claraty: Challenges and steps toward reusable robotic software. *International Journal of Advance Robotics Systems (ARS-journal)*, 3(1):23–30.

[Nesnas et al., 2003] Nesnas, I. A., Wright, A., Bajracharya, M., Simmons, R., Estlin, T., and Kim, W. S. (2003). Claraty: an architecture for reusable robotic software. In Gerhart, G. R., Shoemaker, C. M., and Gage, D. W., editors, *Unmanned Ground Vehicle Technology V. Edited by Gerhart, Grant R.; Shoemaker, Charles M.; Gage, Douglas W. Proceedings of the SPIE*, volume 5083 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 253–264.

[Nielsen and Goodrich, 2006] Nielsen, C. W. and Goodrich, M. A. (2006). Comparing the usefulness of video and map information in navigation tasks. In *Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 95–101.

[Nolle et al., 2001] Nolle, L., Wong, K., and Hopgood, A. A. (2001). *Research and Development in Intelligent Systems XVII*, chapter DARBS: A Distributed Blackboard System. Springer.

[Nourbakhsh et al., 2005] Nourbakhsh, I., Sycara, K., Koes, M., Yong, M., Lewis, M., and Burion, S. (2005). Human-robot teaming for search and rescue. In *IEEE Pervasive Computing: Mobile and Ubiquitous Systems*, pages 72–78.

[OWL Website, 2006] OWL Website (Accessed in April 20, 2006). http://www.w3.org/tr/owl-features/.

[Philippsen and Siegwart, 2003] Philippsen, R. and Siegwart, R. (2003). Smooth and efficient obstacle avoidance for a tour guide robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA 2003)*, Taipei, Taiwan.

[Protege OWL Website, 2006] Protege OWL Website (Accessed in April 20, 2006). http://protege.stanford.edu/plugins/owl/.

[Ribeiro, 2007] Ribeiro, S. Z. (2007). Simulação para Robôs Móveis Aplicada à Desminagem. Master's thesis, Universidade da Madeira, Madeira, Portugal.

[Santana and Barata, 2005] Santana, P. and Barata, J. (2005). Multiagents applied to humanitarian demining. In Pechoucek, M., Petta, P., and Varga, L. Z., editors, *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005), LNAI 3690*, Budapest, Hungary. Springer.

[Santana et al., 2005a] Santana, P., Barata, J., Cruz, H., Mestre, A., Lisboa, J., and Flores, L. (2005a). A multi-robot system for landmine detection. In Bello, L. L. and Sauter, T., editors, *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, pages 721–728 (vol.I), Catania, Italy.

[Santana et al., 2008a] Santana, P., Cândido, C., Santos, P., Almeida, L., Correia, L., and Barata, J. (2008a). The ares robot: Case study of an affordable service robot. In *Pro-*

ceedings of the 2nd European Robotics Symposium, (EUROS 2008), pages 26–28, Prague, Czech Republic.

[Santana et al., 2006a] Santana, P., Cândido, C., Santos, V., and Barata, J. (2006a). A motion controller for compliant four-wheel-steering robots. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics, (ROBIO 2006)*, Kunming, China.

[Santana and Correia, 2005] Santana, P. and Correia, L. (2005). Survival Kit: a constraint-based behavioural architecture for robot navigation. In Bento, C., Cardoso, A., and Dias, G., editors, *Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), LNAI 3808*, pages 435–446, Covilhã, Portugal. Springer.

[Santana et al., 2005b] Santana, P., Mestre, A., Barata, J., and Flores, L. (2005b). Roadmap for mine action robotic technology development. *Journal of Mine Action*, 9.1:89–91.

[Santana et al., 2008b] Santana, P., Salgueiro, M., Santos, V., Correia, L., and Barata, J. (2008b). A knowledge-based component for human-robot teamwork. In *Proceedings of the 5th IEEE International Conference on Information in Control, Automation and Robotics*, Funchal, Portugal.

[Santana et al., 2006b] Santana, P., Santos, V., and Barata, J. (2006b). Dsaar: A distributed software architecture for autonomous robots. In *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 20–22, Prague, Czech Republic.

[Santana et al., 2007] Santana, P. F., Barata, J., and Correia, L. (2007). Sustainable robots for humanitarian demining. *International Journal of Advanced Robotics Systems (ARS-journal)*, Vol. 4(No. 2).

[Santos et al., 2007] Santos, V., Cândido, C., Santana, P., Correia, L., and Barata, J. (2007). Developments on a system for human-robot teams. In *Proceedings of the 7th Conference on Autonomous Robot Systems and Competitions*, Paderne, Portugal.

[Santos et al., 2008] Santos, V., Santana, P., Correia, L., and Barata, J. (2008). Teleoperation mechanisms in a multi-agent system. In *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation (to appear)*, Hamburg, Germany.

[Sierhuis et al., 2005] Sierhuis, M., Clancey, W. J., Alena, R. L., Berrios, D., Buckingham Shum, S., Dowding, J., Graham, J., van Hoof, R., Kaskiris, C. A., Rupert, S., and Tyree, K. S. (2005). Nasa's mobile agents architecture: A multi-agent workflow and communication system for planetary exploration. In *'i-SAIRAS 2005' - The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, volume 603 of *ESA Special Publication*.

[Sycara et al., 2003] Sycara, K., Giampapa, J. A., Langley, B. K., and Paolucci, M. (2003). The retsina mas, a case study. In Alessandro Garcia, Carlos Lucena, F. Z. A. O. J. C., editor, *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, volume LNCS 2603, pages 232–250. Springer-Verlag, Berlin Heidelberg.

[Wang et al., 2004] Wang, J., Lewis, M., and Hughes, S. (2004). Gravity-referenced attitude display for teleoperation of mobile robots. In *Proceedings of the 48th Annual Meeting of the Human Factors and Ergonomics Society*, pages 2662–2666.

[Website, 2006] Website, J. (Accessed in April 20, 2006). http://jade.tilab.com/.

Appendix A

Enabling Technologies

In this chapter a description of the multi-agent support framework that supports this architecture is presented.

A.1 JADE

JADE [Bellifemine et al., 1999] simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines, which do not even need to share the same OS, and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required.

JADEs two major aspects of the conceptual model are: distributed system topology with peer-to-peer networking, and software component architecture with agent paradigm. The network topology affects how the various components are linked together, whereas the component architecture specifies what the components are supposed to expect from one another. JADE is based of the following driving principles:

- Interoperability - JADE is compliant with the FIPA[1] specifications. As a consequence,

[1] www.fipa.org

JADE agents can interoperate with other agents, provided that they comply with the same standard.

- Uniformity and portability - JADE provides a homogeneous set of APIs that are independent from the underlying network and Java version. More, the JADE run-time provides the same APIs both for the J2EE, J2SE and J2ME environment. In theory, application developers could decide the Java run-time environment at deploy-time.

- Easy to use - The complexity of the middleware is hidden behind a simple and intuitive set of APIs.

- Pay-as-you-go philosophy - Programmers do not need to use all the features provided by the middleware. Features that are not used do not require programmers to know anything about them, neither add computational overhead.

From a functional point of view, JADE provides the basic services necessary to distributed peer-to-peer applications in the fixed and mobile environment. JADE allows each agent to discover other agents dynamically and communicate with them according to the peer-to-peer paradigm. From the application point of view, each agent is identified by a unique name and provides a set of services. The agent can register and modify its services and/or search for agents providing given services. Each agent can control its life cycle and communicate with all other peers. Agents communicate by exchanging asynchronous messages, a communication model almost universally accepted for distributed and loosely-coupled communications, a general scheme of JADE agents registry, lookup and communication can be observed in figure A.1. The DF agent is in the main container, each machine can support several containers, yet there cannot be one container in several machines.

At this moment the agent communication uses a Hybrid peer-to-peer architecture, which means that there is a centralized aspect to this approach. However in the future it is intended to use a pure peer-to-peer architecture with no central point, to accomplish this goal it will be used a Federated DF system. The current system communication approach can be seen in figure A.2. Where it can be seen that the DF (yellow page agent) is on the main container, so if

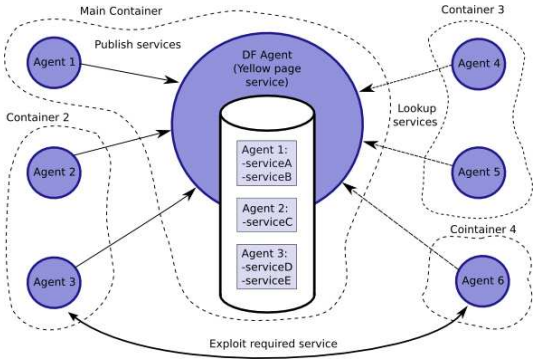


Figure A.1: Registry and Communication sequence

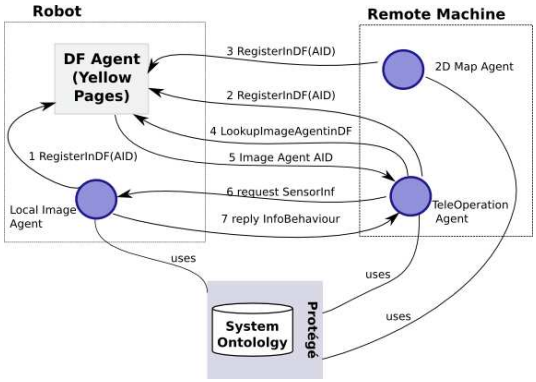


Figure A.2: Current communication approach

this container fails all the other agents will be at a stand still. Although to prove the proposed concept a pure peer-to-peer architecture is not really necessary, for the sake of scalability it fells necessary to try and implement a pure peer-to-peer architecture, as JADE supports it.