Distributed Agents for Autonomous Spacecraft

Stuart Grey

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

School of Engineering
College of Science and Engineering
University of Glasgow

Abstract

Space missions have evolved considerably in the last fifty years in both complexity and ambition. In order to enable this continued improvement in the scientific and commercial return of space missions new control systems are needed that can manage complex combinations of state of the art hardware with a minimum of human interaction.

Distributed multi-agent systems are one approach to controlling complex multi-satellite space missions. A distributed system is not enough on its own however, the spacecraft must be able to carry out complex tasks such as planning, negotiation and close proximity formation flying autonomously. It is the coupling of distributed control with autonomy that is the focus of this thesis.

Three contributions to the state of the art are described herein. They all involve the innovative use of multi-agent systems in space missions. The first is the development of a multi-agent architecture, HASA, specifically for space missions. The second is to use embedded agents to autonomously control an interferometric type space telescope. The third is based on software agents that coordinate multiple Earth observation missions coupled with a global optimisation technique for data extraction.

The HASA architecture was developed in reaction to the over generality of most multi-agent architectures in the computer science and robotics literature and the ad-hoc, case-by-case approach, to multi-agent architectures when developed and deployed for space missions. The HASA architecture has a recursive nature which allows for the multi-agent system to be completely described throughout its development process as the design evolves and more sub-systems are implemented. It also inherits a focus on the robust generation of a product and safe operation from architectures in use in the manufacturing industry.

A multi-agent system was designed using the HASA architecture for an interferometric space telescope type mission. This type of mission puts high requirements on formation flying and cooperation between agents. The formation flying agents were then implemented using a Java framework and tested on a multi-platform distributed simulation suite developed especially for this thesis. Three different control methods were incorporated into the agents and the multi-agent system was shown to be able to acquire and change formation and avoid collisions autonomously.

A second multi-agent system was designed for the GMES mission in collaboration with GMV, the industrial partner in this project. This basic MAS design was transferred to the HASA architecture. A novel image selection algorithm was developed to work alongside the GMES multi-agent system. This algorithm uses global optimisation techniques to suggest image parameters to users based on the output of the multi-agent system.

Acknowledgement

I would like to thank all of my fellow PhD students who I met working in the Space Advanced Research Team at the University of Glasgow, particularly Dr Matteo Ceriotti, Dr Camilla Colombo, Dr Christie Maddock, Dr Nicolas Croisard and Dr Pau Sanchez. I'd also like to thank Dr Gianmarco Radice and Dr Massimiliano Vasile for their supervision, guidance and most of all for giving me this opportunity.

I'd like to thank my parents and my brothers for all of their support in following my heart.

Lastly I'd like to thank Lorna, without her this PhD simply wouldn't have been possible.

Stuart Grey

13th June 2012

Authors Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Stuart Grey

Glasgow, Scotland, 13th June 2012

Table of Contents

List of Tables

List of Figures

Acronyms

ACL	Agent Communication Language
AOCS	Attitude and Orbital Control System
API	Application Programming Interface
ASAR	Advanced Synthetic Aperture RADAR
ASE	Autonomous Science Experiment
BDI	Belief Desire Intention
CAM	Collision Avoidance Mechanism
CD	Cooperative Domain
CRTBP	Circular Restricted Three Body Problem
DARPA	Defense Advanced Research Projects Agency
DUE	Data User Element
EO	Earth Observation
EO-1	Earth Observing Mission 1
EOMD	Earth Observation Market Development
ERCS	Emergency Response Core Service
ESA	European Space Agency
FBS	Function Behaviour Structure
FDIR	Failure Detection Isolation and Recovery
FDS	Flight Dynamics System
FIPA	Foundation for Intelligent Physical Agents
FTP	File Transfer Protocol
GA	Genetic Algorithm
GMES	Global Monitoring for Environment and Security
GNC	Guidance Navigation and Control
GPS	General Pattern Search
GSD	Ground Sample Distance
GUI	Graphical User Interface
GWT	Google Web Toolkit
HASA	Holonic Agent Space Architecture
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
JADE	Java agent development framework
KIF	Knowledge Interchange Format
KQML	Knowledge Query Manipulation Language
L2	Second Sun-Earth Lagrangian point
LEO	Low Earth Orbit
LMCS	Land Monitoring Core Service
MAS	Multi-Agent System
MCS	Marine Core Service
MVM	Mission Vehicle Management

Acronyms

NRT	Near Real-Time
ODE	Ordinary Differential Equation
OOP	Object Oriented Programming
ORCA	optimal reciprocal collision avoidance
PID	Proportional-Integral-Derivative
PROBA	Project for Onboard Autonomy
PROSA	Product Resource Order Staff Architecture
RAM	Random Access Memory
RVO	Reciprocal Collision Avoidance
SA	Simulated Annealing
SAR	Synthetic Aperture RADAR
SL	Semantic Language
SPOT	Système Pour l'Observation de la Terre
TCP/IP	Transmission Control Protocol/Internet Protocol
TPF	Terrestrial Planet Finder

Chapter 1 Introduction

The aim of this doctoral dissertation is to answer the question *"can multi-agent control enable more autonomous space missions?"*

This research is motivated by the fact that large strides have been undertaken in the field of autonomous artificial intelligence and robotics in recent years. The possibility of utilising these technologies in the space domain offers the possibility to extend and enhance the abilities of currently planned missions and to enable previously impractical missions.

The objectives of this dissertation are to give an overview of the current state of the art in autonomous agent based control and how it may be applied to space missions. Three contributions to the state of the art are described herein.

The first is the development of a multi-agent architecture for space missions. Both involve the innovative use of multi-agent systems in space missions. The second is to use embedded agents to autonomously control an interferometric type space telescope. The third is based on software agents that coordinate multiple Earth observation missions coupled with a global optimisation techniques for data extraction.

1.1 Why Autonomy?

There are two key drivers towards more autonomous space missions, performance and cost [1]. Primarily this is due to the autonomous mission's ability to remove the need for human direction and through the reduction or removal of communication with the ground and its inherent delays, especially for deep space missions. The ability of a properly designed autonomous control system to improve performance and lower cost for certain missions has led to the launch of number of missions exploring autonomous control as found in [2–6].

1.2 What is an Agent?

1.2.1 What is an intelligent agent?

Agency as a model was developed to try to encapsulate the way we as humans think. This leads us to ask a number of questions, such as:

What makes something an agent?
What makes something intelligent?

At its simplest an agent can be defined as a combination of perception, reasoning and action or more generally as an entity with human like attributes such as decision making and reasoning. [7]

Perception is the agent's view of its environment. The mode of perception and the environment in which the agent operates are all extremely variable. Perception can be data from sensors, a simulation or from external data sources. An agent must also be able to perceive its environment and any changes that occur through its actions.

Reasoning is where any "intelligence" of the agent is found. An agent may reason about its past, current or future actions, the actions of others or any past current or future changes to its environment.

In order to have agency the agent must be able to act. This can be easily seen using the example of an embodied agent, i.e. a human or a robot with agent based intelligence. The agent is able to move or manipulate objects within its environment and change its position in the environment. Agents can exist entirely in software (an informational agent) where their actions are moving bits rather than moving atoms. Agents can also straddle the physical and purely informational domains and provide intelligent links between previously unconnected actors or services. It should also be noted that in multi-agent systems human agents may play a large role (although they don't have to) and can be adequately described by the theory.

The inputs of the agent, its perceptions, can take the form of prior knowledge, past experiences, goals/values and current observations. These inputs are fed into the reasoning engine and the result is a set of actions undertaken by the agent.

It is tempting to group prior knowledge and past experience together as they can be thought of in the human domain as one and the same thing. In the more abstract domain of agents though there is a clear delineation. The agent's knowledge, defined by an ontology, encapsulates all the data and algorithms it needs to function, whereas the past experiences are a set of past perceptions about specific scenarios linked with the agent's observations about what those scenarios led to.

The goals and values determine what the agent will aim to achieve and what its priorities are. These can be encoded in many different ways and must be chosen to make the agent carry out its desired role, that is they must be designed to encode the specific goals and values of the agent but allow for these to change.

The current observations made by the agent are its observations of its environment. The environment that an agent operates in and how the agent and environment interact are as important as the agent design in terms of creating a successful system. The environment contains the agent itself and multiple artefacts that must be observed or interacted with. The environment can also include other agents which may be treated differently by the agent to how it would treat simple artefacts. It is this specialised interaction that allows for the development of multi-agent systems with another layer of control, namely control over the agent's actions by utilising their societal interactions. This society must also be designed and tuned like all of the other aspects of an agent architecture to try and ensure the successful completion of the agents tasks.

The high number of different domains and design choices that can be performed make a comprehensive formal description of an agent difficult without generalising too greatly. A good starting point is the definition by Franklin and Graesser [8] which states that:

"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda so as to effect what it senses in the future"

From this general but comprehensive definition of an autonomous agent four key aspects of an autonomous agent can be described.

- Reactive: the agent can respond to changes or stimuli from its environment, itself or other agents.
- Autonomous: the agent has control over its own actions.
- Goal oriented: the agent acts purposefully to execute its goals
- Temporally continuous: the agent is a continually running process in whichever environment it is present.

Another definition by Wooldridge and Jennings [9] splits the definition into two parts, that of a weak and a strong notion of what constitutes an agent.

The idea of a weak agent in this case requires that the agent be:

- Autonomous, that is it can operate without external control and has control over its internal states.
- Social, it must be able to interact if necessary with other agents, other systems or humans.
- Reactive, the agent can perceive other agents and its environment and respond in a timely fashion to any stimuli.
- Pro-active, the agent can not only react to the environment but must exhibit goal directed behaviour.

The stronger notion introduces concepts like knowledge, belief and intention. These higher level concepts that are more commonly associated with describing human decision making processes can be used to describe certain classes of agents. Attaching these human like attributes to agents can be of benefit when trying to mimic human style intelligence. In the control engineering and robotics fields these higher level constructs may lead to more complex designs of the agent.

More usefully for our purposes are some other concepts introduced in the notion of a strong agent. These include mobility which means the given agent can move within a given organisational structure or network, this is however not to be confused with physical mobility. This ability introduces many new possibilities when it comes to designing an autonomous distributed multi-agent system. Other concepts such as agent veracity (how accurately an agent's communications reflect the truth) and benevolence (if an agent acts for the other agents benefit) are important but in in this work these are assumed.

As the number of definitions of exactly what an agent is has increased there has been a concerted effort to try and develop a more formal definition based on mathematics and formal logic. When defining agents in these terms we can draw upon a wide range of the work on logic and the nature of decision making in domains such as philosophy. [10]

Drawing on this work Woodridge and Jennings chose to represent an agent as an intentional system. An intentional system is described as "an entity whose behaviour can be predicted by the methods of assigning belief, desires and rational acumen" [11]. Again this provides another way to describe more complex agents and multi-agent systems but does not provide specifics for how to design an agent in any given domain.

The main idea to be taken from all the myriad possible ways of describing an agent is that at a high level many of the concepts may seem nebulous but importantly all these aspects of an agent's structure are linked regardless of what we chose to call the structures. The fact that we can define and group structures within an agent system allows us to tune the agent to a specific job. For example an agent that operates as part of a large population will have to have its architecture focused on communication and the ancillary functions required for successful communication such as negotiation and error checking. If an agent however works primarily alone and is required to perform complex reasoning tasks then describing it in terms of beliefs and desires makes more sense as it allows us to succinctly describe its key features. When designing a domain specific agent architecture the correct combination of architectural structures which best describe the agent and its task without over complication must be chosen.

The concept of an agent has evolved over the last decades into an important tool in artificial intelligence and in computer science as a whole. The key reason for this is its flexibility and novel approach for designing software systems.

Brenner [12] stated that an intelligent agent can be characterised using the following properties.

Internal Properties - Characteristics that determine how it acts:
- Autonomy
- Learning
- Productivity
- Goal-Orientedness
- Reactivity
- Mobility

The idea of autonomy is a key difference between an intelligent agent and a program. The level of autonomy however, will vary greatly, depending on the application and how agents differ themselves within a multi-agent framework. The other internal characteristics are self-explanatory but learning can more usefully be expanded to include both learning and reasoning.

External Properties - characteristics that determine how it interacts with other agents are:
- Communication
- Cooperation
- Coordination

The real challenge in developing multi-agent systems for deployment in real world applications is taking the theoretical description of the agent and implementing it. Agent architectures are designed to bridge this gap and construct working systems that satisfy the relevant area of agent theory. The agent architecture has to deal with lower level concepts than the initial theory may gloss over. An agent architecture must describe the internal structure of the agent and how information enters, leaves and is processed in the agent as well as determining the state of the agent at any given time. An agent architecture has been defined by Maes [13] as:

"A particular methodology for building agents, it specifies how the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. An architecture encompasses techniques and algorithms that support this methodology"

Unsurprisingly there are as many agent architectures as there are theories describing agents ranging from the highly abstract and general to domain specific examples. These intelligent agent architectures can be broadly grouped into three categories that will be expanded on in Chapter 2.

1.3 Chapter Summary

In this chapter the concept of an agent was explored. The idea of the agent operating on both internal and external problems was explored. The basic types of agent architectures were outlined in order to give a framework from which the novel agent architecture is developed in the later chapters. The distribution of key properties of an agent (intelligence, knowledge, decision making and communication were also discussed,

This chapter acts as a broad introduction and acts as the foundation for the discussion of the aspects of multi-agent systems to spacecraft in the following chapter.

1.3.1 Structure of Thesis

Chapter 1 outlines the fundamental concepts of autonomous agents.

Chapter 2 discusses agent architectures and agent system architectures, that is, how agents can interact as part of a multi-agent system.

In chapter 3 current architectures are discussed and the concept of holonic recursive agents is discussed and finally the HASA (Holonic Agent Space Architecture) is proposed.

Chapter 4 describes the DARWIN mission and a HASA based multi-agent system (MAS) is developed for this mission. His chapter also details the multi-agent testing suite developed as part of this thesis and closes with the results of running a number of simulation on the DARWIN MAS using the multi-agent testing site.

Chapter 5 covers the GMES (Global Monitoring for Environment and Security) mission and the development of a HASA based multi-agent system to control it. The GMES mission itself is discussed in detail and the MAS outlined then further defined using the HASA architecture.

1.4 Contribution of thesis

This thesis makes a number of contributions to the field. The primary contribution is the development of a multi-agent architecture based on an extension of holonic and recursive agent architectures. This architecture is then utilised to design the autonomous multi-agent control systems for two missions, each with particular requirements from the architecture. In the development of the MAS for these missions the architecture is shown to be suitable for describing these systems.

1.5 Industrial Collaboration

The first period of this PhD research was undertaken in collaboration with GMV on the distributed agents for autonomy (DAFA) project funded by the European

Space Agency. This resulted in the co-authorship of two technical documents, [14,15]. The first technical document was tasked with giving the rationale for the development of multi-agent systems to space missions as well as creating a shortlist of possible missions that could benefit. The second technical document made the case for each mission on the shortlist to be taken forward in the DAFA project and a simple generic architecture was developed to be used as the project went progressed.

This simple architecture was used to develop a demonstrator for the GMES MAS and is described in [16-18]. In this thesis a new architecture was developed for space missions in general. A MAS was developed for the DARWIN mission based on this architecture. The basic GMES MAS as converted to this architecture and the Image selection algorithm for GMES was designed to work on top of the GMV developed MAS.

1.6 Research Outputs

1.6.1 European Space Agency General Studies Programme 06B34: "Distributed agents for autonomy"

- Technical Note 1 – Identification of Mission Scenarios
- Technical Note 2 – Identification of Distributed Agents Architecture and Selection of Reference Mission Scenario

1.6.2 Conference Papers

- Analysis and design of wsb transfers for the European student moon orbiter mission – D. Novak, W. van der Weg, G. Laguardia, S. Grey, T. Yang, M. Mercier.– 59th International Astronautical Congress, 2008 – Glasgow, UK.

- Design of a Multi-Agent System for Cost Reduction in Multi-Craft Space Missions – S. Grey, G. Radice, M. Vasile, Q. Wijnands – 60th International Astronautical Congress, 2009 – Daejeon, Republic of Korea.

- Image Selection Algorithm for GMES Mission – S. Grey, G. Radice, M. Vasile, Q. Wijnands – 60th International Astronautical Congress, 2009 – Daejeon, Republic of Korea.

- Design and testing of an autonomous multi-agent based spacecraft controller - S. Grey, G. Radice, M. Vasile – 61st International Astronautical Congress, 2010 – Prague, Czech Republic.

1.6.3 Journal paper in review

- Global Optimization Techniques in Multi-Agent Image Analysis - Journal of Applied Earth Observation and Geoinformation.

Chapter 2 Background

2.1 Agent Architecture

2.1.1 Deliberative architectures

The deliberative (logic based) architecture for creating agents is a classical approach to building a knowledge based intelligent system [19]. The deliberative architecture must contain a symbolic model of its environment and be able to reason based on logical concepts about its environment [20]. This reasoning can take the form of logical reasoning, symbolic manipulation or pattern matching or any combination depending on the architecture involved. This approach, at its simplest, creates a symbolic representation of the environment and reasons about it by syntactically, manipulating this model so that it can be thought of as a logical deduction. If logical theory is adhered to the whole process can be reduced to a case of logical proof. This offers many advantages when verifying an agent system which for many multi-agent systems is a key stumbling block to implementation. The downside of this approach is that as the environment and the agents representation of it increases in complexity then the logical deduction becomes more protracted and with dynamic environments the logical deduction can become impossible. The fact that a purely deliberative agent does not perform well in a dynamic environment as well as the fact that although a logical deduction can be made a useful result cannot be guaranteed within a certain timeframe limits the effectiveness of these architectures. The computational complexity involved with logical theorem proving for any non-trivial case within an acceptable time window means it is unlikely if it can be utilised effectively in practice.

The decision making process in deliberative agents is based on the assumption of calculative rationality [21, 22], that is, the assumption that the environment will not change significantly when a decision is being made and that the resultant action is still rational when the decision making concludes.

The deliberative architecture creates agents that can reason about their environment and their goals and then create definite plans to achieve these goals. The deliberation takes two forms. They deliberate about ends, that is, whether to attempt a goal, and means, that is how to achieve a given goal.

Structurally deliberative agents are not constructed as a monolithic whole but instead usually created from a set of components which may include elements such as planners, executors and knowledge bases.

In order for deliberation to take place there must be representations of both the world the agent operates in and representations of the actions it may take. This collection of actions and the world model is manipulated as a whole. This approach requires the ability to represent actions and derive the results of any given action using the model without actually performing them. The results of any deliberation in the agent are a set of actions that will conclude in the desired result.

One of the most common approaches to achieving a deliberative agent is to equip the agent with a reasoning engine, that is, the ability to reason. At its most basic that gives the agent the ability to make plans based on its knowledge to achieve its goals. The intention of an agent is expressed by the creation of a plan. If no plan is created then there is no intention of achieving the goal. Agents constructed in this way are described using the BDI (Belief, Desire, Intention) type architectures. In BDI agents we make the assumption that the entire context in which the agent operates and its environment can be modelled using mental attitudes which contain beliefs, goals, capabilities and rules. [23]

As previously mentioned however these mental attitudes will quickly become inaccurate in a dynamic environment and the environment may change to a point where they are invalid.

The beliefs component of a BDI agent may come from a number of different sources. For example the agent's beliefs can be influenced by input from a user, the actions of other agents (cooperative or uncooperative interactions) or feedback from its environment. Beliefs work by constraining the possible actions an agent may perform from the complete list of all possible actions. The newly constrained list of actions includes all of the actions that the agent may perform, with an associated possibility of success within a given time frame. This is important in the successful operation of the agent because it greatly reduces the possibilities for action and thus reduces the size and scope of the plans being generated which leads to quicker response by the agent. This in turn means that there is less chance of the environment changing in the meantime and thus invalidating the plan. This approach also ensures that any plans generated are relevant to its current situation and priorities [24].

The fact that beliefs can be thought of as both static and transient can introduce problems. Transient beliefs are defined as being largely dependent on the current state of the environment and thus are highly susceptible to change. In order to keep beliefs relevant transient beliefs must be re-evaluated at regular intervals. This interval depends on the change that makes the belief irrelevant. If beliefs are no longer valid then all of the plans and goals that are dependent on these beliefs must be reconsidered by the agent [25].

An agent's desires can be thought of as its goals, the new environmental state the agent desires to bring about or the future goals and tasks it wishes to undertake. In many cases the goals of an agent are supplied by a user. In the case of an autonomous agent however it may be able to create its own goals. These goals are created based on its current information about its environment, its internal states and the states of the other agents in the multi-agent system. An agent may of course have multiple desires and as with its beliefs, the desires will constrain the possible choices it may make.

An intention of an agent in the BDI architecture is a desire which the agent is committed to achieving. The commitment to achieving the desire is shown by the generation of a plan to complete the desire. The question arises about when an agent should reconsider its intentions and thus its current plans. One approach is to recompute its intentions at every opportunity. If reconsideration of plans and reasoning in general is computationally cheap then this approach makes sense but often reconfiguring plans after each action could make the agents response slower than the rate of change in the environment.[26]

The rate at which the agent reconsiders its intentions, environment, internal state, plans and goals is a key variable used for tuning a given agent to a given environment and scenario. A balance must be struck between creating plans and checking their execution. It is desirable that an agent drops intentions that are no longer valid and should thus re-evaluate its intentions. An agent that spends a lot of time re-evaluating however will not be able to spend much time actually carrying out its designated tasks. This dilemma is the key problem in balancing the proactive and reactive behaviours in agents. This problem was studied by Kinny and Georgeff [27] and they found that if the rate of world change is small then bold agents that do not stop to reconsider their plans will be more successful but in an environment that changes more rapidly the cautious agents that re-evaluate their plans will be more successful.

The key lesson is that different environments require very different decision making strategies. In purely static environments goal directed behaviour will produce a good result but in dynamic environments the need to modify intentions on the fly takes precedence. When an agent realises that due to an environmental change or otherwise that a goal is unobtainable then it should no longer pursue that course of action. Ideally the factor that causes the goal to be unobtainable should however not be created by the agent. In order to stop this case happening there must be incentives in place to achieve its goals and mitigate against the effects of actions on their uncompleted goals.

In summary the advantages of a deliberative architecture are that a correct and possibly optimal solution can be found using logical proofs as long as the problem can be successfully encoded. The disadvantages are that an accurate world model must be constructed (which is only possible in a well-known, or largely static environment) and that there is no guarantee of how long an action will take and thus it is hard to predict performance.

2.1.2 Reactive (Behavioural) Architectures

Reactive type architectures were developed as a clear alternative to logic based deliberative architectures. Reactive architectures forgo the logical representation of their environment and the idea of solving it through logical manipulation and proof finding. The "intelligence" of a reactive architecture comes from its extremely close link to its environment and that the desired behaviour emerges from the interaction between many simple behaviours and the environment. The reactive architecture is specifically designed to operate in a rapidly changing environment where a more deliberative architecture would struggle to perform adequately due to the time taken for deliberation and introspection.

Reactive agents have very few beliefs compared to deliberative agents and in many cases have none at all. Reactive agents also do not have goals but instead have a set of behaviours that are triggered by events in the environment (which includes other agents). The subsumption architecture developed by Rodney Brooks [28] consists of a number of reactive behaviours or action functions that are modelled using finite state machines. This architecture was developed to try to avoid the problems generated by the need to represent the agents' knowledge about its environment. In a reactive architecture there are no centralised functional models such as reasoning, learning, etc. such as those found in a deliberative architecture. Instead a given reactive agent consists of a distributed, decentralised set of competence modules (behaviours). These competence modules do all of the reasoning, learning, perception and representation required to achieve a specific behaviour.

Competence modules are all connected directly to the actuator or sensors that they are associated with and all of them run in parallel. This distributed parallel structure helps reactive agents overcome some of the problems found with deliberative agents. The close coupling between the reactive agent and its sensors/actuators means that a sensor input directly triggers an action in the agent without the need to create or query a representation of the problem. Reactive agents also produce a guaranteed fixed response to any given input which makes the system predictable and more robust. Due to the distributed nature of the agent architecture, multiple actions can be triggered by independent inputs in parallel (within the computational limits of the system). Multiple independent actions triggered by different perceptions can be combined into a single composite action. This approach however can introduce a number of problems, such as finding only local minima of problems and the introduction of cyclical behaviours. These problems can be mitigated against by introducing noise or other random inputs into the agents' behaviour which pushes the behaviour away from the local minima and breaks any cyclical behaviour.

With a distributed structure it is possible that multiple behaviours will be triggered by the same stimuli. Therefore the agent must have the ability to choose between a number of different behaviours if a conflict arises. Behaviours can be chosen based on how mission critical they are but other behaviours may just be mutually exclusive such as if they both require the use of a given actuator to perform different actions. Whether such blind response to stimuli shows intelligence is up for debate. This type of reactive response can be thought of analogous to reflexes and instincts in biological agents.

Reactive behaviours are modelled as condition-action logical rules. That is if a condition or set of conditions is met then an action is carried out. In this type of behaviour the agent keeps no representation of its environment or actions. This type of behaviour can be extended so that rather than only looking for inputs to meet conditions the behaviour also checks some state internal to the agent. This check of an internal state allows the agent to utilise data about its environment from the past or from other agents. To detect and represent changes to the environment in such a way we need rules and behaviours that set and change this representation rather than carry out an action on the outside world. Given the addition of a relatively small number of these "internal behaviours" that affect the state of other behaviours, much more complex external behaviours can be achieved. This structure allows the agent to remember its environment and recover from failures in other behaviours. The obvious drawback is that the maintenance of a symbolic representation of the environment is computationally expensive as seen in the deliberative case.

To summarise, reactive architectures are designed to react quickly to changes in the environment. If purely reactive responses are required then there is no need for a symbolic representation of the environment. Simple representations of the environment can be used and coupled with behaviours that act on other behaviours rather than solely on the external environment allow reactive agents to carry out complex tasks. The key downside to purely reactive agents is that no alternative plans are formed so that there is no consideration of alternatives or redundant actions.

In a behavioural architecture the solutions to any given problem must be coded in advance and all possible interactions between the interconnected behaviours of the system mapped. Reactive architectures tend to favour scenarios where the environment is observable (or at least partially) and dynamic. A reactive agent's goals are usually time dependant and have differing levels of utility and cost.

2.1.3 Hybrid (Layered) Architectures

As described so far, deliberative and reactive agents are at opposite ends of the spectrum where type of environment, dynamism and reasoning ability are concerned. In situations where the mission is at one of these extremes then a purely deliberative or reactive architecture can be deployed and any shortcomings are not exposed. Hybrid architectures were developed in order to try to combine the reactive and deliberative components within one agent structure. This is achieved by separating the deliberative and reactive components into different layers that operate separately but communicate with each other [29]. This level of abstraction allows for more complex agent models to be created and is flexible enough to be used to model agents in many different domains. Each function of an agent is decomposed into one of many layers. Reactive behaviours are the same as those in the reactive architecture and are responsible for tasks where a fast and robust response is required and are all collected in a reactive layer. The deliberative behaviours as outlined in the deliberative architecture are collected in their own layer and are charged with organising the sequencing the behaviours of the reactive layer. In deliberative and reactive architectures the main design issue is the design of the behaviours required to carry out any given mission. In a hybrid architecture the interaction between the layers must be designed and managed as well.

A common approach is to insert a third layer between the reactive and deliberative layers called the planning layer which mediates between the low and high level functions. In this case we have three layers: the deliberative layer, the planning layer and the reactive layer. A perception subsystem takes the inputs from the agents' environment and feeds the required data to each of the three layers. An action subsystem takes the actions from all of the layers and applies them to the environment. Other subsystems may be added depending on the usage scenario for the agent that act as checks and balances to this structure and aid in integration with other systems.

The middle planning layer in this structure, or executive layer, has a number of tasks. In order for the agent to operate, the planning layer must be able to handle tasks such as task decomposition, task scheduling, task allocation, synchronisation, execution monitoring, exception handling and resource management. The planning layer takes high level plans or goals from the deliberative layer and decomposes them so that they may be interpreted and carried out by the correct reactive components as shown in Figure 2-1.

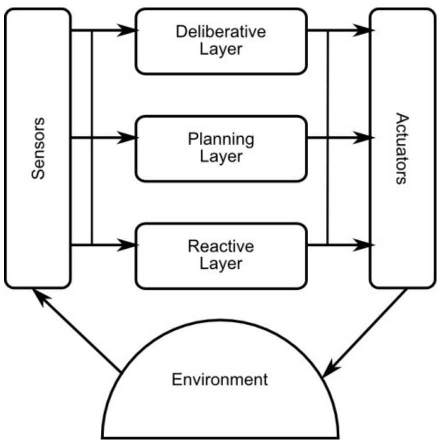


Figure 2-1 Basic layered agent

Layers can be arranged either vertically or horizontally. In a vertically layered structure inputs and actuators are processed by at most one layer each. In a horizontal structure each layer is attached to the input and output of the agent as shown in Figure 2-2.

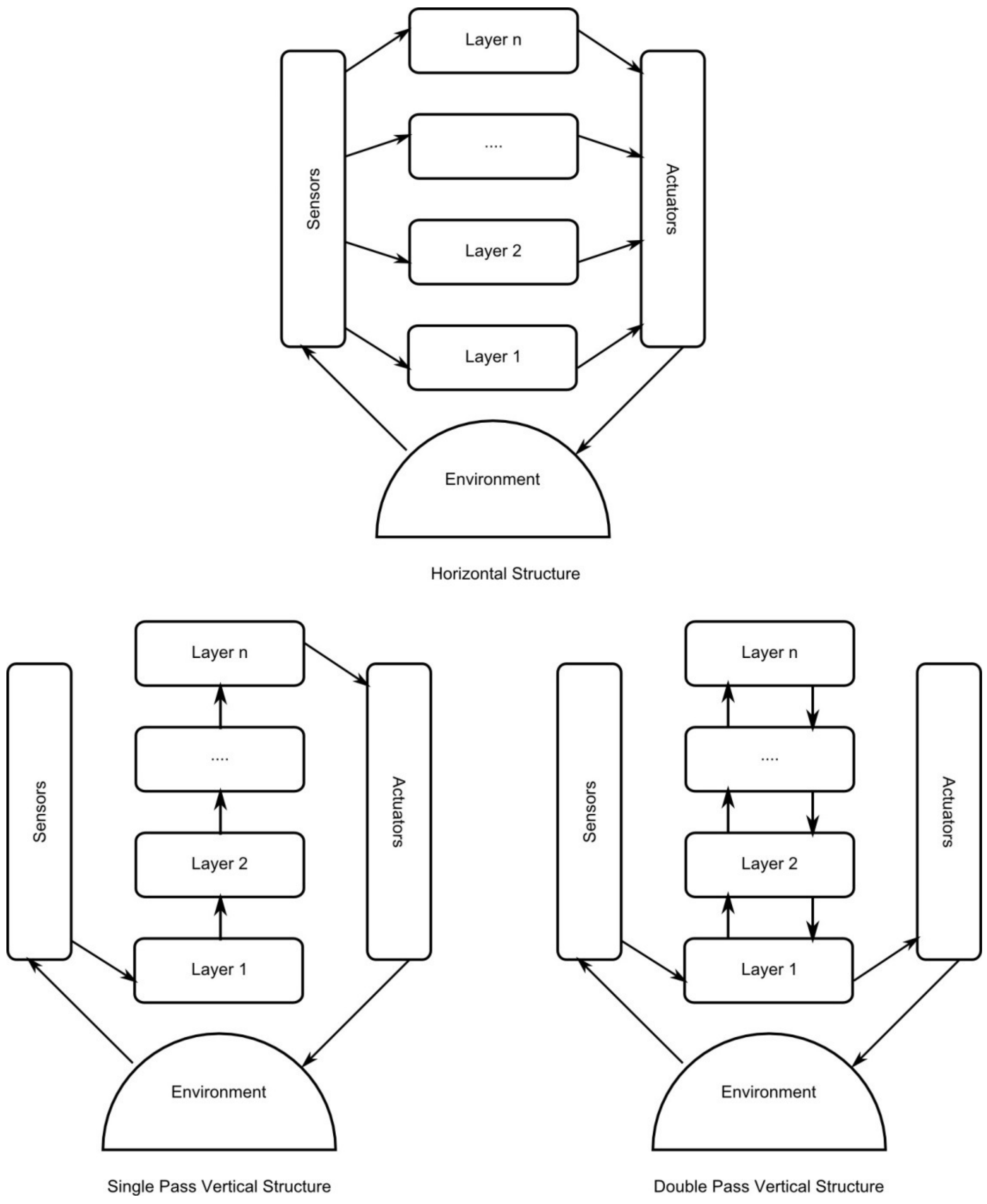


Figure 2-2 Various layered agent structures

In horizontal layering the behaviours of the agent take the input from the environment and create some sort of output, whether they are deliberative or reactive. In the vertically layered structure the inputs enter on a given layer and messages are passed to other layers. This allows for sophisticated structures to be constructed. For instance orders can be sent from deliberative layers to reactive layers to carry out specific tasks or sensory information can be sent from reactive layers to deliberative layers to be analysed. Examples of different layered agent configurations see Figure 2-2.

The key advantage to horizontally layered hybrid architectures is their conceptual simplicity. To ensure that the behaviour of the agent is coherent a mediator function that monitors all of the layers and prevents any clashes or unwanted events must be implemented. This approach requires knowledge of each layers' possible interactions with every other layer and as such the complexity of the mediator function increases exponentially with the number of layers.

The problems associated with constructing an adequate mediator function are potentiality alleviated by using a vertical structure. Vertical architectures can be split into two broad types: single pass and double pass architectures (Figure 2-3). In a single pass architecture, control flows sequentially from layer to layer until the final layer produces some sort of output. In the double pass architecture, information flows sequentially up from the bottom layer to the top layer where decisions about actions are made. The actions to be performed are then passed down back to the bottom layer where the actions are actually carried out. In both the single and double pass approach the complexity of the interactions between the different layers is reduced but at the cost of higher chance of failure if one of the layers fails as message passing could be disrupted.

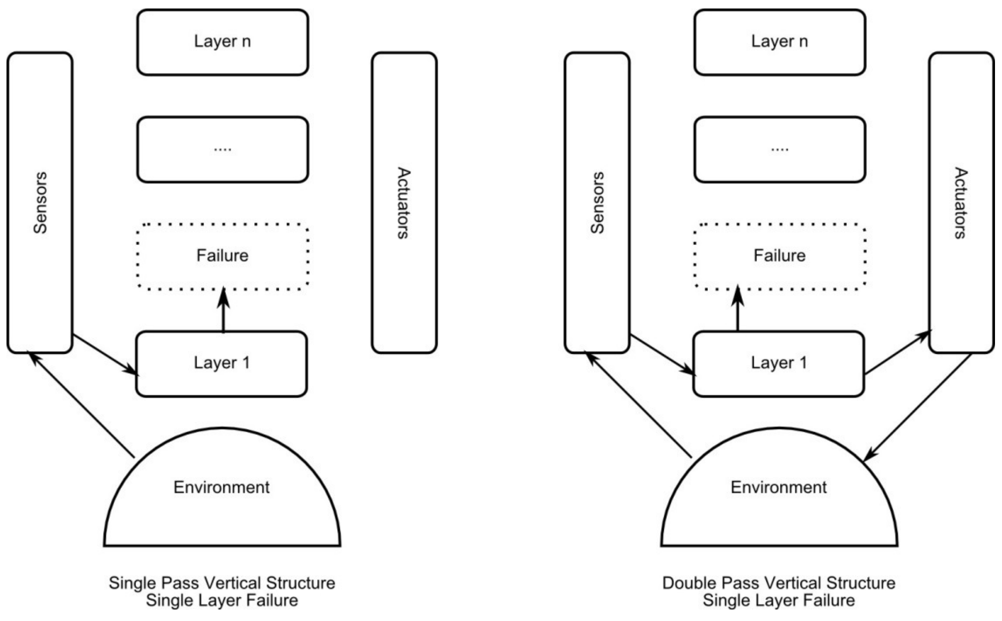


Figure 2-3 Failure within a layered agent

2.1.4 Decentralised Layer Control

In decentralised layer control all of the layers operate concurrently and independently acquiring inputs and outputting actions. In hierarchical control the layers operate sequentially with tasks coming from higher deliberative layers to the reactive layers. In concurrent control the layers operate concurrently but may influence the layers immediately adjacent to them (Figure 2-4)

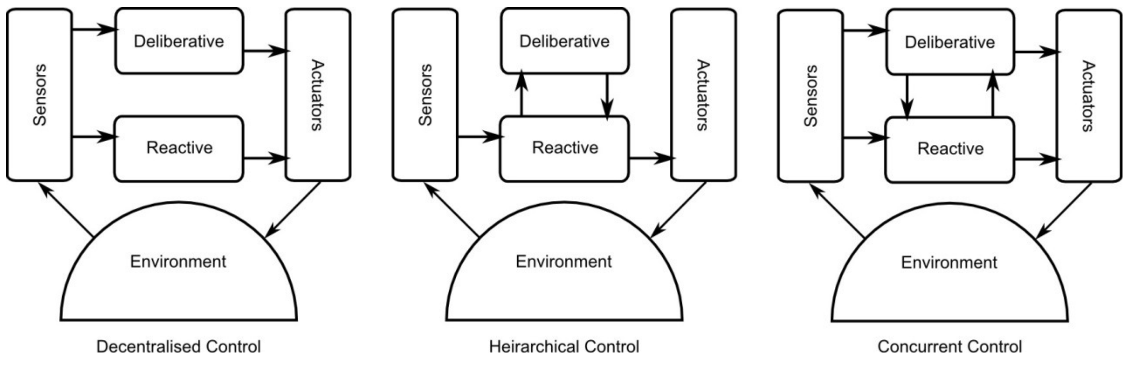


Figure 2-4 Different agent control strategies

The use of both reactive and deliberative layers brings with it challenges when it comes to representing data and synchronising time-scales as a deliberative layer may well have full representation of its environment but work on slow time-scales and a reactive agent may well have no representation of its environment and work on extremely fast time-scales. Regardless they must interoperate effectively.

2.1.5 Comparison of architecture types

Of the three broad agent architecture classes outlined above hybrid architectures are the preferred choice for many applications. The natural decomposition of the agent into a number of different reactive and deliberative layers means that a wide range of agents can be designed to meet any reactive, deliberative or social needs required by the designer. The trade-off required when using a hybrid architecture is that although they can provide a balanced solution to a specific design problem the semantic clarity of the purely deliberative and reactive architectures are lost. The use of deliberative layers also requires that complex models for the environment be made.

In summary, deliberative architectures can utilise elegant logical semantics to encode the agent's processes and its external environment. The use of logical semantics also allows any behaviour to be predicted as logical proofs can be found. On larger and more complex problems however logical reasoning takes up a lot of computational power and in particular the planning algorithms used by deliberative agents can be difficult to scale. The overall complexity of a deliberative agent can be larger than those of other types due to the reasoning and learning requirements and the need for a suitable environmental model. The environment that the agent operates in must be relatively static in order for the deliberative agent to be able to generate a useful environmental representation.

Reactive architectures can have very simple structures which are easier to understand and this in turn makes them easier to implement. This structural simplicity also means that reactive agents are on the whole much cheaper computationally. The fact that a reactive agent does not have a full model of its external environment means that it must therefore act only on limited local information which could lead to inaccuracies. Also due to the lack of a complete model of its surroundings and operating scenario its decision making will be on a relatively short term time-scale. The disadvantage of a reactive architecture is that there is very limited opportunity to implement any sort of reasoning or learning functionality other than that which arises from interactions between simplistic components.

Layered architectures benefit from conceptual simplicity, in part because the functionality of the agent is decomposed into multiple specialist layers which can be developed independently. The use of both deliberative and reactive components means that the hybrid agent still needs an accurate model of its environment. The use of many heterogeneous components adds complexity to the design and depending on the type of layer structure chosen there may be little scope for recovering from one layer's loss.

2.2 Distributed Agent Architectures

Previously we touched on the idea of multi-agent systems and that the interactions between agents can be a highly important aspect of a system's ability to operate effectively. We define a multi-agent system as a collection of agents operating in the same environment for some group outcome. Cooperation is not necessarily required and many problems are better served with a multi-agent system comprised of competing agents, particularly for problems involving finite resources, interaction with humans or where buying and selling in a marketplace is necessary. Agents operate as a collective whole to perform tasks that are outwith the ability of a single agent to perform. A task may be outwith the ability of single agent due to speed constraints or lack of complete knowledge of its environment. A multi-agent system must be specifically tailored to its domain; this is especially true in that multi-agent systems developed to solve single problems are very different from those developed to solve multiple problems. Single problem multi-agent systems tend to involve a significantly large data set and the multi-agent system carries out many complex analyses on that data set. Multiple problem domains, of which autonomous spacecraft control is one, are characterised by less need of analytical capacity but a more complex structure, hierarchy and grouping of agents. The groups within the multi-agent system and any layers within the agents can be treated like self-contained problem solving units and a key problem is effectively sharing information about the environment and solutions to problems between these logical units.

The key point to make is that groups of agents, agents and the components of agents can often be treated in a very similar way. This leads to a so called recursive agent structure [30] as shown in Figure 2-5.

In this structure the same terminology is used to describe all of these groups and it can be helpful to treat the sub-layers of an agent as agents themselves depending on what they are designed to do and treating groups of agents as a single autonomous unit can help to greatly simplify the description of complex multi-agent systems. Obviously the terminology to describe a recursive multi-agent architecture must be developed in order to clarify the structure rather than obfuscate it and this is by no means a trivial task.

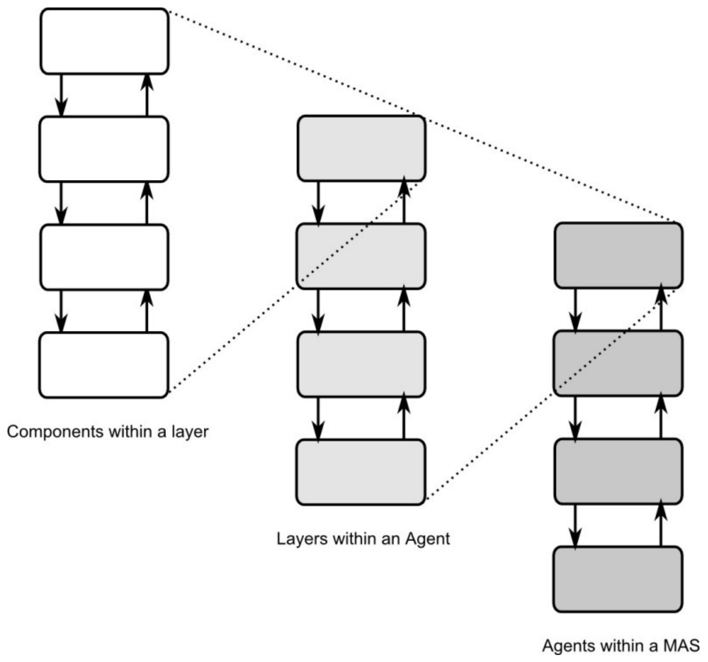


Figure 2-5 Recursive agent structure

There are of course other ways to group and describe agents within a multi-agent system and this means that a multi-agent system is inherently more complex that a single agent. A number of key questions must be addressed when developing a coherent and useful multi-agent system. The level of cooperation and why the agents need to cooperate must be addressed. Often this is due to insufficient computational resources or insufficient problem solving capabilities on a single agent. How the agents cooperate or compete must also be decided. This means making design decisions on how agents communicate, how specialised each agent is, how resources and tasks are shared, how actions are coordinated and synchronised and how any conflicts or clashes are resolved. In any MAS communication is a key factor and how messages are constructed and checked by agents must be resolved. The organisational structure of the agents must also be designed and this factor alone can have a wide reaching impact on the performance of the individual agents and the multi-agent system as whole.

Organisational relationships describe the different relationships that occur between agents and different classes of agents. If an agent has another agent as an acquaintance then that means that the Agent A has a representation of the other agent, Agent B. This representation or knowledge of the agent could just be simple identification data or a detailed representation of its internal systems and processes. A step up from being purely an acquaintance is actual communication between agents. For this to happen the agents must be at least acquaintances as they must know the other exists and agent A sends messages to agent B. An operative relationship means that an agent needs information from another agent in order to perform the given task, meaning to perform a task, agent A needs information from agent B. A subordinate relationship means that tasks are delegated by an agent, agent A, to another agent, agent B, to be competed. The introduction of subordinate relationships introduces a level of hierarchical control in the multi-agent system. Hierarchical control introduces a branching structure with agents passing tasks from root to leaves, that is, agents higher up the hierarchy may pass messages to one or more agents lower in the hierarchy.

The coupling between agents dictates the level that agent organizations can modify themselves. If the coupling between the agents is fixed then the agents' role and relationships will not change over time. If the coupling between agents is variable the relationships and thus the organisational structure can evolve over time although only within predetermined limits.

As well as the structure and organisational concerns, which task a given agent can perform within the group and how different tasks may be achieved is an important consideration in the design of a multi-agent system. Within this framework, specialisation can be thought to describe the number of actions an agent can perform in relation to all of the actions needed to be carried out by the multi-agent system. Redundancy indicates the number of agents that can perform any given task.

With a high level of redundancy and unspecialised agents we have a redundant generalist type organisation. In this organisation, agents can perform many actions and each action can be performed by many agents. With high redundancy and highly specialised agents we get a redundant specialist organisation where each agent can only perform a limited number of actions but many agents perform each action. With low levels of redundancy and unspecialised agents we get non-redundant generalist organisation where agents can perform many actions and each action is only performed by a few agents. If we have a high level of specialisation and low level of redundancy we get a non-redundant specialist organisation where each agent can only perform a few actions and each action can only be performed by a few agents.

Many multi-agent systems are designed to be deployed in dynamic environments where traditional architectures may be less successful. In order to operate in a dynamic environment the organisational structure of the multi-agent system must be flexible. The problem arises that as the organizational structure becomes more flexible then it becomes more difficult to predict what another agent will do.

To try and overcome this problem multiple coordination frameworks have been developed to make sure that agents within a multi-agent system interact effectively and operate as team to reach their complex team oriented goals. The main purpose of these coordination frameworks is to make sure that the agent's plans do not conflict with each other while ensuring the successful pursuit of goals, both individual and societal.

There are two broad strategies for ensuring coordination in a system like this. In the first method there is full collaboration between the agents in pursuit of the common goal. Rewards are shared between agents and resources are shared to try to maximise group performance. In the second method the agents work for their own individual gain, even at the expense of others and most interactions involve conflict resolution, bidding for resources or tasks and negotiations.

Regardless of the organisational structure chosen, the ultimate objective of any multi-agent system is to work towards a set of goals by achieving globally coherent behaviours. Key to making a system like this efficient is the ability of any given agent to reason about the actions and state of other agents within the system.

How agents are grouped to carry out tasks, how intelligence and knowledge are distributed through these groups, how decisions are made and how these decisions are distributed within the system contribute to the overall effectiveness of a multi-agent system.

2.2.1 Levels of Distributed and Local Intelligence

Within a multi-agent system the level of intelligence of the individual agents can vary greatly depending on the need for reactive type or deliberative behaviours and agents. The desired level of group intelligence however can be achieved in a number of different ways using different levels of individual agent intelligence and different distribution strategies. The level of intelligence of one agent (local intelligence) and the level of intelligence of the group as a whole (distributed intelligence) has a large impact on the way decisions are made and the time-scales in which they are made. This distribution affects how goals and tasks are assigned as well as performance criteria such as response time and redundancy.

Agents can be extremely simplistic and only react to a single input from the environment or a specific prompt form another agent but in a well designed multi-agent system with a coherent and efficient society, high levels of intelligence can emerge as the carefully choreographed interactions between the agents enable high level operations.

Agents that have the capacity to plan are a level above the simple reactive agents and can be thought of as locally intelligent in they have at least some knowledge of their surrounding environment and that they use this knowledge to formulate their plans of action. Interactively intelligent agents can utilise the knowledge of others within the system to generate their plans, giving them access to knowledge outside of their own local group.

Agents can not only generate plans for themselves but also for other agents. This ability is key to many multi-agent systems and especially the master-slave architecture. In a master-slave architecture a single agent has direct control over a number other agents and commands are only ever sent from master to slave, as shown in Figure 2-6.

If the decision making and command ability is instead spread throughout the multi-agent system then the architecture can be described as distributed. There are a number of different variations on the distributed architecture. A common approach is to have a peer to peer command hierarchy where commands may travel to any combination of peers, as shown in Figure 2-6.

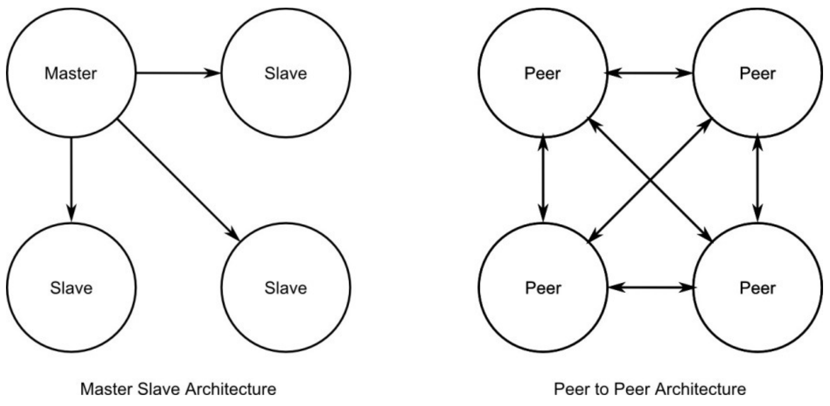


Figure 2-6 Communication strategies

2.2.2 Communication Process and Architecture

All multi-agent systems, be they purely tasked with information retrieval or controlling autonomous robots have the need to communicate, with each other and with any users. Many attempts have been made to standardise the methods for this type of communication, agent communication languages (ACL).

A Defense Advanced Research Projects Agency (DARPA) knowledge sharing project led to the introduction of the Knowledge Query Manipulation Language (KQML) [31]. KQML is a language that is designed to facilitate the exchange of knowledge and information within software systems. This need to transfer knowledge is what differentiates KQML and its ilk from simpler and more widely used data exchange protocols. The language was initially developed for the construction and operation of large software knowledge bases which could be easily shared and reused. It was however, soon re-purposed as an agent communication language. KQML is both a message format and a communications protocol that allows agents to communicate and share knowledge in real time between each other and users. KQML can be used as an application layer to allow users to interact with an intelligent system (or agent) or for two or more intelligent systems/agents to communicate with each other.

KQML gained popularity, especially in academia but was eventually superseded by the Foundation for Intelligent Physical Agents (FIPA) agent communication language that was designed from the ground up to describe both how agents would interact with each other and how they would be executed in an agent platform. The FIPA standard has been employed in a number of agent platforms and was made a formal Institute of Electrical and Electronics Engineers (IEEE) standard in 2005.

The FIPA-ACL is based on speech-act theory [32] which states that messages represent actions. These actions can be communicative acts and are known as speech acts or performatives.

There are four base performatives in the FIPA ACL:

- Inform an agent of a proposition
- Request an action of an agent
- Confirm a belief
- Disconfirm a belief

As stated earlier the ability for agents to communicate with each other effectively has a large bearing on the performance of the system as a whole. The content of the language must be suitably expressed in a content language such as the FIPA- Semantic language (SL) [33] or FIPA-Knowledge Interchange Format (KIF) [34] and then encoded to allow it to be effectively transmitted. There are two other mandatory structures for an agent system to meet the FIPA-ACL standards. The agent system must have both an agent directory and a service directory accessible to the agents. The agent directory is a shared information repository where each agent publishes data about themselves to allow for effective communication. The service directory is a repository in which agents and other services can find applicable services. Such services could include message transport services, agent directory services and application specific services.

When describing an agent's communication languages there are also other features that must be taken into account. Most agent communication languages like the one above are prescriptive, that is they describe actions to be undertaken by other agents but they could also be descriptive and be based on describing what is occurring.

Any meaning within the agent's messages must also be as unambiguous as possible and as such conventions for messaging must be put in place. Any subjectivity in the messages sent or received must be accounted for and this is frequently done by introducing societal norms to the system [35]. These societal norms are in place so that all of the agents know how to react to any given message and as importantly how others will react to any given message.

The pragmatics of the communication, that is how it is structured, must also be considered [36].

The system must also be developed to take into account that messages may not be received and interpreted in isolation but are instead received in the context of the environment, other agents and other messages. The language used by the agents must also be large and complete enough to meet all of its requirements but still be of manageable size.

2.2.3 Autonomy and Representations

The nature of autonomy is described with great clarity in [37]. We talk a lot about agents and autonomy but our definitions must be clear to avoid confusion as the architectures and different types of constructs we design become more numerous. Agency as used in our domain describes a system whose actions lead to some other state in the world. The goal oriented action can however cover a wide range of systems from the complex such as a human to the relatively simplistic such as a chemical compound reacting with an oxidising agent.

Obviously we have to further refine our definition of an agent, and this can be done by having an internal representation which stores its goal and may have the ability to self-regulate. "Self-regulated agents are goal governed agents, who given a certain goal are able to achieve it by themselves: planning, executing actions, adapting and correcting actions" [21] This definition more succinctly describes the type of "cognitive" agent that we will be working with but it must be noted that for any agent definition there will always be counter intuitive systems that meet the description but may not to our eyes seem to meet them, such as a simple thermostat more or less meeting our previous requirements of a basic cognitive agent. This description of an autonomous agent is building towards a usable definition for our domain but still has limitations and must be further refined. In our case an agent is *not truly autonomous unless it has autonomous goal setting abilities*. The impact that autonomy has on the definition of an agent is vast and the definition of autonomy must also be set before work on designing an autonomous agent system begins.

At its most basic level autonomy of a system is defined by its relationship with others, that is, a system can only be autonomous with respect to another system. Agent autonomy is frequently defined as being able to carry out its own wishes but for clarification we will use the term executive autonomy for this kind of autonomous nature. *Executive autonomy defines the ability to achieve goals by itself*.

So if an agent is to be autonomous this autonomy must be defined by its relationships to some other system. The autonomy may be from some physical system or structure or it may be from some other agent or agent structure. We can now look at our agents and try to decide if they are autonomous by the physical or societal context. We can think of some interesting examples of agent autonomy by thinking of a generic multi satellite mission which is controlled by a distributed autonomous system. It can be envisaged that most of the agents will be societally autonomous from the other agents in the system but this intuitively will almost always be the case, otherwise why would we not just make a monolithic system?

When we talk about autonomy in space missions we mostly mean that the system is autonomous from ground control. An autonomous systems does not always have to act alone however. In this case the autonomy is both societal, as the system is autonomous with respect to the human agents on the ground and physical as the system is autonomous with respect to the physical connections and data from the ground as well. Agents within a distributed spacecraft missions can also be physically autonomous in another way. The agents may not be tied to a particular piece of hardware and are thus able to move from spacecraft to spacecraft or node to node. This gives the agent the ability to change its environment and operate independently of certain hardware constraints.

Agents must operate within some defined environment (even if the definition changes) and as such they are limited to operating and being autonomous within this environment. An agent must therefore operate at some level based on the environment it is situated in and cannot be autonomous if it receives no inputs from its environment at all.

An interesting issue arises however in that although the relationship with the environment is vital to the overall functionality of an autonomous agent it may not directly affect it. That is by operating in an environment the environment may influence the agent but not cause its goals to change directly. This is known as the Descartes problem [38], how do we ensure that our autonomous system is neither entirely dictated by the environment (non-autonomous) but also not oblivious to the environment (non-situated).

2.2.4 Social Autonomy

We previously discussed the fact that agents can be socially autonomous, that is they operate independently of other agents. This concept is key to multi-agent systems but must first be further defined to avoid confusion. One definition of social dependence is that the agent is dependent on another agent; that is it relies on another agent to operate. This level of dependence will vary greatly between agents but it can thought that each agent within an operational multi-agent system will be dependent on many others otherwise our time would be better spent developing multiple monolithic systems. This way of looking at social autonomy basically equates to self-sufficiency and can be quantified by looking at how many of a given agent's actions are dependent on other agents.

Another definition of social autonomy is the relationships and interdependency between the goals of the agents in the multi-agent system. This definition makes the goals of the agents the key driver for autonomy which makes sense for a space based systems, as goals must be generated, coordinated and distributed throughout the system in order for it to operate effectively. How the goals are spread and the relationships between them define the social autonomy of the system.

2.3 Transition from a traditional to an agent based architecture

An approach to transitioning to multi-agent systems must be carefully considered to allow for a smooth modular transition from traditional architectures. Having a more structured progression or roadmap from traditional to agent based systems would hopefully increase adoption and acceptance of these systems.

A simplified roadmap for this progression can be described using the following two steps. Firstly we can enhance the traditional architecture. We would achieve this by replacing components in the architecture with agent based components but still use the traditional interfaces. In this case components can be swapped out and improved while still allowing for verification using the tools that already exist for that architecture. As more components are replaced there should be provision for allowing agent based interaction between these replaced components to enhance performance gains. Most importantly, at this stage not all of the components are replaced so mission critical or problematic components can be left as is and the system will still operate, hopefully at a higher level due to the inclusion of agent based components. Some of the initiatives in this respect are covered by [39, 40].

The second step is then to transition to goal based mission operations which fully exploit the social and intelligent features of the agents to again increase performance over traditional systems.

Appendix A - Traditional Spacecraft Control Structure describes in more detail the traditional approach and structure of a spacecraft control system.

2.4 System Architectures

In the discussion of multi-agent systems and multi-agent design the term "agent architecture" can sometimes have a number of different meanings and connotations. When working with multi-agent systems we are frequently looking at both the structure of the agents themselves and the structure of the multi-agent system as a whole. To try and avoid any ambiguity we will use the term “agent architecture” to define the internal structure of the agents and "system architecture" to describe how the agents are organised and interact with each other to create a cohesive multi-agent system.

A number of different agent paradigms have been introduced. These paradigms include deliberative and reactive agents and the whole spectrum of hybrid agents in between. The different approaches to autonomy are extremely helpful in trying to define different types of agents but there are other classes of agent that may be better described by their internal organisation. These internally defined agents offer more options and potential considerations for the agent architecture we want to define for autonomous space missions.

2.4.1 Internally Defined Agents

There are five main categories of internally defined agents, Modular, Subsumption, Blackboard, Production System and Layered. The simplest and conceptually the most simple is the modular architecture [41].

2.4.1.1 Modular Architecture

In the modular architecture the agent consists of multiple modules that each correspond to a particular action associated with that agent, be it planning, negotiation, perception, execution etc.

2.4.1.2 Blackboard Architecture

The blackboard architecture as introduced in [42] was designed for use in single intelligent agent systems. The basic idea behind the blackboard structure is that there is central store for all of the data needed by the agent. The blackboard contains all of the current system states and solution data. This blackboard approach has been applied to multi-agent systems where a central data store accessible by all the agents but independent of all of them may be desirable but the problems of a single central point of failure means that a more distributed system may be more desirable.

2.4.1.3 Production System

The production system approach [43] takes its structure from the system developed for controlling industrial production processes and has an agent acting as the production system within the larger multi-agent system. The production system approach offers some intriguing possibilities. Looking at our distributed multi-agent system as a collection of products and processes allows us to design a MAS tuned to the safe and efficient production of our desired product, scientific data.

2.4.1.4 Layered Architectures

Layered architectures [44] allow for a mix of reactive and deliberative components within a single cohesive whole. Typically the more deliberative components are at the top of the hierarchy and commands flow down towards the more reactive components and data or results flow back up to the deliberative components to inform their future decisions.

2.4.1.5 Subsumption Architecture

The subsumption architecture [28] consists of a number of finite state machines grouped together on layers. These finite state machines are purely reactive and a key part of the subsumption architecture is that there is no environmental model in the agent. Instead the environment itself acts as the model as the agent only reacts to the world around it.

2.5 Types of System Architecture

The internal agent architecture can be taken by itself to help to define any given single agent or single agent system. Whenever two or more agents are working in concert or in competition however, then great care must be taken to design their interactions so that they can effectively carry out their tasks.

Multi-agent systems are defined by their hierarchy, be it homoarchical or heterarchical. In the homoarchical case (Figure 2-7) there is only one possible way that the agents may form a hierarchy whereas in the heterarchical case (Figure 2-8) there are many different possible hierarchical structures the system can take depending on its circumstances, the task in hand or its operating constraints.

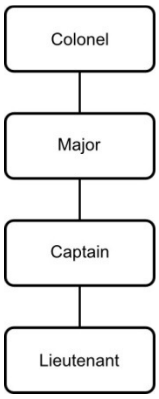


Figure 2-7 Homoarchical structure

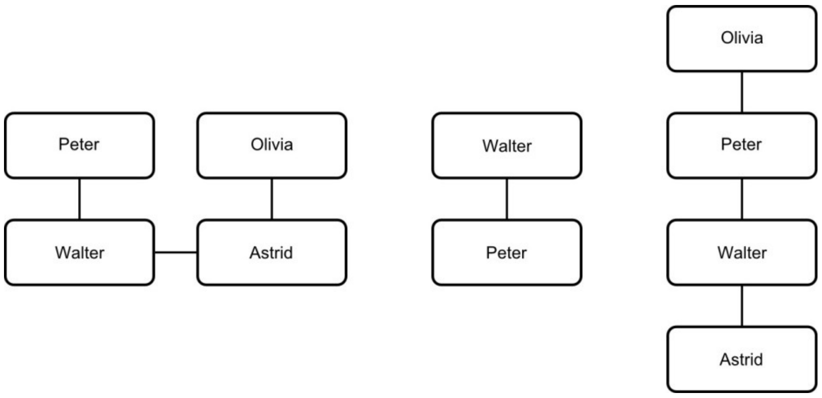


Figure 2-8 Heterarchical structure

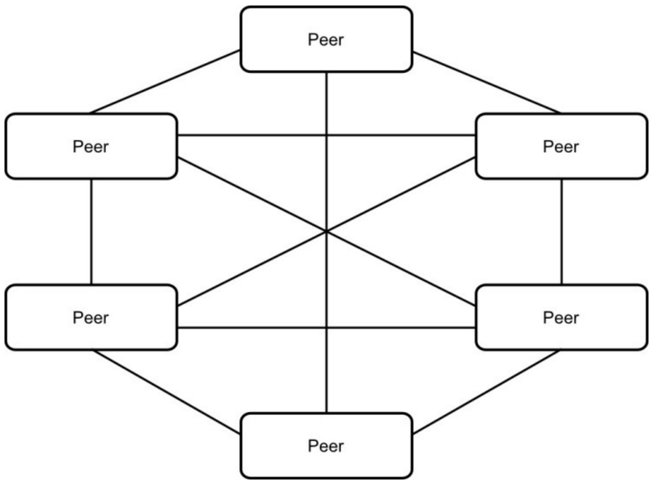


Figure 2-9 Non-hierarchical structure diagram

Much of the early work on heterarchical or non-hierarchical systems such as Figure 2-9 (although heterarchical is not the same as non-hierarchical they do share many of the same features) was based on distributed data processing which led to work on cooperation between agents and then to distributed sensor networks [45, 46, 47] This work led to many communication protocols, of which the Contract-Net is probably one of the most well-known [48].

The benefits of a non-hierarchical or heterarchical approach to distributed computing are mainly in flexibility and redundancy in the case of changing conditions or component failure. There are disadvantages to this system architecture however, most importantly there are issues with heterarchical systems not being able to find global optima for any given problem space and as the systems become more complex and the possible interactions between them more varied the system as a whole can only be predicted at a very high level. This reduction in predictability means that these systems are not seen as suitable for mission critical tasks or processes.

The hierarchical or homoarchical systems lie at the other end of the spectrum. The strict nature of the hierarchy and its inflexibility means that it is more predictable and therefore easier to coordinate. The disadvantage of this type of agent system architecture however is that it is somewhat inflexible and in return for increased coordination there is a reduction in the reliability gains from redundancy in the heterarchical designs. In order to try and overcome the limitations of operating at either end of the heterarchical/homoarchical spectrum a set of compromise architectures have been developed, collectively known as the federated architectures [49].

These federated architectures (Figure 2-10) tend to have no central data storage and instead operate through the passing of messages between agents. This is different however from a purely distributed approach as there are specific agents tasked with undertaking very particular roles in these system architectures. There are four main types of federated architecture and we will discuss each, they are the facilitator, broker, matchmaker and mediator approaches.

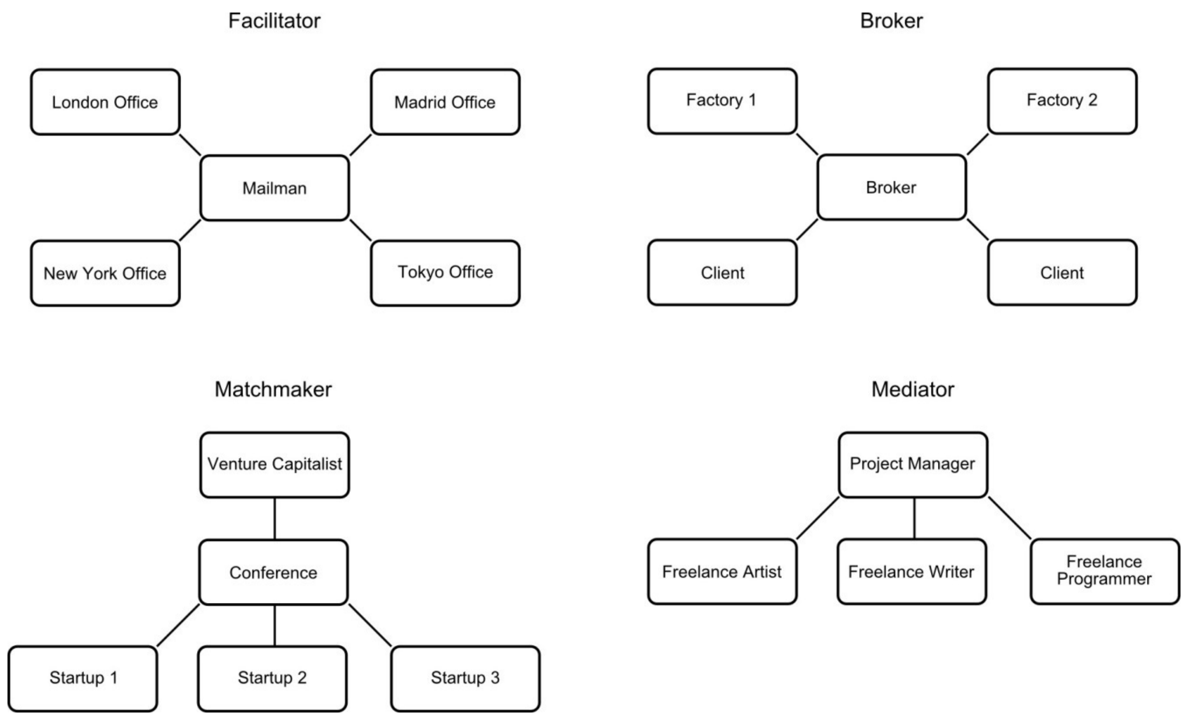


Figure 2-10 Federated Architectures

The facilitator approach uses facilitator agents that enable communication and coordination between agents. The key aspect of this system architecture is that agents may only communicate to each other though the facilitator agents and no direct communication is allowed. This constraint allows for a much more controlled system while still being able have the benefits of a distributed system [50]. The facilitator agents do not just pass messages on to other agents but also check the messages, translating them if necessary ensuring that all of the agents operate as an efficient whole. This facilitator approach is found in the GMES agent architecture design, but in the GMES case it takes second place to the broker system which is much more suited to the transactional requirements of GMES.

The facilitator approach, while enabling the communication between agents to be carried out in a separate layer, does not inherently have the capacity for more complex societal structures. In the facilitator approach a single agent talks only to a single facilitator agent who takes on the responsibilities for communication with the other facilitators. In some societal structures that are particularly suited to use in multi-agent systems a many-to-many communication approach is desirable rather than one-to-one. A broker agent is an example of this many-to-many communication strategy. In a broker approach to multi-agent systems there a number of broker agents which act as facilitators. The agents in the multi-agent system advertise their services and make requests through any number of brokers. The broker agents then match the requests to the services as best they can. Broker agents can include functionality to not only make direct matches between services and requests but also to serve as a mediator in negotiations between services and requests. This can also involve the instigation of collaboration between agents to meet a particular request or to create combined requests that make better use of the advertised service.

The matchmaker approach is similar to the broker framework but instead of acting as the middle man for all communications between the agents it brings together the agents making the requests and offering the services then leaves the agents themselves to carry out any required negotiation. This approach is best suited to systems where there is less negotiation and collaboration between agents and where the performance of the system as a whole may be improved by the removal of the superfluous intermediary agents. The matchmaker approach can lead to the creation of a yellow-pages type structure. In this case the services of the agents are advertised in a single location and the agents making the requests all look in this location for their desired services. The mediator approach is aimed at better coordinating the actions of the agents in the multi-agent system. The mediator agent uses the techniques of brokering and matchmaking to try and create groups of agents that collaborate within the sub group or cluster. This clustering is carried out during execution and is not strictly predefined in the system level architecture. Once the mediator agent has created the desired collaborative cluster it must then ensure that it operates in an effective manner. This will involve more complex behaviours such as mediation to reduce or combat deadlock scenarios between agents in the group and to increase the operating efficiency of the group where possible.

Many modern agent systems incorporate aspects from many or all of the federated structures as well as having a mix of reactive and deliberative agents. Real time distributed control applications lend themselves to multi-agent systems that have a wide range of components which add intelligence to the system. The intelligence arising from the interactions between the agents on a societal level is where a multi-agent system differs from other intelligent control techniques.

Chapter 3 An Agent Architecture for Space Missions

3.1 The need for a new architecture

3.1.1 GMV Architecture

The DAFA architecture design document [51] from GMV describes the architecture and implementation of a MAS for GMES. The DAFA project was focused on building a proof of concept for a MAS for an ESA space mission. The architecture that is discussed in this work is highly implementation focused and does not try to describe the MAS in any consistent way that may allow for reuse in other space missions. This is an extremely common approach. The architecture itself is basically the default generic architecture used by the SeSam [52] software that is used in the implementation of the MAS. As shown in the implementation of the DARWIN MAS SeSam can be used to implement any architecture but in the DAFA GMES case no real architecture is specifically outlined or used. This lack of a clear and definitive architecture was a key reason for developing a new multi-agent architecture in this work to allow future DAFA like projects to be more easily described and for common elements to be transferred.

A discussion of the generic architecture used for the completion of the DAFA project by GMV can be found here [15].

3.1.2 General Architectures

A look at a number of multi-agent architectures follows and their suitability as a basis for a space mission architecture is discussed.

Alami [53] posits a general architecture for autonomy comprising two layers, a decision layer and an execution layer. Each layer can be composed of a number of sub-layers depending on its application. The actual architecture is a basic layered structure and the authors description of an architecture is in fact a description of the tools that were developed to implement the architecture. This is a key distinction that has to be made. An architecture describes the structure and relationships required by the system but does not specify explicitly their implementation. This architecture does utilise the idea of a generic module that is then specialised. This use of templates is useful as it allows for flexibility and heterogeneity between component parts but forces the modules to have certain similarities. The idea of a specific executor module/agent/layer is also a useful one and allows for the system to be decomposed based on its operating time frame as well as its role.

The Cougaar architecture [54] consists of a collaborative workflow of coordinated agents. Again the architecture is relatively simple, this time based on a blackboard for communication and agents that accept a number of modular plugins. The line where the architecture finishes and the implementation begins is especially blurred in the case of Cougaar. The designers have created a very capable multi-agent framework based on a simple architecture but the architecture itself is still very general and relies on the addition of extra components in order to be specialised to a particular task or problem.

The work by Criado [55] extends the BDI architecture to allow for the use of normative contexts, primarily with the aim of agents acquiring norms from the environment. The work offers a concise description of the architecture but its generality makes it hard to apply this to the very specific requirements of a space mission.

Of the more general architectures in the literature the ANA architecture [13] best encapsulates the approach to architecture definition that is required for real progress with autonomous agents in space. Although it is relatively general it still describes a novel structure for agent interaction and the agents themselves.

Another problem with current MAS architectures is demonstrated in the Hilaire [56] paper on adaptive agent architectures for holonic multi agent systems. The paper makes a compelling case for the use of a holonic structure as the basis for an agent architecture. The specifics of the architecture however are based on the conceit of describing the system in terms of an immune system. This is a common problem and the terminology used only works to obfuscate an otherwise very powerful architecture.

3.1.3 Robotic Architectures

Architectures developed specifically for robotics can offer insights into the requirements of how autonomy and actuation can be handled within a multi agent architecture. The CLARAty architecture [57] for instance is typical in that it encapsulates a layered autonomous design and makes the development of autonomous robotics platforms more standardised. The problem with using an architecture like this one in multi-agent systems is that any communication and cooperation between agents and robots has to be added on as a an extra layer rather than being included and integrated at an architectural level.

3.1.4 Space architectures

In the Ghallab paper [58] they make a case for the suitability of the LAAS architecture as described in the Alami paper for space missions. Again the architecture strictly defines the implementation of its components and the underlying structure and does not take into the account the special requirements of a distributed space mission. This focus on implementation and individual autonomy means that the architecture could be implemented fairly rapidly for a given mission but it might not give all of the benefits of an architecture developed with distributed space systems in mind.

The AGATA architecture outlined in the Verfaille paper [59] describes a highly modular architecture based on four key components, state tracking, object tracking, decision making and decision execution. The architecture take a very control oriented approach and makes the system highly modular. The downside of the approach is again not taking the opportunity when designing a multi-agent architecture to really incorporate all of the advantages of multi agent systems such as communication and cooperation. The architecture does include a robust description of the decomposition of the agents into smaller and smaller modules which enables the MAS to be viewed at multiple levels of abstraction.

Much of the work on multi-agent architectures for space has been highly specific to a single task within the overall space mission. An example of this is the OCAMS project [60] which designed and created an agent architecture for mission control. A robust architecture was developed but it is only really applicable to mission control tasks and could not be easily adapted to other tasks.

3.1.5 Manufacturing architectures

The architectures developed for the manufacturing industry, especially the holonic architectures [61, 62], offer some innovative approaches to coordinating and optimising multi-agent systems. The concept of the product being encapsulated as an agent with its own autonomy is one that has not been implemented in the space domain but offers potential for a logical and flexible structure as all of a space missions science and formation tasks can be thought of as products. Adopting some of the paradigms from the manufacturing architectures enables a new space architecture to make the most of the manufacturing sectors focus on reliability and modularity. Unfortunately there are number of reasons that the architectures cannot be used wholesale. The environment of the manufacturing MAS is basically static when compared to that required by a space mission. That is, there are discrete inputs and outputs to the system but the environment that the agents operate in is not continuous or liable to fundamentally change. The manufacturing architectures also take a very different view of real-time control. In the manufacturing architectures the actual actuation and control of the equipment is handled by the controllers of the equipment themselves whereas in a space mission the real time control must be handled by the agents in the MAS.

3.2 Holonic Control Architectures

A holonic control system consists of a number of holons, each holon is identified as a discrete autonomous agent, in a similar way to an agent as previously described. Holonic Systems have a recursive structure so that they can be thought of as an agent architecture as well as an agent system architecture [63]. In this way holonic systems are very similar to certain multi-agent systems and in some respects can be thought of as a subset of multi-agent systems, or at least another way to describe these autonomous distributed systems. The individual holons within a holonic system collaborate to achieve a common goal. The key difference between a holonic system and software agent system is that the holons themselves can be decomposed into an information processing part (which is akin to the software agents in a typical software agent MAS) but also a physical processing part. This close coupling of software and hardware makes holonic architectures very well suited to describing systems charged with distributed control of physical devices. These physical devices can vary in complexity and have varying software control requirements, the flexibility of the holonic architecture enables us to model a variety of space missions.

Holonic systems make use of functional decomposition in order to divide a larger system into many more simple parts. The observation was made in [64] *“systems evolve much more readily from simple systems if there are stable intermediate forms than if there are not"*, this means that if the system has many different layers of aggregation then a complex system can be easily built up from simple components in subsystems which are themselves complete. Since the holonic architecture is built on the idea of decomposition of a problem into smaller parts that have themselves purely informational or informational and physical components we are also able to define holons as comprising of a number of sub-holons that still meet the full definition of a holon. This recursive structure allows for the system to be fully defined at a number of different levels using the same nomenclature and provides us with many stepping stones as we build our autonomous control system.

Holonic systems are a bottom up approach to system control but as they can be thought of a subset of multi-agent systems that have a bias towards physical integration many of the techniques and architectures that have been developed for multi-agent systems can also be applied to holonic systems.

Certain strategies however have been developed specifically for holonic systems. Cooperation in holonic systems can be achieved by using a "cooperative domain" (CD) [65]. The CD is defined as "a logical space in which holons communicate and operate, that provides the context where holons may locate, contact and interact with each other". Two different types of cooperation can occur within this CD. Simple cooperation means that a given holon must respond to the requests of another holon (even if the response is negative). Complex cooperation is where holons work to achieve a joint goal.

In the complex cooperation paradigm the higher level deliberative software level must be sufficiently integrated with the lower level physical devices(s). This can be thought of as the classic distinction between high level and low level control. In the low level control domain, functions are kept as simple as possible with the view to reducing response times as much as possible. It is also at the low level that processes or actions can be simply automated to negate the need for higher level control over certain functions. This low level of control is based around the optimal use of any given hardware component and a great deal of highly applicable work has been carried out in intelligent process and manufacturing control of low-level hardware that can be successfully applied to the control of spacecraft hardware. The higher level control will, by its very nature, be more specialised to its particular domain. However, although its implementation may differ by domain the key tasks of ensuring inter-holon cooperation, collaboration and negotiation will be found in most intelligent controllers. As the holon control architecture can be thought of as a subset of agents as a whole we can use the standard agent descriptions and definitions, such as FIPA to define the deliberative layer of the holons. There are many international standards for defining the low level control aspect of the holon although the majority of these are designed for use in manufacturing and process systems. They can however act as a good starting point for defining an agent architecture for any application.

3.2.1 Standards

One of the benefits of looking at the process and manufacturing domain for inspiration is the desire for these systems to adhere to standards. The standards in industry are in place to ensure interoperability and to provide a level of confidence as to any given system performance. They also provide a series of requirements that a system must meet to not only be safe but also to operate effectively in the desired environment. For instance the International Electrotechnical Commission (IEC) 61499 standards for distributed industrial automation have the following as requirements and many of these can be directly applied to a distributed autonomous space control system with the exception of the intellectual property based requirements.

- Component-Based
- Support encapsulation/protection of Intellectual Property (IP)
- IP Portable across Software Tools and Runtime Platforms
- Distributed
- Map IP modules into distributed devices
- Integrate IP Modules into distributed applications
- Functionally Complete
- Control/Automation/Diagnostics components
- Machine/Process Interface components
- Communication Interface components
- Human/Machine Interface (HMI) components
- Software Agent ("Holonic") components
- Extendable
- Encapsulate new types of IP
- Create new IP through Functional Composition of existing IP modules
- Multiply the value of IP through widest possible deployment
- Benefits available to all market players

3.2.2 High-level vs. Low-Level Control

In [66] Deen provides an architecture that links the high-level and low-level control layers. The PROSA (Product Resource Order Staff Architecture) reference architecture ([62]) uses concepts designed for the manufacturing applications and is based on the idea of creating products using resource and ordering holons. This approach is applied to scientific space missions in the architecture developed in this work where a product must be created based on user requirements while being closely observed and controlled autonomously.

3.2.3 Requirements for intelligent control

In [67] Balasubramanian outlines four key requirements for a distributed intelligent control architecture (in this case for manufacturing but this also applies in the more general case). Firstly the control system must be able to operate, at least to some level, in real time and must be able to satisfy deadlines with real-time constraints. The control system must also be distributed, have a consistent communication strategy and be consistent across physically separated nodes. The control system must be able to supply support for event driven control so that it can react autonomously to the occurrence of events. The controller must also offer intelligent control in the form of reactive behaviours at lower levels and deliberative behaviours at higher levels.

The common theme of the distinction between the higher level software functions and the lower level hardware can be seen in the frameworks outlined and there are many more agent/holon based architectures that show this layered structure. As such the designs for the multi-agent architecture will be layered and will be fully outlined in section 3.3.

3.2.4 Reconfiguration of Multi-agent Systems

Reconfiguration has to occur whenever the hardware or software of the system being controlled changes. From the software viewpoint a multi-agent system can be seen as a highly dynamic system and in many cases change will be desirable and should be able to be accomplished with minimal adverse effects. In the cases of space missions which can be comprised of multiple spacecraft then the hardware can change as well. This ability to reconfigure hardware can be useful in a number of ways. In the GMES mission the spacecraft are highly heterogeneous and operated by different bodies. It can be expected that spacecraft will be added to the system as well as retired. The multi-agent system has to be able to handle all of these possibilities. In the DARWIN mission it is expected that the hardware will be largely homogeneous between the space telescopes, the only exception is the collector hub that will differ from the other telescopes. A hardware level reconfiguration can still occur in this scenario though as failures of specific modules may require a reconfiguration of the system to make up for the lost hardware. Distributed telescope missions such as Darwin are inherently expandable and it can be envisaged that in such a mission the number of telescope satellites could be increased to enhance the scientific return. Any multi-agent control system should be able to handle this addition of new hardware.

The work of Labeyrie [68] in particular outlines the possibilities of a great many interferometric telescopes together in a large coordinated formation. These “hyper-telescopes” would contain up to hundreds of separate craft but offer unprecedented imaging abilities and have a high level of redundancy within the formation.

The reconfiguration of software, especially as it consists of independent autonomous entities will be much more common. The ability for multi-agent systems to reconfigure is one of their main strengths and can be a natural product of their ability to share knowledge and tasks and create local groupings to provide a required service or make a request. In more traditional software control systems reconfiguration can require the whole control system to be replaced, even if the change is only to a small part. If the control system implements a truly object oriented approach then this can be avoided to some extent by simply replacing the objects that need to be changed but this still requires the control system to be shut down, the changes inserted and then restarted. This forced restart can cause many problems on an operating platform. In a multi-agent system however a new or modified agent can be uploaded to a given node and accepted as part of the multi-agent system. Even if this step requires a restart, the other nodes will continue to operate and once the updated node is back on-line the new agents can propagate through the multi-agent system. This desire to keep the system online as much as possible is another benefit of using aspects of manufacturing control systems in our architecture.

Early work such as that by Kramer [69] postulated that reconfiguration should be split across a number of operating layers. In this way the reconfiguration commands are introduced at a high level and then decomposed by the system itself at a lower level. This lower level decomposition is handled by the components or at least groups of components. This approach is taken because it is assumed that in a hierarchical structure problematic interactions may only become apparent at particular levels of abstraction so a step by step implementation of the reconfiguration process allows the system to check for possible problems with the reconfiguration. In this early work reconfiguration was decomposed into multiple "change transactions" that specified the creation and removal of nodes and relationships within the system.

For the development of reconfiguration ability for our space mission control system the ideas of separation and the implementation of layers are used. A multi-agent system (whether it is based on the holonic model of coupled physical and software entities or not) is easily made into a layered structure as it is by its very nature comprised of many different components working on different levels of task. It is therefore desirable that any reconfiguration ability works with this structure rather than against it. One possible approach is that taken by Leveson [70] where the different requirements of system control and system recovery are catered for by the separation of the control components and the recovery components into independent layers in the system. This approach makes sense as it separates the top down control sequence where commands are sent towards the lower reactive levels for execution from the bottom up safety process where faults and clashes are sensed at a low level first and then propagated up to the deliberative layers.

3.3 HASA

The holonic architecture developed as part of this thesis is an extension of and reworking of the PROSA architecture [62]. The PROSA architecture is a holonic agent architecture developed for manufacturing systems. Its central paradigm is that of creating products based on customer orders. The whole manufacturing and control system is modelled and the PROSA architecture offers a generic holonic architecture that does not proscribe any hierarchy. The PROSA architecture was chosen as starting point for the Holonic Agent Space Architecture (HASA) outlined herein for a number of reasons. The key drivers in the manufacturing systems that PROSA was deployed to control are safety, extensibility and autonomy but above all else the ability to reliably create a given product. These concepts are also key drivers in many scientific space missions. The PROSA architecture is very general architecture and as such acts only as a starting point for the development of the HASA architecture.

The fundamental differences between HASA and PROSA are:

- Replacement of the order holon in PROSA with an executor and planner holon in HASA to enable the better modelling of the structure of space missions.
- An emphasis on real-time control rather than the completion of orders.
- Continuous rather than discrete time operation.

Thinking about space missions as a manufacturing process can also give us added insight in how a multi-agent controller may be designed. It is quite logical to think of any given space mission as creating a product. Often the product will be data of some sort, most likely scientific readings or images. The product could also be movement of the spacecraft or the provision of a certain resource in the case of communication link. As such, in the HASA architecture, a product can be defined as belonging to one of three groups, data, actuation or resource.

There are four basic types of holonic agent in the HASA architecture as shown in Figure 3-1. They are planner agents, executor agents, resource agents and product agents.

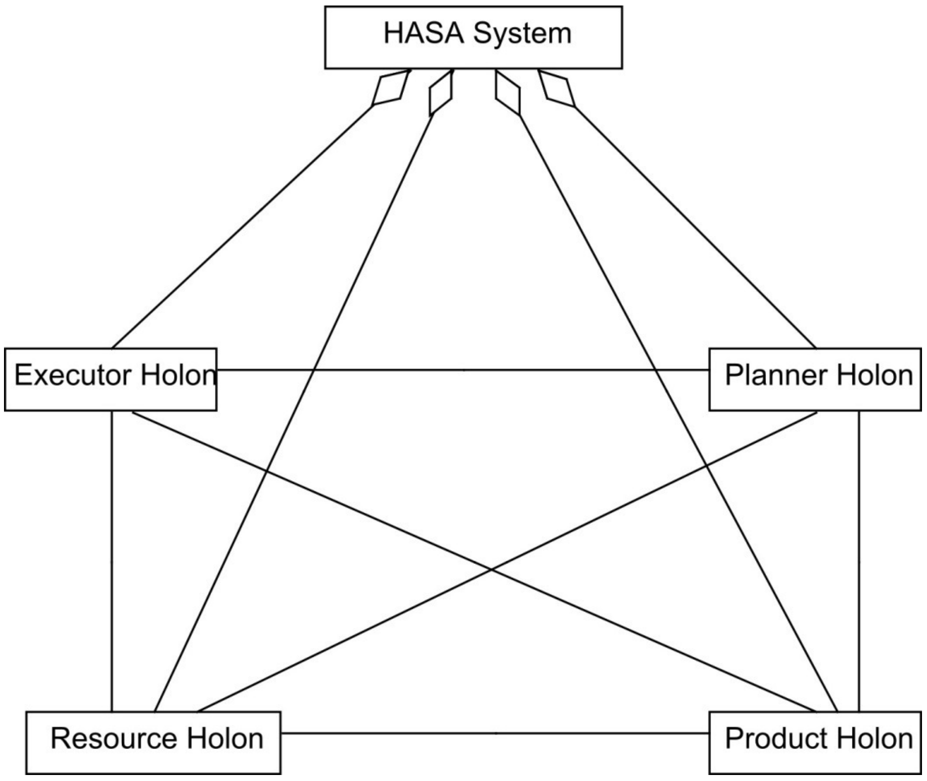


Figure 3-1 Basic types of holonic agents in HASA

At its most basic level the resource agent consists of a resource that is used to construct some product. An executor agent monitors the creation of a product and the planner agent decides where and when any given product will be created. The product agent encapsulates a model of the product, how it must be created and to what specifications.

A resource agent can take many forms. It can be thought of as a traditional resource, as in a computational resource to be utilised or a resource that is expended, such as energy. The resource could also be an actuator or a sensor or at a higher level a specific spacecraft in a formation. It should be noted that the key concepts of planner, executor, resource and product apply at many levels of aggregation of the system and this vertical self-similarity or recursive nature of the architecture is a key benefit of the holonic architecture.

A planner agent is an abstraction of the deliberative aspects of the systems. Again it is used to describe entities at many different levels of aggregation so a planner agent could be an agent deciding how to schedule and organise the acquisition of a certain formation or at a much lower level, deciding what data to pass on from a single sensor.

An executor agent is an agent that manages the execution of processes in real time. The processes to be undertaken will be decided by a planner agent and the executor agent will work with one or many resources to carry out these processes as dictated by the planner agent.

A product agent encapsulates what exactly a given product is and how it is to be produced. This can include processes and required resources as well as sub-products it needs to create its product. The holonic nature of the HASA architecture is fully recursive so a product can quite validly be made of many sub-products, each of which is a fully operational product agent.

3.3.1 Aggregation

A number of different agents will be grouped together in a given hierarchy dependant on what product they will be constructing or generating. This hierarchical group can be thought of as a bigger agent with its own identity. The recursive nature of the architecture is here the key element and allows us to view the systems as a whole over many different levels of detail (Figure 3-2). In the HASA architecture the hierarchy is one where an agent's membership to an agent group is not fixed. Agents can belong to many different groups and due to the recursive structure each group is comprised of sub-groups. Aggregate agents may be specifically designed but scenarios can be envisaged where aggregate agents may emerge as a result of the self-organisational abilities of some agents. The multiple levels of aggregation apply to the four types of agents in the architecture where, for example, a given planner agent can be thought of as consisting of a number of smaller planner agents undertaking certain tasks, some executor agents overseeing the planning process and resource agents representing the knowledge or algorithms needed to create the plan. A product can be thought of as consisting of a number of sub products, each of which may consist of other products, until the product can be fully described.

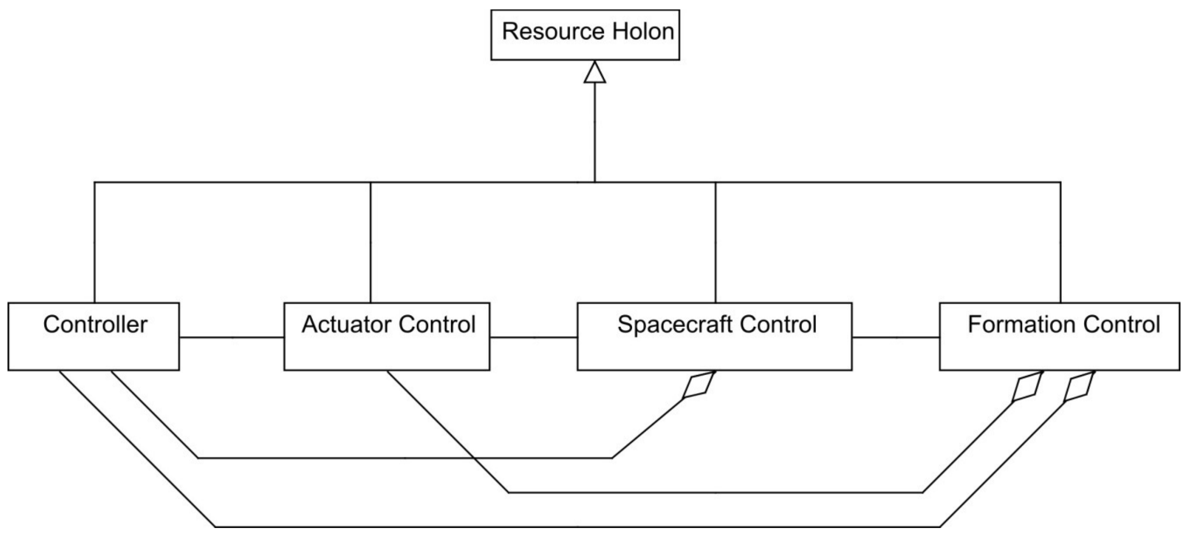


Figure 3-2 Aggregation of holonic agents in HASA

3.3.2 Specialisation

The four core types of agent can also be sub divided into a set of more specialised agents, defined by their individual characteristics and the sub agents that they are composed of, as shown in Figure 3-3 which represents the attitude control system of a spacecraft and Figure 3-4 which shows a resource holon.

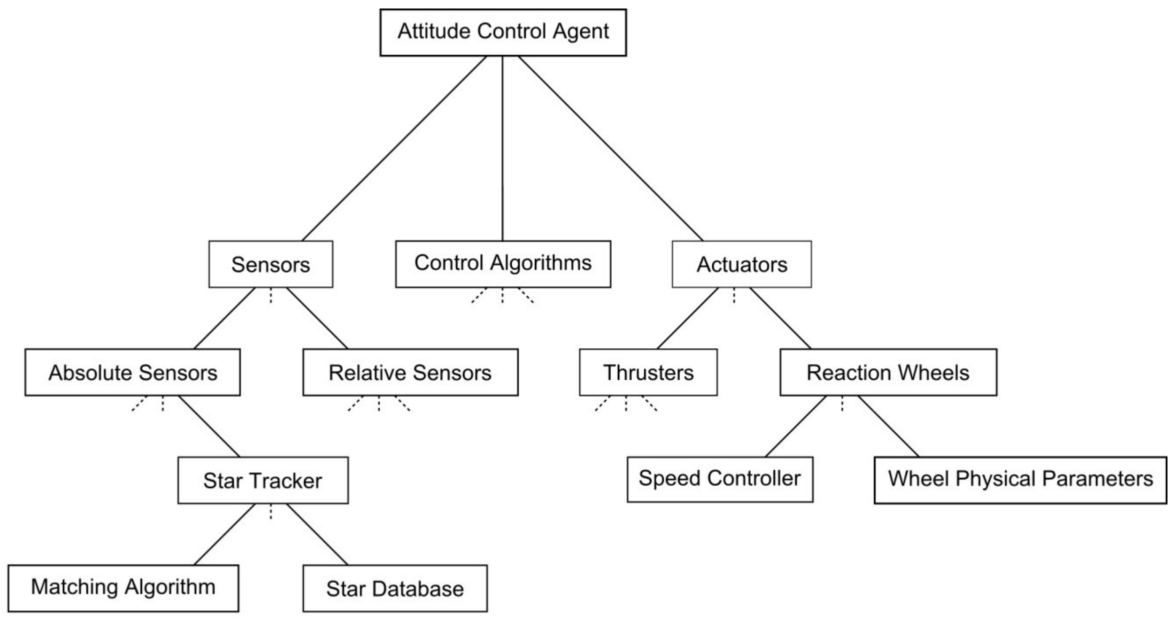


Figure 3-3 Possible sub-agents of an Attitude Control Agent

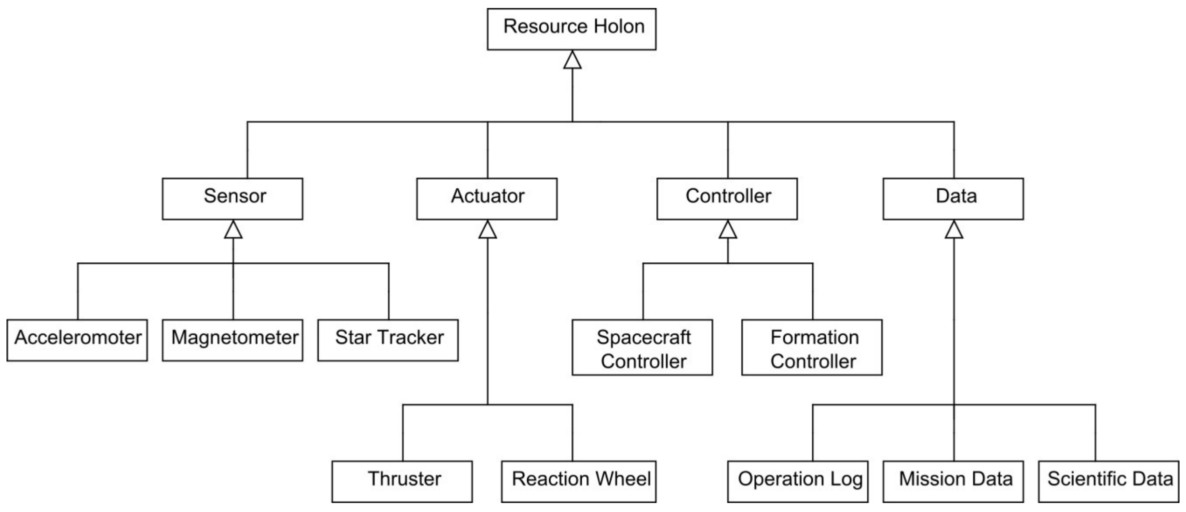


Figure 3-4 Specialisation in holonic agents

3.3.3 Data managed by basic agents.

A planner agent’s key data structure is a plan. This is defined as set of processes that lead to the creation of a product. As the planner agent encapsulates the deliberative aspect of an intelligent system. Characteristics such as goals are also stored and manipulated in a planner agent.

An executor agent's key data structure is a set of processes. These may be the set of processes currently running, the set of processes need to create a product or a set of processes carried out in the past. The executor agent will also contain knowledge about each process that it has undertaken, is undertaking or plans to undertake. This allows the planner to use data on past performance and predicted utilisation when creating a plan for the present.

A resource's key data structure is a set of limits or rules to its use. This could be used to describe a finite resource's capacity or the rules that govern the physical actions of an actuator.

A product's key data structure is the model of a product and its associated quality requirements. This product model can take many forms. If the product is something like a scientific image, the product model will include all of the requirements needed for that image such as time constraints, resources and target data needed. If the product was a manoeuvre for the formation then the product model would encapsulate the dynamical model of the formation and the individual spacecraft, thus allowing a specific product to be created.

3.3.4 Functions performed by basic agents.

The planner agent creates and manipulates plans based on the state of the current resources and running processes, the executor agent executes these plans using the available resources and the state of other processes.

3.3.4.1 Knowledge exchange

- The resource and planner agents exchange product knowledge. That is how a product was create/achieved and if it was fit for purpose.
- The planner and executor agents exchange plan knowledge.

- The executor and the resource agents exchange process execution knowledge. That is how to achieve a given process.
- The resource and the product holon exchange process knowledge, that is, how to perform a process on a resource.
- The planner and the product holon exchange product performance knowledge. This is based on the quality metrics from both the product and the planner.
- The executor and the product holons exchange production knowledge. That is how to produce a certain product.

Figure 3-5 below shows how knowledge moves through the system and which types of holon produce and consume what type of knowledge. This in turn shows the key structures of the HASA architecture.

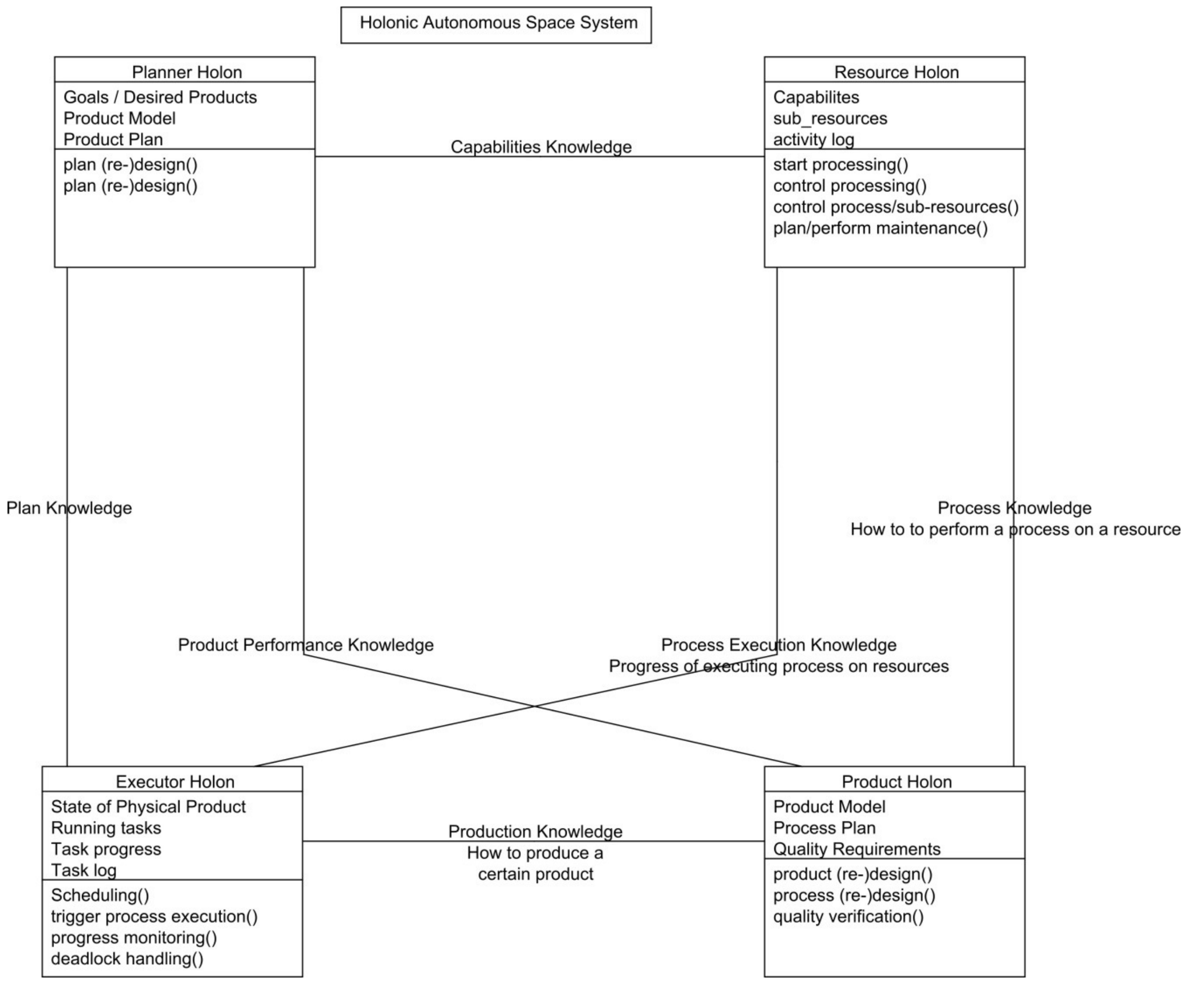


Figure 3-5 Functions and Knowledge exchanged by each holonic agent type

3.3.5 Self-Similarity in HASA

Horizontal self-similarity means that at a given level of aggregation all of the agents of given core type are similar to each other. This means that they operate in a standard way using standard interfaces. This allows for complex structures to be designed but still permits standard interaction. This self-similarity applies to the key communication strategies and data structures of the agent but is not designed to hinder specialisation.

Vertical self-similarity means that the basic types of agents share common aspects regardless of the level of aggregation. This means that the multiple smaller planning agents that go to create a single larger planning agent will all share the same methods of communication and data structures. This allows for the recursive nature of the HASA architecture.

3.3.6 Agent Environment

As discussed previously the environment of a multi-agent system is the key factor in its operation. In the HASA architecture the environment is a specialised type of resource agent that can be utilised to aid in creation of products. Manipulation of the environment is classed as a product in this architecture.

3.3.7 Benefits of the HASA architecture

The key benefits of the HASA architecture when compared to the other architectures available are:

- **Recursive**: The holonic structure allows for a complex system to be described at varying levels of abstraction, allowing for development of the system using a top down or bottom up approach. This recursive nature also facilitates the use of external modules in the system as they can be easily described in the architecture.

- **Real-time**: The specific executor agent handles real-time operations and acts as a bridge between the more deliberative and reactive aspects of the system and is rarely found in the manufacturing architectures.

- **Product-oriented**: An emphasis on creating a viable product is paramount for space missions and this concept has never been used in the robotics/space domain previously.

Chapter 4 Multi-agent system for DARWIN

4.1 Introduction

In the previous chapter the design of a multi-agent architecture was discussed. In this chapter a design for a multi-agent control system using the HASA architecture for a formation flying mission is developed. This is for two main reasons, firstly to show that the architecture can be applied to a wide range of missions and secondly to show that a control system, that requires a set of relatively complicated simulation models, developed using this architecture can be adequately tested.

In the first part of this chapter the DARWIN mission will be outlined as well as the current expected baseline for its level of autonomy, formation flying in the domain of space missions will be discussed before the design of a multi-agent system to control this type of mission is described in detail.

The design of a multi-agent testing suite is developed and the testing suite implemented in order to test the multi-agent control system. The multi-agent system described herein is an extension of the work presented here [71]. A more comprehensive look at the ESA missions that were considered for multi-agent control can be found here [14].

4.1.1 The DARWIN mission

The DARWIN mission concept offers an excellent example of a formation flying mission that could also benefit from on board autonomy to improve its science return and operational reliability.

The DARWIN mission's primary purpose is to detect and characterise Earth-like extra-solar planets (exoplanets) and search for signs of life in the exoplanet's atmosphere. This is to be achieved by the use of a formation of interferometric space telescopes. DARWIN would operate in the mid infra-red band and by using nulling interferometry between the telescopes be able to compensate for the massive differences in light intensity between the exoplanet and the star it is orbiting [72].

The DARWIN mission's implementation of nulling interferometry requires that a number of independent space telescopes all observe the same point in space, in this case the candidate star, simultaneously. The light from the star is collected at each telescope and then transmitted to a central hub. Precise phase shifts are applied to the light coming from each telescope with the aim to cause destructive interference on the light coming from the star but constructive interference on the light coming from the faint exoplanet [73, 74].

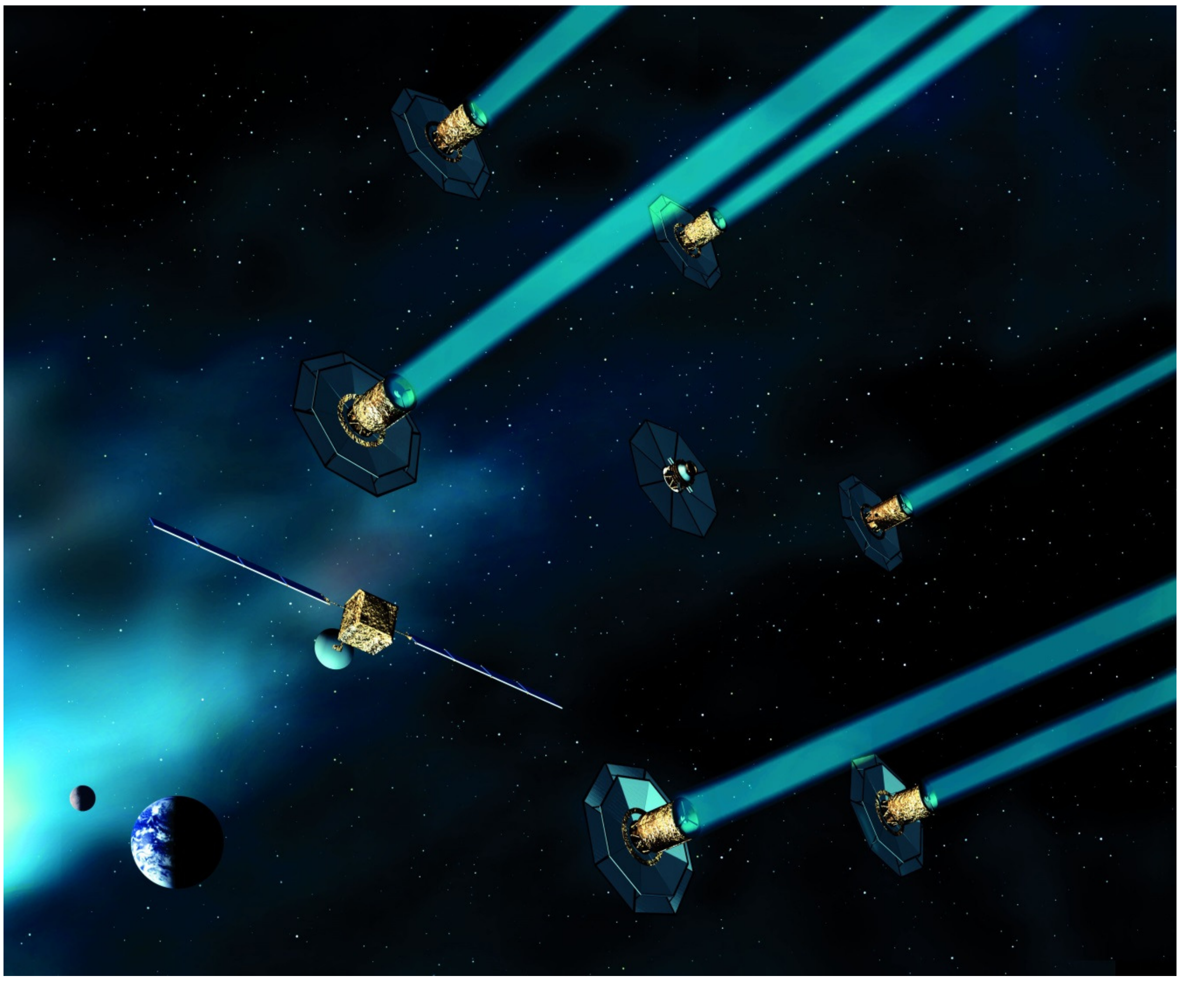


Figure 4-1 Artists impression of the DARWIN mission. Credit: ESA

As such the mission concept comprises of a number of space telescopes, (3 or 4 is the current baseline) and one central hub collecting the light from the telescopes, processing the data and providing communications back to Earth. The current mission concept baseline suggests that the spacecraft would be launched by either a single Ariane-5 launch or two separate launches by Soyuz-Fregat launchers [75]. After separation from the launcher the flotilla of spacecraft will acquire a coarse formation and proceed on their planned trajectory to the second Sun-Earth Lagrangian point (L2). Once the flotilla arrives at the L2 point the imaging formation will be acquired. Once the spacecraft are very close to their final position they will acquire the final formation with sub millimetre accuracy using radio frequency and laser metrology devices on each spacecraft and the hub. The formation will need to be able to perform manoeuvres to reconfigure and rotate the formation to image new stars without losing the relative positional accuracy gained from using the metrology devices.

The DARWIN space telescope concept is based on the Bracewell nulling interferometer concept but with key changes. The Bracewell nulling interferometer [76] operates when two small physically separated telescopes both point towards a star and collect light. The light hits the telescopes as a wave front and if the optical paths are exactly equal, constructive interference occurs when the two beams are combined. If a phase shift of half a wavelength is added to one of the beams (the phase shift must be achromatic in order not to lose any information in the resulting image) then destructive interference occurs and the light from the star is cancelled. If the telescopes are then rotated around an axis pointed towards the target star they will constantly keep the stars light in destructive interference as it is at the centre of the field of view but periodically allow the light from the exoplanet to be visible.

The DARWIN system builds upon this basic premise but is significantly more complex. The Bracewell concept does not take into account the possibility of a zodiacal cloud around the target star which produces what is known as extra-zodiacal light [77]. This exo-zodiacal light is not negligible when trying to image such a faint target as the exoplanet. For example the integrated zodiacal light emitted from the Sun's zodiacal cloud is equal to three hundred times the brightness of the Earth in the near infra-red band in which DARWIN will operate. The concepts required to minimize or eliminate the exo-zodiacal light require at least 3 telescopes. The rotation of the interferometer in the Bracewell design is a relatively slow way to modulate the signal coming from the exoplanet and requires a constant rotation of each telescope. This rotation adds another layer of complexity to an already complex formation flying and attitude control problem. The DARWIN concept includes the partial recombination of the light beams within sub-interferometers and movable mirrors permit the internal modulation of the signal. This kind of modulation is faster and easier in terms of configuration control

A DARWIN/Terrestrial Planet Finder (TPF) type mission will allow for direct observation and spectral analysis of an exoplanet but in order to directly image the planet an interferometric mission with much larger baselines and light collecting abilities would be needed. DARWIN is however a step on this road and would provide priceless scientific data as well as paving the way for future interferometric missions [78].

During the transit to the L2 point it is envisaged that the DARWIN formation will fly in a sphere of roughly 30km diameter holding a loose formation. The maximum inter-satellite distance is limited by the operational range of the satellites relative position sensors. The control system at this point will aim to keep to the target trajectory while minimizing any risk of collision between the spacecraft. The acquisition of the final formation will be carried out by the mission's on board autonomous control systems. The autonomous systems will utilise all of the available metrology data and other sensor data to be able to compute the set of manoeuvres to be carried out in order for the spacecraft to acquire the particular formation needed for the mission.

4.1.2 Baseline for DARWIN mission autonomy

The DARWIN mission from its very conception has required a high degree of autonomy in order to operate effectively. The exact nature of many of the autonomous systems that would help to operate DARWIN has not been finalized. What has been envisaged at this early stage of mission development is an architecture encompassing differing levels of autonomy which are encapsulated into 3 distinct levels, a decision level, an execution level and functional level. The exact nature or architecture of the planning, execution and FDIR systems has not been finalized. This requirement for a high level of autonomy but no hard definition of how it should be achieved offers a very good opportunity to design and develop an autonomous system based on newer technologies such as multi-agent systems for this mission.

The DARWIN mission has many different requirements for autonomy throughout its many components and subsystems. An autonomous navigation system is required to allow the mission to undertake the complex formation flying aspects of the mission and to enable a fast enough response for retargeting and formation reconfiguration. In order to operate autonomously for long periods, an autonomous FDIR may be required and in order to operate effectively and will need to be connected to all of the mission subsystems. The nature of the distributed telescope mission means that many tasks will have to be undertaken in parallel aboard different spacecraft, in order to be able to carry out these tasks concurrently and still successfully synchronise the spacecraft's operations an autonomous task execution system must be developed and can be thought of as part of the higher level autonomous planning system. The execution level must be able to respond to requests in an expedient manner and the planning system must also be tightly linked with the FDIR subsystem.

DARWIN's architecture and use of autonomy, and thus its feasibility, will depend heavily on the results from the European Space Agency's Project for Onboard Autonomy (PROBA) series of spacecraft, most notably PROBA-3.

PROBA-3 is the third in a series of spacecraft developed and deployed to validate novel technologies in space systems. It is focused on the system's need to perform reliable and precise formation flying and will test hardware and techniques using two micro satellites. The technologies under scrutiny involve autonomous formation flying, autonomous FDIR and the use of RF and optical metrology for formation control. In PROBA-3 the guidance, navigation and control systems and the formation flying systems will operate autonomously and be entirely space based but with ground based verification. This allows the ground team to oversee the formation flying sequences and to intervene if a problem occurs. In the PROBA-3 mission there are only two vehicles making up the formation so it only has a single axis. This formation allows for simpler autonomous formation flying controllers but is considerably less complex than the multi satellite formation envisaged in the DARWIN mission concept [79]. PROBA-3 is expected to launch in 2015-2016 timeframe [80].

The baseline for the DARWIN mission states that the mission should be autonomous in the following ways:

- The mission should be able to autonomously undertake the coarse navigation to its final destination orbit. This aspect will be continuation of the work carried on PROBA-3 in autonomous collision avoidance along with autonomous loose formation flying.
- The mission should also be able to autonomously make science observations from a plan and modify the observation plan if necessary. It is envisaged that high level plans will be uploaded from the ground but the mission must have the facility to autonomously modify the plan to either re-factor the plan based on the current state of the mission or to optimise the plan further.
- It is also envisaged that the mission will have an autonomous guidance, navigation and control (GNC) system capable of carrying out the formation flying manoeuvres required for successful operation of the mission. The GNC system must, as well as operating autonomously, be able to accept high level commands from the ground such as acquire a specific formation, slew formation, resize formation etc.
- The mission must also have an autonomous failure detection, isolation and recovery (FDIR) system capable of handling any potentially harmful scenarios to the mission as a whole or to individual subsystems. To operate effectively the FDIR must have a fast and accurate fault detection and diagnosis system as well as a robust decision making system for computing solutions or repairs in response to failures.

The autonomous formation flying control will be decentralised for transit and will then transition to a centralised structure when operations begin. Overall formation control will be assigned to a single central agent which is in communication with all of the other agents in the system. The collision avoidance agents however will be fully decentralised and Independent from the formation flying agents.

4.1.3 Formation flying

When operating as a formation a consensus must be reached between the constituent satellites. This consensus is vital as all of the spacecraft must agree on how they share information such as position, velocity etc. For further reading on this topic see [81, 82]

There has been a significant amount of work carried out on the use of consensus schemes to allow for the more efficient creation and maintenance of vehicle formations. Some of the findings in this body of work are only applicable to the specific type of vehicle that is being studied, such as wheeled vehicles, terrestrial unmanned aerial vehicles etc. but a lot of the general ideas can be applied to spacecraft flying in formation. The ability of an algorithm to find a consensus and the benefits of this are different depending on the topology of the formation in question. For example in the case of a circular pursuit formation which is favoured for many space missions the topology itself is greatly simplified into a uni-directional ring which allows for very efficient formation flying manoeuvres [83]. A common approach is to find the consensus of the whole formation and then use this point as the equivalent to a single vehicle and the other vehicles in formation just try to achieve a certain offset from this point. In [84] an approach that allows for the analysis of consensus problems and stabilization problems is outlined.

As well as finding a consensus between agents and thus between spacecraft, the spacecraft must also be correctly aligned and positioned with respect to each other. For a mission such as DARWIN this is of vital importance as the optical interferometry system relies on extremely accurate inter-spacecraft distances and that all of the spacecraft are observing the exact same point on the target, in this case the candidate star that is hoped to have an exoplanet in orbit around it. It can be envisaged that a fully connected topology would be able to achieve a synchronised attitude of the spacecraft but this may be expensive to operate in terms of communication packets sent and received and thus time. A sparser communication topology with spacecraft only communicating with its direct neighbour or neighbours to synchronise attitude may be simpler [85].

As well as being able to effectively acquire the formation position and attitude, there may be time constraints on the system in which a synchronised arrival at the desired target may be beneficial. If not required implicitly by the mission, a synchronised approach to acquiring and changing formation may well simplify the problem and reduce the communication and computational load.

Using a consensus and other explicit topologies is not limited to just the formation flying problem. The design decision as to how distributed to make the decision making system as a whole is dependent on how much confidence the designer can put in the system's ability to reach a consensus on any given problem. Decentralised decision making has a number of advantages over centralised decision making, namely that there is redundancy if something goes awry and there is no single point of failure for the system. The disadvantage is that in reality it relatively difficult to guarantee that consensus will be reached within any given time frame. If the topology of the system is simplified without damaging the speed or reliability of the system then certain assumptions may be proved and time-scales given for normal operations.

The DARWIN structure is relatively simple so a simple completely connected topology can be implemented so that all of the agents in the system can communicate with each other.

The idea of using highly simplified control laws and each agent only acting locally to produce coherent large scale behaviour [86] is very appealing for multi satellite space missions. These flocking laws are typically based on very little data, most famously that of the inter-craft distance and the current velocity of the spacecraft and the adjacent spacecraft. All of these variables can be calculated at a high rate and thus a flock of spacecraft might be seen as very attractive. The downside of these flocking laws is in their behaviour under a wide range of conditions, their stability and their efficiency. A lot of work must be carried out to understand how perturbations to the system are propagated through the flock and it is entirely reasonable to assume that certain types of disturbances will cause unexpected consequences. In this case the very simple laws that govern the flocking may not have the ability to avoid collisions or other adverse events. These events would be catastrophic for a space mission and the uncertainty inherent in these methods makes them unlikely to be used in their most basic state, which is most suitable for terrestrial unmanned aerial vehicle (UAV) systems.

4.2 DARWIN MAS design

The following multi-agent system design was designed for a DARWIN [87] type mission but can be thought of as an example implementation of the previously discussed HASA architecture that implements general formation flying abilities. In this multi-agent system there are a number of different agent types: the Planning agent, the Formation Flying Command agent, the Formation Flying execution agent, the Feedback agent and the Negotiation agent. We will briefly outline their operation here before defining them in more detail as part of our architecture in the next section.

4.2.1 Planning Agent

The planning agent is one of the most important agents in the multi-agent system as it contains and is in charge of all the primary deliberative and decision making processes for the mission. Whereas in the other agents the tasks that will be performed are fairly standard in the spacecraft control domain and a fair amount of reuse from other systems can be envisaged the planning agent will have to be developed uniquely for this multi-agent system. This is because the decision making processes must be designed to exactly make use of the structure of the physical mission (multi-satellite) and the architecture of the multi-agent system, as no system of this type has been developed before then the planning agent must be developed from the ground up to fully take advantage of the system. This also means that the planning agent will also need a lot of developmental time compared to the other agents and must also be more thoroughly tested as it is more deliberative and thus more unpredictable in operation than the more common reactive components. In this agent, plans are developed from overall mission goals and objectives which are supplied from the ground and should not really change over time. The planner then uses the environmental model to decide which actions out of all of the actions that would lead to the completion of its objectives are feasible. The format of the plans must take into account the fact that the planning process itself will be a distributed activity and as such the plans will be fragmented and provision must be made to allow for the recombination of the individual plans into a coherent global plan. The plans will be object based and can also be further optimised for the mission itself, for example in this case the plan is primarily concerned with controlling the relative position, attitude and speed of the all the spacecraft in the formation. As such the main plan types will be to acquire formation, acquire a target, test plans for validation and verification purposes and high level emergency actions that come outside remit of the lower level reactive emergency procedures and the collision avoidance mechanisms (CAM).

4.2.2 The Formation Flying Command agent

The next agent in the hierarchy is the formation flying command agent. This agent can be thought of as an intermediate step between the high level plans that are generated in the planning agent and the actual execution of a formation flying manoeuvre. The formation flying command agent takes as its inputs the plans that have been ratified at the planning level by all of the planning agents on all of the spacecraft. This is a key requirement as the formation flying command agent can only undertake to achieve a given formation as long as all of the spacecraft are in agreement. This should not be taken to mean however that there is no possibility of altering the formation once the acquisition has started, only that that the commencement and any change to the formation has to be agreed at the highest level by all the spacecraft. The level of abstraction used by the formation flying command agent is different to that used at the lower level and is not purely comprised of the requisite thruster profiles and timings.

Instead the formation change is encoded in its entirety to include not just the exact positions that are to be attained but possible alternate positions and emergency procedures for the different legs of the formation acquisition. It is also important to note that it is not a particular formation that is encoded but a change between two specific formations that relies on specific starting point for each spacecraft and that each spacecraft be operating correctly in the desired state. These formation changes can be manipulated at the highest level and exchanged between different agents allowing for them to be suggested, negotiated, refined and then, if consensus is reached, enacted. In order to generate the formation change profile with sufficient accuracy the formation flying command model has a number of specific requirements. It must have full access to the spacecraft's system status information so that it can gauge the relative performance of each spacecraft and must also have a more accurate physical model of the spacecraft and their environment in order to compute exactly how to achieve a given formation.

4.2.3 The Formation Flying Execution agent

The formation flying execution agent takes care of the aspects of the formation change not covered by the formation flying command agent, most notably the tasks associated with the execution of and real time monitoring of the formation change. At an abstract level the formation flying execution agent converts the formation change plan generated by the formation flying command agent into specific manoeuvres that are broken down to the individual thruster level.

The formation flying execution agent also has the responsibility for the real time monitoring and control of the formation change. Whereas the formation flying command agent is a highly deliberative agent that accepts plans and creates and manipulates formation change procedures the formation execution agent is much more reactive. The structure of this type of real-time operation agent will be very different to that of the higher level agents in that it must be able to constantly monitor a wider range of inputs and be able to react reliably within a very short period of time if required in an emergency or if a change of desired formation is necessary. Where the formation flying command agent creates a formation change procedure for the formation as a whole the formation flying execution agent instead concentrates on meeting the requirements outlined in the formation change procedure but only for its own spacecraft. This extra level of granularity allows the agent to more effectively use spacecraft specific constraints and resources as well as taking into account the spacecraft's performance history in its processes. Again a slightly different physical model will be required with more detail of the physical characteristics of the single spacecraft as at this level the control of the formation as whole is not considered and is carried out at the formation command level.

It can be seen that the physical and environmental models used by the formation flying execution agents is very similar to that used by the agents above it in the hierarchy but with smaller scope and higher level of fidelity. It can therefore be envisaged that instead of having many different models (if our structure is suitably modular) we can use the same models for all levels as long as certain portions are accessed and taken into account by the correct agents. This ability to share resources between agents but only access and compute on the required parts offers real advantage to the system designer as fewer individual modules and models have to be developed.

As well as receiving commands from the planning agent through the formation flying command agent there must also be the capability of the formation flying execution agent to execute commands directly, these low level commands will be mainly used in the testing of the system within the simulation as part of the extensive unit testing of the system but this direct line to the actuators may also be used by the emergency components of the systems. These two lines of communication however must be kept separate as the availability of direct control of the spacecraft introduces more possibility for unchecked manoeuvres to be carried out.

4.2.4 Feedback agent

The feedback agent and the negotiation agent that will be discussed later are part of the "checks and balances" in the system that monitor the system from a level at least once removed from the operation but retain the ability to act at any level of the system if intervention is necessary. As previously discussed a clear distinction between the more deliberative components such as planning and the execution of manoeuvres is present in the system. The feedback agent is in charge of monitoring and finding any problems that may occur or may have occurred in the real-time portion of the system. In order to carry out this task it must have complete access to the other agents but has different needs when it comes to environmental and physical models.

A decision must be made as to whether the feedback agent has its own models of the physical spacecraft and the environment or relies on the models in the agents it is checking or both. An advantage of the feedback agent having its own versions of the environment model and physical models for checking is that the feedback agent can act as an independent checking mechanism, the disadvantage of this approach however is that the feedback agent will need models that equate to each agent that it will be monitoring. The problem arises however if the models in both the active agent and the feedback agent are the same as any errors could occur equally in both as a common mode failure so the feedback agent will offer no real insight. At the other extreme the feedback agent can have no real environmental or physical models itself but instead just have a set of checks and criteria that the agents it monitors must adhere to. This process would be much faster than computing the calculations of another agent in parallel and would give the feedback agent the level of independence required to find errors occurring in other agents. This second approach can be further extended to not just operating with hard coded checks on the other agent but also, if the need arises, the agent can perform test calculations using the agent's models to test if a particular error is caused by the models used or the agent structure. It may also ask other agents of the same type as the monitored agent to carry out these checks to see if the failure is common to all of the agents. As well as these more deliberative actions the feedback agent will also encompass the traditional collision avoidance mechanism (CAM) system found in spacecraft which is a highly reactive system. This means that the feedback agent will have to decide in a very short time-scale to take action if it deems that a collision between any of the spacecraft is likely. In order to be enacted in the quickest possible time these collision avoidance actions must be very simple and as standard as possible as the mission's reliability greatly relies on these commands being executed and their ability to reduce the probability and potential severity of a collision.

As well as working on very small time-scales the feedback agent will also work to try to determine if any errors emerge in the system over time. With a highly complex interacting system such as the multi-agent system in question it is likely that there will be certain drifts in the performance of the sub components and agents. The Feedback agent must be able to notice these longer term changes and inform the sub system or agent of its error or take further action to stop it if necessary. This process will be carried out by the agent starting with hard coded bounds of the system's expected performance in the chosen metrics, it will allow for some drift but this will always be monitored and taken into account throughout the system.

4.2.5 Negotiation agent

The negotiation agent is the non-real-time partner of the feedback agent and has a similar general remit in that it is designed to facilitate the smooth operation of the multi-agent system but instead of trying to mitigate the technical faults like the feedback agent the negotiation agent is instead designed to stop and recover from any social faults in the multi-agent system.

These social faults are unique to autonomous agents and agent based systems where each agent undertakes its own actions as part of a larger system towards a common goal. In this design each spacecraft has a full suite of agents to allow for independent operation and facilitate independent error detection and resolution. The negotiation agent is needed in order for these local groups of agents to work together and to overcome any deadlocks or disagreements about the higher level goals and plans of the system.

The fact that many of the agents are repeated throughout the system allows for the system to operate with a high level of redundancy, this is only true however if errors can be detected and consensus reached by the agents if one of their kind has failed or is operating incorrectly. The simple way for this to be achieved is for the negotiation agent to operate as mediator and allow for voting between agents to discern which agents views are to be incorporated into any action. This sort of process works for the lower level agents but the higher level agents will work in a slightly different way. The higher level agents will select from amongst themselves a prime agent (or the negotiation agent will choose one if no consensus is reached) and then the prime agent will operate for the whole formation. The planning agent is an example of this approach where one planning agent will have to take overall responsibility for combining all of the disparate plans generated within the multi-agent system and then creating a single comprehensive plan. The other planning agents however will also be constantly running the same calculations to check the prime agents operation is correct and the negotiation agent will revoke an agent's prime status if the other agents disagree with its results.

The negotiation agent will also be used by many of the other agents in the system to break deadlocks and enable the system to operate effectively. It can be easily envisaged that two spacecraft may disagree about their position in the formation and this disagreement would stall any efforts to compute and enact a formation acquisition procedure. In this case the negotiation agent would have to consult with any non-involved formation flying agents and decide which agent is correct. If none can be chosen in this way the negotiation agent will have to try to take into account other factors such as the historical reliability of the agents in question and their past performance and then make a decision based on this data. This agent has the most deliberative aspects after the planning agent but also shares the most with traditional informational agents as some of the most common problems with all agent systems is social deadlocks and similar problems [88–90].

4.3 Agent Structure and Interaction

Figure 4-2 shows a general overview of how the agents on each craft interact with each other.

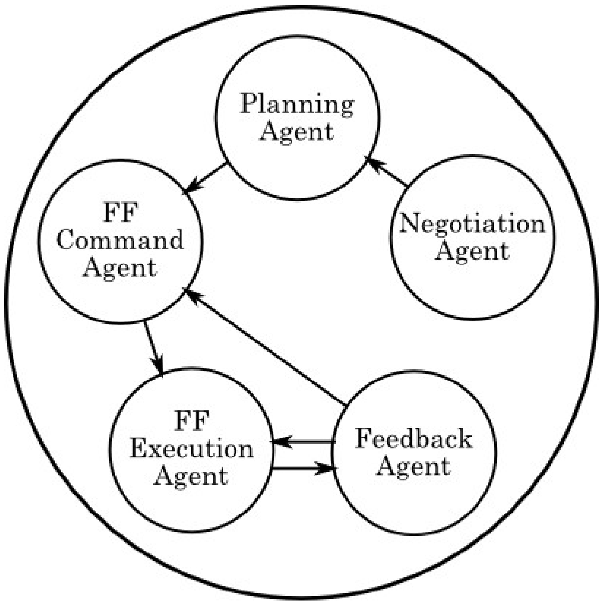


Figure 4-2 Agents on a single DARWIN spacecraft

The agents for the DARWIN mission were initially prototyped in the SeSam [52] multi-agent simulation environment. This approach offered a number of benefits over proceeding straight to coding the agents in the target programming language, firstly that the structure and interaction of individual components can be easily seen and modified and secondly that these designs can be used to create skeleton code structures in Java removing some of the work needed for implementation. The following figures (4-2 to 4-7) show the structure of the agents in SeSam.

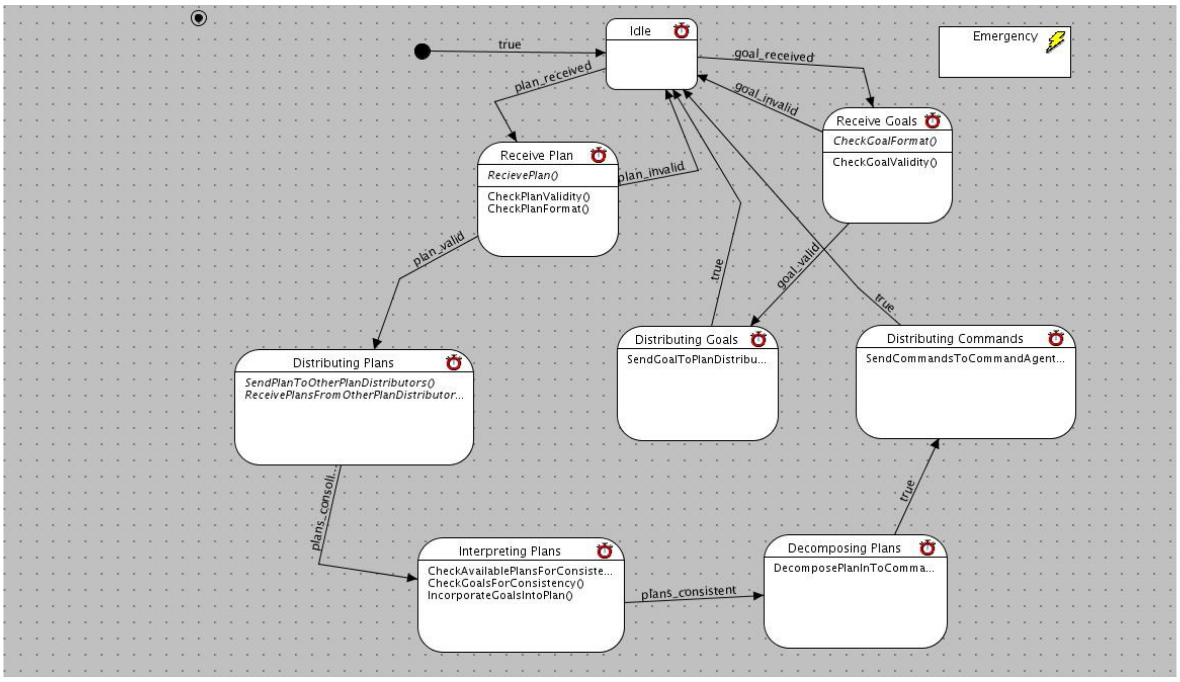


Figure 4-3 Planning Agent Level 2

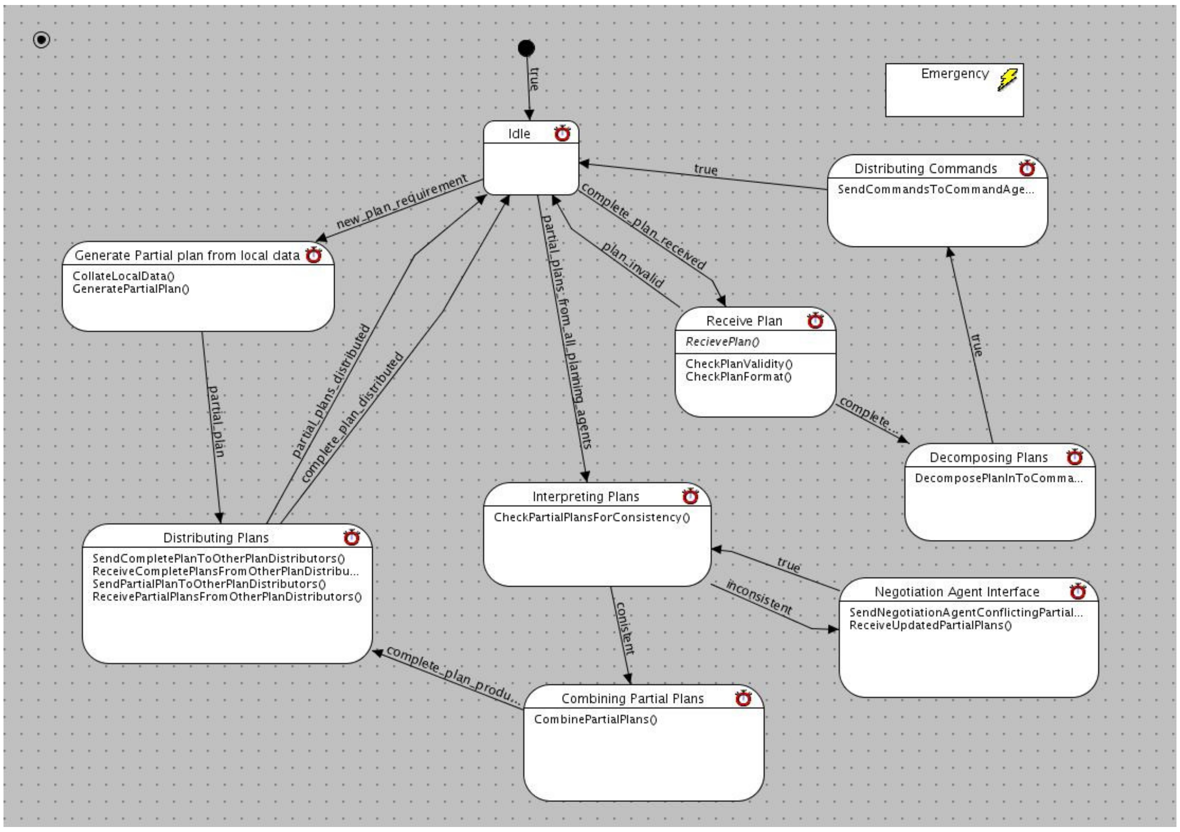

true
Idle
Emergency
true
Distributing Commands
SendCommandsToCommandAge...
new_plan_requirement
complete_plan_received
plan_invalid
partial_plans_from_all_planning_agents
Generate Partial plan from local data
CollateLocalData()
GeneratePartialPlan()
Receive Plan
RecievePlan()
CheckPlanValidity()
CheckPlanFormat()
true
partial_plan
partial_plans_distributed
complete_plan_distributed
complete...
Decomposing Plans
DecomposePlanInToComma...
Interpreting Plans
CheckPartialPlansForConsistency()
Distributing Plans
SendCompletePlanToOtherPlanDistributors()
ReceiveCompletePlansFromOtherPlanDistribu...
SendPartialPlanToOtherPlanDistributors()
ReceivePartialPlansFromOtherPlanDistributors()
true
inconsistent
Negotiation Agent Interface
SendNegotiationAgentConflictingPartial...
ReceiveUpdatedPartialPlans()
consistent
complete_plan_produ...
Combining Partial Plans
CombinePartialPlans()

Figure 4-4 Planning Agent Level 2

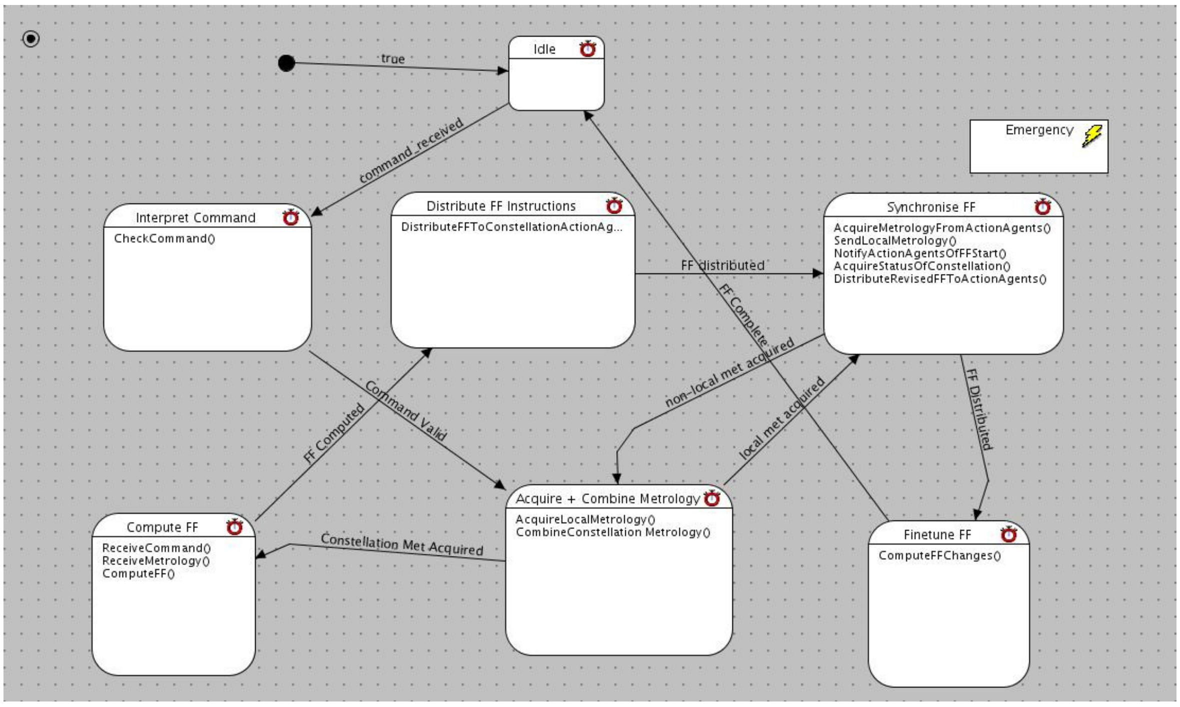


Figure 4-5 Formation Flying Command Agent

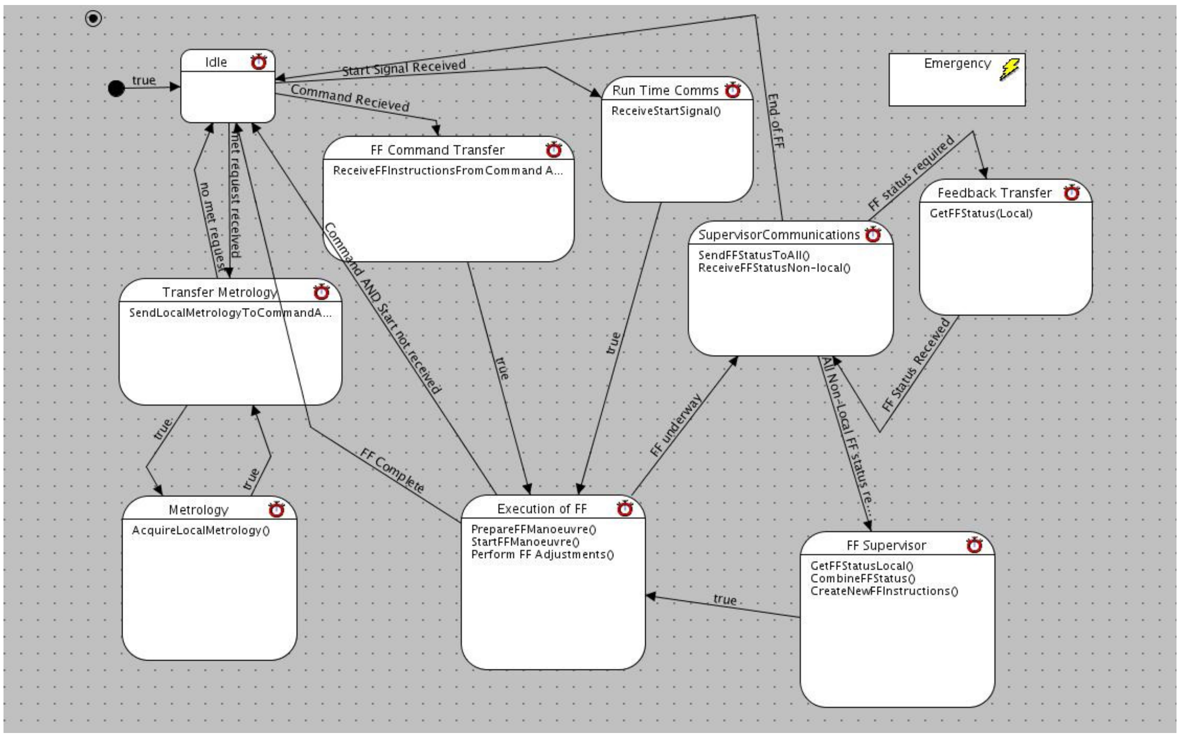


Figure 4-6 Formation Flying Execution Agent

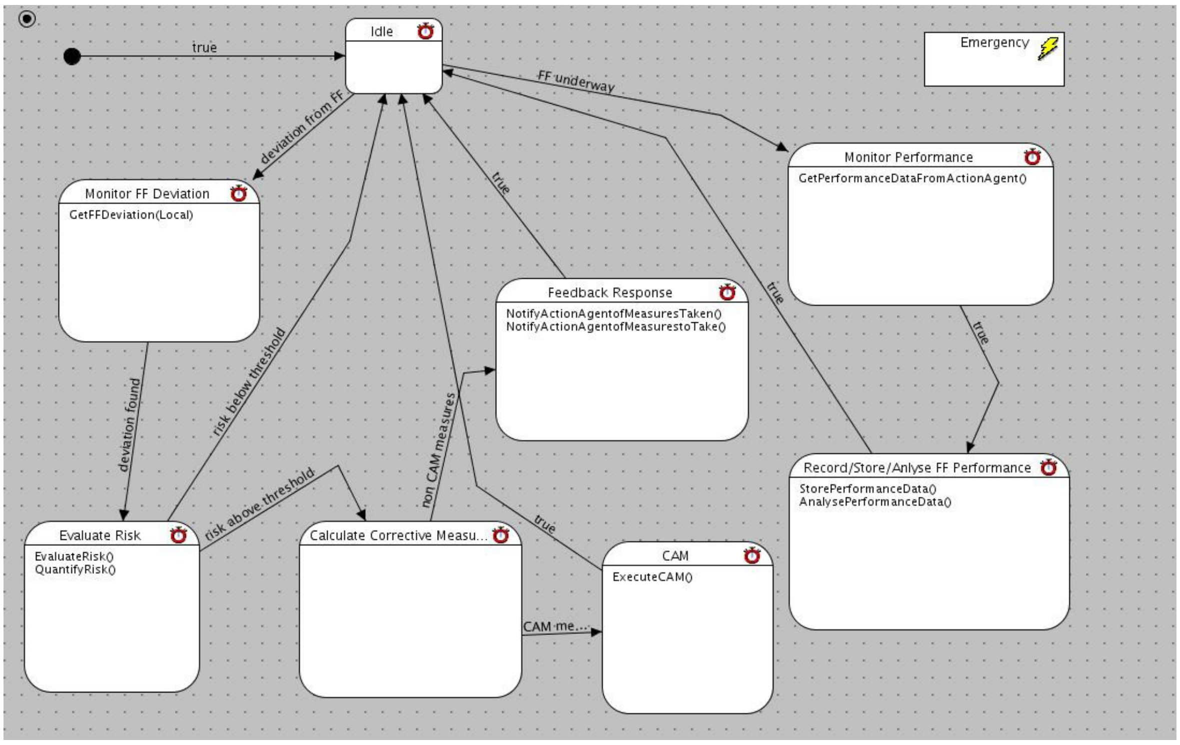


Figure 4-7 Feedback Agent

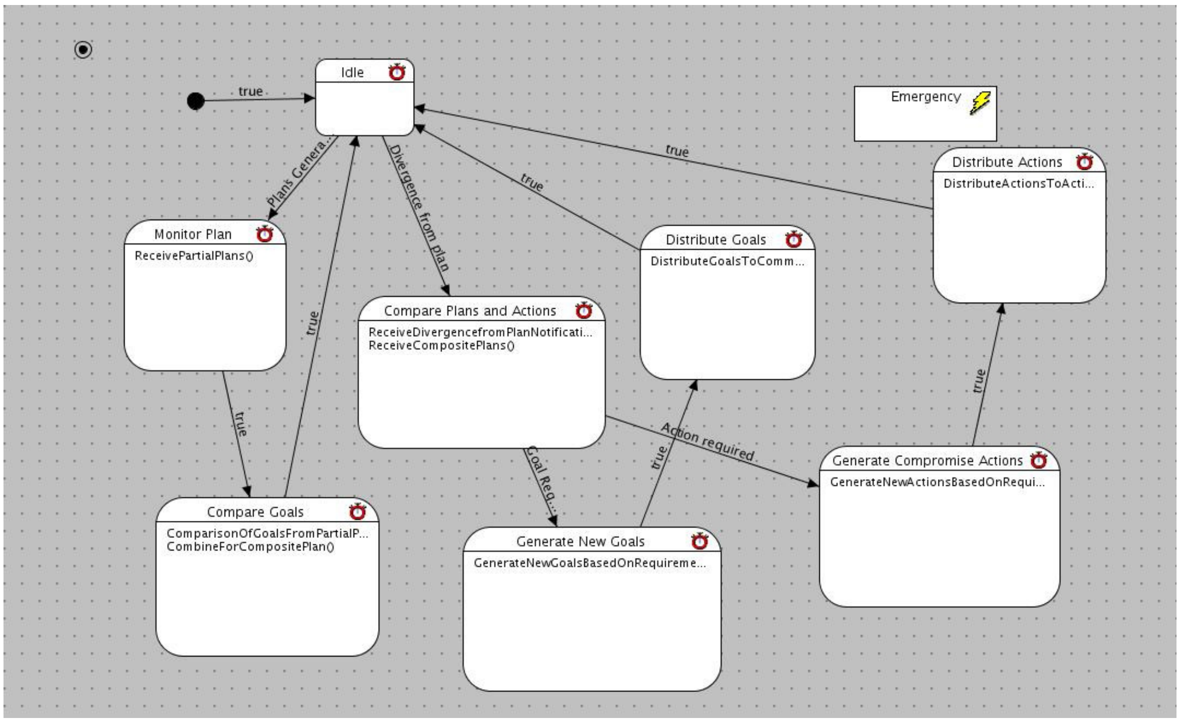


Figure 4-8 Negotiation Agent

4.3.1 Quantification of multi-agent system attributes

As well as measuring the performance of the system in the simulation testing suite the systems should also be characterised beforehand so we can try to find some correlation between system parameters and system performance. One of the key measures we want to make is that of system autonomy. We can grade autonomy using levels such as those used by [91] which are fully autonomous, boss, cooperative, underling, instructable and remote control. Using this approach we can use the rank of the agents within a given multi-agent system to uniquely identify the system.

Another useful metric that is very important is the amount of resource sharing that a given agent or component undertakes. One of the key reasons why multi-agent systems can operate is that they can share and use finite resources. An example given in [91] is that of a vision system that could easily apply to many resources in our distributed space mission. Let us say that we have two behaviours using the same vision system, A and B. A can be a collision avoidance behaviour that is part of the larger collision avoidance system and B is a scientific event detection behaviour. Let us, for this example, assume that the behaviours are mutually exclusive. This means that the behaviours must share the resource (in this case the vision system). Let us say that a system has a certain capacity, in the vision system's case this can be thought of as the frequency capacity or how many times it can be used in a given time. If for this simple example we assume the cost of switching between behaviours is negligible and the minimum operating frequency for A is A_t and for B is B_t. If C is less than $A_t + B_t$ then the capacity of the vision system must be increased. If C is greater than $A_t + B_t$ then the two behaviours can operate together without penalty. If the actual operating frequency achieved can be written as A_t' and B_t' then we can use $\frac{|A_t - A_t'| + |B_t - B_t'|}{C}$ as our resource sharing metric.

4.4 Development of a multi-agent simulation suite

4.4.1 Simulation

When trying to determine the validity of a multi-agent system it is not just the lower level actions that must be considered. One of the key benefits of distributed intelligent systems is the ability for emergent behaviour to arise. It is this emergent behaviour however which cannot be adequately validated using most formal methods. Instead the system has to be tested in order to adequately gauge its response for any given scenario. Ideally the software would be tested on the final hardware and within the final environment in which it will operate but in practice this is rarely feasible, especially in the case of space missions.

The field of simulation is a particularly broad one and encompasses many different areas. The many different variations of simulation have usually been developed to try to simulate particular systems and as such there is very little work on "general" simulators as for worthwhile simulation of a system to be carried out it must be tailored to that specific system.

4.4.2 Test suites

In the development of simulation test suites there are a number of approaches the system designer can make in order to try and simulate his system. The system can be simulated entirely in software or with some components modelled in actual hardware. The latter is known as simulating with "hardware in the loop" and is frequently seen as an effective middle ground between 100% software simulations and testing the control system on the actual hardware. Most simulation begins by being carried out completely in software for a number of reasons, the main one being the ease of deployment and the ease of modification of the simulation and the second is the reduced cost this ease of use entails and the zero risk it poses to expensive hardware. If the simulation is developed following a modular structure then the designer has the ability to introduce hardware in the loop to the system while not having to change the entire simulation system. This approach entails simulating the individual hardware components that make up the system as accurately as possible so the control system is tested using sensor values and executes its action through the appropriate actuators. It is then easier to bring real hardware into the simulation by replacing those individual software components with their respective hardware part where appropriate and feasible. The approach taken in this work is based on a 100% software approach but with multi language and multi-platform compatibility which allows for the easier hardware integration in the future.

The evolution of this concept is the development of hardware test beds. Where the hardware for a given domain can be thought of as relatively standard across a given problem domain then a generic hardware test bed that can test multiple different types of control systems can be created. A good example would be a motor car which is modified to allow full software control over the steering, acceleration, brakes, gears etc. If the interfaces to these actuators are correctly developed then many different kinds of automated driving software could be tested on this one piece of hardware. Similar hardware test beds have been developed for space missions: these range from the generic which have typical spacecraft subsystems such as power, thermal, etc. to formation flying test beds where robots or spacecraft analogues operate to acquire their formation in an environment specifically designed to test the formation flying algorithms using close to real hardware (cold gas thrusters, real sensors etc.) but in a reduced degrees of freedom environment such as on a "frictionless" two dimensional plane.

Much like multi-agent systems themselves simulation systems can be distributed or centralised. A centralised simulation system is better suited to single spacecraft whereas a distributed system is obviously more suited to a multi satellite or formation flying type mission. A centralised simulation system works by having all of the modules comprising the simulation environment, the sensors, the actuators etc. all in one place. This approach allows for a more easily tested simulation suite and also makes development, maintenance and modification easier. The downside is that this sort of system is not suited to modelling distributed systems where multiple satellites and multiple processes are concurrently operating. For this type of mission a distributed simulation system makes more sense. In this case the simulation is run on multiple nodes, roughly equating to the number of nodes on the system being tested. In this way the abilities and characteristics of the distributed system can be ascertained and any benefits over a centralised system hopefully quantified. Communication between the nodes should be in the protocols that will be used in the hardware where possible, frequently these protocols are wrapped in a standard transmission control protocol/internet protocol (TCP/IP) layer and the simulation runs over a distributed computing network. The benefits of this approach are that any improvements arising from the distribution of the system can be gauged and that for complex system the computational load of the simulation can be distributed. The simulation suite developed in this work is decentralised.

4.4.3 Basic types of simulation

Simulations can be further grouped as:

- Containing Static or Dynamic Models
- Containing Stochastic or Deterministic
- Containing Discrete or continuous time models
- As modelling aggregates or individuals

Dynamic models are models that change over time. This type of model is obviously used for the simulation of space systems that involve movement through space and thus orbital dynamics or any states that change over time. This includes all autonomous agent based systems (as these must inherently operate over time in order to be autonomous). Static simulation models are used when modelling a single point in time for a system. This type of simulation is used for the optimisation of models and statistical simulations such as Monte Carlo analysis or for statistical learning techniques such as neural networks or support vector machines. Dynamic models are used for the core of the test suite but static models can be introduced to increase the agent's functionality, for instance by introducing a learning model for some agent or agents.

The type of model in the simulation can also be further categorised. A stochastic model is a model whose behaviour cannot be entirely predicted as it involves the interaction between the models previous state and some randomised elements. A deterministic model on the other hand can be entirely predicted based on the current or previous states of the system. A chaotic model can be thought of as a deterministic model but with a resulting behaviour that cannot be entirely predicted and which is highly sensitive to initial conditions [92].

The key modules in this suite are deterministic but stochastic processes are introduced by some aspects of the agent's interactions so depending on the scenario and the agents involved the system can be either deterministic or stochastic.

Another key differentiator between models is whether they operate over continuous or discrete time. Discrete models operate by changing their state or variables only at certain (discrete) points in time. These discrete time points can be regular intervals or only coincide with the occurrence of certain event. In a continuous model the system states and variables change constantly and may have one of an infinite number of values. This is the desired approach for modelling the physical world but a discrete model is much more practicable when taking into account computing constraints and the difficulty of modelling complex systems on analogue computers. The models in this system are based on discrete models with the simulator numerically integrating between epochs.

Another choice that has to be made is whether the simulation simulates every entity to its fullest extent or instead simulates groups or aggregates of entities. Cases where both would be applicable can be imagined. In a relatively small scale system with a limited number or agents or modules then a full simulation can be carried out in a reasonable time frame. If the system was much larger a full simulation could still be carried out but the decision would have to be made as trade-off between simulation fidelity and computational time. At the other extreme in the case of a simulation that involves a large number of human protagonists that interact with each other and the system it can be useful to model these as groups or in aggregate in order to reduce complexity with limited impacts on the fidelity of the simulation as a whole. The impact of using aggregates and groupings on increase computational speed at the cost of fidelity should be measured by conducting small scale simulations using the individual entities and the aggregates and comparing results. It may be hard however to carry these results over to larger scale simulations as the effects of individual interactions at the larger scale may have more impact than at a smaller scale. As one of the key aspects of the work is the inter-agent interaction the simulation suite will simulate every agent to its fullest extent.

The question raised above regarding the relationship between the fidelity of the simulation and the computational cost of the simulation is the key question in the field of simulation. It is relatively easy to create simulation systems that model the extremely low level behaviour of a system and each of its individual components, but frequently the computational cost is too large. With the constant increase in computing power available to the average user the number of simulations that can be run in a reasonable time increase but it is still highly desirable to instead optimize the simulations and run as many as possible within a particular time frame. There are number of different strategies for making a simulation more efficient. In this work the highly parallel nature of the system being simulated is used to our advantage as it naturally allows large parts of the simulation to be run in parallel, hardware permitting.

4.4.4 Simulation Structure

The actual structure of simulation is extremely important to its efficiency. At the most basic level the simulation is either run in series or in parallel. In series the answer to a calculation is fully computed before moving onto the next step. In the parallel case multiple calculations are carried out at the same time. With the proliferation of multi-core and multi-threaded processors the parallel approach is much faster but many systems cannot be simulated entirely in this way as many simulations require the previous answer to a calculation before being able to perform the next calculation. Distributed control systems and especially multi-agent control systems do however lend themselves to parallel computation. In operation the actions of the agents are carried out in parallel so it stands to reason that they should be able to be carried out in parallel in the simulation. There has been a great deal of work on multi-agent simulation [93-96] and this is especially important when there is a mix of software, hardware and human agents. In our planned autonomous multi-agent control system there are no human agents in the autonomous phase and the hardware is under direct control of the software agents.

Instead of taking a step back and trying to abstract and simulate a software agent system the system designer can omit the development of this extra simulation layer and gain a more accurate insight into the multi-agent systems operation and performance. This is carried out by not simulating the multi-agent system but instead coupling the actually running multi-agent system to a simulation that constitutes its external environment. In this way the multi-agent system's performance can be more accurately gauged while still having full control of the simulation environment.

4.5 Simulation Suite Architecture

The following work was based on the work presented in "Design and Testing of an Autonomous Multi-Agent Based Spacecraft Controller" [97].

Any agent that has some deliberative aspect, that is it deduces outcomes or solves problems about its environment, must by implication have some sort of environmental model. The environmental model for a multi-agent system can take many different guises but all environmental models share certain key features. It helps to define the environment as that in which the agent will be operational. In the robotics domain and thus similarly the space domain this includes the "outside world" but must also contain the hardware that the agents operate on and the agents themselves. The fact that a multi-agent system's environment can include both hardware and software components as well as some definition of the external world can make definition of these environments difficult. As with most of the more complex ideas and constructs in multi-agent systems the environment itself can be split into sub-sections but they must all have the same basis. The agents obviously have relationships with the other agents in the Multi-agent system but may also have relationships with agents or entities outside the MAS such as ground controllers. The ground controllers in this case should also be included in the environment. The agents also have relationships with the hardware on the mission, at the most basic level they have a controlling relationship with the actuators and an observant relationship with the sensors. The agents also have relationships to the outside world in that environmental disturbances such as atmospheric drag, gravity fields, etc. can alter the orbital parameters of the satellite.

The agent's relationships with the outside world may not be as literal as those found in the classic mobile robotics literature in that they are not pushing boxes or (at least we hope) not directly physically interacting with each other. Instead the relationship is more that the outside world acts as a set of modifiers on the agent's actions and the agent must understand these modifiers in order to successfully deliberate about what actions to take or gain insight into what has happened in the past.

For example the relationship an agent has with the external world may define certain constants or sub-relationships that affect for instance the motion of the robot. For a simple wheeled robot these may include the coefficient of friction or the way that momentum is calculated. They are not restricted to purely modelling the physics of the situation but can also include data about the scenario, so if the wheeled robot relied on solar power then the agent will have some relationship that tells it the light levels in different areas or at different times that will allow it to deliberate about what is the best course of action to take.

This idea of defining a multi-agent systems environment as a series of relationships is extremely flexible and can be applied to both pure software agents as well as situated (robotic etc.) agents. The level of abstraction gained by the definition in terms of relationships has another key benefit. A well-defined modular structure with many common components between systems makes for a much greater ease of implementation into an actual system.

In this way relationships between agents equate to interfaces between methods and behaviours and act as a good starting point for the design of the control software. An interesting by-product is that if we are defining our environment for the agent system in a clear and modular way, this can help us with the simulation of the multi-agent system. If we are sensible with our development then it stands to reason that the set of relationships that are used to define the outside world in the agent's domain can be used as an environment in which to simulate the multi-agent system. At its most simple, instead of a multi-agent system having knowledge of the real world through sensors and then operating in the real world through its actuators the relationship model developed for the agent can be used to model the outside world. This can be summarise succinctly by the statement, *"if we already have a model of the external world to allow the agent to reason then why don't we use the same model to replace the external world in our simulations?"*

The description and similarity between the environmental models can lead to confusion so a nomenclature has been devised to ease the description of these types of models. At its most basic we can think of three different models, the intelligence model M_i, the simulation model M_s and the agent model M_a. For an agent controlled robotic platform operating in some environment we can model it by a number of agent models which are coupled to one or more intelligence models. These then operate within the external world when the robot is operating. These intelligence models contain the relationships that describe the external environment. We can then think of replacing the real world in the first example with a simulation model which encapsulates the nature of the real world through description of its relationships with the agents. As such it can be seen that the M_s and M_i will have very similar structures and will share a common basis.

Formally we can say the simulation model, M_s can be defined as:

$$M_s = < D, S_d > \quad (4\text{-}1)$$

where D is a dynamical model comprised of the agent's relationships with the outside world (friction, momentum, equations of motion etc.) and S_d is a sensor data generator that is required to convince the multi-agent system that it is operating in the real world and not in a simulation. The agent model, M_a can be described as:

$$M_a = < G, K_a, M_a, C_i > \quad (4\text{-}2)$$

where G are the agent's goals, K_a is the agent's knowledge, M is the agents memory and C_i is a coupling with an intelligence model M_i. The intelligence model M_i can be defined as:

$$M_i = < B, D_i, K_b, M_b, C_i > \quad (4\text{-}3)$$

where B is an intelligent behaviour, D_i is the dynamical model used by the behaviours (it can have exactly the same structure as the dynamical model in the M_s), K_b is the knowledge required by the behaviour B and M_b is the behaviour model itself.

So the D_i equates to the dynamical model required by the intelligent behaviour and is thus very similar to the dynamical model D that is used in the simulation model M_s. This requirement for a dynamical model as part of the greater intelligent behaviours of the agent can be made due to the fact that all our agents will be situated. That is, they will operate on real hardware that interacts with the outside world. Even though not all agents will require direct constant access to a dynamical model go their environment in order to operate, such as the agents in charge of formation flying or manoeuvre execution it can be envisaged that there will be occasions where nearly all agents may have use of predictions based on the dynamical model or may need to reason about the events of the past by using the dynamical model. As such, although it is not strictly necessary it is deemed prudent to have the dynamical model as part of the intelligent behaviour model M_i definition.

Any agent can therefore be described as:

$$A =< M_b, C_s, C_i > \tag{4-4}$$

where M_b is the behaviour model, C_s is a coupling between the agent and a simulation model and C_i is a coupling between the agent and intelligence models where $|C_s| = 1$ and $|C_i| >= 1$. This structural definition allows for a given agent to access multiple intelligence models depending on its requirements.

With a set of properly defined agents utilising a number of intelligence models, all operating with a single simulation model then testing can commence. The simulation model itself consists of a dynamical model and sensor data generator and can be used by the system designer or tester to create a set of scenarios that will be tested. As well as the scenario definition the relationships between all the agents and the structure of the agents themselves is also pre-defined and allows for many different tests to be carried out.

4.5.1 Describing Scenarios

During the early design stages of a multi-agent controlled system the designer wants to be able to use the test suite to quickly and easily evaluate possible multi-agent system configurations. As explained before, the multi-agent system itself can be defined in many ways; the structure and architecture of the system as a whole and of individual agents can have a massive effect on the success of a given multi-agent system so testing variations of the same multi-agent system at an early stage can be highly beneficial.

As well as changing the configuration of the agents and the multi-agent system the designer can also use the dynamical models and the structure of the simulation and intelligence models to gain a deeper insight into some specific traits of autonomous systems. This can be achieved by introducing and managing a particular disparity between the simulation model and the behavioural model.

To take a step back we can see the benefit of sharing the dynamical model between the simulation model and the intelligence model. In this case the simulation model is acting as our replacement for the real world. As such the simulation model as a whole and thus the dynamical model at its heart must have a fidelity as high as is practicably possible to make the simulation results as accurate as possible.

If we reuse this high fidelity dynamical model for the intelligence model, and thus the intelligence of the agents, then we are saying that the agents have access to this high fidelity model when operating the real world. In reality the hardware constraints of the real space mission will be orders of magnitude more restrictive than that of the desktop hardware and computing clusters available for the simulation of the mission. As such we may choose to purposefully lower the fidelity of the model used by the agent's intelligent behaviours in order to more closely match that which we will be able to deploy in the real spacecraft. As well as modifying the dynamical model used by the agents to match the available hardware we also modify the dynamical model in other ways to allow us to gain specific insights into the operation of the multi-agent system.

We can lower the fidelity of the dynamical model used by the agents in a number of ways. Most intuitively we can think about applying another layer to the model in which the values generated are rounded or errors introduced from some external random number generator. The highly structured and modular construction of our dynamical models however also allows us to quite simply reduce the number of terms used in any given calculation, so for instance terms in the higher fidelity models dealing with other bodies or higher order harmonics may be excluded in order to better reflect the type of model available to the spacecraft in operation.

In conjunction with the sensor data generator the dynamical model can also be manipulated to help with the definition of more complex scenarios for the test suite. For instance the dynamical model can be modified to model the failure of any of the actuators on the spacecraft or indeed any of the sensors of the spacecraft. As well as modelling complete failure partial failure of components can also be modelled and subsequently the design's ability to deal with these errors ascertained.

As we can use the test suite at the preliminary stage of the design we can also use it to make decisions about what level of fidelity we need for the different agents by running the simulation repeatedly with different fidelity models and comparing performance between the systems on a the same suite of test scenarios.

The design of multi-agent simulation system has to take into account many different aspects. One of the key decisions to be made is whether the simulation should be generic or domain specific. One of the key reasons for building a multi-agent simulation system should be the time it saves future users when it comes to simulating their own multi-agent systems. As such the more multi-agent systems that can be modelled the more utility it offers future users. On the contrary however it takes a not inconsiderable amount of work to develop a multi-agent simulation for any given domain and although the kernel of generic multi-agent simulation system may be small, there will be requirement for domain specific libraries. Another disadvantage of generic simulation systems is that they will not, by their very nature, be optimised for one particular domain.

It is desirable of course to find some middle ground and try to make our simulation system as generic as possible while still being able to adequately carry out the simulations we require. In our case we want to be able to effectively simulate spacecraft and in particular formation flying spacecraft. This domain has widely divergent needs when compared to purely informational agents but our simulation system could be easily applied to more situated agents, in particular mobile robotics. The clear design philosophy behind the development of this simulation system, that is modularity and good object oriented design, means that without any extra work the simulation system can be used on other mobile robotics applications or seemingly different domains such as operational analysis which can be modelled with only a change to simulation and intelligence models of the agents.

The idea of trying to pursue a more generic structure for simulation models can however provide benefits. The development of a more generic system means that the designer must try to make the system as transparent as possible and maximise the ability of future users to add components or even change large portions of the system to suit their needs. Striving for extra clarity and extensibility in this way will inevitably improve the interfaces and construction of the code being generated. One of the most common problems, and indeed one that our system is trying to address is that a great deal of the work carried out in the scientific community is opaque to other researchers either because they do not have access to the code in question to repeat the tests carried or more commonly that the code produced has been developed with the specific purpose of carrying out a very well defined test and is totally impenetrable to any other user. This impenetrability usually comes from the software's non-existent or badly reduced documentation and the use of non-standard interfaces to other libraries.

One of the aims of our simulation system is to allow the user to use external libraries for certain parts of the system. At its most general this means that simulation models and intelligence models can be externally developed and plugged in to the system. The reason for this is that it is envisaged that our simulation system will be used at an early stage and primarily in the comparison of different architectures or intelligence approaches for the autonomous control system. One of the key aspects of the system developed here that is different to many others is that it is designed from the ground up to support multiple different platforms. The idea behind this was not to preclude using libraries or modules that a user may want to test purely because they were developed in another system or using a different programming language.

In order to carry out simulations using different libraries on different platforms we have to use a fairly specialised structure to enable all of the component parts to talk to each other. The structure however is perfectly compatible with multi-agent systems. In our simulation system communication between the different modules or libraries (not the agents themselves) is carried out by simple client server structure. This structure allows us to interface disparate technologies while still having full control over the communication and the interfaces.   It also allows us to easily deploy the system over multiple nodes as the client server structure is already there to allow this.

4.5.2 Discrete versus Continuous simulation systems

The client server structure allowing heterogeneous modules and components to interact with the simulation does however force our hand into making one specific decision; that is the decision of whether to make the simulation continuous or discrete in time. The asynchronous and heterogeneous nature of our system means that continuous real time simulation is not possible, but with a successfully developed discrete time system the simulation system can perform extremely varied simulations on differing hardware while still producing useful results.

One such discrete multi-agent simulation system is the Swarm system, originally developed in the mid-nineties [95]. Swarm has many interesting features but is a generic simulation tool and is able to model purely informational agents as well as situated agents and everything in between. The modelling formalism that the Swarm system adopts is to model independent agents interacting via sets of discrete events. Another interesting aspect of the Swarm system is its ability to define agents recursively. In the Swarm system groups of agents are known as swarms. Each agent in any given swarm can however itself be a swarm (sub-swarm) and consist of may sub agents itself. This recursive nature of the system allows for high level structures to be defined and the detailed structure of the system defined by replacing components parts of the high level swarms with sub swarms that add more fidelity to the model. The recursive nature can also be multi layered and can go any number of levels deep. The Swarm system is also able to dynamically create and destroy swarms which would allow for the dynamic increase or reduction in fidelity of the model or computational time as the simulation demands it. The recursive nature of the HASA architecture allows us to gain these benefits in our simulation suite.

4.6 Formation Flying for DARWIN

One of the problems facing the designers of future formation flying missions is the problem of how to adequately control multiple spacecraft in close proximity. Formation flying missions vary greatly and multiple satellite missions can have both scientific and commercial objectives. A current example of operating formation flying mission is the TerraSAR-X and TanDEM-X mission which uses interferometry to precisely control a close formation mapping of the earth to produce a more accurate digital elevation model for geophysical and environmental science [98, 99]. Another example is the PRISMA mission [100] which is designed as a formation flying and in orbit servicing technology demonstrator. As can be seen it is not just test beds and technology trials for formation flying that are being developed and flown and the technology is rapidly improving. The hardware is also already in place for interferometric formation flying which means that the main stumbling blocks for mission such as DARWIN is the obvious financial and political support and the development of sufficiently reliable autonomous control system.

As touched on previously the control requirements for a multi satellite mission greatly exceed those of a single satellite mission. As well as the development and implementation of more complex control laws a great deal of consideration must be given to the communication structure and to the decision making abilities of the system. To summarise the problem of multi spacecraft control does not rely solely on the development of the correct control laws but also on other factors. Luckily all of these factors can be described and implemented by using a multi-agent control system in coherent way. The communication abilities are inherent to a multi-agent system and as previously discussed decision making processes can also be developed to give the agents and the system as a whole the desired level of autonomy. The control laws will be encoded in the agents themselves as knowledge and enacted by the reactive parts of the system.

Understandably there has been great deal of work carried out in the field of controlling a formation of spacecraft. This work varies from classical control work [101] to extensive work on an agent based approach to formation flying [102, 103].

4.7 Testing the simulation suite

The simulation suite that we have developed allows for the testing of different formation flying strategies. As the system is modular any given multi-agent system can be coupled with any formation flying control components and tested. The simulation system also allow the components to be run on different nodes or to have been developed in different programming languages so if the software for the given formation flying strategy has already been developed it can be easily integrated into the system without re-writing it.

4.7.1 Artificial Potential fields

One such formation flying strategy that has been tested is that of using artificial potential fields to govern the motion of the spacecraft in the formation. The use of artificial potential fields is extremely popular in the field of mobile robots in general and in spacecraft control. Good examples of the current work can be found in [104–107]. At its most general the method works by setting up a series of attractor and repulsor nodes in the spacecraft state space. A set of heuristics are then chosen to determine how the spacecraft reacts to the attractors and repulsors. The influence of the attractor and repulsor nodes are governed by set of shaping parameters that vary the effect of the nodes over distance to achieve the desired control response.

The artificial potential functions implemented were those found in [108] and are described below.

Suppose we have a set of spacecraft ($1 \leq i \leq N$) that are interacting by an artificial potential function, U. The gradient of the artificial potential function defines a virtual force acting on each individual spacecraft. With a spacecraft mass, m, spacecraft position,x_i and a spacecraft velocity, v_i the dynamics can be described as:

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i \tag{4-5}$$

$$m\frac{d\mathbf{v}_i}{dt} = \nabla_i U^s(\mathbf{x}_i) - \nabla_i U^r(\mathbf{x}_{ij}) - \sigma\mathbf{v}_i \tag{4-6}$$

The virtual force experienced by the spacecraft is therefore dependent on two artificial potential functions and a dissipative term. The first artificial potential function is the steering potential:

$$U^s(\mathbf{x}_i) = -\frac{1}{2}\mu\left((x_i^2 + y_i^2)^{\frac{1}{2}} - r\right)^2 + \frac{1}{4}\left((x_i^2 + y_i^2)^{\frac{1}{2}} - r\right)^4 \tag{4-7}$$

The second is the repulsive potential:

$$U_{ij}^R = \sum_{j,j\neq i} C_r \exp^{-|x_{ij}|/L_r} \tag{4-8}$$

In this example we used 3 types of nodes. The attractive 'gather' type node has a relatively low intensity but a large radius and is used to guide the spacecraft from their starting points to the general area of the final formation. The exact formation itself is governed by attractive 'dock' nodes that have a high intensity but low radius. These are designed to guide the spacecraft to an exact position. There are also repulsive 'avoid' nodes which are used in the potential field to stop the spacecraft stopping at centre of the gather node for instance. Each spacecraft is treated as an avoid node by all the other spacecraft to avoid collisions.

The agents on any given spacecraft have access to their M_i as well as data about the position and state of the other spacecraft in the formation. All of the communication between the agents is carried out within the MAS and consists of the agents passing simple FIPA/ACL messages to one another in a peer-to-peer fashion.

The functions governing the artificial potential fields are encapsulated within each Mi and as such are repeated across all of the spacecraft. In the case outlined in this paper the structure of the M_i for each spacecraft is identical but this need not always be the case and changing this would allow us to test the negotiation and conflict resolution abilities of the MAS.

As a proof of concept the artificial potential field controller was combined with the DARWIN MAS and tested in a 2D simulation with no gravity. As a test scenario the spacecraft have random starting points within a set radius of the desired formation. The problem for the agents is to acquire the desired formation from their relative starting points. The agents use the distributed artificial potential field to move towards the target positions while avoiding each other as shown in Figure 4-9. In this example there is one centred 'gather' node with a radius of 200km and a normalised magnitude of 0.5 and one 'avoid' node with a radius of 20km and a normalised magnitude of -1 (the magnitude is always negative for repulsors). Defining the desired formation are three 'docking' nodes with a radius of 30km and a normalized magnitude of 1. Each agent is also viewed as an 'avoid' node with a radius of 30km and a normalised magnitude of -1.

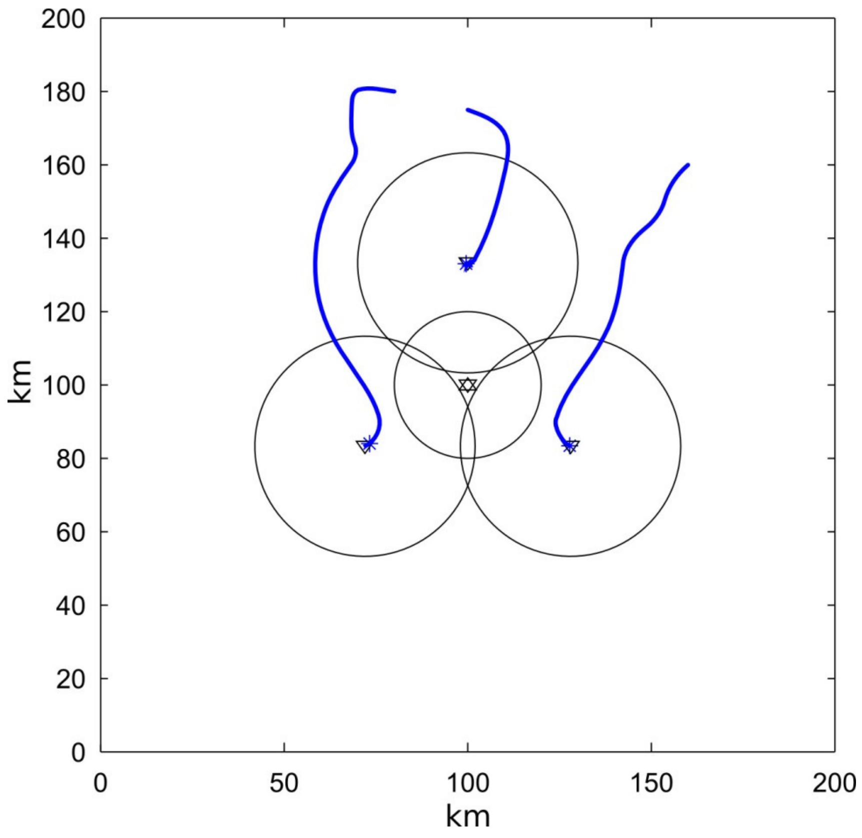


Figure 4-9 3 spacecraft start from random locations within the area of interest shown (200km by 200km) and successfully acquire their formation.

This scenario was run 500 times and the results of all the successful tests sorted by duration are shown in Figure 4-10. The MAS controlled spacecraft successfully acquired their desired formation within the time limit (48 hours) in 69.5% of these tests. This time limit was chosen as a realistic time frame for acquiring or changing a formation on the order of 10s of kilometres. Figure 4-11 shows a heat map of the final position of the spacecraft which did not successfully get to their desired position. This was generated to show the most common areas where a spacecraft ends up when it fails to acquire its desired position.

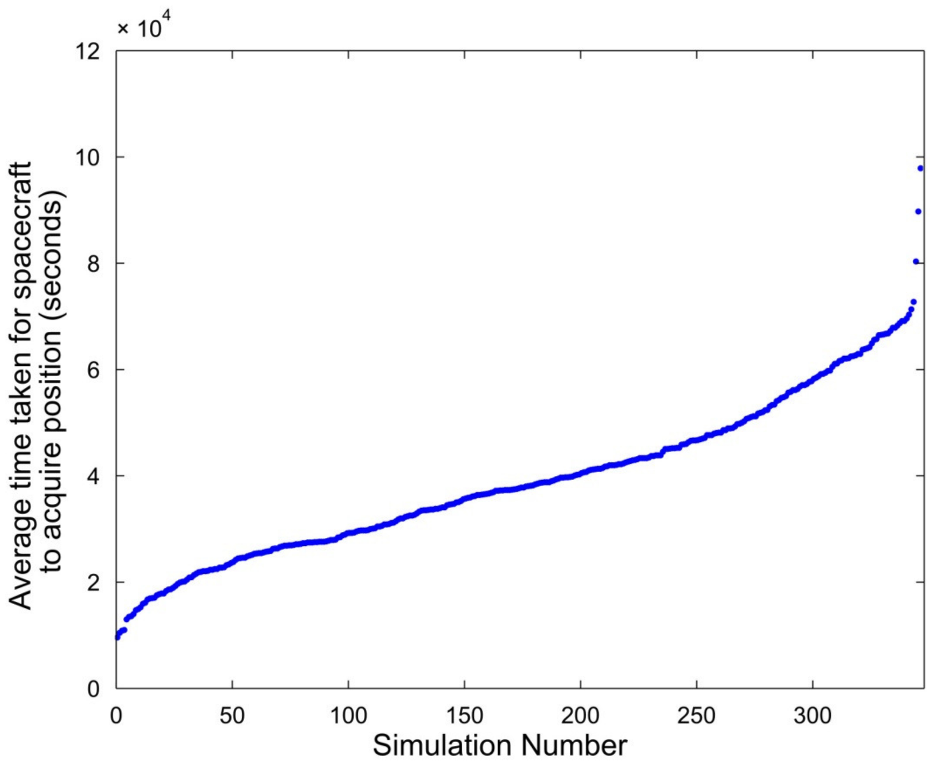


Figure 4-10 Distribution of times for spacecraft to acquire formation, ordered from lowest time to highest time.

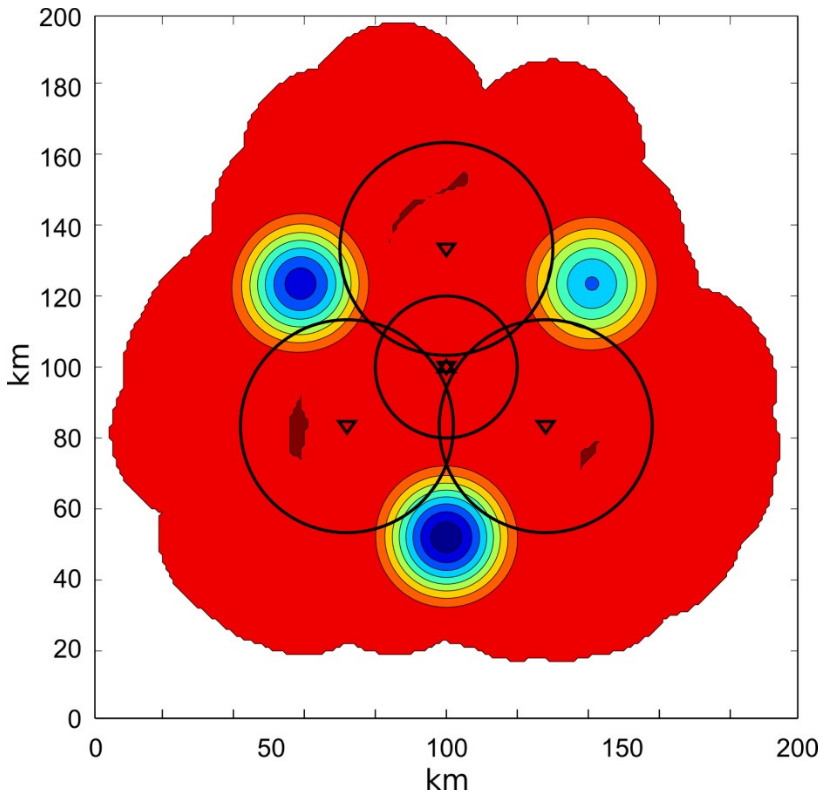


Figure 4-11 Plot showing the distribution and density of the final position of spacecraft that failed to acquire the formation

The success of a testing suite is independent however of any success of the controller. The success of the testing suite is based on its ability to help us find scenarios where the MAS fails to achieve the desired formation. In this case whenever the simulation failed the parameters are stored so we can investigate these failures and what might have caused them.

The most common failure mode in this scenario was two of the agents acquiring the correct position but the third agent stopped when it found the equilibrium point created by the other agent’s repulsive forces rather than its intended destination .

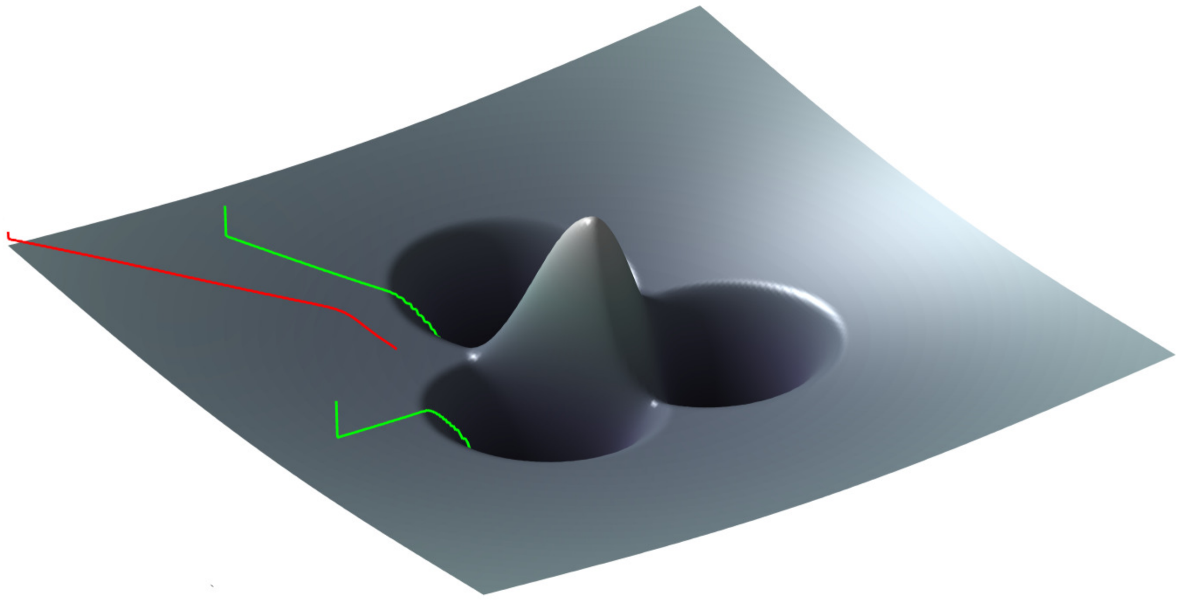


Figure 4-12 Local equilibrium point leading to incorrect final position for the red agent

The test suite has proven useful even in this simplified case when trying to find failure modes for an autonomous multi-agent system. As the system under test increases in complexity the number of unknown failure modes will increase and we hope this tool will aid us in finding them and helping us to validate the system.

Grouping and identifying the different failure modes of the formation raises some interesting issues. In this simplistic case the failure modes can be grouped as either in formation or out of formation errors. In formation errors can then be grouped based on rotational, reflectional and translational symmetry. Out of formation errors are those in which the formation is no longer valid (Figure 4-13).

Figure 4-12 shows the artificial potential field as a surface where peaks are repulsive nodes and troughs are attractive nodes. In this figure the spacecraft can be thought of marbles rolling on this surface under the effect of gravity. Two of the spacecraft quickly enter two of the attractive nodes but one instead finds a position at a local minimum from which it can’t escape.

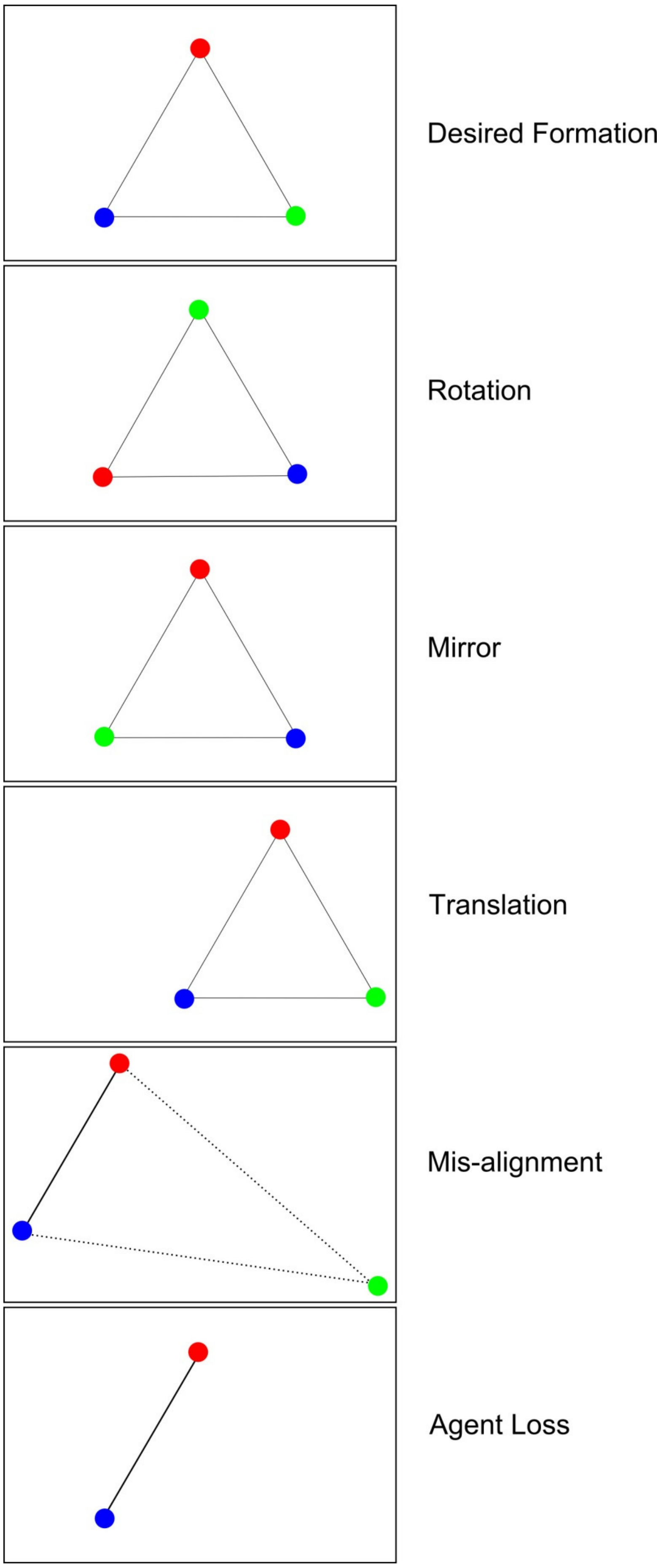


Figure 4-13 3 agent failure modes

4.7.2 CRTBP

The next step was to implement a more realistic dynamical model. The circular restricted three body problem (CRTBP) involves a coordinate system that rotates around the centre of mass (the barycentre) of the system comprising the two main bodies, in this case the Sun and the Earth. In this rotating frame the position of the Sun and the Earth appear static from the point of view of the satellite as their orbits are presumed to be perfectly circular. In order to further simplify the model the problem is formulated in non-dimensional units chosen for our convenience. In this problem it us useful to define the unit of length as the distance between the Sun and the Earth (as the Earth's orbit around the sun is defined as circular this stays constant). The unit of time is chosen so that the Sun and the Earth have an angular velocity about the centre of mass that is equal to one. This means that one full orbit of the earth has a period of 2π.

The x and y coordinates of the Sun and the Earth are:

$$x_{sun} = -\mu \quad y_{sun} = 0 \tag{4-9}$$
$$x_{earth} = 1 - \mu \quad y_{earth} = 0$$

The gravitational potential that the satellite experiences due to the mass of m_1 and m_2 in our normalised units is:

$$U = \frac{\mu_1}{r_1} - \frac{\mu_2}{r_2} - \frac{1}{2}\mu_1\mu_2 \tag{4-10}$$

Where the normalised mass of the three body system is:

$$\mu = \frac{m_2}{m_1 + m_2} \tag{4-11}$$

We can safely assume that as $m_1 \;>>\; m_2$ then in our normalised system:

$$\mu_1 = 1 - \mu \tag{4-12}$$

$$\mu_2 = \mu$$

The primary-spacecraft distance, r_1, and the secondary-spacecraft distance, r_2, can be defined as follows:

$$r_1^2 = (x + \mu_2)^2 + y^2 + z^2 \tag{4-13}$$

$$r_2^2 = (x - \mu_1)^2 + y^2 + z^2 \tag{4-14}$$

If we then define C$_1$ and C$_2$ as:

$$c_1 = -\frac{\mu_2}{r_2} \tag{4-15}$$

$$c_2 = -\frac{\mu_1}{r_1} \tag{4-16}$$

The equations of motion in the rotating frame are given by [109]:

$$\ddot{x} = 2\dot{y} + x + c_1(x - \mu_2) + c_2(x - \mu_1) \tag{4-17}$$

$$\ddot{y} = -2\dot{x} + y + (c_1 + c_2)y \tag{4-18}$$

$$\ddot{z} = (c_1 + c_2)z \tag{4-19}$$

Now that we have the equations of motion of a spacecraft interacting with the Sun and the Earth we must find a suitable orbit.

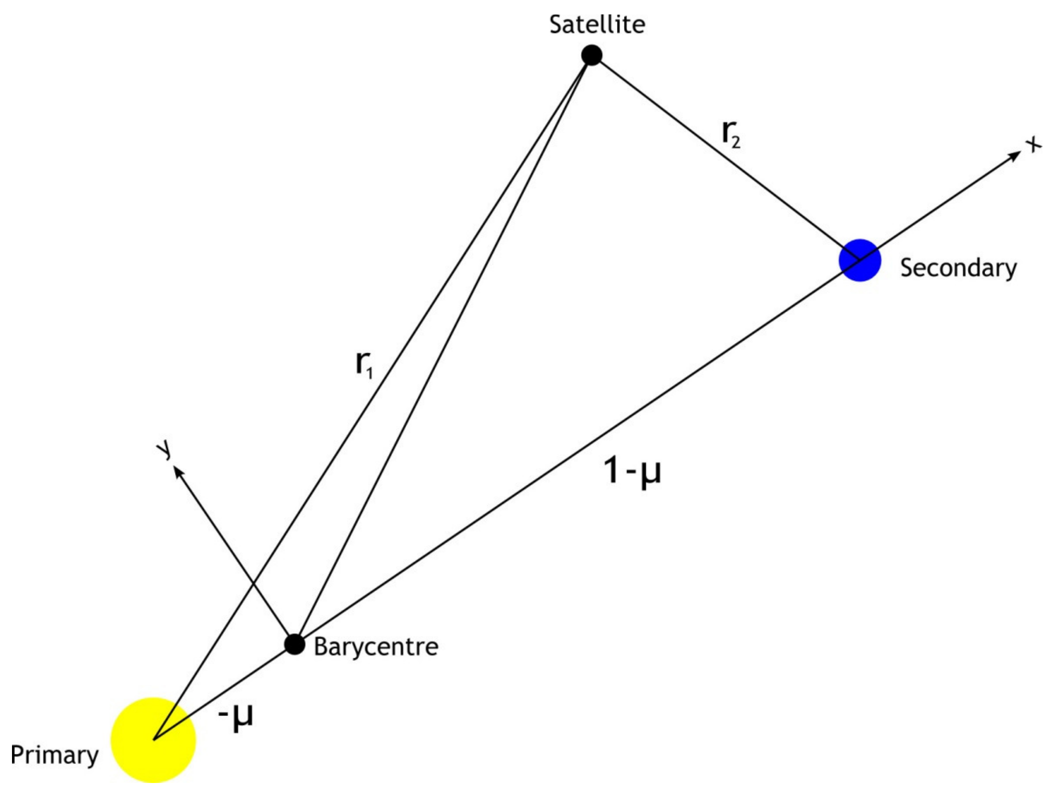


Figure 4-14 CRTBP

4.7.3 Halo Orbit

For our example of the DARWIN mission we need to model our spacecraft formation in orbit around the second Sun-Earth Lagrangian point (L2). There are 5 Lagrangian points in the Earth-Sun system as shown below. They are points where a third body (our satellite) would experience zero net force as it followed the orbit of the Earth.

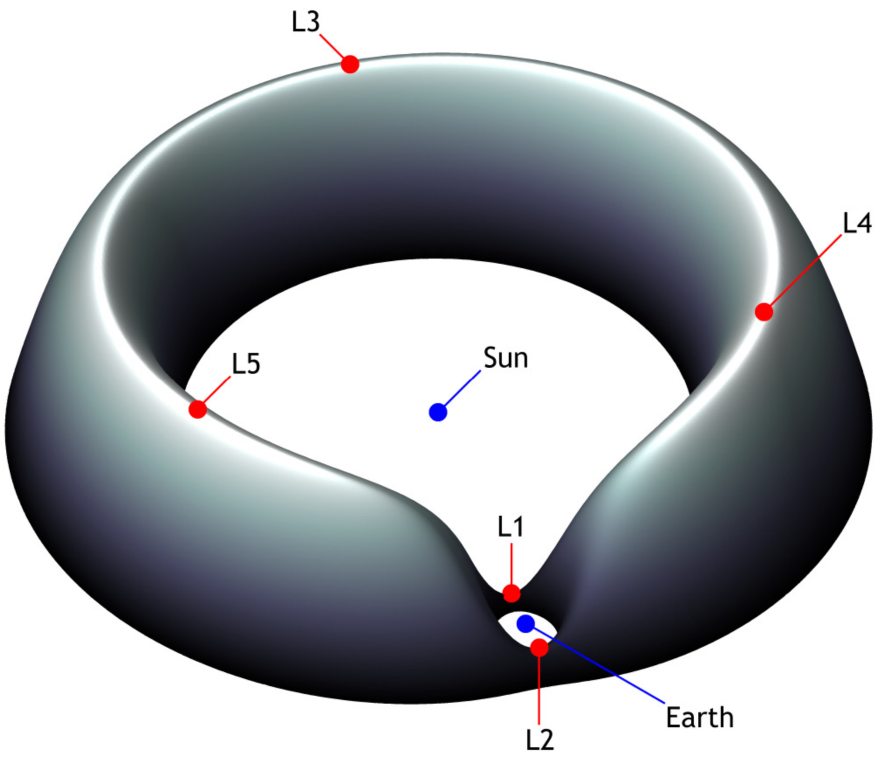


Figure 4-15 Gravitational potental in the rotating Earth-Sun system

In Figure 4-15 the x and y dimensions equate to those in the CRTBP and z shows the magnitude of the gravitational potential. The Lagrangian points are situated at local minima and maxima of this surface.

The L2 point has been chosen for missions such as DARWIN and the terrestrial planet finder (TPF) for a number of reasons [110, 111]. An orbit near L2 is easy and inexpensive to get to from Earth. A halo orbit around L2 also offers the spacecraft a near constant geometry with the Sun the Earth and the Moon always behind the spacecraft which is of great benefit when the mission has very heat sensitive instruments and fine tolerances to operate within. A halo orbit will also provide a near constant communications geometry with Earth due to its constant distance at around 1.5 million km. The slightly lower energy required when inserting a satellite into a halo orbit when compared to a heliocentric orbit as well as the ease with which additional or replacement satellites could be sent to the halo orbit makes it highly appealing for interferometric telescope array missions.

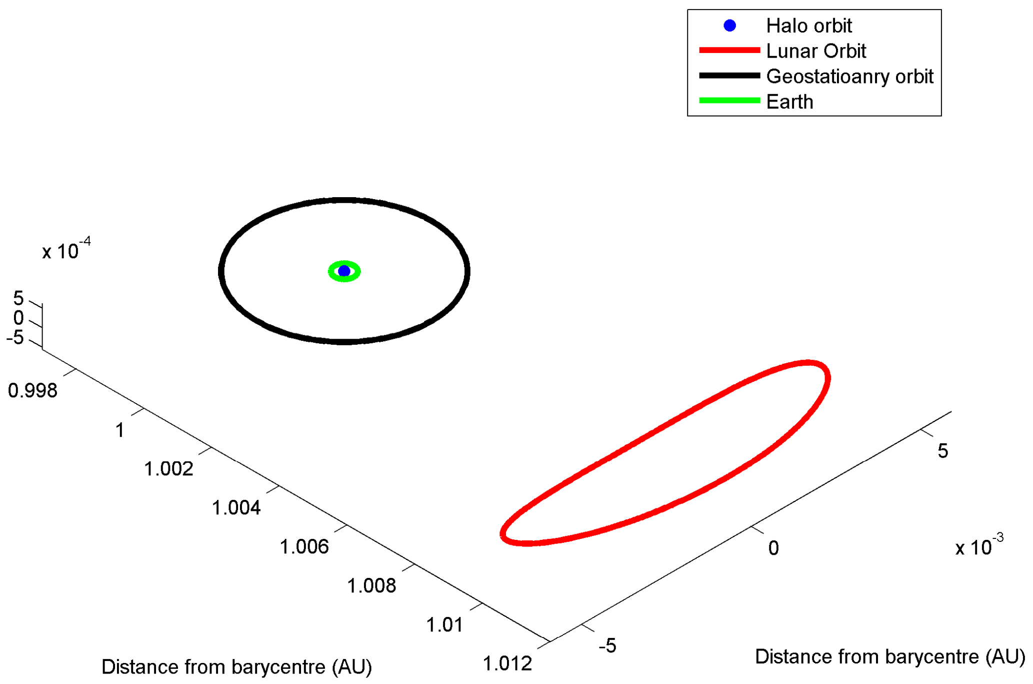


Figure 4-16 Halo orbit relative to the earth and lunar orbit, isometric view

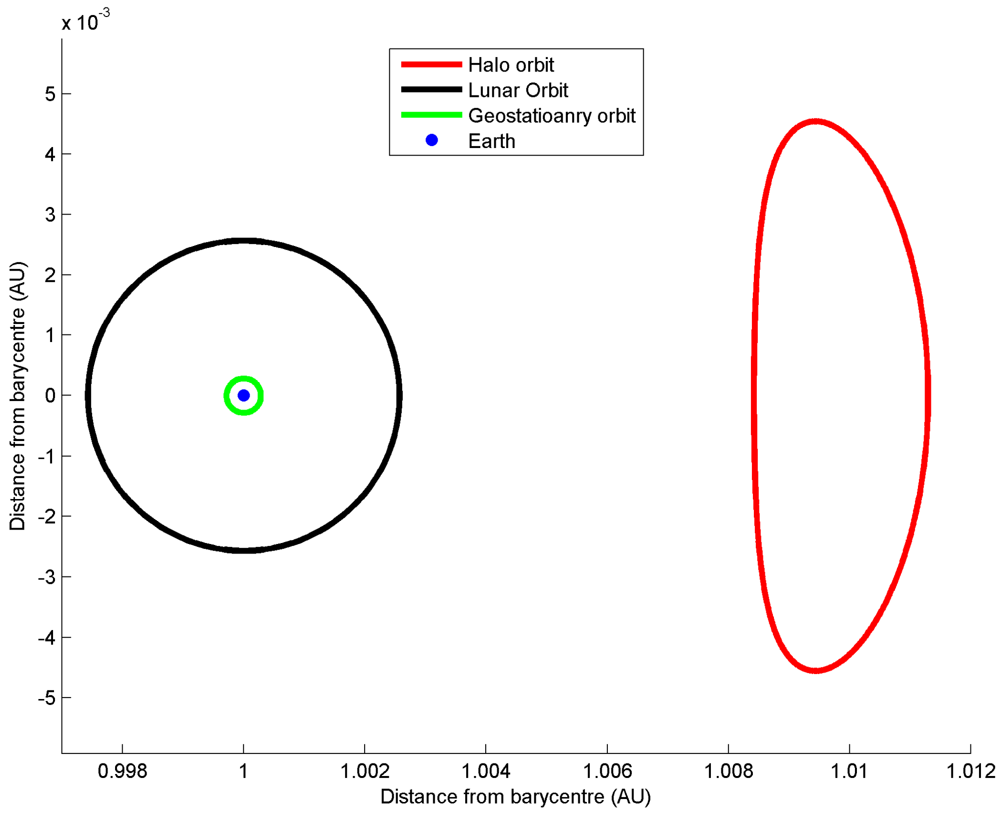


Figure 4-17 Halo orbit relative to the earth and lunar orbit, top view

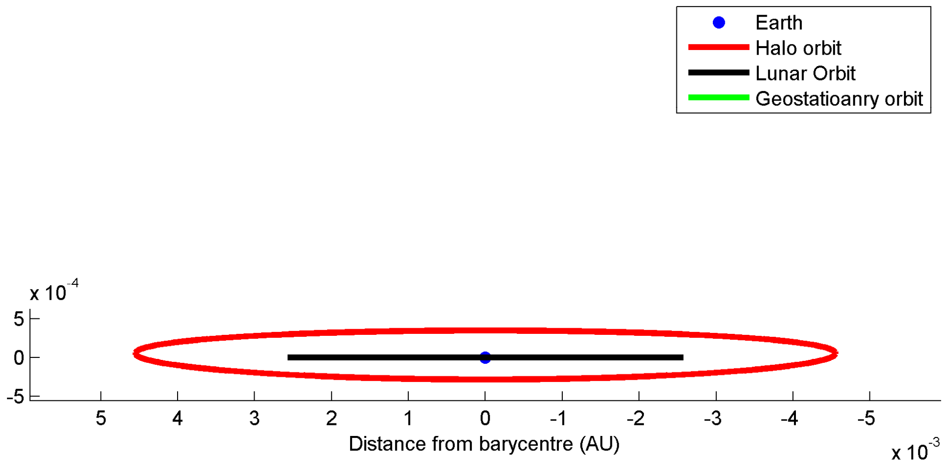


Figure 4-18 Halo orbit relative to the earth and lunar orbit, front view

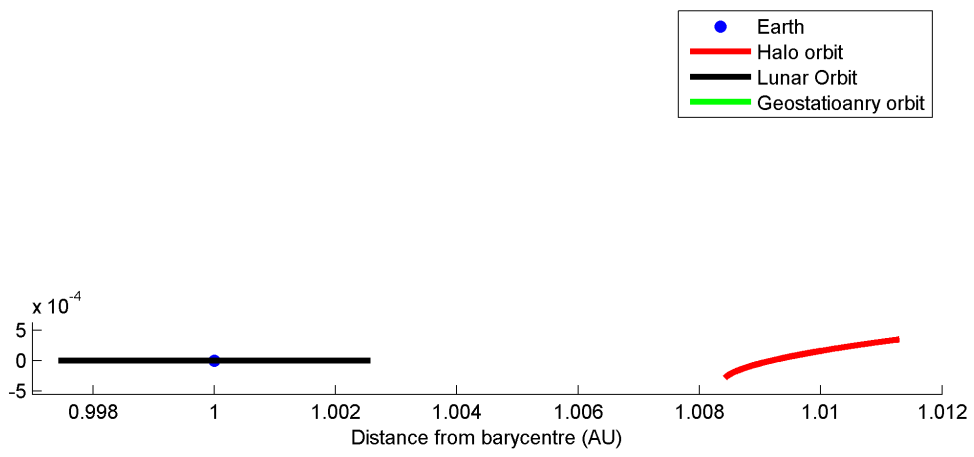


Figure 4-19 Halo orbit relative to the earth and lunar orbit, side view

Figure 4-16-Figure 4-19 show a typical halo orbit (in red) around the L2 point. For scale an Earth geosynchronous orbit is shown in green and the orbit of the moon is shown in black.

The disadvantages of a halo orbit are that the orbit is unstable and will thus require station keeping manoeuvres and its distance from the Earth may lead to own communications issues.

4.7.4 Generation of the Halo orbit

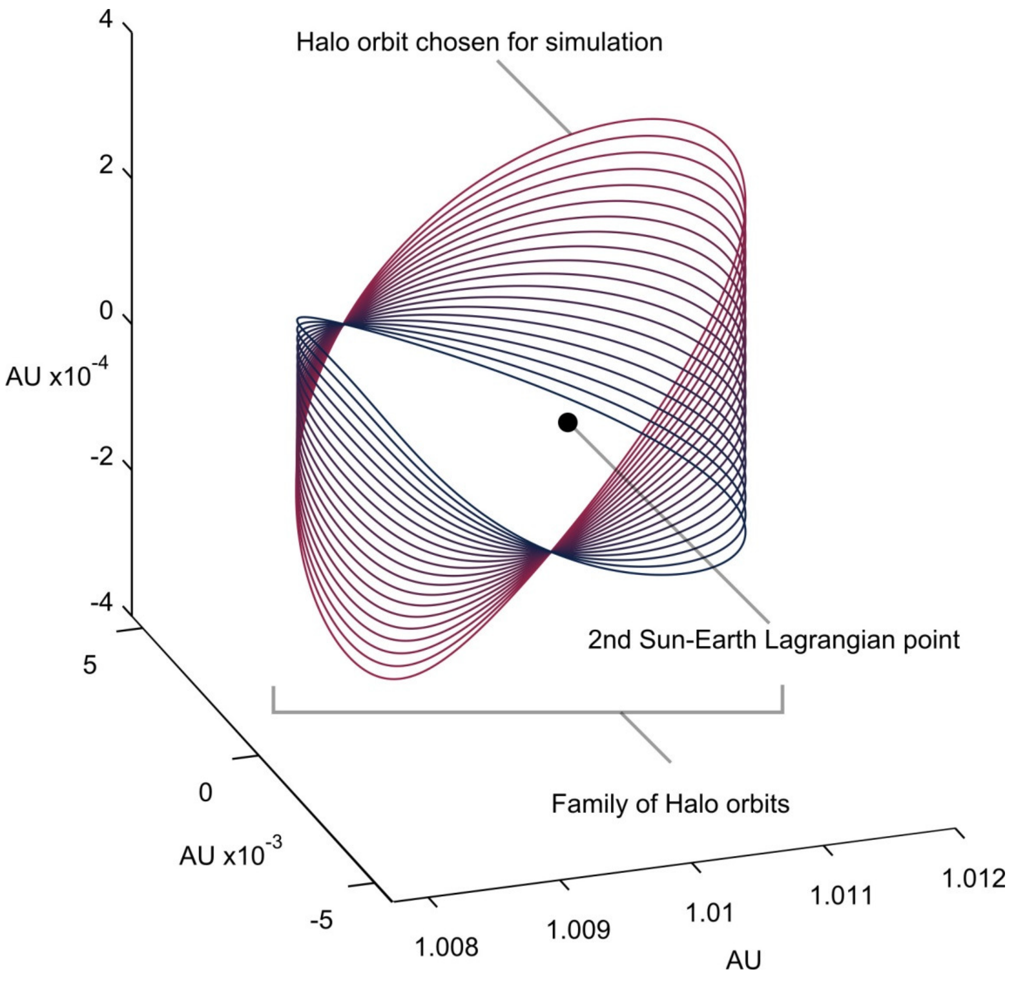


Figure 4-20 Halo orbit family

The procedure described in [112] and in [113] was used to generate the Halo orbits used for the simulation of the DARWIN mission. The process works by refining an initial estimate of an orbit by using the Newton method. The initial estimate for an orbit is defined by its initial position, initial velocity and its period: $(x_0, y_0, z_0, \dot{x}_0, \dot{y}_0, \dot{z}_0, \tau)$. We will hold the x_0 coordinate fixed and search for $z_0^*, \dot{y}_0^*$ and τ^* such that $\dot{x}^*(\tau^*)$, $\dot{z}^*(\tau^*)$ and $y^*(\tau^*)$ are all zero.
Define $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ by

$$f(z, y,\dot{}\tau) = \begin{bmatrix} \phi_4(x_0, 0, z, 0, \dot{y}, 0, \tau) \\ \phi_6(x_0, 0, z, 0, \dot{y}, 0, \tau) \\ \phi_2(x_0, 0, z, 0, \dot{y}, 0, \tau) \end{bmatrix} \tag{4-20}$$

Where

$$f(z,y,\dot{\tau}) = \begin{bmatrix} \phi_1(x,y,z,\dot{x},\dot{y},\dot{z},\tau) \\ \phi_2(x,y,z,\dot{x},\dot{y},\dot{z},\tau) \\ \phi_3(x,y,z,\dot{x},\dot{y},\dot{z},\tau) \\ \phi_4(x,y,z,\dot{x},\dot{y},\dot{z},\tau) \\ \phi_5(x,y,z,\dot{x},\dot{y},\dot{z},\tau) \\ \phi_6(x,y,z,\dot{x},\dot{y},\dot{z},\tau) \end{bmatrix} \tag{4-21}$$

To find the initial conditions for the halo orbit it is sufficient to find $x^* = (z_0^*, \dot{y}_0^*, \tau^*)^T$ satisfying the equation

$$f(z_0^*, \dot{y}_0^*, \tau^*) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4-22}$$

The Newton method for refining the orbit can be expressed as:

$$\mathrm{x}_{n+1} = \mathrm{x}_n - [Df(\mathrm{x}_n)]^{-1} f(\mathrm{x}_n) \tag{4-23}$$

With $\mathrm{x} = (z, \dot{y}, \tau)$ and $\mathrm{x}_0 = (z_0, \dot{y}_0, \tau_0)$ and the differential equal to:

$$Df(x) = \begin{bmatrix} \frac{\partial}{\partial z}\phi_4 & \frac{\partial}{\partial \dot{y}}\phi_4 & \frac{\partial}{\partial \tau}\phi_4 \\ \frac{\partial}{\partial z}\phi_6 & \frac{\partial}{\partial \dot{y}}\phi_6 & \frac{\partial}{\partial \tau}\phi_6 \\ \frac{\partial}{\partial z}\phi_2 & \frac{\partial}{\partial \dot{y}}\phi_2 & \frac{\partial}{\partial \tau}\phi_2 \end{bmatrix} \tag{4-24}$$

$$= \begin{bmatrix} \Phi_{(4,3)} & \Phi_{(4,5)} & g_4(x_0, 0, z(\tau), 0, y\dot{(}\tau), 0) \\ \Phi_{(6,3)} & \Phi_{(6,5)} & g_6(x_0, 0, z(\tau), 0, y\dot{(}\tau), 0) \\ \Phi_{(2,3)} & \Phi_{(2,5)} & g_2(x_0, 0, z(\tau), 0, y\dot{(}\tau), 0) \end{bmatrix}$$

Where $g\colon U \subset \mathbb{R}^6 \to \mathbb{R}^6$ is the vector field of the CRTBP.

$$g(x,y,z,\dot{x},\dot{y},\dot{z}) = \begin{bmatrix} g_1(x,y,z,\dot{x},\dot{y},\dot{z}) \\ g_2(x,y,z,\dot{x},\dot{y},\dot{z}) \\ g_3(x,y,z,\dot{x},\dot{y},\dot{z}) \\ g_4(x,y,z,\dot{x},\dot{y},\dot{z}) \\ g_5(x,y,z,\dot{x},\dot{y},\dot{z}) \\ g_6(x,y,z,\dot{x},\dot{y},\dot{z}) \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ 2\dot{y} + D_xU \\ -2\dot{x} + D_yU \\ D_zU \end{bmatrix} \tag{4-25}$$

With the newton method set up as described then if the initial x_0 is close to a halo orbit then $x_n \to x^*$ as $n \to \infty$.

Once a periodic Halo orbit has been obtained further Halo orbits are found by incrementing parameters of the known halo orbit and re-running the Newton method to find a new periodic orbit.

The initial orbit was taken to be a Lyapunov orbit (that is, an orbit that is planar in x-y and has no z component) and its initial conditions $(x,y,z,\dot{x},\dot{y},\dot{z})$ are shown below.

$$\mathrm{x}_0 = \begin{bmatrix} 1.00675137755428 \\ 0 \\ 0 \\ 0 \\ 0.01867323092996 \\ 0 \end{bmatrix}$$

The initial conditions of this orbit are then given a slight out of plane (z) component and refined using the Newton method to produce the following orbital initial conditions.

$$\mathrm{x}_0^* = \begin{bmatrix} 1.00842815565444 \\ 0 \\ 0.0001 \\ 0 \\ 0.00981039306520 \\ 0 \end{bmatrix}$$

To produce a family of Halo orbits the initial z position is decremented by 0.00002AU and then the orbit is refined again. This process is repeated as many times as required to get the desired family of orbits.

4.7.5 Multi-agent control

Multi-agent controlled formation flying for a DARWIN type mission was investigated by coupling the MAS outlined previously and running it in the Java agent development framework (JADE) [114, 115] with the CRTBP model. The structure of the simulation suite allows for easy replacement of one simulation model with another.

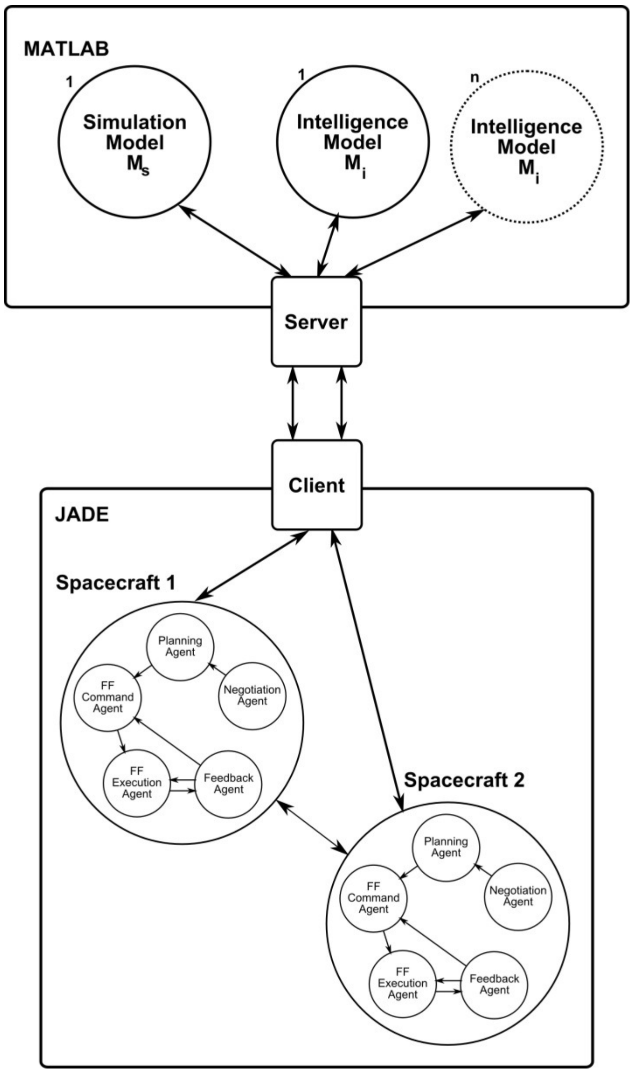


Figure 4-21 Agent test suite structure

Figure 4-21 shows the agents for the DARWIN multi-agent system running with JADE interacting with intelligence and simulation models running on another process, in this case Matlab.

The station keeping of the spacecraft was achieved by writing a Proportional-Integral-Derivative (PID) type feedback controller to counteract any drift due to the fact that halo orbit is unstable. The PID controller was written in Java as a pure Java object. This was to show the compatibility in the test suite between the Jade agents, Matlab models and pure Java objects.

4.7.6 PID Controller

A PID controller is a type of feedback controller made of three constituent parts. The controller takes in the desired output of the system as an input (known as the reference signal). The difference between the reference signal and the output of the process is known as the error and it is this error value that is fed into the controller. The controller then generates a signal based on this error and outputs it to the process. This closed loop runs continuously.

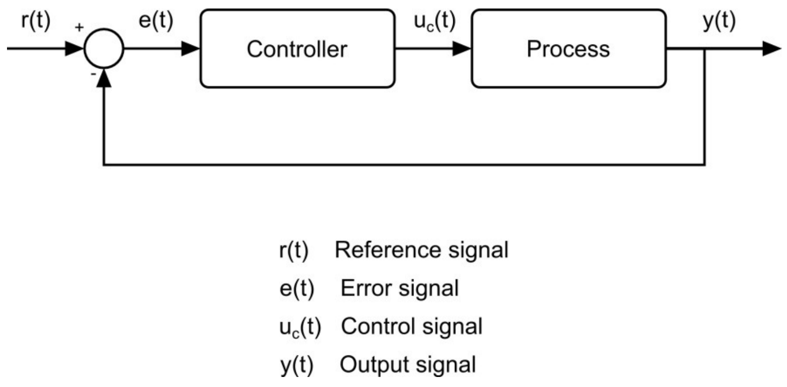


Figure 4-22 Feedback Controller

In the proportional (P) part of the controller the error signal is multiplied by some value K_p before being output. In the integral (I) part of the controller the past error values are integrated over time and then multiplied by a gain K_i before being output. In the derivative (D) part of the controller the rate of change of the error is multiplied by a gain K_d before being output [116]. The equation of a generic PID controller is:

$$u_c(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt} \tag{4-26}$$

A complete PID controller diagram is shown below.

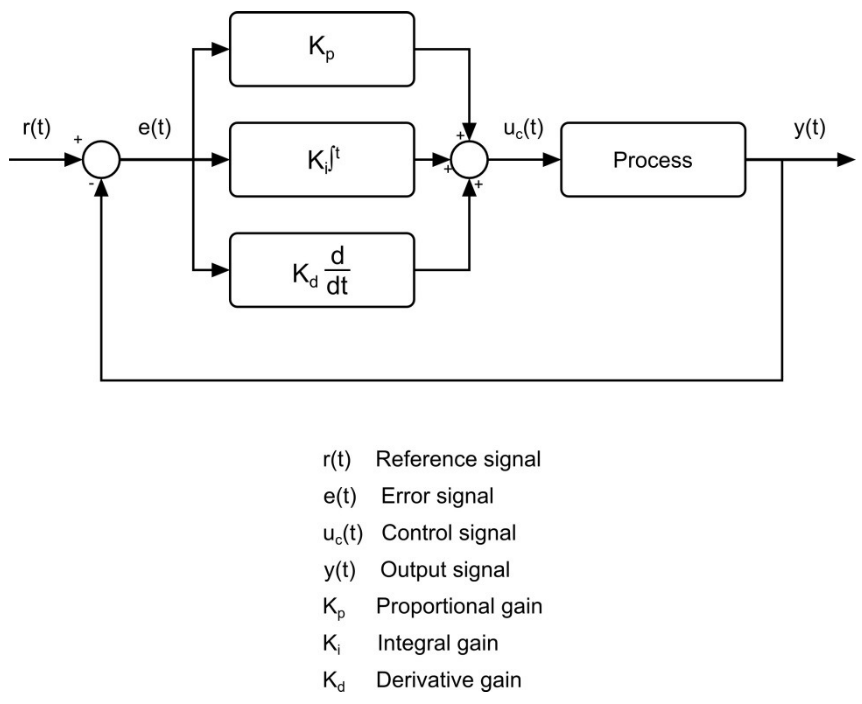


Figure 4-23 PID feedback controller

Control Term	Reference Tracking	
	Transient	Steady state
P	Increasing $K_p > 0$ speeds up the response	Increasing $K_p > 0$ reduces but does not remove steady state offset
I	Introducing integral action, $K_i > 0$, gives a wide range of response types	Introducing $K_i > 0$ eliminates offset in the reference response
D	Derivative action $K_d > 0$ gives a wide range of responses and can be used to tune response damping	Derivative action has no effect on steady state

Table 4-1 Effects of individual terms of a PID controller

It was this PID controller that was implemented in the test suite for orbital maintenance and individual spacecraft control.

4.7.7 Simulation Models

In order to show the workings of the multi-agent testing suite two different simulation models were developed. In the first the dynamics of the system were modelled in Matlab and propagated by the testing suite using Matlab's own built in ordinary differential equation (ODE) solver (ODE113).

In the second simulation model the CRTBP dynamics are modelled using Simulink. The Simulink model is shown in appendix C. This approach may seem counter-intuitive compared to implementing the equations of motion as simple Matlab function but it offers up some interesting possibilities. One benefit of using Simulink is that high level coding knowledge is not required and complex models can be constructed piece by piece in a relatively intuitive way.

The other benefit of using Simulink is that the simulation model designer has access to a highly developed suite of tools in Simulink. To illustrate this point the station keeping controller was implemented as a single PID controller control block in the Simulink model and its gains automatically tuned using the built in tools.

Both models interact with the MAS in the same manner and use the same mathematical model.

During the construction of the JADE/Java/Matlab model it became apparent that the level of numerical precision was an important limiting factor in determining the accuracy of the discrete time based simulation suite such as this. For most models the key variables can be expressed precisely enough using Javas built in double variable type but the normalised non-dimensional units used in the CRTBP required a higher level of precision in order to avoid injecting errors and propagating them. As such the BigDecimal class was used to represent all of the variables in the system. The BigDecimal class offered the required precision at a higher computational cost.

The velocities and accelerations acting on a craft following of L2 Halo reference orbit (from here on referred to as the 'reference orbit') are shown below (Figure

4-24 and Figure 4-25) and show the unsurprising cyclical nature of the velocities and accelerations in our rotating frame (deviations in x are dark blue, y are green and z is red).

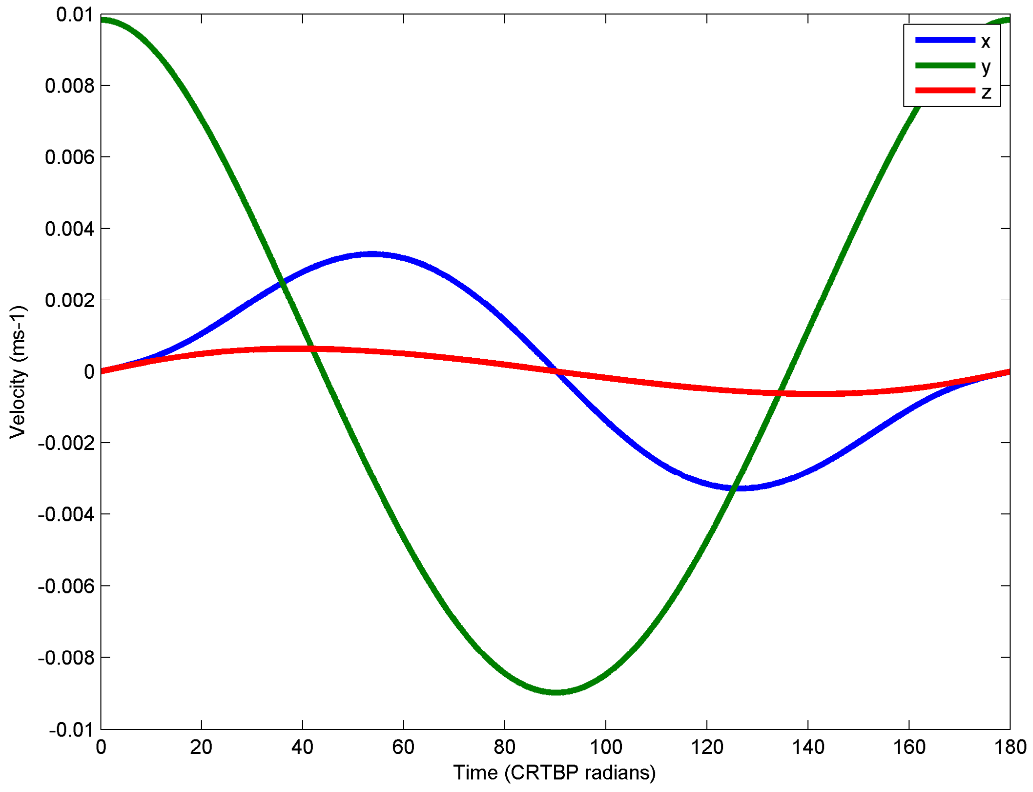


Figure 4-24 Accelerations during Halo orbit

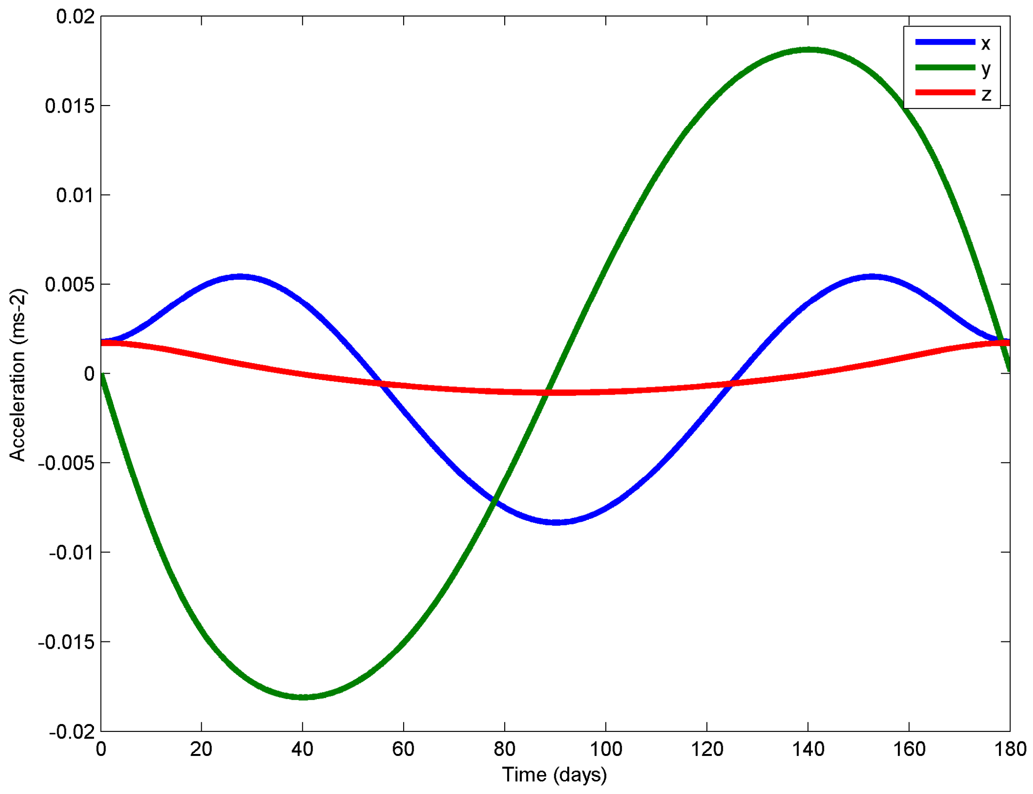


Figure 4-25 Velocities during Halo orbit

In Figure 4-26 a craft is placed a given distance in one axis away from the reference orbit and the model propagated through time. The results show the craft drifting away from the reference orbit when no orbital correction manoeuvres are proscribed.

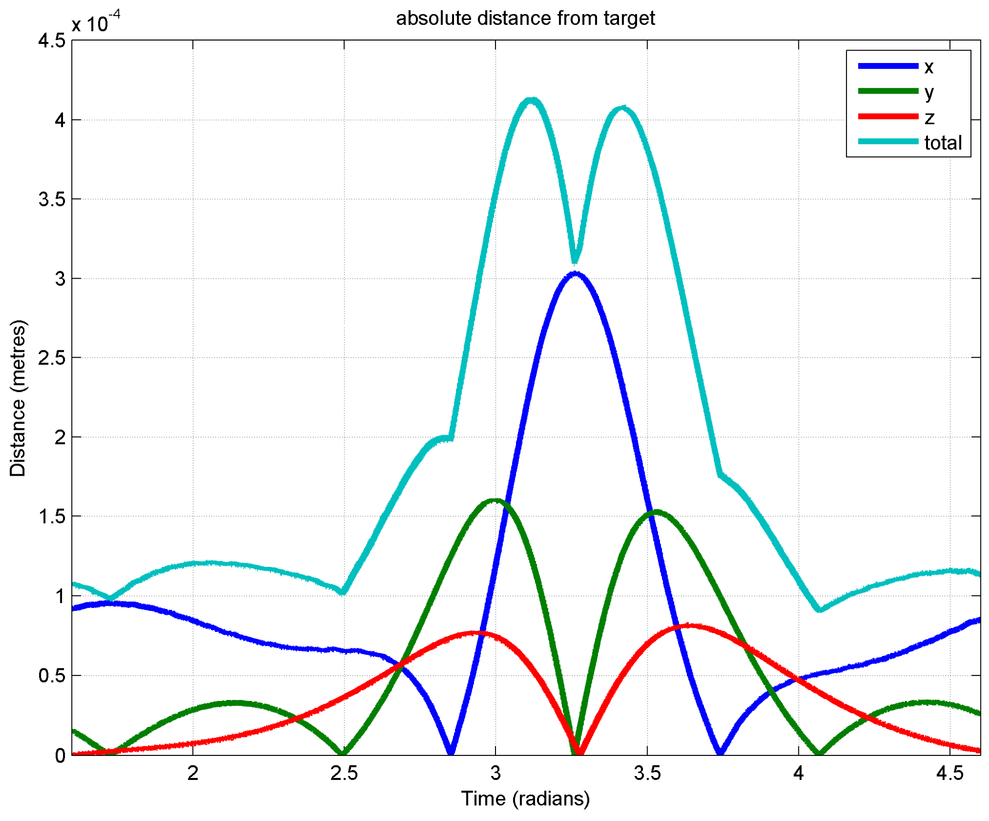


Figure 4-26 Absolute distance from reference orbit over one orbital period

It should be noted that the station keeping manoeuvres required to keep the spacecraft on the desired orbit around the L2 point are relatively small (15.55 ms^{-1}) and can be easily accounted for in a mission's Δv budget. In this simulation the station keeping thrusts are modelled as instantaneous and carried out at each time step. It is noted that a continuous thrust strategy would give better results in a real mission but that method is not implemented here and does not detract from the conclusion that are made about the test suite and the multi-agent system for controlling the DARWIN mission.

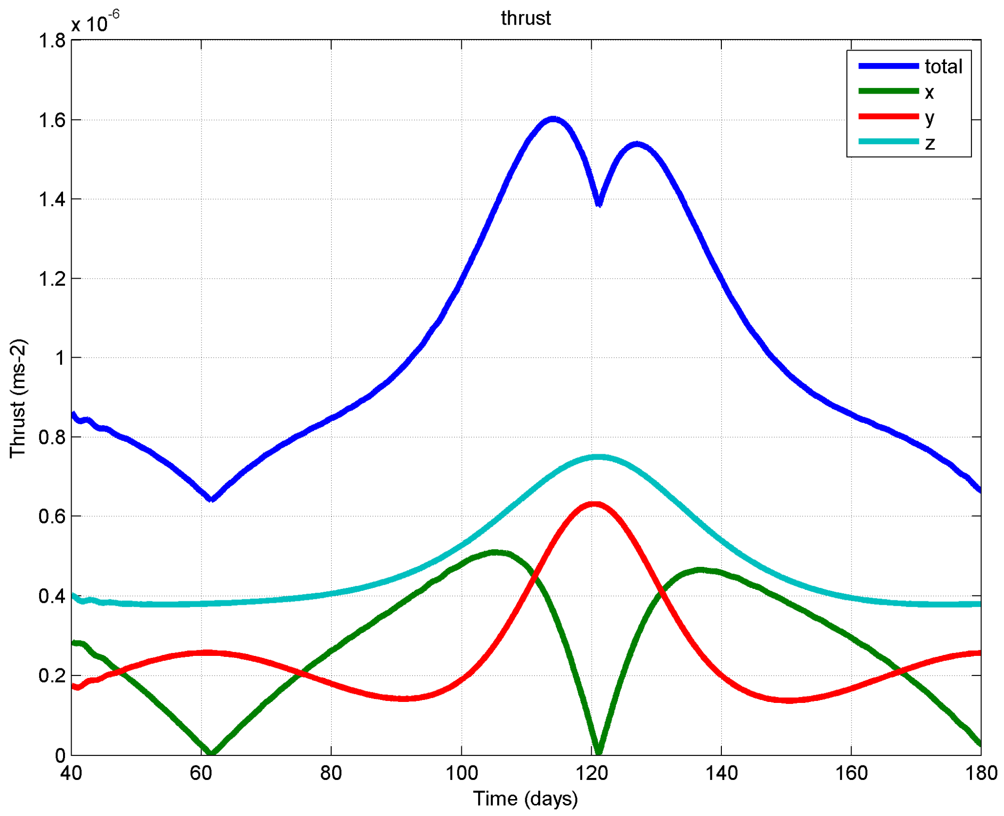


Figure 4-27 Thrust of spacecraft during station keeping on a single orbit

4.7.8 Test scenarios

In order to try to demonstrate that the multi-agent test suite can be used to help efficiently model and test multi-agent controlled space missions a number of test scenarios have been developed. These test scenarios are designed to serve a number of purposes, their primary purpose is to demonstrate that the architecture of the test suite is robust enough to handle multiple agents and simulation models and to show its ability to scale to progressively larger and simulation structures with many agents and simulations running in concert.

The test scenarios were designed to test the multi-agent test suites full range of modes, most notably the ability of the test suite to allow for the seamless interoperation between the autonomous agents and components running in Java, C++ and Matlab.

The philosophy behind designing these tests was to start at a very basic level and then add one component or change at each level and observe how the simulator coped with the slowly increasing complexity of the simulation.

The scenarios were built up starting from simple validation tests with one agent interacting with its associated environment model. In all of the test cases developed, the environmental model used was the CRTBP model previously outlined. The first test case comprised a single satellite following the previously mentioned Halo orbit around the second Lagrangian point. The agent was not able to produce any actuation and was just tasked with recording to its local data store pertinent information about its state at any given time. This test was run to check that there was a consistent flow of data from the simulation to the agent and that all of the basic simulation parameters were correctly applied such as time step and simulation to agent communication protocols.

In the second test case the setup is the same as the previous case but the agent was now tasked with orbital correction manoeuvres to maintain it on its desired orbit. This was the only addition to this test case and this allowed for the testing of the external controller used by the agent, in this case a closed loop proportional, integrator derivative controller written in Java.

Three examples of code used in the test cases are shown in appendix B. The code is taken from the test cases where the satellites are in an icosahedron formation.

The next logical step is to increase the number of agents in the simulation, as such another spacecraft as added and placed in an offset orbit from the reference orbit, the agent controlling the spacecraft has access to an instance of the same orbital maintenance PID controller as in the previous test.

This test was also successfully completed. These first tests prove that the simulator can handle more than one agent operating within a shared environment while using identical controllers.

4.7.8.1 Numerical integration

ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver in computing $y(t_n)$, it needs only the solution at the immediately preceding time point, $y(t_n - 1)$. In general, ode45 is the best function to apply as a "first try" for most problems [117].

ode113 is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than ode45 at stringent tolerances and when the ODE file function is particularly expensive to evaluate. ode113 is a multistep solver - it normally needs the solutions at several preceding time points to compute the current solution [118].

A related test was also developed to show that the simulation model was not limited to pure Matlab and was generally software independent. In this test the simulator and controller were written in Matlab's Simulink package (See appendix C). The Simulink package is frequently used to design controllers due to its abundance of advanced built in controller 'blocks' and features. The equations of motion were implemented using blocks and feedback loops and a PID controller added to control the spacecraft. The benefit of using Simulink is apparent because it allows the PID controller block to 'self-tune' depending on the users requirements. This facility was utilised and the controller tested in the simulation.

A number of other position hold type scenarios were developed. The first was with 4 spacecraft in a square formation, equidistant from the reference orbit again using the basic Matlab PID controller.

A test was then developed to try to push the test suite. In this case 12 satellites were placed at the vertices of an icosahedron and required to maintain their positions relative to the reference orbit for a full orbit of the Lagrangian point. This is a more interesting test as instead of being limited to a single plane as in the previous square formation the formation of spacecraft spans all 3 axes and allows the user to observe the changing forces and required corrections over a significant number of spacecraft.

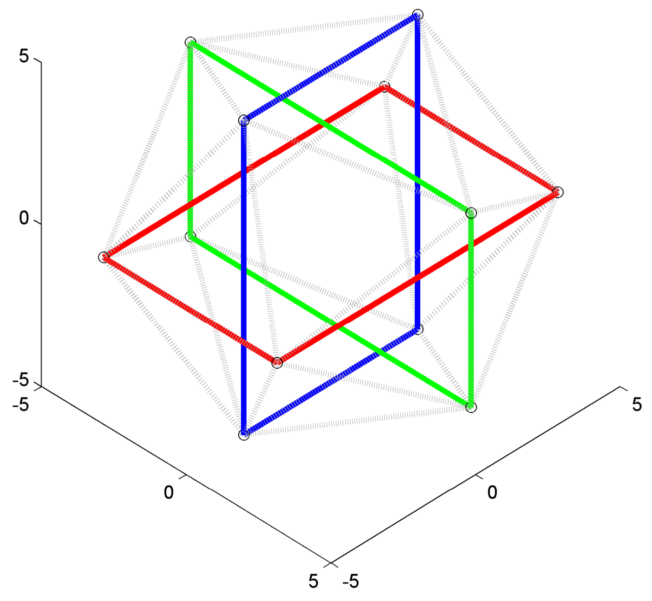


Figure 4-28 Icosahedron structure, can be thought of 3 sets of 4 agents, located on the x, y and z axis in CRTBP frame

x	y	z
0	-1	-1.6180
0	-1	1.6180
0	1	-1.6180
0	1	1.6180
-1	-1.6180	0
-1	1.6180	0
1	-1.6180	0
1	1.6180	0
-1.6180	0	-1
-1.6180	0	1
1.6180	0	-1
1.6180	0	1

Table 4-2 Positions of points on an icosahedron

A number of scenarios were developed utilising the icosahedron structure (Figure 4-28 and Table 4-2).

The first started with the 12 satellites at the vertices of the icosahedron and had as a target an icosahedron that had been deformed in the x axis as shown in .

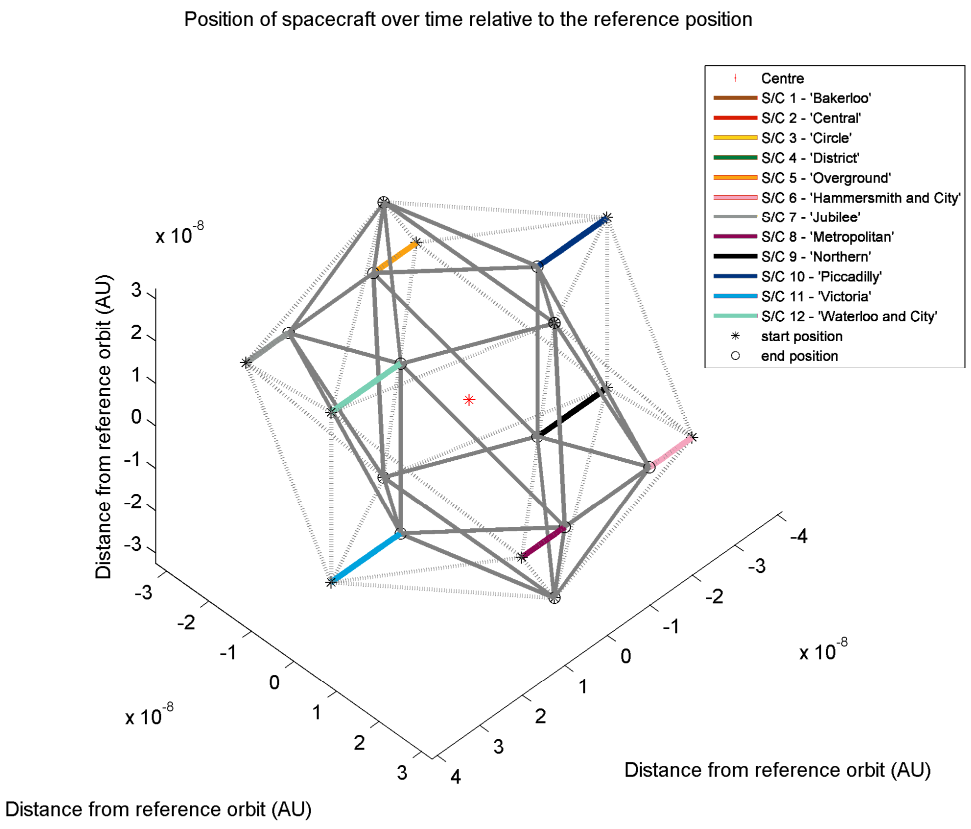


Figure 4-29 Icosahedron formation deformed in x axis

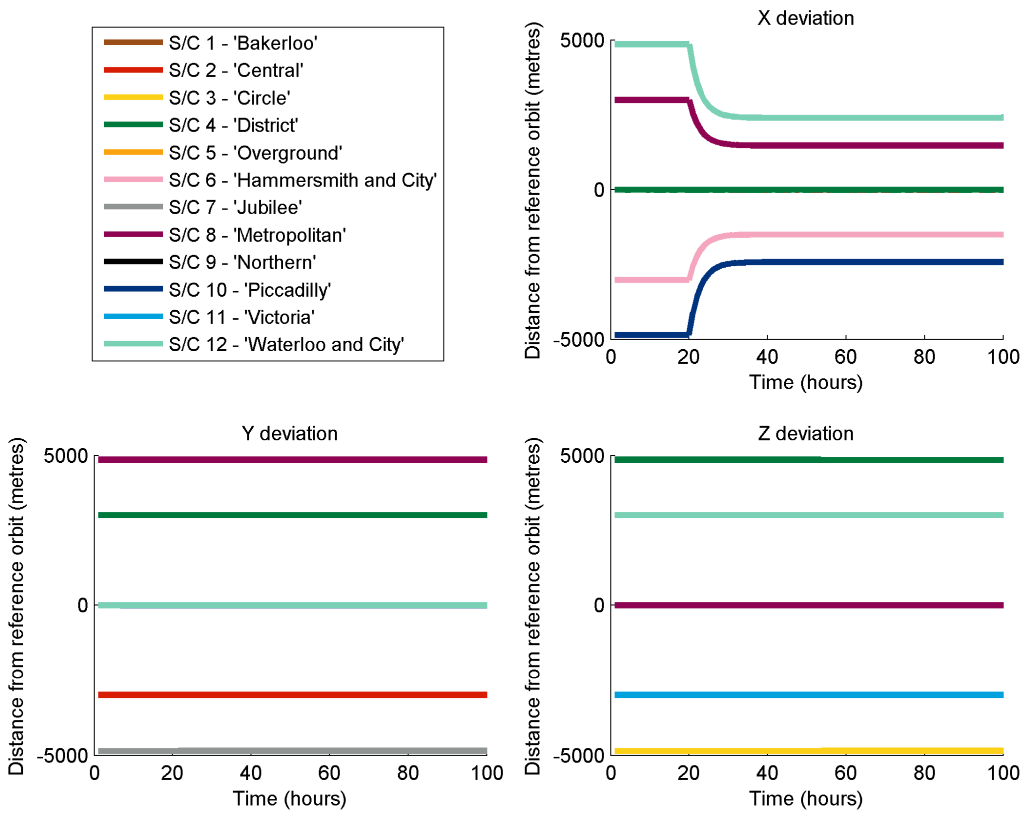


Figure 4-30 The track of the individual satellite orbits in the CRTBP frame and each agents deviation from the reference halo orbit in the icosahedron reduction in x scenario

	Initial Positions (km)			Final Positions (km)		
Spacecraft	X	Y	Z	X	Y	Z
1	0.0000	-3.0000	-4.8541	0.0000	-2.9996	-4.8533
2	0.0000	-3.0000	4.8541	-0.0001	-2.9996	4.8533
3	0.0000	3.0000	-4.8541	0.0001	2.9996	-4.8533
4	0.0000	3.0000	4.8541	0.0000	2.9996	4.8533
5	-3.0000	-4.8541	0.0000	-1.4923	-4.8561	0.0000
6	-3.0000	4.8541	0.0000	-1.4923	4.8508	0.0000
7	3.0000	-4.8541	0.0000	1.4923	-4.8508	0.0000
8	3.0000	4.8541	0.0000	1.4923	4.8561	0.0000
9	-4.8541	0.0000	-3.0000	-2.4146	-0.0042	-2.9995
10	-4.8541	0.0000	3.0000	-2.4146	-0.0042	2.9995
11	4.8541	0.0000	-3.0000	2.4146	0.0042	-2.9995
12	4.8541	0.0000	3.0000	2.4146	0.0042	2.9995

Table 4-3 Table of values at beginning and end of icosahedron deformation in x axis.

The next scenario again saw 12 satellites on the vertices of an icosahedron but its target icosahedron had been reduced in size in all 3 dimensions.

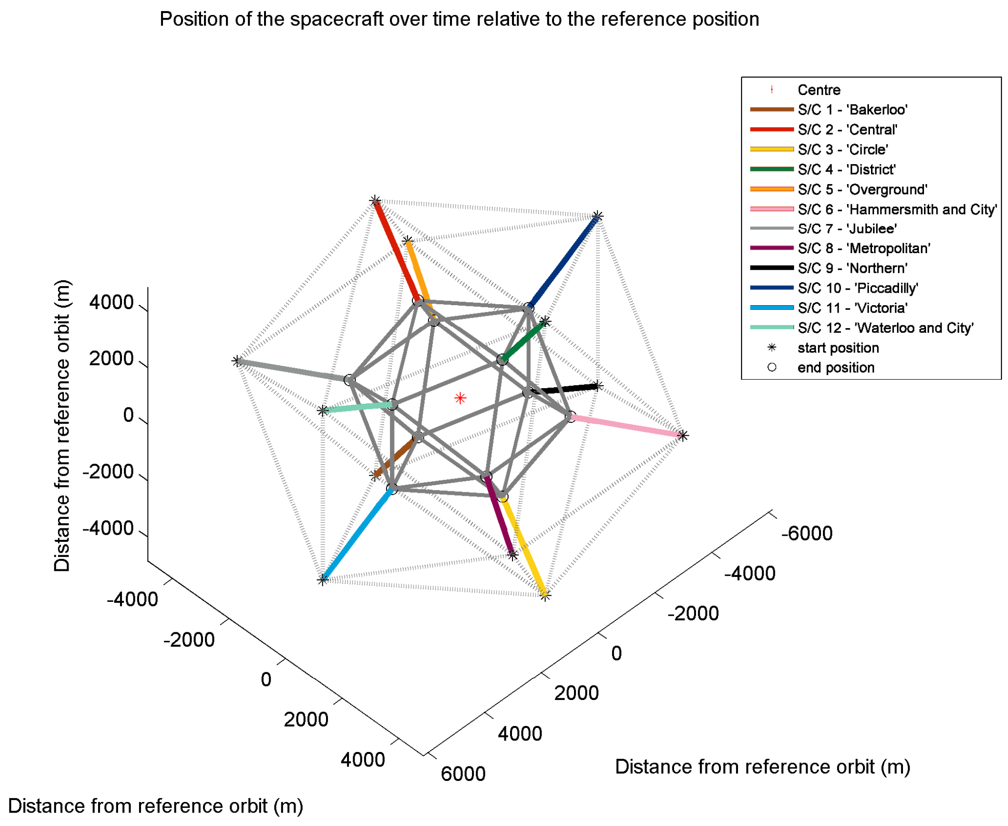


Figure 4-31 Icosahedron formation deformed in all three axes

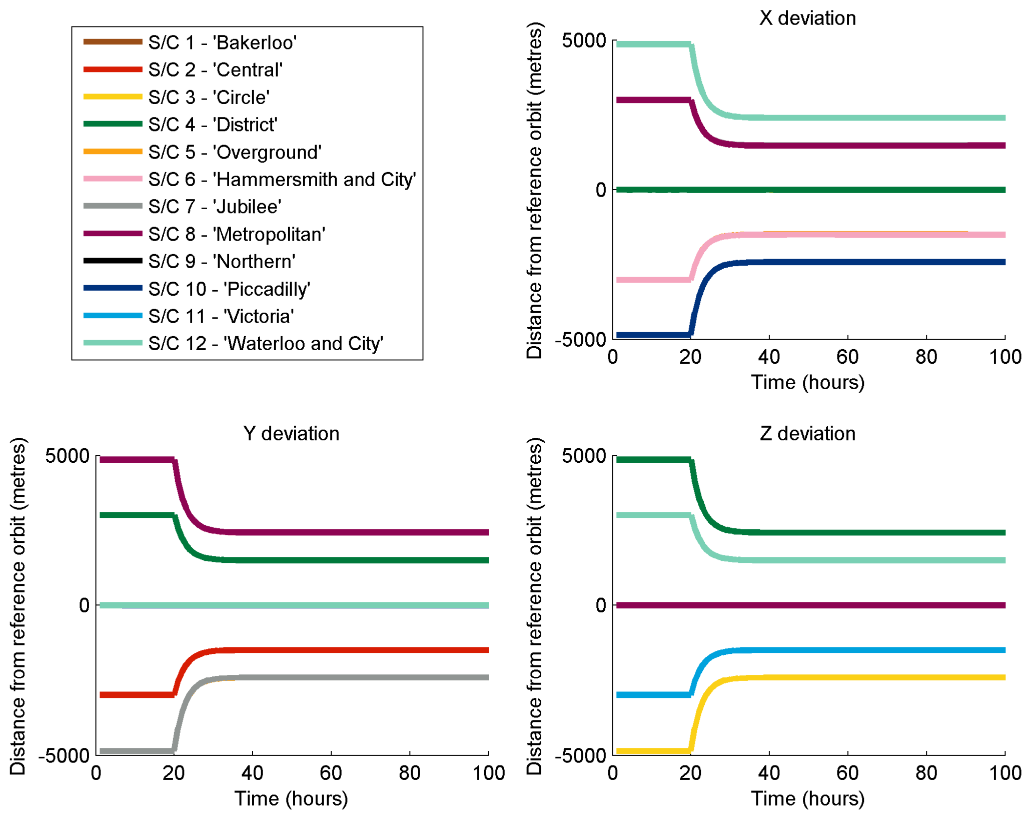


Figure 4-32 The track of the individual satellite orbits in the CRTBP frame and each agents deviation from the reference halo orbit in the icosahedron reduction in 3 dimensions scenario

	Initial Positions (km)			Final Positions (km)		
Spacecraft	X	Y	Z	X	Y	Z
1	0.0000	-3.0000	-4.8541	0.0026	-1.4914	-2.4130
2	0.0000	-3.0000	4.8541	0.0026	-1.4914	2.4130
3	0.0000	3.0000	-4.8541	-0.0026	1.4914	-2.4130
4	0.0000	3.0000	4.8541	-0.0026	1.4914	2.4130
5	-3.0000	-4.8541	0.0000	-1.4881	-2.4157	0.0000
6	-3.0000	4.8541	0.0000	-1.4965	2.4105	0.0000
7	3.0000	-4.8541	0.0000	1.4965	-2.4105	0.0000
8	3.0000	4.8541	0.0000	1.4881	2.4157	0.0000
9	-4.8541	0.0000	-3.0000	-2.4146	-0.0042	-1.4913
10	-4.8541	0.0000	3.0000	-2.4146	-0.0042	1.4914
11	4.8541	0.0000	-3.0000	2.4146	0.0042	-1.4914
12	4.8541	0.0000	3.0000	2.4146	0.0042	1.4913

Table 4-4 Table of values at beginning and end of icosahedron deformation in all three axes.

In the next scenario the same starting point was used but the target formation was a ring aligned to the x-axis.

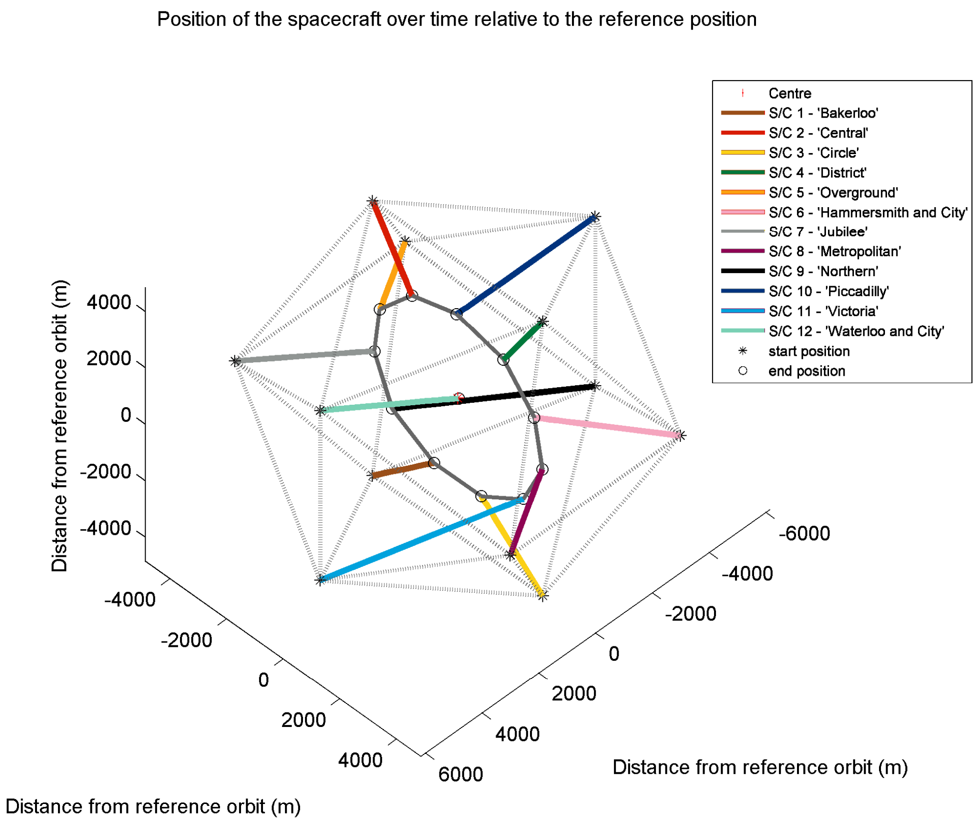


Figure 4-33 Icosahedron formation changing to ring formation

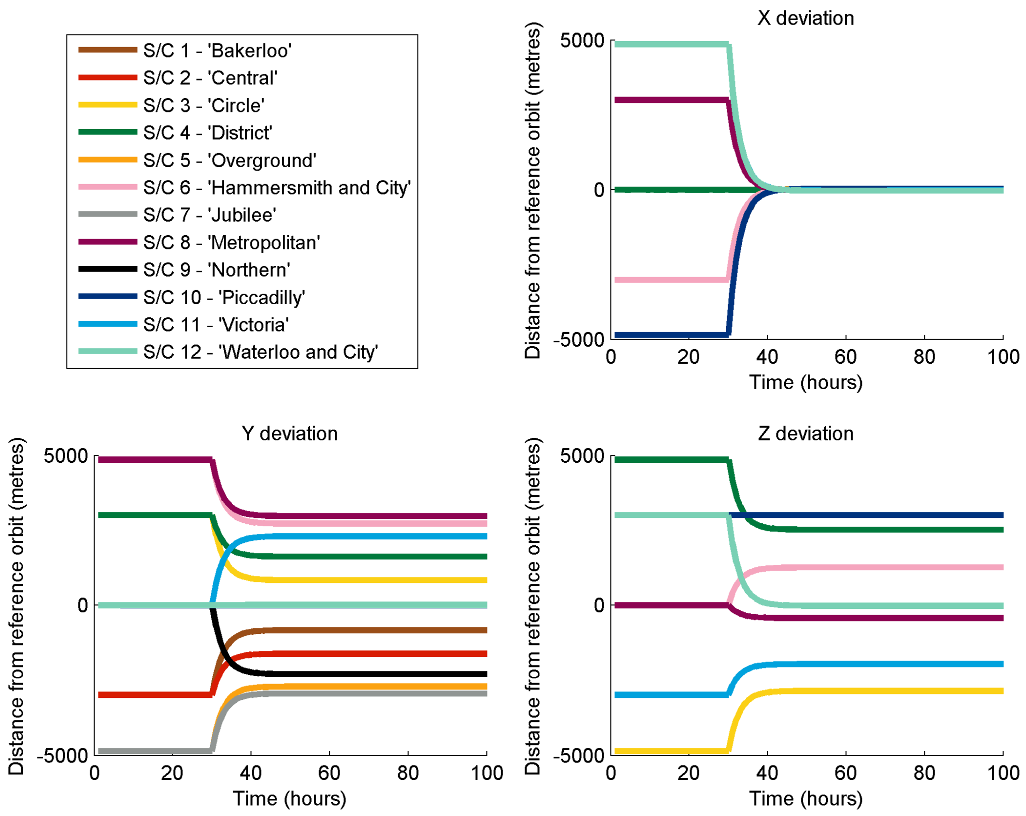


Figure 4-34 The track of the individual satellite orbits in the CRTBP frame and each agents deviation from the reference halo orbit in the icosahedron to ring scenario

	Initial Positions (km)			Final Positions (km)		
Spacecraft	X	Y	Z	X	Y	Z
1	0.0000	-3.0000	-4.8541	0.0039	-0.8327	-2.8667
2	0.0000	-3.0000	4.8541	0.0024	-1.6138	2.5100
3	0.0000	3.0000	-4.8541	-0.0038	0.8327	-2.8667
4	0.0000	3.0000	4.8541	-0.0025	1.6138	2.5100
5	-3.0000	-4.8541	0.0000	0.0205	-2.7217	1.2532
6	-3.0000	4.8541	0.0000	0.0129	2.7110	1.2532
7	3.0000	-4.8541	0.0000	-0.0134	-2.9529	-0.4293
8	3.0000	4.8541	0.0000	-0.0200	2.9636	-0.4293
9	-4.8541	0.0000	-3.0000	0.0230	-2.2885	-1.9583
10	-4.8541	0.0000	3.0000	0.0270	-0.0086	2.9995
11	4.8541	0.0000	-3.0000	-0.0230	2.2885	-1.9583
12	4.8541	0.0000	3.0000	-0.0271	0.0086	-0.0172

Table 4-5 Table of values at beginning and end of icosahedron changing to ring formation.

A key aspect of the above results is the fact that the agents were only ever modifying their position relative to the reference orbit. In reality the difficulty of formation flying and multi spacecraft missions comes from the requirement for interaction between the spacecraft themselves.

The logical next step was to test the addition of intelligence models in the agents, giving them decision making abilities in the context of the scenario, and observe the results. As such the next set of test scenarios reduced the number of satellites but now tested the collision avoidance abilities of the multi-agent control system. The first collision avoidance test consisted of 2 satellites either side of the reference orbit. After a certain period of time one of the craft moved towards the reference orbit position which was inside the radius of the second crafts collision avoidance mechanism, making it move away from its initial orbit. It should be noted in this first test only the second spacecraft had a collision avoidance mechanism and the first satellite could be thought of as non-operational in that it was not responding to any communications between the agents.

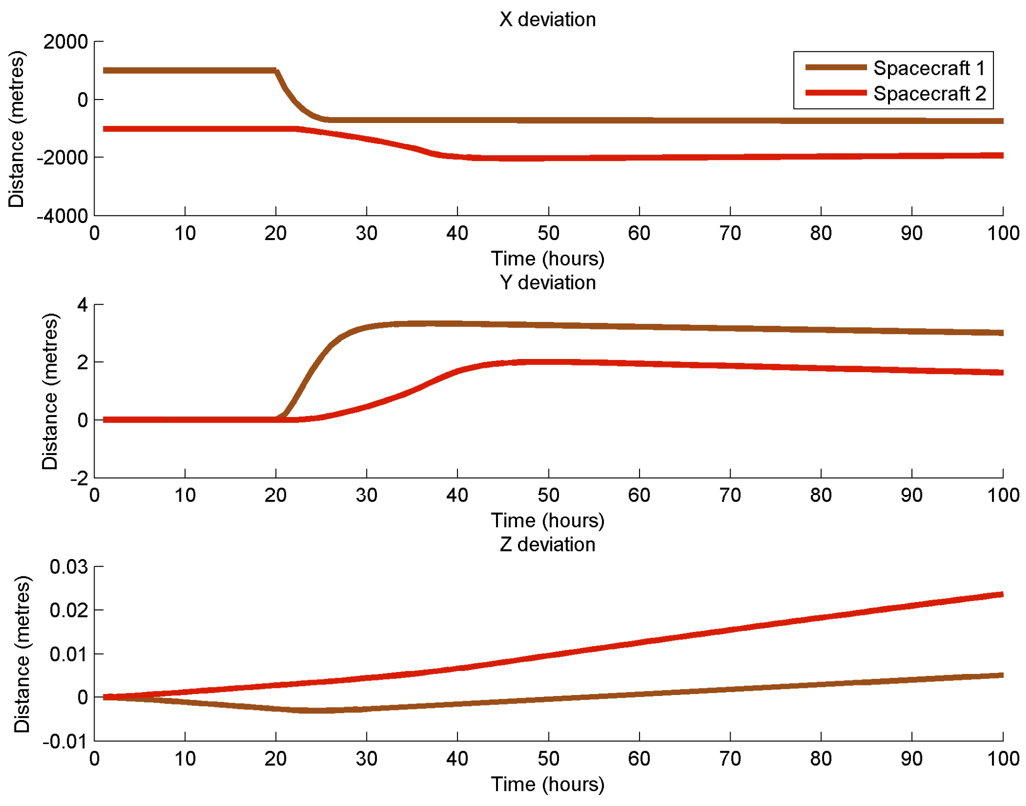


Figure 4-35 Collision avoidance test 1

In the second of the collision avoidance tests the same initial set up was used but both satellites had the CAM activated so they both reacted equally to each other's presence.

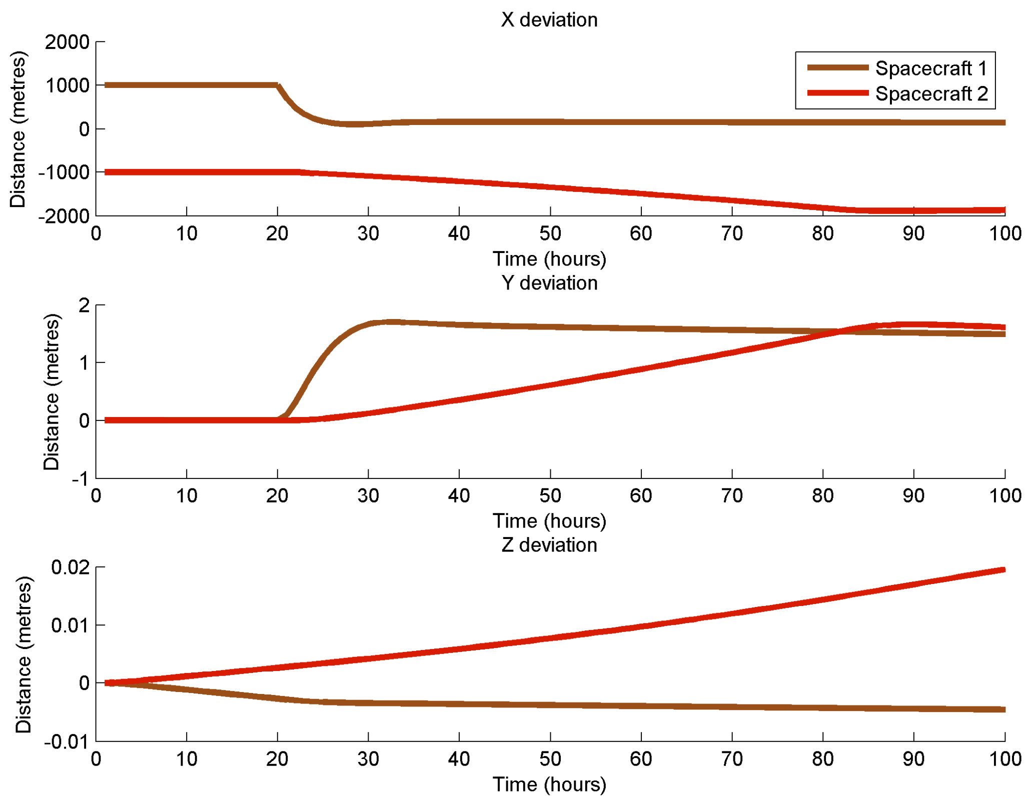


Figure 4-36 Collision avoidance test 2

The next series of collision avoidance tests were with the 12 satellites flying in the icosahedron formation. The first of these involved the 3 adjacent satellites on the icosahedron simultaneously swapping places in a circular fashion: i.e. the first goes to the position of the third, the second goes to the position of the first and the third goes to the position of the second. This meant that there was no direct conflict so the CAM should not have to be utilised.

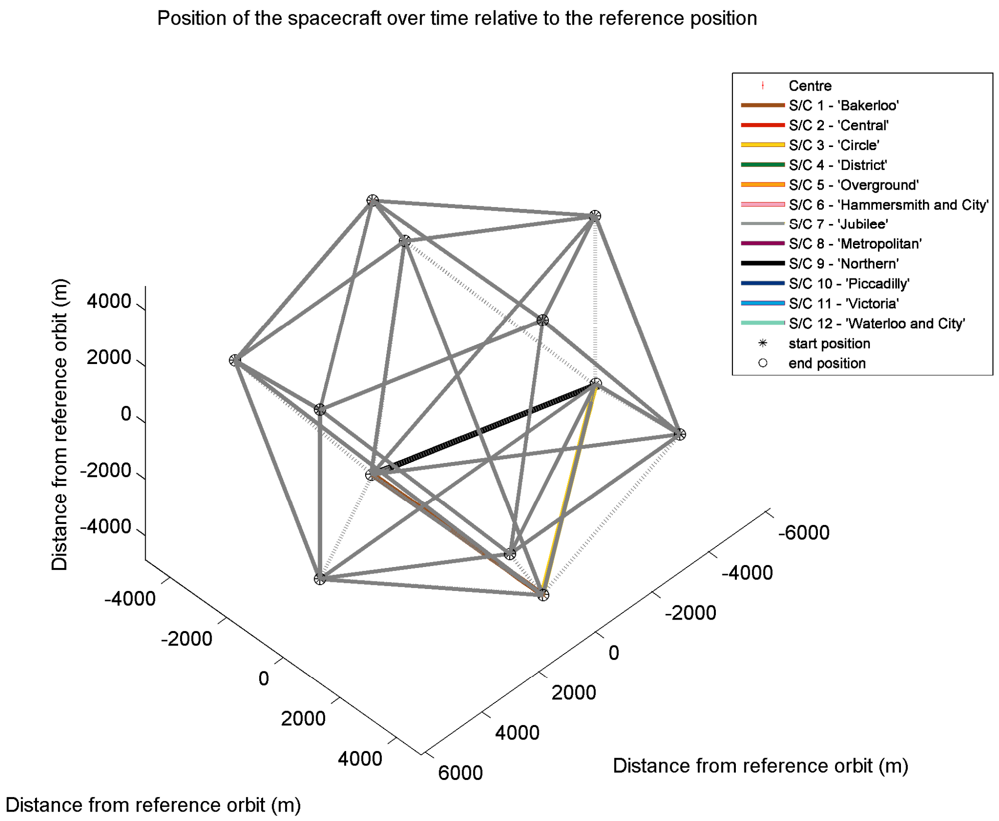


Figure 4-37 Icosahedron 3 way postion swap

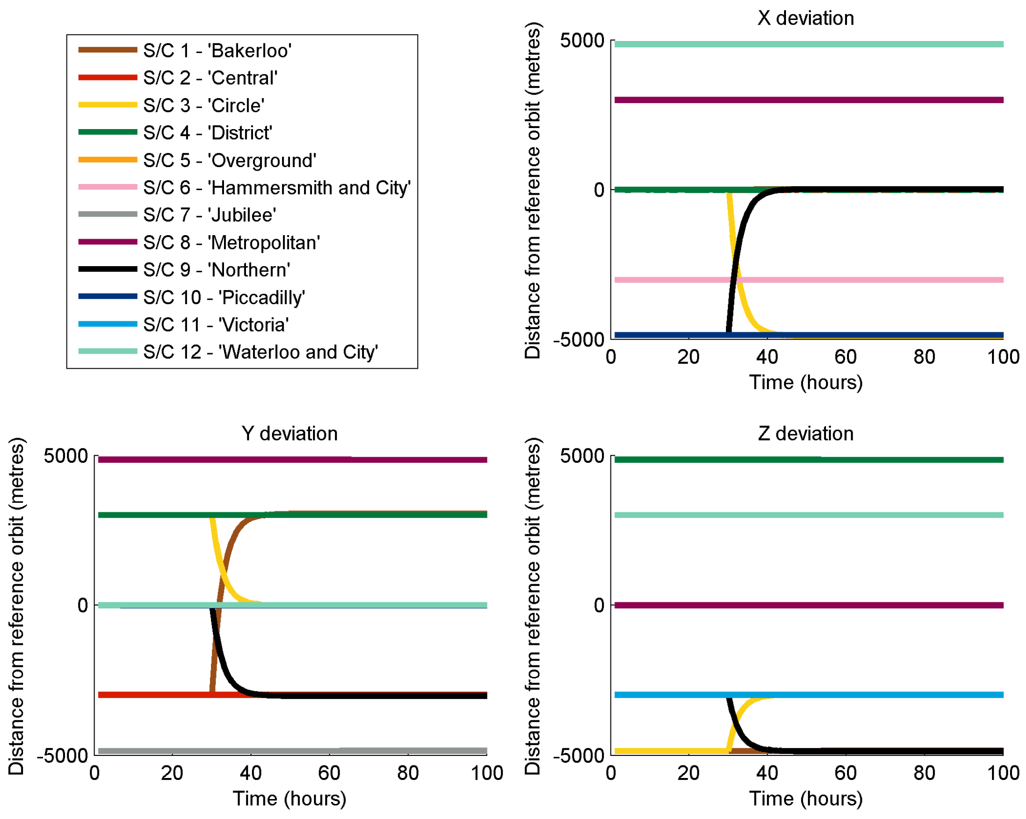


Figure 4-38 The track of the individual satellite orbits in the CRTBP frame and each agents deviation from the reference halo orbit in the icosahedron 3 way position swap scenario

	Initial Positions (km)			Final Positions (km)		
Spacecraft	X	Y	Z	X	Y	Z
1	0.0000	-3.0000	-4.8541	0.0107	3.0340	-4.8533
2	0.0000	-3.0000	4.8541	-0.0001	-2.9996	4.8533
3	0.0000	3.0000	-4.8541	-4.8882	-0.0086	-2.9888
4	0.0000	3.0000	4.8541	0.0000	2.9996	4.8533
5	-3.0000	-4.8541	0.0000	-3.0011	-4.8534	0.0000
6	-3.0000	4.8541	0.0000	-3.0011	4.8534	0.0000
7	3.0000	-4.8541	0.0000	3.0011	-4.8534	0.0000
8	3.0000	4.8541	0.0000	3.0011	4.8534	0.0000
9	-4.8541	0.0000	-3.0000	0.0218	-3.0254	-4.8639
10	-4.8541	0.0000	3.0000	-4.8559	0.0000	2.9996
11	4.8541	0.0000	-3.0000	4.8559	0.0000	-2.9996
12	4.8541	0.0000	3.0000	4.8558	0.0000	2.9994

Table 4-6 Table of values at beginning and end of a 3 way position swap

The next icosahedron test was two agents at opposite sides of the sphere encompassed by the shape swapping places. In this case the two satellites could collide at the centre of the formation if no CAM is deployed so this is test of the CAM in larger formation but with only one possible collision scenario.

The next test was each of the craft trying to swap places with its oppositely positioned counterpart. In this case the PID does not perform well and collisions are highly likely. The simple PID based controllers are effective in simple cases but with more complex agent interactions a more sophisticated avoidance approach may be needed. A more complicated collision avoidance test allowed us to further test the ability of the test suite to utilise external libraries.

4.7.9 ORCA

In this case the external library used was the reciprocal collision avoidance (RVO2) library for collision avoidance among multiple agents. The RVO2 library is based on the idea of reciprocal n-body collision avoidance. A distinction must be made between collision avoidance and motion planning (which assume the environment of the agent is known) and collision detection which is purely the detection of the intersection of geometrical objects. Collision avoidance aims to control agents to avoid obstacles in a complex environment involving other mobile agents.

The RVO2 library is based on the theory of optimal reciprocal collision avoidance (ORCA) [119]. ORCA does not require communication between agents although it does assume perfect sensing. In our multi-agent environment perfect sensing is assumed and is achieved through communication between agents. ORCA also assumes that the agents are fully holonomic (not to be confused with holonic), that is they are free to move in any direction at any time. This constraint is not very onerous in a spacecraft formation flying scenario. ORCA is found to be sufficient for collision avoidance if every other agent in the simulation also implements ORCA. The ORCA algorithm finds a solution in the velocity space that guarantees collision avoidance, if no are found then the safest possible velocity is found.

4.7.9.1 ORCA Basics

The following description of the ORCA algorithm is taken from [119]. For two robots A and B, the velocity obstacle for A induced by B for a given time window is a set of all relative velocities of A with respect to B that will result in a collision between A and B at some moment before the end of the time window. It is formally defined as follows. Let $D(\boldsymbol{p}, r)$ denote an open disc of radius r centred on $\boldsymbol{p}$;

$$D(p,r) = \{q \mid \| q - p \| < r\}, \qquad (4\text{-}27)$$

$$VO_{A|B}^{\tau} = \{\mathbf{v} \mid \exists t \in [0,\tau] :: t\mathbf{v} \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\} \qquad (4\text{-}28)$$

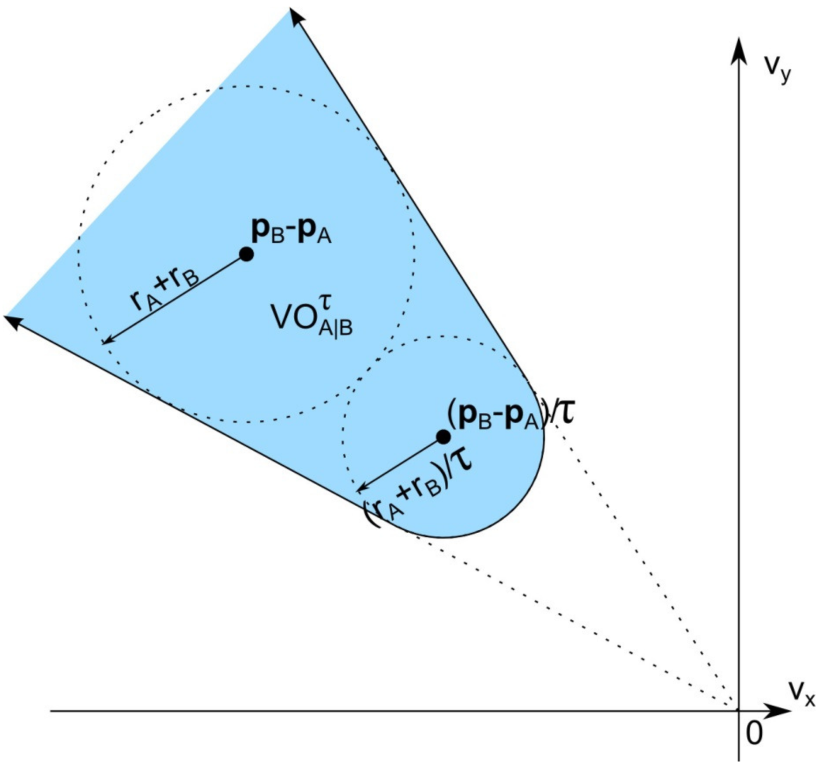


Figure 4-39 Velocity obstacle in the velocity space

The geometric interpretation is shown in Figure 4-39. The definition of the velocity obstacle implies that if $V_A - V_B \in VO_{A|B}^{\tau}$, or $V_B - V_A \in VO_{A|B}^{\tau}$ then A and B will collide at some point before τ if they continue moving at their current velocity. Conversely if $V_A - V_b \notin VO_{A|B}^{\tau}$, robot A and robot B are guaranteed collision free at least for τ time.

Let X $\oplus$ Y denote the Minkowski sum of sets X and Y [120];

$$X \oplus Y = \{x + y \mid x \in X, y \in Y\} \tag{4-29}$$

Then for any set V_B, if $v_B \in V_B$ and $v_A \notin VO^{\tau}_{A|B} \oplus V_B$ then A and B are guaranteed to be collision free at their current velocities for at least τ time. The set of collision avoidance velocities for A given the velocity set of B is:

$$CA^{\tau}_{A|B}(V_B) = \{\mathbf{v} \mid \mathbf{v} \notin VO^{\tau}_{A|B} \oplus V_B\} \tag{4-30}$$

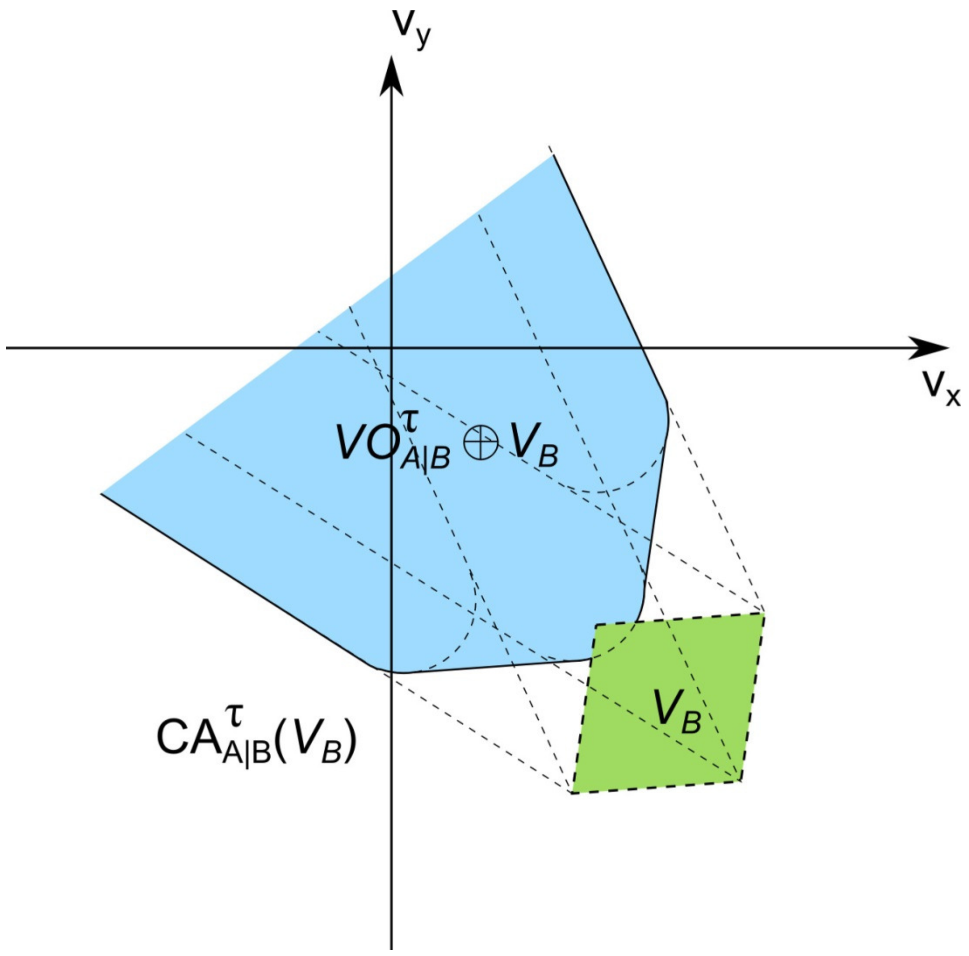


Figure 4-40 Minkowsky sum of robot velocity and a velocity obstacle

The velocities of A and B are reciprocally collision avoiding if:

$$V_A \subseteq CA^{\tau}_{A|B}(V_B) \text{ and } V_B \subseteq CA^{\tau}_{A|B}(V_A) \tag{4-31}$$

The optimal reciprocal collision velocities $ORCA^{\tau}_{A|B}$ and $ORCA^{\tau}_{B|A}$ are designed to maximise the velocities closer to a desired optimisation velocity (V^{opt}_{A} and V^{opt}_{B}) which in general is the robots current velocity. The set is defined as follows for all radii > 0:

$$| ORCA^{\tau}_{A|B} \cap D\left(\mathbf{v}^{opt}_{A}, r\right) |=| ORCA^{\tau}_{B|A} \cap D\left(\mathbf{v}^{opt}_{B}, r\right) | \geq \min(| V_A \cap D\left(\mathbf{v}^{opt}_{A}, r\right) |, | V_B \cap D\left(\mathbf{v}^{opt}_{B}, r\right) |) \qquad (4\text{-}32)$$

In practice the robot acquires the radius, position and optimisation velocity of all of the other robots and computes $ORCA^{\tau}_{A|B}$ with respect to each other robot B. The set of velocities permitted for A with respect to all robots is the intersections of the permitted velocities induced by each other robot. All of this is carried out in real time and was successfully integrated into the multi-agent test suite.

Traditionally the fact that the library is written in C++ for the windows platform would pose a serious problem for the test suite designers but as our multi-agent test suite is built from the ground up to allow heterogeneous modules to communicate with the agents it was a fairly trivial exercise to get it working. This test was run twice with different settings for the collision avoidance algorithm, the first giving the spacecraft very little tolerance for proximity to another spacecraft resulting in many of the craft leaving the formation in order to return at a later time (Figure 4-41). The second test case allowed the agents to have a higher tolerance and the satellites moved towards the opposite corner but performed the required manoeuvres as they approached the centre of the formation (Figure 4-42).

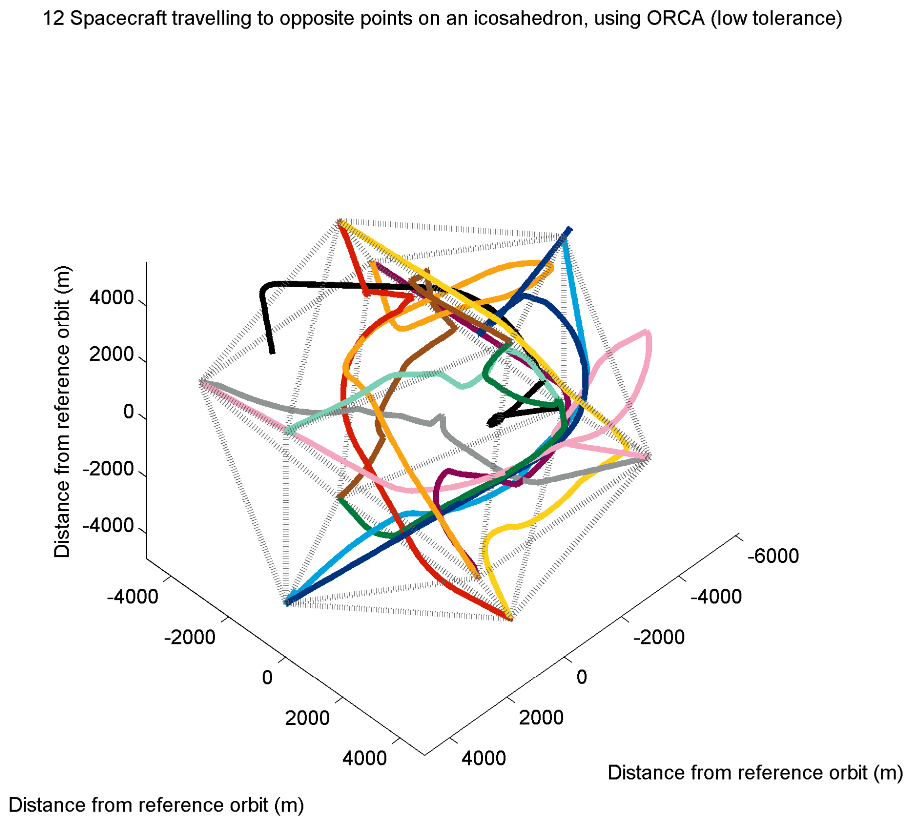


Figure 4-41 RVO collision avoidance with low tolerance for proximity

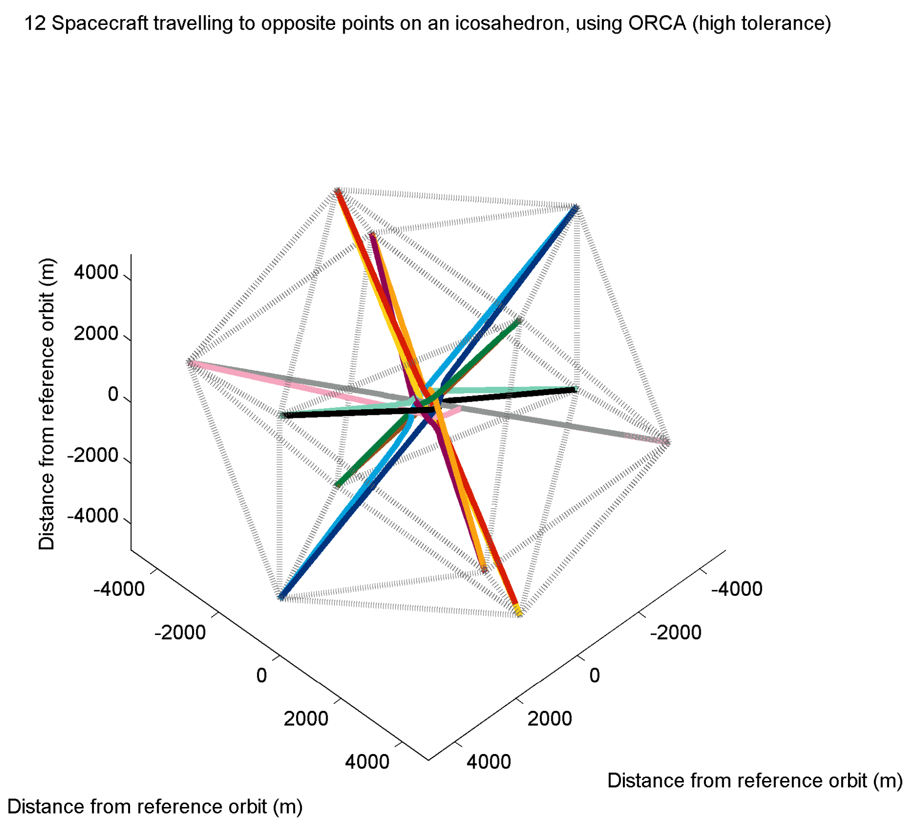


Figure 4-42 RVO collision avoidance with high tolerance for proximity

The system seemed to cope well with using this library for twelve agents so a similar but more computationally taxing scenario was developed to really test the testing suites scalability at the cost of drastically reduced realism.

4.7.10 Scalability

A test was developed to show the scalability of the multi-agent suite. In the final test 812 spacecraft were positioned at points on an imaginary sphere and required to go to the opposite point on the sphere. Similarly to the previous tests this would lead to a large number of collisions in the centre of the sphere. This produced a massive data set (Figure 4-43) and it was particularly informative to single out a subset of agents and track their progress to monitor the success of the collision avoidance algorithm (Figure 4-44).

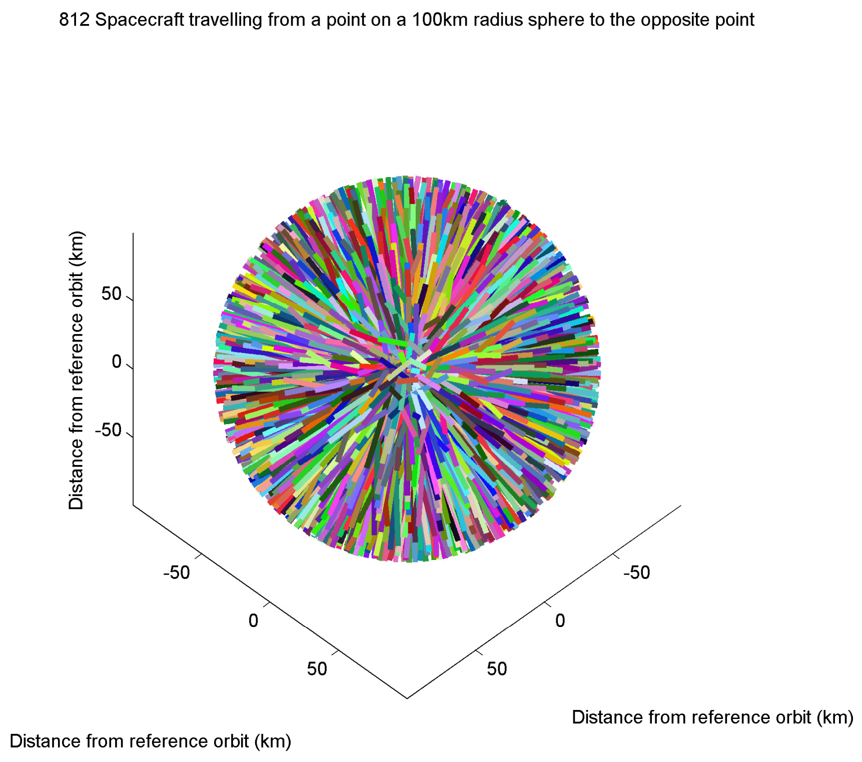


Figure 4-43 812 spacecraft swapping posotions across a sphere

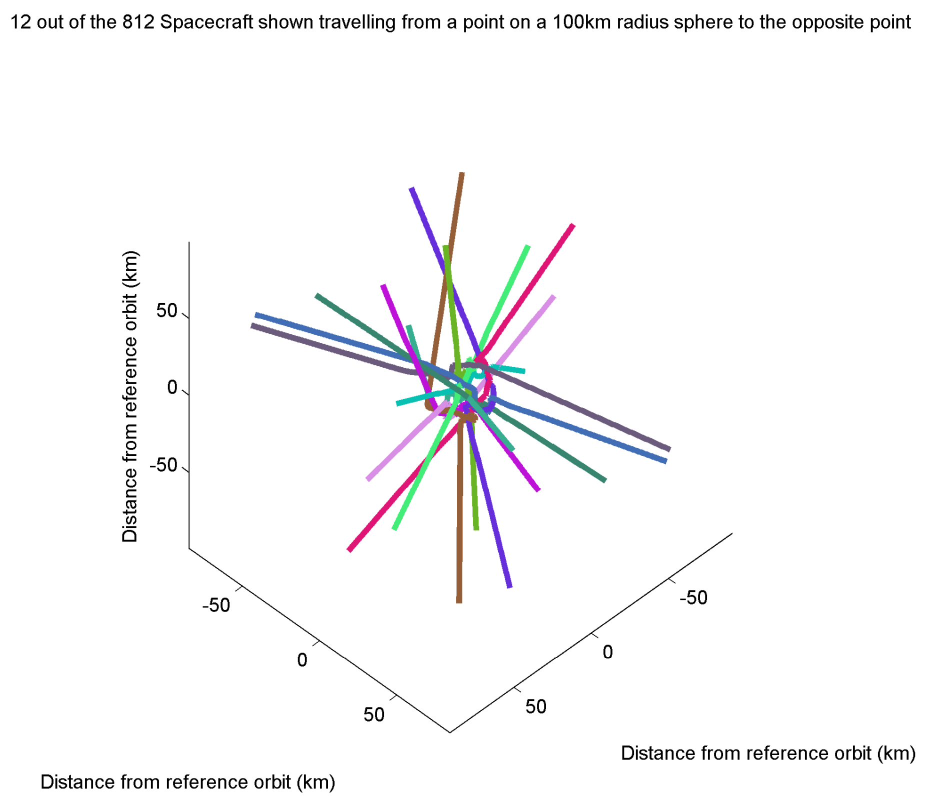


Figure 4-44 The trajectory of 9 of the 812 spacecraft

The use of the ORCA library in this scenario was a success and all of the agents avoided collisions. This ability of the test suite to accommodate novel models and algorithms regardless of platform is one of its key strengths. It also shows that the DARWIN MAS can utilise a variety of different intelligence models in order to carry out its tasks.

4.8 Chapter Summary

This chapter introduced the DARWIN mission and developed a multi-agent system to control it from first principles using the HASA architecture. A simulation suite was developed to try and test the MAS. Three different control methods were implemented, artificial potential fields in a limited two dimensional context and a PID and ORCA controller utilising the full three dimensional dynamics form the CRTBP.

Unfortunately the implementation of the control system in a more traditional way was outside of the scope of this work. As such no direct comparisons can be made between the traditional architectures performance and the multi-agent systems performance. This work does however clearly show that a multi-agent system can be used to control such a formation flying mission and that it is extremely flexible as to how the environment is modelled and how its control laws are implemented.

Chapter 5 Multi-agent System for GMES

5.1 Introduction

In Chapter one I covered what agents are, how they interact as part of larger multi-agent systems and the different variations of these systems. In Chapter three I introduced a new agent architecture based on the idea of holonic or recursive agents specifically designed to meet the demands of space missions.

In this thesis two missions were chosen as candidates for the development of an autonomous agent based control system, specifically the HASA architecture outlined in the Chapter three. The HASA architecture has the flexibility to enable its deployment to missions with different hardware and software systems and varying autonomy requirements.

In this chapter the focus turns to the GMES [121] mission. As outlined in Chapter one, agents can be broadly characterised as software/informational agents or hardware/embedded agents. Software agents have no real physical presence and operate autonomously by manipulating data and information rather than manipulating a physical environment. Hardware agents operate in and act upon a physical environment. An interesting aspect of the GMES mission is that both types of agents are implemented side by side with slightly higher number of software agents.

There are many different scenarios where a more autonomous approach to data management would be of benefit. Data management in this context involves creating, moving and prioritising data. Data must be managed in this way in all space missions and especially in the scientific missions which are the focus of this work. GMES was chosen as part of the work on the ESA contract "Distributed Agents for Autonomy", a collaboration between the University of Glasgow and GMV, SA.

The GMES mission was chosen as a good example of mission where autonomous software agents could greatly assist the mission’s key role of data management but also where autonomous hardware agents could help in the acquisition of the primary scientific data as well.

The GMES mission itself will be described in detail followed by a discussion of what a multi-agent system for GMES could look like, this is based on the work by GMV [14, 15, 18]. The multi-agent system is then defined using the HASA architecture defined in Chapter three. As the basic MAS was developed by GMV a complementary image selection algorithm that runs alongside the MAS to provide relevant data to the end users was developed. The image selection algorithm includes global optimisation methods to aid the user in making an informed choice and is described with the operation of the whole system explored for a number of test cases.

5.2 GMES mission

5.2.1 GMES services overview

GMES (Global Monitoring for Environment and Security) is a European Space Agency led program to design and build information services to deal with the scientific and security data generated by the European Union. GMES will be based on the information gathered from Earth observation satellites as well as data gathered on the ground. GMES will enable greater access to this data as well as preparing, coordinating and analysing data for end users. By utilising the GMES system the European Union will be able to make better informed decisions on our environmental future in the short, medium and long term. The overall aim is to improve the quality of life for European citizens by using both environmental and security data.

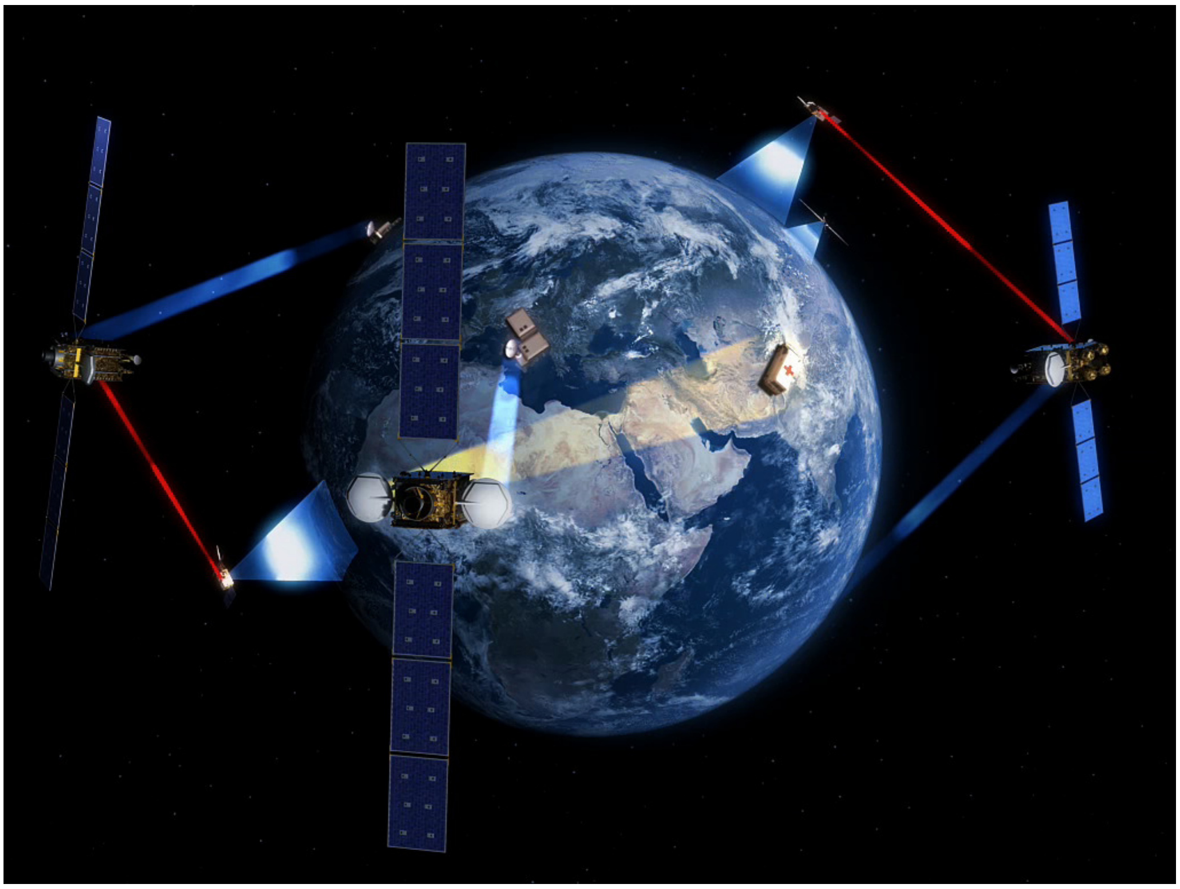


Figure 5-1 The GMES mission will consist of a number of earth observation missions working in concert. Credit: ESA

GMES has been a continuous project for number of years now and is now in its 7th framework program. As such it is hoped that GMES will be operational and able to meet user's requirements in 2013 [121].

It is expected that GMES will be used by both the public and private sectors within the European Union. GMES could help in areas that concern the greater public good and the correct response to natural events such as earthquakes, forest fires and flooding or it could enable European firms to gain a competitive advantage by utilising this observational data. Decision making in this sense is based around the three concepts of anticipating events, intervening in events and controlling events. The GMES system will enable decision makers to acquire data in a reliable manner and make this information available to end users so events can be better anticipated, intervention can be swifter. As a result the end user has more control over a situation. All of the possible clients of GMES, both private and public will have customised data services which will help them to make informed decisions in their own fields and areas of interest.

At a high level GMES observes the Earth's main environmental subsystems, the land, sea and air. A multitude of services will be built on top of this base data structure to service the wide range clients envisaged for the system, services that add value to the data gathered by GMES by fusing it with other data sources and tailoring the final product to the individual customer. In general terms the GMES services can be grouped into three main categories.

- Mapping: Mapping services will include both topographic data and road mapping but also data that can help build a better picture of our environment such as land use, forestry monitoring and mineral and water resources. This mapping service typically involves a highly exhaustive coverage of the Earth's surface and periodic archiving of data.

- Support: Support services for emergency management such as natural disasters but also for civil protection of the population and property. In this service the data must be as up to date as possible if intervention is going to be successful.

- Forecasting: Forecasting services can be applied to fields such as fisheries, air quality or crop yields. This service would have to provide a systematic set of data from a selected area over a long period of time to support the development of the models of these systems that are required for accurate forecasting.

It is hoped that the wide scope and fast response of the proposed GMES system will allow for more efficient use of human resources and infrastructure. Some of the proposed services that will be included in GMES are monitoring of:

- Coastal water quality
- Land use
- Area and density of forests

- Land use globally to predict food shortages and to increase the European Union's food security
- Risks of flooding and fire,
- The oceans to enable early detection of oil leakages and to aid clean-up
- Soil movements to predict landslides and track erosion and top soil migration
- The levels of ice in glaciers, icebergs and the ice in the sea, lakes and rivers

Three GMES services have been put forward for early development and deployment, they were chosen based on their maturity, the level of demand for the service (both long and short term) and the speed at which the user community would start using the services.

The Emergency Response Core Service (ERCS) will focus on providing rapid mapping and infrastructure assessment services to users. This service is primarily aimed at humanitarian groups and relief agencies and will provide access to geo-spatial databases for the region concerned to get the client up to speed. It will also offer assessment of any events that have taken place or are predicted to take place and any possible impacts arising from them. It will give the client access to real time monitoring tools for the duration of the crisis.

The Land Monitoring Core Service (LMCS) will provide both regular and complete satellite coverage of the European Union and will provide an up to date land cover database covering the entire region. This will include mapping at the European scale to enable the implementation, review and monitoring of different EU policies such as water directives, biodiversity strategies, common agricultural polices as well as adherence to international treaties such as the Kyoto protocol. The service will also provide mapping at a local scale for city planning, construction, noise monitoring etc. as well as looking for hotspots where there are rapid changes in construction, land use or agriculture.

The Marine Core Service (MCS) will meet the requirements for oceanic environment data required for national monitoring, European directives and international treaties. The MCS will hopefully lead to better management and exploitation of oceanic resources, improvement in safety and efficiency of maritime transport with the monitoring of shipping lanes and naval operations, anticipating and mitigating against possible man-made environmental disasters, enhancements in basic marine research including both the climate and the ecosystem for seasonal climate prediction and implementation of specific policies regarding coastal management.

5.3 Current Status of GMES

The overall mission architecture consists of four major elements:

The Space Component comprises both space and ground elements. The GMES space component will consist of both existing missions provided by ESA, ESA partners and Eumetsat as well as new missions which are known as the five sentinel satellites.

The in-situ component comprises the development and operation of the ground based and airborne data gathering networks.

The Data Integration component comprises the data assimilation infrastructure which is charged with data fusion tasks, processing the data in a standardised manner and all the functions associated with cataloguing and archiving the GMES data.

The service segment provides services such as data procurement for clients, information generation and delivery as well as services to monitor compliance with requirements and to be able to meet the necessary quality assurance targets.

GMES can be broadly thought of as comprising an infrastructure component and a services component. These two broad components will be expanded on in the next section.

5.3.1 GMES Space infrastructure

The GMES infrastructure will comprise of a large (approximately twenty) number of heterogeneous satellites designed and operated by a wide range of entities. The satellites within the GMES system will also change over time as older platforms are retired and new platforms are launched. The two main groups of Earth observation satellites are polar and geostationary satellites. These two types of Earth observation platforms are seen as generally complimentary due to their individual advantages and disadvantages. Geostationary satellites offer nearly full world coverage expect for the polar regions and offer data of a region in a continuous way by sampling at a high rate (in the order of minutes). The high orbit needed for geostationary satellite however means that the resolution of the produced images may not be high enough. For instance the geostationary Meteosat satellite has an imaging resolution of between 2km and 5km depending on the sensor and latitude of the target [122] . Polar satellites orbit at much lower altitudes and this enables them to achieve much higher resolutions of between 30m and 150m in the case of the Envisat advanced synthetic aperture RADAR (ASAR) [123].

A polar satellite’s path will also cover the entire surface of the Earth, including the polar regions, but this could take at least a few tens of orbits. The disadvantages of polar satellites are that because of their rapid progression over the surface of the Earth they cannot provide continuous data for one particular area of the Earth’s surface.

5.3.2 GMES satellites

A number of new satellites have been suggested to enhance the capability of GMES as a whole, these are the sentinel satellites. The proposed nature of the sentinel satellites are listed below [124].

Sentinel-1 - The Sentinel-1 spacecraft will operate in a Sun synchronous dawn/dusk orbit. It will carry a synthetic aperture radar (SAR) which will operate in the C-band and will offer wide swath medium resolution coverage. It is proposed that Sentinel-1 will also be able to support emergency observation requests.

Sentinel-2 - The Sentinel-2 space craft will have a Sun synchronous orbit which will pass the equator between 10-14h in ascending or descending passes. It will observe the Earth in multiple spectral bands within the solar part of the spectrum. It will offer wide swath on the order of 100kms and a medium spatial resolution. Sentinel 2 will also be able to support emergency requests.

Sentinel-3 - Sentinel 3 will operate in a sun synchronous orbit and provide optical data for imaging sea colour, sea temperature and global land monitoring at a coarse resolution. No support for emergency requests is envisaged.

Sentinel-4 /5 - Sentinel 4 will be in a geostationary orbit and will be equipped with a payload for monitoring the composition and state of the atmosphere. Requirements for Sentinel-4 have still to be confirmed but it likely that it will have sensor operating in the ultraviolet or the visible range of the spectrum. The composition of the sentinel 5 has still not been decided but it will operate in low earth orbit. [125]

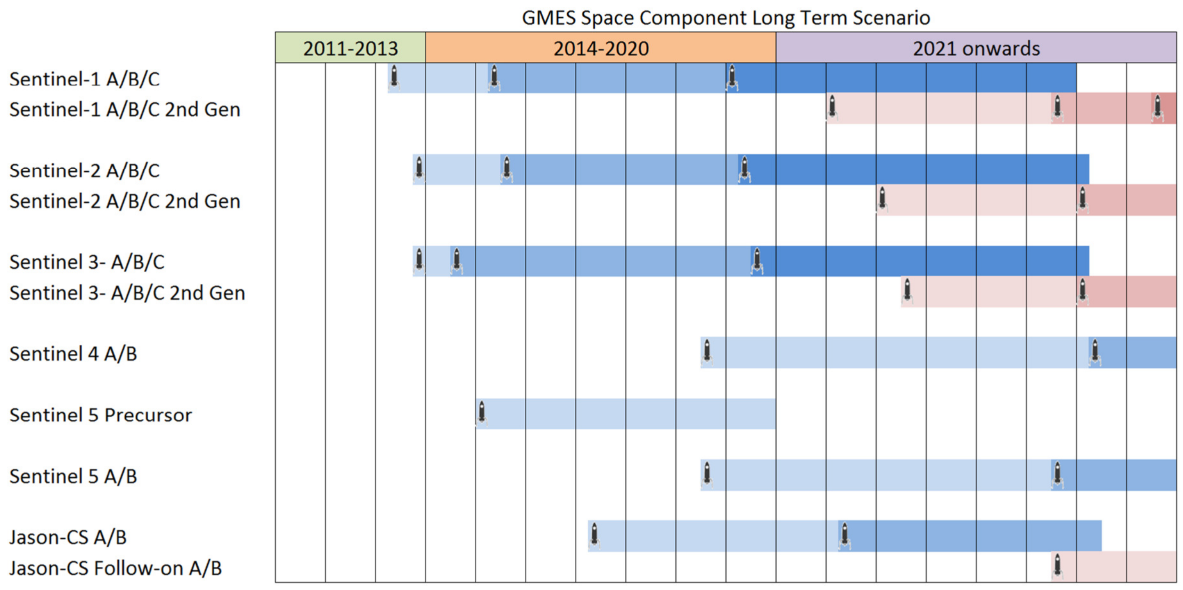


Figure 5-2 GMES Space components timeline

5.3.3 Sensors

There are two broad groups of sensors used in planetary observation satellites, radar sensors and optical sensors.

Radar sensors transmit energy towards the surface of the Earth and record the reflections. Radar operates both day and night and has a high tolerance to atmospheric conditions such as cloud cover or dust storms. There are three types of radar sensor. Synthetic aperture radar sensors allow for very high resolution imaging. Using techniques such as interferometry (the phase change of the reflection) and polarimetry (the polarisation change of the reflection) very accurate images can be acquired. SAR sensors are widely used for object recognition and detecting variations on the ground. Radar altimeters measure the height of the surface of the Earth or sea and are used for land and ocean topography. Wind scatterometers utilise radar to measure wind speed and direction. Radars are extremely useful for Earth observation because of their near independence to atmospheric conditions as well as their day and night availability however they are functionally limited to detect only shapes and surface variations.

Optical sensors utilise receptors that are sensitive to certain frequencies of the electromagnetic spectrum. Often the sensors are tuned to the visible light spectrum but many worthwhile Earth observation can be obtained by using frequency ranges in the ultraviolet or infra-red bands, for instance monitoring sea temperature using thermal infrared and monitoring the atmospheric ozone using the ultraviolet band [126]. Different spectral bands can also be combined on the spacecraft to provide a range of data applicable to the spacecraft's mission. A hyper-spectral sensor is a sensor that senses many spectral bands simultaneously and uses the reflections or transmissions received to identify elements by their spectrum, not their shape.

Optical sensors offer the possibility for very high resolution images and a wide variety of products can be created by combining the data from a number of different bands. Optical sensors are however extremely weather dependant and any atmospheric phenomena such as clouds, rain or dust can severely alter or entirely preclude the ability to obtain good results. Optical sensors must also have daylight to operate as they do not transmit any light and are of limited use at night.

5.3.4 GMES products and Image Processing

Raw data from the satellites must be processed before it meets the requirements of the users. Typically the data is passed through a number of different levels, each extracting more information before it is ready for the user. For instance, in the determination of land cover the data may go through the following stages [127] :

- Geometric Corrections
- Radiometric Corrections
- Geographical/Algorithmical Scale
- Temporal Scale
- Change Detection

5.3.5 GMES Service Provision

The services provided by GMES can be thought of as either core services or downstream services. Core services are the collection of services providing raw data such as sea temperature data whereas downstream services reflect some added value on the information such as tracking a feature over time. There are a number of different operational modes in GMES.

In the routine imaging operational mode an imaging schedule is produced well in advance by the GMES management personnel. This plan is then checked on the ground and if it meets the quality requirements uploaded to the spacecraft roughly every two weeks. The images are received through the x-band antennas of the ground stations covering the spacecraft's orbit and are then sent to the central control where they are available for download by users roughly 24 hours after they were received on the ground.

The emergency operational mode allows for emergency requests to take precedence over the routine imaging operations. An emergency request would contain a list of images that must be obtained in the shortest possible time. The emergency request image acquisition plan is uploaded outwith the usual routine image schedule. Images are only acquired once and downlinked to the ground as soon as the spacecraft becomes visible to a ground station or an inter-satellite link becomes available. The target is to have the emergency request uploaded to the appropriate satellite within 12 hours of receipt.

Most providers of Earth observation data subscribe to the "charter on cooperation to achieve the coordinated use of space facilities in the event of natural or technological disaster" [128]. The charter outlines the provider's duty to deliver timely data to aid those afflicted by a natural or man-made disaster.

In the near real time (NRT) operational mode a plan for both image acquisition and image downlink are uploaded giving priority to both. The image once downlinked to the ground is given high priority of processing, thus allowing acquisition to delivery times in the region of 3 hours.

In the direct downlink operational mode the image data is downlinked as it is being acquired. This operational mode offers the lowest latency possible but with little or no processing of the data. It will also only be available when a ground station is in sight of the satellite for low Earth orbiters (LEO).

The price of Earth observation data products is largely governed by the operational level, with library imagery being the cheapest to acquire and direct downlink the most expensive. For example the cost of a particular Système Pour l'Observation de la Terre (SPOT) image is increased by €3900 if it is required in the next 24 hours [129]

5.3.6 Earth Observation Service Providers

There are a large number of different service providers of Earth observation (EO) data and no easily accessible central repository for data regarding availability of services as each private company or public body has its own policies and procedures for getting images. As this is the case it is only possible to be confident of knowing where to look for EO data if the client is well versed with this field and knows which catalogues to search. It is therefore desirable to open up the access to EO data to a wider range of users and to allow them access to large amounts of data.

Currently GMES and the GMES service element are designed to try and widen access to EO data but there is also the Earth observation market development (EOMD) programme which aims to promote the use of EO data to new customers and promote partnerships and the Data User Element (DUE) which aims to foster a strong relationship between the EO data providers and the EO data clients.

EO data providers can be classed as either image providers who allow clients to access the raw image data or operations providers who process the images from the image providers into new products. Providers are more frequently allowing users to access their images through web based interfaces but images are also still distributed by email, file transfer protocol (FTP) or physical media.

Two examples of web based catalogues are the EOLI catalogue [130] and the CREPAD [131] catalogue. The EOLI catalogue is an interactive tool for accessing ESA's earth observation data. CREPAD allows access to the EO data collected by the Instituto Nacional de Técnica Aeroespacial in Spain. In other cases the requests are not automated and must be submitted in writing. The fact that the request has to go through another layer of bureaucracy obviously reduces the system’s response time substantially.

Increasingly there have been new initiatives to try to improve the provision and distribution of EO data. This can be achieved through easier access, improved processing time/algorithms or more specialised and targeted services. The fusion of multiple data sources has also enabled for more effective products to be created. Portals such as ESA's Service Support Environment have been deployed to provide a common platform for image and service providers to distribute their current products and promote future products [132]. Some of the common services found in the newer generation of EO services are:

Clipping services are where a desired area is described by the user (typically as a polygon) and then the area described is clipped from a supplied image. In this way the desired image data is kept by the user and any unwanted data masked out. Clipping is frequently used to reduce the amount of data in an image before a time intensive processing step or to highlight information about only a specific area.

Conversion services allow for the conversion of a raster image in to a format that is compatible with a client’s system. This is useful for outreach purposes in the case of conversion to a common format or to allow the image data to be processed using an external provider who only accepts certain file formats.

Services also exist for the re-sizing of images to more convenient resolution for dissemination etc. Images may be re-projected if a user or process requires a non-standard projection.

More complex products include forest monitoring services in which EO data is combined with GIS (Geographic Information Systems) to give a more objective view of forest management. The same process can be applied to land management in general but can include data such as land use, soil moisture, leaf cover etc. which can be useful for studying the effects of global environmental change. Urban land use can be monitored in the same way.

The use of intelligent agents aims to combine and improve the ease of access to this wide range of products and is the focus of this work on GMES.

5.4 Current GMES baseline operations concept

The GMES system architecture consists of the space segment, the flight operations segment, the payload data segment, the GMES service segment, a data access integration layer and interfaces to any external systems.

The GMES service segment acts as the primary interface between the GMES system and the end users. The GMES service segment handles the user requests as its input. These user requests will contain one or more data requests which can either be for products that are already available, in which case the products are given to the user or they can be for products that have not yet been captured in which case the data request must be added to the acquisition plan for the next cycle. Requests may be sent directly to certain spacecraft's ordering systems or instead assigned to the GMES service segment to acquire. The output of the service segment is the data products themselves that have been acquired and processed by GMES and these are made available to users through the service segment or through the sentinel ordering service. The GMES service segment provides users with access to all of the available services which usually consist of either a set of raw images or a set of images with some post processing steps applied to add value.

The Data Integration layer is charged with handling the interface between the GMES system and any external systems. These systems can include, but are not limited to, third party missions and in-situ data sources.

The GMES payload data segment consists of a number of sub units. The multi mission planning unit is at the heart of the payload data segment and the heart of the GMES system itself. The multi-mission planning component gathers information on user requests, ground station availability, spacecraft availability (including flight dynamics information) and generates an acquisition plan in concert with the mission specific planning components in order to meet the GMES global objectives. The planning process for different spacecraft will be heterogeneous as to levels of autonomy or interface and the multi-mission planner must take this into account. The mission planning interface to third party mission will take place through the data access integration layer.

The data processing unit is charged with taking in raw image data and generating the desired products. It is intended that the data processing unit is data driven so that rather than carry out tasks at pre allocated times the data processing takes place as soon as the data for that processing is made available. Emergency and near real-time requests will be given priority for data processing over routine image requests. Processing power will be dynamically allocated to the tasks on a priority basis.

The sentinel ordering unit takes orders for the sentinel craft directly from the users and propagates them through the system as a whole.

For each mission in the GMES system there is a flight operations segment. The typical flight operations segment will consist of a monitoring and control unit, a flight dynamics unit and a ground station unit. The monitoring and control units act as an interface between the GMES and the spacecraft hardware. The flight dynamics unit manages all activities pertaining to the spacecraft's orbit and attitude. As such it will compute the manoeuvres required for orbital maintenance and for achieving mission goals. It will also propagate the spacecraft's orbit and flag any possible future orbital events. The ground station unit is in charge of transmission of all telecommands to the spacecraft and the receipt of telemonitoring data from the spacecraft. Another key task of the ground station unit is the management of ground station availability and the creation of plans to maintain contact with the spacecraft. The ground station unit will also have to interface with existing systems for scheduling of ground stations to ensure there are no clashes with other missions.

To further clarify the structure of the GMES baseline the following example of an emergency data request is outlined. In this scenario the user makes an emergency request. This request will usually consist of a geographical area to be imaged, a specific date and time for the first acquisition, date and time information about any subsequent imaging as users will often want images at set intervals to monitor a changing problem or monitor an evolving situation. The type of image will also be selected (infra-red, synthetic aperture radar image, visible etc.). For the fulfilment of an emergency order it is envisaged that multiple spacecraft will work in concert to fulfil the user's request in the timeliest manner possible. This differs from the usual scheduled approach and will require a significant amount of cooperation between the spacecraft.

Once the emergency request has been gathered it is passed to the multi-mission planning component which will then, in collaboration with the flight operations segments generate a plan to acquire the product that has been requested. Depending on the current status of the GMES system and the nature of the request the plan may involve the acquisition of images at the next possible opportunity or possibly the start of a near real-time feed of data to the end user. There may also be a combination of scheduled, near real-time and real-time tasks depending on the request. It can be envisaged that the initial imaging may be taken in near real time but subsequent follow up imaging can be scheduled as normal. For an emergency request, priority will also be given when it comes to the data processing of the images, be they scheduled or near real-time.

Once the data processing step has been finished, the product generated and made available to the ground segment the user is informed of its completion. The product will then be given to the user as per the service level agreements that are in place for that particular product. Service level agreements are guarantees between the users and the providers of the data to ensure that the user gets the products they subscribe to or pay for.

The service level agreements provide guarantees of service availability from the provider to the customer. The provider also sets limits on the response time and uptime of the service. The client has some recourse if the provider fails to meet any of these limits.

The previous example of a request for imagery and the response of the GMES system was used to illustrate two main points. Firstly any emergency requests are fed through the multi-mission planner as not all spacecraft can respond to emergency requests directly. If the spacecraft that can respond to emergency requests cannot fulfil the entire request then the multi-mission planner updates the plans of the other spacecraft to cover the shortfall. Secondly it is expected that many emergency requests will require imagery for an area over time at a frequency supplied by the client. Given that for example the sentinel spacecraft revisit time is 12 days then there will have to be significant cooperation between spacecraft to offer updated imagery in timely manner. The level of cooperation needed to achieve the desired products, particularly in the case of near real time products, and the broad range of demanding requests that can be made makes the GMES system an ideal candidate for the introduction of a multi-agent system.

5.5 Distributed Agent Approach to the GMES system

The following approach was developed in [15, 51].The baseline GMES system as outlined in the previous section incorporates a high level of automation but a low level of autonomy. It is this kind of scenario where it is envisaged that autonomous distributed agents can improve system performance. When developing a multi-agent control system for a given mission it may be seen as preferable to replace the entirety of the previous control system. One of the benefits of using an agent oriented design is that it can be compatible with existing systems, especially those based on the service oriented architecture. The baseline system for GMES is based on a service-oriented architecture and as such for the sake of achievability of the implementation only certain aspects of the system have been replaced with agent based systems. This approach increases the likelihood of adoption by reducing the amount of work needed to create the agent based systems as they are smaller but also from a political standpoint as a mission like GMES naturally has a wide range of stakeholders and many of the mission will be operated by different entities meaning that cooperation on the scale needed to rewrite the control system for each would be impractical.

As GMES is a long term project working with a wide range of spacecraft both operating and under development, a categorisation of the types of spacecraft is beneficial as the different classes of spacecraft bring with them different capabilities and restrictions. As such the spacecraft envisaged for GMES can be classed in one of three categories. Class one spacecraft such as the sentinel spacecraft have predefined planning that cannot be modified and users can only get information by subscribing to the imagery from that particular satellite. Class two spacecraft have planning systems so that their planning can be modified to adapt to user requirements in a near real time acquisition case where re-planning is required. Class three spacecraft have the next level of autonomy on board in that they can autonomously detect scientific events or events of interest over particular areas and modify their own planning to better acquire the data of interest. The most taxing scenarios envisaged for GMES require near real time data from a number of different spacecraft of different classes. In the next sections two typical cases are described in detail. The first is to provide the best possible data product in a near real-time time frame by taking into account the capabilities of the different spacecraft involved (which may well be of differing classes of autonomy) and the restrictions on the product required. In this scenario an agent based system that allows for competitive bidding on data products with the aim of reducing response time and reducing mission costs due to re-planning of missions is outlined. The second use case is the autonomous detection of scientific events in which multiple class three autonomous spacecraft cooperate to improve the chance of detection of a scientific event.

5.5.1 Generation of data products in near real time scenario

In this scenario we will look at the GMES response to a near real-time emergency request. In this type of request it is envisaged that many different data products could be required to meet the needs of different clients involved with responding to the emergency, these could include land or sea based civilian responders or governmental groups.

In this scenario all of the different classes of spacecraft will compete to give the best possible product for the user once the specific request has been made. The desired product will have many parameters that must be assessed such as deadline, resolution, area of interest, etc. For simplicity at this stage only the data service has been modelled as an agent based system and the customer facing services left as they are. It can be assumed for the specifics of this case that the service providers generate and deliver the correctly formatted requests and the actual heavy lifting is carried out by the data provision service.

This scenario has a number of key actors. There will be number of users or a user that makes the actual emergency request. The request itself will consist of number of variables that the user wants met and level of priority given to each variable. In this case alternative products that meet some of his desired variables and not others using this fuzzy definition of his requirements can be suggested. Each user or user group, depending on how they access the system, will be represented by an interface agent that will negotiate on its behalf. A multi-mission broker agent is tasked with negotiating with the mission agents on behalf of the user agents and will generate requests, based on the user's requirements, that are compatible with the spacecraft.

Both the Sentinel type mission developed specifically for a GMES type scenario and third party missions will need proxy agents to act on their behalf. These proxy agents will represent their respective mission in negotiations with the multi-mission broker agent. The proxy agents will have a standard interface for communication with the broker agent but allow for different architectures and baselines on the different spacecraft in the system. The Sentinel missions depending on the final level of autonomy may well have the capability to interact directly with the broker but the implementation of proxy agents despite this makes for a system that is easier to develop and maintain as new missions are added.

The multi-agent approach to GMES aims to improve the responsiveness and availability of the system. The interaction between the multi mission broker agent and the proxy agents representing each spacecraft will be designed as a cooperative system which can be thought of as a number of consumers (represented by the broker agent) negotiating for the use of a finite set of resources represented by the proxy agents of the spacecraft. The cooperation between the agents needed for this strategy to work is based on both the local intelligence of the individual agents tasked with autonomous planning and autonomous on board science as well as the distributed intelligence that arises from a successful negotiation structure. The distribution of decision making in this design raises the possibility of maximising the social welfare within the system while reaping the robustness benefits of a fully distributed system.

Within this system design where agents negotiate the use of limited resources for the collective good there are two possible scenarios that would require full use of the distributed problem solving abilities inherent in the multi-agent system. The first is the situation where a number of different consumers make incompatible requests for the use of a spacecraft resource. Through a negotiation sequence where the multi mission broker agent takes into account the different consumers priorities, the history of a given consumer and the possibility of finding any alternatives that satisfy the consumers, the distributed agent system can find a result that satisfies the greater good. The second situation is that of a request being made by a consumer for information that requires the collaboration of 2 or more proxy agents (and thus spacecraft) in order to fulfil its request. As such, the negotiating multi-agent system needs to parametrise a number of variables to be able to effectively find an optimal solution. It must parametrise the different needs of all its consumers so that they may be compared as well as parametrising the capabilities of its resources in such a way that an agent is not only aware of its own resources but also of the resources available through other proxies. The resources available to and required by an agent are typically resources related to the payload of a given satellite, computational resources on a spacecraft or on the ground and services supplied by other agents.

The constraints and interrelations between different elements in the system must also be parametrised along with the priorities of the system and any expected results in order to get a meaningful result from the negotiation step. This system involving the negotiation of consumers who compete to utilise resources is well suited to an agent based solution. As the decisions made by the multi-agent system are based on deep knowledge of each element's priorities, restrictions, resources and interlinks it is envisaged that the system would be able to provide a higher quantity of useful data to users as a whole versus the traditional undistributed approach. If such a multi-agent system is successfully implemented then it can be hoped that the system would see an improved success rate of acquisition requests, a higher level of resource utilisation, a clear and transparent planning system for serving data to users on a scheduled basis and an ability to combine data from a number of sensors to provide data for emergency services in a more timely manner.

Examples of the multi-agent based negotiation system will now be described to clarify its operation.

In the first simple case a user requests a product which is then negotiated with the GMES proxy agents. For this example the user must be logged into the GMES system in a way that complies with the appropriate service level agreements as well as all missions that are capable of providing data to the user taking part in the selection process. Initially the user will select an area of interest for his desired data product. After the region is selected a number of image and data characteristics must be chosen from a set of predefined values and ranges appropriate to the systems as whole. These characteristics and fuzzy requirements are defined in more detail in the section on implementing the image sorting algorithm for GMES. Based on these requirements the multi mission broker agent will initiate communication with each proxy agent about its ability to provide data that meets the requirements of the user. The proxy agents response will be based on the spacecraft’s current status including the current capabilities of its instruments, its current mission planning status (is it currently engaged in other activities?) the status of its sub-systems and other local parameters. The possibilities or bids generated by each proxy will then be displayed to the user with the system itself prioritising the prospective images based on the image meta data (This is expanded on in the image selection algorithm). The user can then choose the image or set of images they wish to have acquired. At this point the missions associated with the images the user wants taken will be informed and a request for specific images uploaded to the spacecraft's plan. Once the data products have been acquired they will be delivered to the customer, either as they are ready or when all of the data in a given dataset has been acquired. If the user’s requests cannot be met by any of the spacecraft then the multi mission broker agent informs the user of this at the earliest possible juncture. If a malfunction occurs and the requested image if not taken or the image request is de prioritised due to an emergency request the user is also notified at the earliest possible juncture.

5.5.2 Detection of an event

In this example a heterogeneous group of spacecraft are tasked with monitoring a particular geographical area for a particular event. The event to be detected will be pre-programmed into the system by way of a number of parameters that the spacecraft are observing for particular changes. The aim of the spacecraft is to work collaboratively to reduce as much as possible the response time from the given event occurring to the user being notified. To help explain how event detection could work we will discuss how events could be detected using generic examples. An interface that allows the user to specify to the spacecraft exactly what scientific event they are looking for and in which area, is envisaged. The spacecraft used for scientific event detection will be class three spacecraft as these are the only spacecraft that have the ability to autonomously change their status in response to specific sensor readings.

This type of event detection can only be carried out using the latest autonomous spacecraft as envisaged for future GMES missions. As such no direct comparison can be made between this specific case and the GMES baseline. The key difference is that in using the class three spacecraft the events can be detected and changes instigated all without the intervention of a ground station whereas in the closest comparable baseline scientific event detection scenario no event can be detected until the data is downloaded and analysed on the ground. For comparison between the autonomous and the baseline cases it is assumed that the users in the baseline case will process data acquired though the standard scheduled planning process to see if any scientific event has been detected.

In this case the three main actors are a set of users, the multi mission broker agent and a number of third party missions. Each user in the process will be represented by an interface agent who will forward any requests, negotiate on behalf of the user and update the user on progress or results delivery. The requests sent by the user will include a detailed description of the parameters associated with the triggering of the scientific event along with data as to the priority of the event (this is of increased importance where multiple users are making science event requests), the area to be searched for the described event and times for validity periods or deadlines.

As in the previous example the multi mission broker agent will have the task of negotiating with the mission proxy agents on behalf of the users taking into account the availability and capability of each mission as related to the user request. The negotiation will attempt to satisfy all of the user interface agents while maximising utilisation of the spacecraft via the proxy agents.

The other key actors in this case are the third party missions with autonomous science detection abilities. Earth observation spacecraft with these autonomous capabilities are not yet common place but it is foreseen that future mission will have increasing levels of autonomy, enabling this type of autonomous science detection through GMES.

The pre-conditions for this scenario is that the users involved are logged into the system in a manner compatible with the service level agreements describing interface and privileges etc. There must also be at least one class three spacecraft available for selection by the user and only those missions that are able to meet these requests are available (in that they are not undertaking some other higher priority task).

The normal flow of this specific test will be as follows. The user through the appropriate user interface agent selects a region of interest in which a search for a given scientific event will be undertaken. Once the region has been selected the event is defined using a number of parameters depending on the characteristics of the event under observation. It is envisaged that there will be a number of predefined events that the user can select but that the user will also have the ability to define their own scientific events based on a combination of parameters. The system will then show the user what spacecraft will be able to collaborate on this task (if any). This selection is created through the multi mission broker agent and is based on the projected orbital paths and capabilities of each class three spacecraft. The users will then select from these spacecraft the individual satellite that they want to collaborate with on this request and the selected mission will be asked to begin the detection of the described scientific event. Once the triggering conditions are met on any spacecraft the spacecraft will autonomously alter its plan to start analysing the corresponding in data and a notification will be sent to the ground segment that detection has occurred. The payload data will then be downloaded from the spacecraft when appropriate, with class three spacecraft this will be relatively soon after first detection. The data is then analysed on the ground and if the event is confirmed the user is informed and given the data regarding the scientific event. There are a number of alternatives to successful completion of the request as described above. There is the possibility of a false positive from the spacecraft and it is up to the ground control analysis to make sure this is not passed to the end user. With the limited number of autonomous class three spacecraft available it is likely that the situation where there are no available autonomous spacecraft available to undertake the request will be common. Checks will also have to be made on the criteria used to define given scientific event.

5.6 Multi-agent system design for the GMES mission

As previously described the GMES mission can be seen as a good candidate for introducing autonomy in order to improve performance over the baseline design. In the previous section a brief outline was given of a distributed multi-agent system design for GMES. In this section we will propose our distributed multi-agent system for GMES in full before describing the novel image selection algorithm and graphical user interface that completes the link between the GMES users and the GMES spacecraft constellation. Again this work is a continuation of [15, 51].

The multi-agent system (MAS) designed for GMES has two primary functions, to enable the flow of requests for data to the correct spacecraft and to enable the flow of the desired data back to the proper user. The MAS in this design is always on and in contact with the spacecraft through their appropriate proxies as well as any active users through their user proxy agents. The structure of the

MAS is based on the spacecraft proxies being informed of a user's request and then making bids comprising a set of images from that spacecraft that best meet the user's criteria. The multi-mission broker agent collects all of these bids and resolves any conflicts so that only those images that can be acquired remain. This set of images is then sent to the user. We will discuss our proposed method for displaying and ranking the images in the next section.

5.6.1 Ground Segment

5.6.1.1 Multi-Mission Layer

The MAS ground segment can be thought of as consisting of two layers, the mission layer and the multi-mission layer. The multi-mission layer consists of the agents that coordinate the flow of requests and data and instigate any collaboration or resolve conflicts between spacecraft. The user proxy agent is the first point of contact between the user and the GMES MAS. The user proxy agent gathers the image requirements from the user. These requirements will consist of information such as the desired time of acquisition, the deadline for acquisition, the frequency of updates to the imagery (if at all), resolution, spectrum etc. The user proxy agent may operate through any number of different interfaces including but not limited to, request forms, web based user interfaces or direct access via a workstation. The user proxy agent may not gather all of the information it needs from the user whether through a limitation in the interface or through lack of knowledge of the user. In this case the user proxy agent must be able to make assumptions as to the missing parameters and this will be achieved by making intelligent choices based on similar results and standard values for any given characteristic.

All of the now complete and formatted request data is then sent from the user proxy agent to the (multi-mission) broker agent. The broker agent's primary goal is to negotiate the provision of image data from the mission proxies to the end users. The broker agent will communicate the requirements from the user and will negotiate a set of images from each mission proxy that the associated spacecraft can acquire based on its current plans, capabilities and other requests on the system from the same user or other users.

5.6.1.2 Mission Layer

At the mission layer we have all of the agents that enable the system to act on image requests and determine if request for image acquisition in the future is feasible.

A mission proxy agent will be associated with each mission within the GMES constellation. The mission proxy will negotiate with the broker agent to try to provide a set of images that best meet the user's requirements but also make best use of the system resources (Payloads, Computation, Communication, etc.) as a whole. The negotiation will be based on the spacecraft's parameters such as when it can record a given image, the constraints on its payload and its commitments to other users in the meantime. Each mission proxy will be tailored to its individual mission in order to take into account its individual capabilities and constraints. The mission proxy agent is also tasked with coordinating with the other mission proxies to provide products that meet the user's requirements. There may be requests that require imagery of an area at intervals that cannot be achieved by single spacecraft. In this case the spacecraft must collaborate to accomplish the goal. As such each mission proxy agent must have knowledge of the status and abilities of all of the other mission proxy agents in the system. It is in this way that a multi-agent system really benefits GMES as it allows for more stable and fast creation of coalitions between agents. The proxy agents must work together in order to provide the decision making capabilities required to meet the user demands while maintaining availability and performance for other users. The mission proxy agent is also tasked with providing and requesting information from the other mission level agents such as requesting orbital position predictions form the flight dynamics agent or coordinating the planning agents of its own mission and that of another to provide adequate coverage.

There are a number of other agents at the mission level that support the decision making capabilities of the proxy agent and thus the MAS as a whole.

The flight dynamics agent provides information about the spacecraft's orbit and attitude at the current time and predictions of its state into the future. This allows plans that incorporate the satellites predicted fly over time to be created. It will also allow the mission proxy agent to determine if the spacecraft’s current orbit and attitude allow for any given collaborative set of images to be taken.

Each spacecraft also has a dedicated ground based planning and scheduling agent. As discussed previously the planning agent constructs a plan for the spacecraft which consists of a detailed time line listing the activities that the spacecraft should carry out. As with the other agents at the mission level there will be close negotiations in order to satisfy each agent. The planning agent receives instructions from the mission proxy and takes into account data from the other mission agents such as the flight dynamics agent. The negotiation between the planning agent and the mission proxy with full access to the appropriate data can lead to the creation of a plan that respects all of the constraints of the system but also tries to maximise utility to the user. There are also specific agents tasked with coordinating uplink to the spacecraft and coordination of communications with the ground stations, these agents must also collaborate and negotiate with their counterparts for each spacecraft. The data processing agent is tasked with generating the science products at the correct time and with sufficient quality to meet the service level agreements requirements. The data processing agent has contact with the user interface agent and the agents at the mission level. The data processing agent will negotiate with other agents for resources to reduce the time it takes to produce any given data set.

5.6.2 Space Segment

Many of the agents in the space segment are counterparts to the ground based agents. They could be abstracted into individual components that span both ground and space segments but for clarity they have been separated. The communications agents on each of the spacecraft operate with the single ground station agent which resides at a multi-mission level. The on-board communication agents is only concerned with the communication from its own spacecraft and its primary functions are to gather the data that is to be transmitted, transmitting the data and receiving and passing on requests or commands from the ground segment.

Likewise the on-board planner agent complements the ground based mission planner. In this case there is one mission planner per mission so the ground and space based planners form a pair specifically for that mission. The on-board planner takes the high level instructions from the ground based planner and creates plans for execution on the spacecraft. When making these plans it can take into account data from the other on board agents such as the monitoring agent or execution agent that may not have been available to the mission planner agent. This behaviour also allows for the detection of scientific events as the on board science agent can directly influence the spacecraft plan without intervention for the ground.

The executor agent executes the plans generated by the mission planner and refined by the on board planner. The executor agent decomposes the plan into a series of actions that it finds can meet the current constraints of the system.

The on board monitoring agent is tasked with monitoring the spacecraft for any anomalous changes in state that could indicate a failure or a reduction in capability. The complexity and design of the spacecraft making up the GMES constellation will vary greatly and the monitoring agent will be specialised accordingly. The monitoring agent will collect this data and will notify other agents of the spacecraft's ability to acquire a given image or even operate as planned.

The on board payload agent is tasked with actually acquiring the data to meet the users initial requests. As well as having access to the payload sensors it must also cooperate with the communications agent in order to get the data to the ground.

The on board science agent allows the spacecraft to acquire data when a scientific event is detected without any intervention from the ground. The on board science agent is described in more detail later in this section.

5.7 Definition of the GMES MAS using the HASA architecture

In the following section the design for a GMES MAS will be outlined. This design is based on the HASA architecture outlined in Chapter three. As previously described the agents in the HASA architecture can be of one of 4 types: Product, Executor, Planner and Resource. Below the high level agents are defined and grouped based on their segment (ground or space) and their mission level.

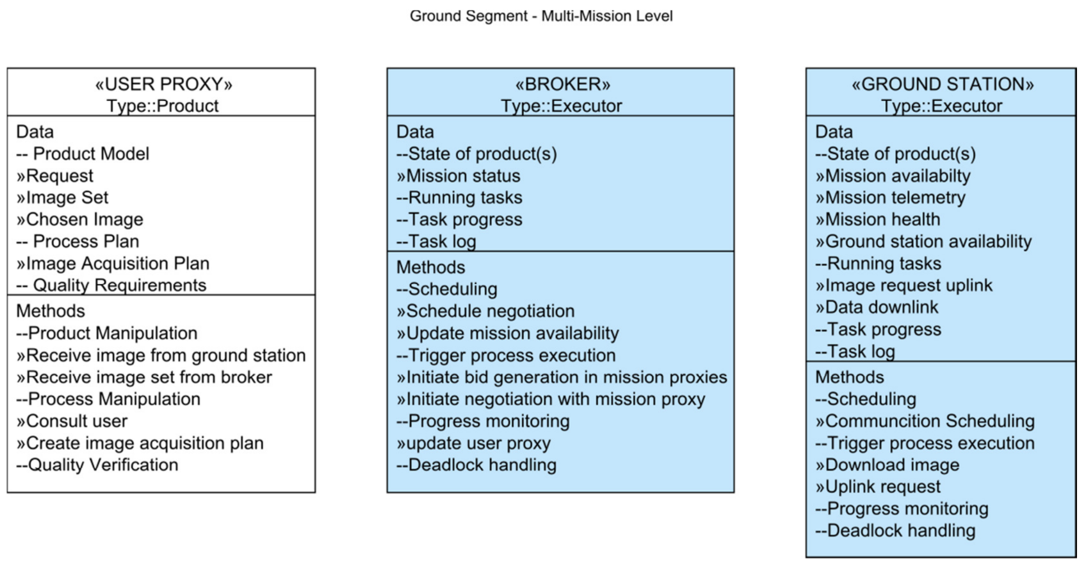


Figure 5-3 GMES HASA ground multi-mission level agents

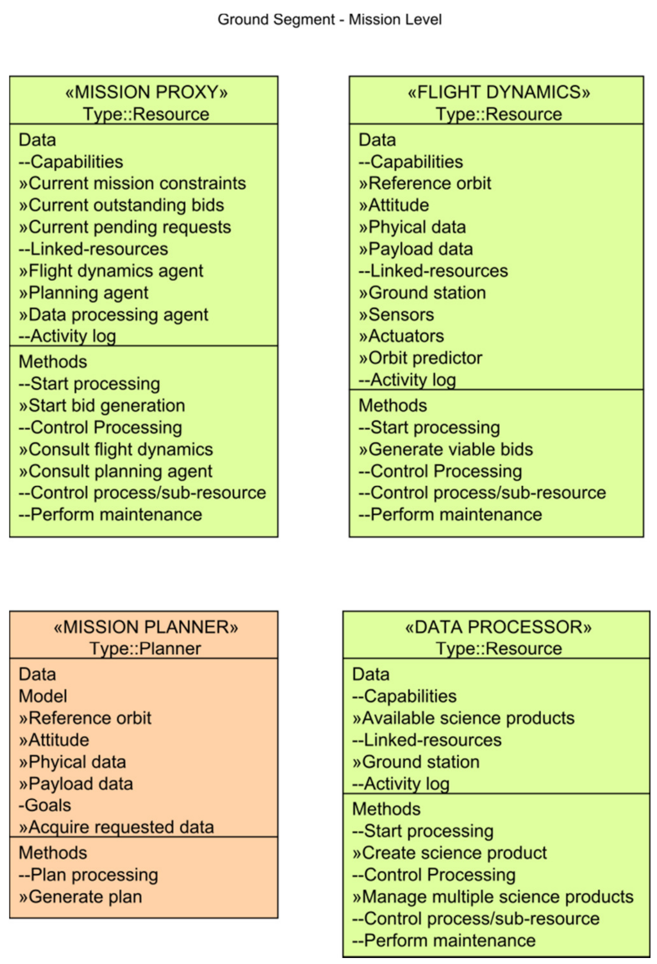


Figure 5-4 GMES HASA ground mission level agents

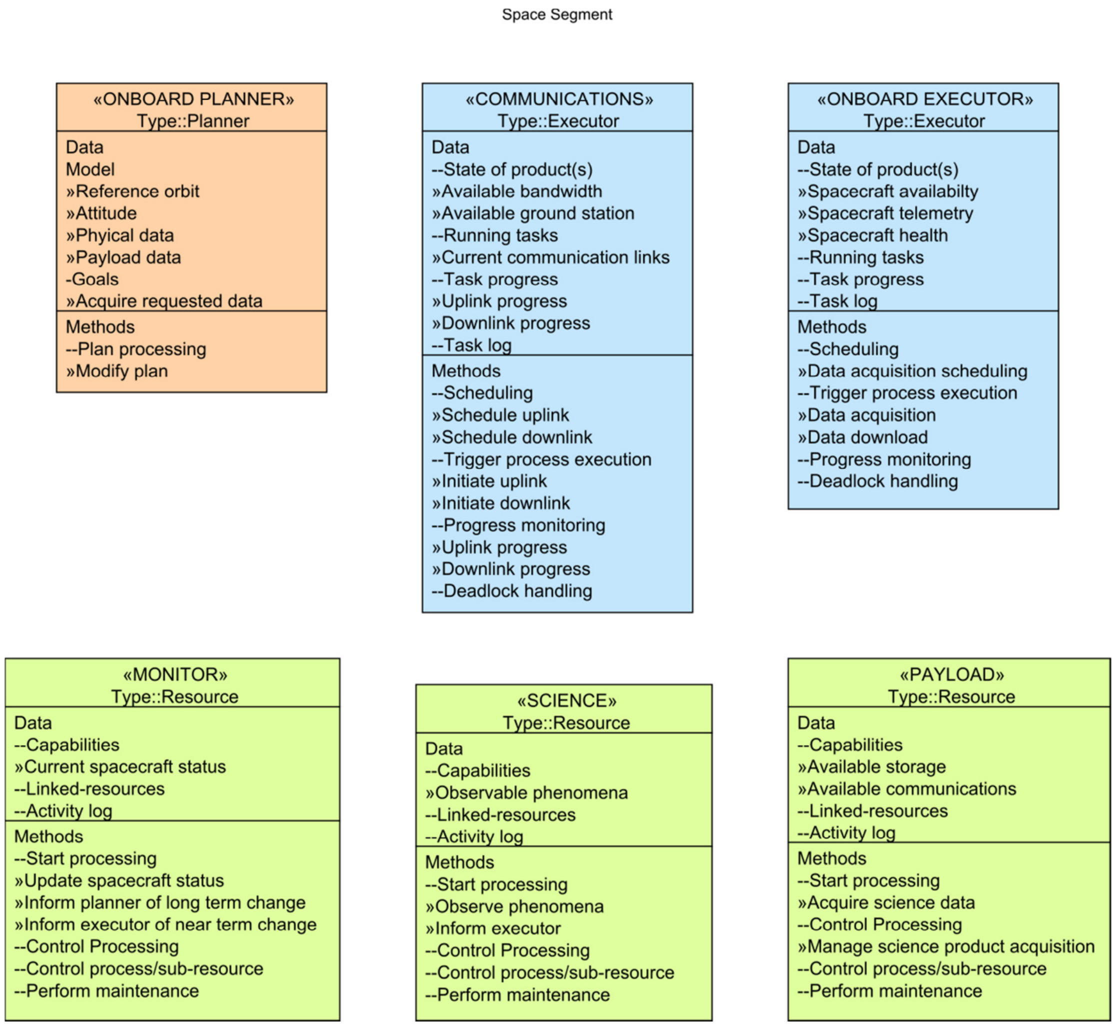


Figure 5-5 GMES HASA space segment agents

Figure 5-6 shows the distribution of the agents more clearly between the ground and space segments and the mission and multi-mission layers.

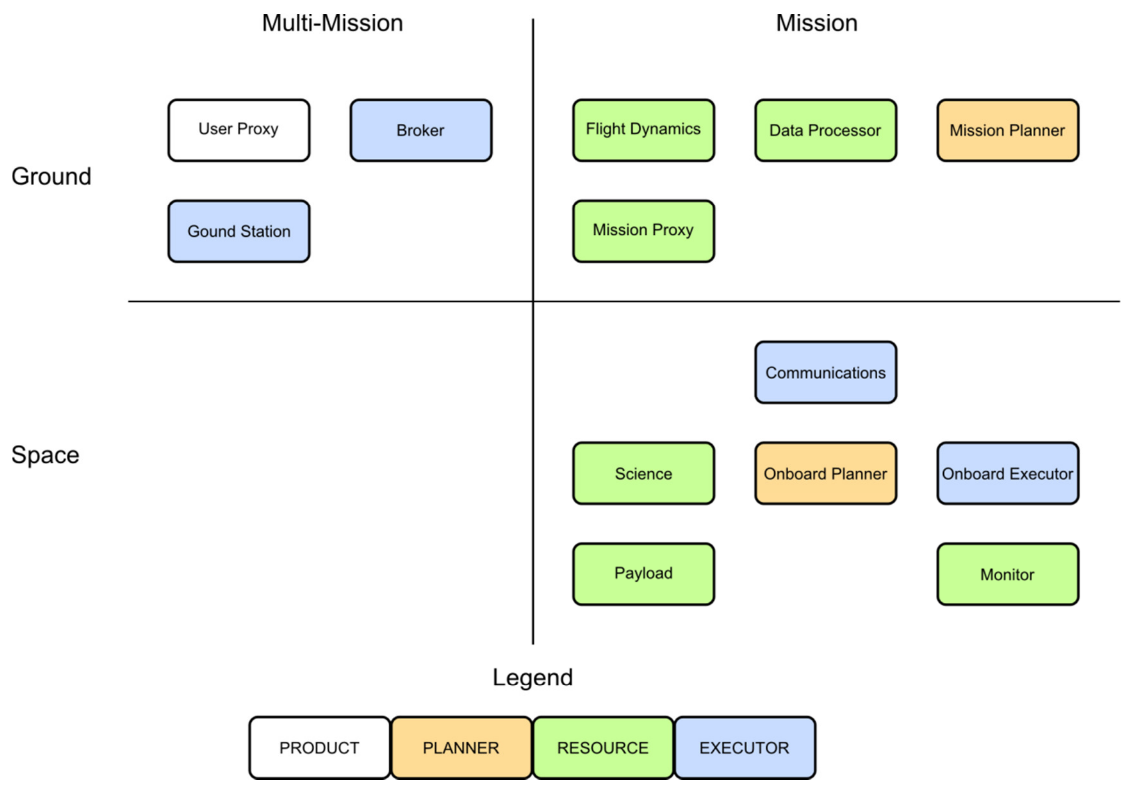


Figure 5-6 Distribution of agents in HASA GMES

5.8 Limitations and constraints on spacecraft autonomy

There are number of constraints and issues that must be understood in order to successfully design an autonomous space system [133]. For a typical spacecraft in Low Earth Orbit (many of GMES spacecraft will be of this ilk) there are severe limitations on communication with the ground. For example a typical LEO spacecraft has approximately eight to ten minute communication windows. The communication opportunities for the spacecraft that will constitute GMES are not as severe as those applying to deep space missions (such as the DARWIN mission for example) which sometimes means that there is no communication for weeks rather than hours but even for the LEO case regular communication cannot be guaranteed. This lack of communication opportunities is the reason that on-board autonomy is so valuable.

Another challenge with implementing an autonomous control system or any autonomy on a spacecraft is the spacecraft's inherent complexity. The spacecraft that will be used for the GMES mission contains thousands of components and even with the push towards using more commercial off the shelf components there will still be one of a kind, or mission specific components and relationships in for example a specialised payload sensor. This complexity leads to many intricate interrelationships between components and systems. These must all be characterised and understood by any autonomous control system.

The complexity of the spacecraft and the limited resources available means that there is limited observability of the spacecraft. This means that any autonomous control system must be able to operate with only partial information about the state of the spacecraft. It is important that key spacecraft parameters (temperature, power, storage etc.) are monitored but continual monitoring of every single spacecraft components down to the nut and bolt level would be a waste of on-board resources. An autonomous control system does however have more information than a ground controller due to the elimination of the downlink communication bottle neck.

As stated before the resources available to any autonomous control system are limited. One of the biggest limitations is that of computing resources. A typical spacecraft processor offers roughly 25 million instructions per second and small amount of random access memory (RAM) (usually 128-256 megabytes). This is far less than is available on even the simplest computer on the ground. To compound matters, most of this computer power is used in the day to day operation of the spacecraft and only a small amount is allocated to any autonomous system (in the case of Earth Observing Mission 1 (EO-1) about 4 MIPS or about 16% of the total computing resources [134]).

One of the primary challenges that an autonomous system must overcome is the additional risk of operating without human intervention. This risk stems from not only the uncertainty that comes with using a novel autonomous system but also from the extremely high value of the spacecraft and the overall mission cost. Any failure by the autonomous system could lead to severe recriminations. Obviously a fault leading to a catastrophic failure of the mission is the worst case and even a failure that leads to a delay could cost the operators of the satellite enough money to dissuade them from deploying an autonomous system in the first place.

5.9 Autonomous science agent in operation

Previously the need and possible operation of an autonomous science agent in the context of GMES was discussed. As a baseline the autonomous science system that has been deployed EO-1 will be outlined.

The autonomous system on the EO-1 mission is called the autonomous science experiment (ASE). The ASE is equipped with a set of high level goals that are supplied form the ground and these correspond to the science targets that it should be monitoring. The on-board planner for the EO-1 mission, CASPER generates and operations plan based on these high level targets. CASPER is based on a model based planning algorithm [133] which enables it to plan for a wide range of operational scenarios but still be able to respond effectively to unforeseen events. The operational plan in this case consists of a plan to monitor scientific targets on the ground using the on board instrumentation. If during these planned operations any new science event, as defined in the high level instructions for the spacecraft, is detected then a new science goal is autonomously generated. The planning system, CASPER, must then integrate this new science objective into the operational plan in order to re-image or reacquire the event depending on the nature of the science event itself. The plan is then executed to acquire the newly prioritised science data. This cycle is then repeated as the spacecraft works through the current operational plan.

The key difference between the autonomous science detection used in EO-1 [133] and that proposed in this work is the use of multiple science agents distributed throughout the GMES constellation.

The benefits of an autonomous science agent are made clear in the EO-1 data shown in Table 5-1 [135].

Process	Total Process Data Acquired (MB)	Data returned by ASE (MB)	Savings Factor
Volcanism	33750	294	115

Cryosphere	38100	304	125
Flooding	25500	239	106
Total	**97350**	**837**	**116**

Table 5-1 Data reduction from EO-1 autonomous agent

The key factor in which a distributed approach can improve on a single autonomous science agent is in the speed of acquisition and monitoring of new events. In the EO-1 case once the anomaly is detected the agent adds a monitoring task to its plan for its next pass. In the distributed case once a spacecraft detects an event it can be added to its own queue but also propagated to other spacecraft who may decide to add it to their own queue. This means that multiple spacecraft can be autonomously tasked with monitoring the science event leading to improvements in response time and update rate.

A key aspect of this type of distributed system is that it will improve with scale. As more autonomous science agents run on more spacecraft, events will be detected faster and will be covered more completely by more spacecraft.

The key requirement for the operation of this type of distributed autonomous science agent is autonomous communication between the science agents on all of the spacecraft.

This will be carried out by passing all possible science targets from the spacecraft's science agent to its on-board planner and to the mission planner and from there to the multi-mission planner from where they can be disseminated to the other spacecraft through the normal tasking process.

5.10 Image Selection Algorithm

The following image selection algorithm is an extension of the work presented here [136]. As the basic MAS for GMES was implemented by GMV a novel image selection algorithm was developed to interface to the MAS in order to build on their work.

In this system the images supplied by the GMES constellation themselves are not used; instead we use a set of variables to describe the images. This allows us to describe images that have been acquired and that may be acquired in the future. These variables or metadata describe the images key characteristics from the customer point of view.

The variables describing the image products are:

- Desired area of the image - n-sided polygon describing the desired area of the image
- Cloud cover - percentage of the area of the image occluded by cloud
- Hard deadline for the image – the latest possible time for the image to be captured
- Desired time for the image - target time for the image to be captured
- Resolution - the size of the area described by one pixel in the image or Ground Sample Distance (GSD)

The images that most closely match the user request form each spacecraft's bids within the MAS. Each bid has values for each of the above variables and it is the algorithm's task to sort these bids into a preferred order for the customer.

The image sorting algorithm consists of two main parts. The first part uses the customers' criteria to find the five images that most closely match the desired variables. The deadline is used as a hard limit and if an image exceeds it it is excluded. In the case of all other variables the closer the variable is to the desired value the better.

The area variable is treated differently. The desired area is supplied by the customer as the points of an n-sided polygon made up of longitude and latitude coordinates. The area covered by the images is also an n-sided polygon ('n' doesn't have to be the same). The algorithm calculates the overlap between the polygons to generate a value which is then used to help rank the images (larger value for overlap is ranked higher).

Once all of the images have been received they are ranked or excluded based on their metadata. If no images are left after this step then the second part of the algorithm comes into play. In reality this will be quite a common occurrence because the set of images available may be quite small and the user may unknowingly set too stringent requirements for the exclusion of images based on deadline or cloud cover.

The algorithm operates as follows. The user defined inputs are collected from the user through the graphical user interface (GUI). These inputs are:

- Area Threshold
- Deadline
- Desired Observation Time
- Resolution
- Cloud Cover Threshold
- Area Weight
- Desired Observation Time Weight
- Resolution Weight
- Desired Area (As polygon coordinates)

The other input to the algorithm is the set of images supplied by the broker agent.

A first pass is carried out on the image data before any images that do not meet the thresholds are excluded. This step involves calculating the area of overlap between the desired image and images in the dataset. The images and the desired area are encoded as n-sided polygons with latitude and longitude coordinates for each point. To calculate the overlap we first find out if any of the vertices of either polygon lie within the other polygon. Secondly we check each side of the image polygon to see if it crosses a side of the desired polygon. Once these enclosed vertices and crossing vertices have been identified, the overlap polygon can then be determined by finding the convex hull of the vertices as shown in Figure 5-7.

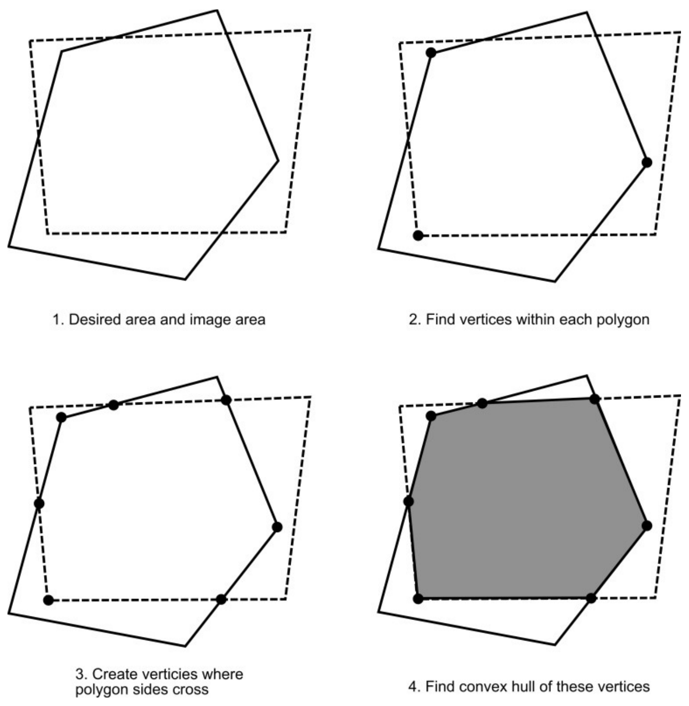

1. Desired area and image area
2. Find vertices within each polygon
3. Create verticies where polygon sides cross
4. Find convex hull of these vertices

Figure 5-7 calculating the overlap of polygons using the convex hull

Each proposed image is then compared against the thresholds for area overlap, the deadline, the cloud cover and the resolution. Rather than remove the images that do not meet the thresholds, the images are kept but excluded from the subsequent steps as we want to include them in the secondary global optimisation step.

For each image in the set the difference between the desired time and the image time and the difference between the desired and image resolution are calculated. These values along with the area overlap which was previously calculated are normalised and then multiplied by the user supplied weightings These weighted values are then summed and the remaining images sorted on these values. The top five images are then presented to the user.

5.10.1 Global optimisation step

If no suitable images are found a multi-dimensional global optimisation on the images is performed by the algorithm. This optimisation is not to find images per se but to suggest to the user which variables could be changed and by what amount in order to find suitable images. In this way the customer can decide which constraints to relax. This approach gives the user results outwith the initially supplied thresholds. This therefore allows the agent to suggest inclusive threshold values to the user. Any global optimisation algorithm requires an objective function, that is a function that produces a scalar value that summarizes the performance of the system for a given value of the variables that are being optimised. That is the aim of the global optimisation algorithm is to minimise the objective function $L(\theta)$ where θ is a vector of system variables. $\theta \in \Theta$ where Θ is the domain of allowable variables for θ.

In our case

$$\theta = [A_i, T_i, R_i, CC_i] \tag{5-1}$$

where:

A_i = Image area

T_i = Image time

R_i = Image resolution

CC_i = Percentage cloud cover in image

The objective function is:

$$L(\theta) = (O(A_i, A_u)A_w + (T_i - T_u)T_w + (R_i - R_u)R_w \\ + (CC_i - CC_u)CC_w) \tag{5-2}$$

Where:

$O(A_i, A_u)$ = Overlap function of the current image and the user's desired area A_u

T_u =Users desired time

R_u = Users desired resolution

CC_u = Users desired percentage cloud cover in image

This objective function is used for comparison with a number of global optimisation methods.

5.10.2 Comparison of Global Optimisation techniques

Global optimisation methods are mathematical approaches that aim to find the minimum of any given function. In most cases this is the global minimum of a multi-variable function or set of functions. There are a great many different global optimisation methods with distinct advantages and disadvantages when it comes to different optimisation problems. In general though, all of the methods aim to find the global minima as quickly as possible without becoming stuck at one or more local minima.

Global optimisation techniques can be broadly grouped into 3 categories, deterministic, stochastic and meta-models.

Deterministic global optimisation methods utilise methods that have no random aspect and given the same data and initial set of conditions will always reach the same optimum value. These methods commonly operate based on the local gradient of the function at any given point. An example of this approach is the Lipschitzian method [137]. A method such as the Lipschitz method requires a constant, the Lipschitz constant which determines the extent of the search and thus determines whether the search favours local search or global search. For many optimisation problems a suitable value for this constant cannot be specified beforehand so the search may not be truly global. To overcome this possible problem, a modified deterministic global optimisation method was developed based on the Lipschitzian model but without the requirement for a Lipschitz constant. This new method was named the DIRECT method [138] and works by searching and using all values of the Lipschitz constant simultaneously.

5.10.2.1 Simulated Annealing

Stochastic methods all utilise some random factors in their search. This makes the methods non-deterministic in that for any given set of initial conditions and a given data set the stochastic methods may not produce the same answer. The stochastic methods utilise the random variables in order to “climb out” of local minima and proceed to the global minimum. A good example of a stochastic method is the simulated annealing (SA) method. Simulated annealing takes its name from the approach used in metallurgy and thermodynamics where a material is heated to a high temperature then the temperature is slowly reduced. As the temperature is reduced, the internal elements of the material naturally try to find lower levels of internal energy. In this analogy the global optimum is the lowest internal energy possible for the material. The stochastic nature of the method allows for small movements away from the current position, even “uphill”, i.e. away from the local minima. In this way the method may be able to escape local minima and search the entire function space.

5.10.2.2 Genetic Algorithms

Heuristic or meta-heuristic based global optimisation methods operate by trying to maximise some objective function. This is done by measuring the “quality” of candidate solutions and then aiming to improve the candidate solution and again testing its “quality” using the objective function. The actual method used to try and improve the candidate results is at the core of the heuristic or meta-heuristic. A well-known meta-heuristic method is the genetic algorithm (GA) method. In this method a set of candidate solutions are generated. Each candidate is then given a fitness value based on its parameters (which map to the variables needed to be optimised). The candidates with the highest values of fitness breed and produce new offspring based on their parents. This process is repeated over many generations to try to find the solution with the highest possible fitness value. The genetic algorithm method is also stochastic as at each generation a certain amount of mutation of the offspring occurs allowing for variation within population and the discovery of solutions outside the area described by the initial candidate set.

5.10.3 Comparison of algorithms

Four different global optimisation methods were compared for inclusion as part of the image selection algorithm. As a baseline a standard sort algorithm was also run to show any improvement in calculation time if any over the method used to initially rank the images. The four methods chosen for comparison were, DIRECT, general pattern search (GPS) [139], simulated annealing (SA) and genetic algorithms (GA).

As is shown in Figure 5-8 the global optimisation methods are all outperformed by the standard sort on small data sets. This was to be expected and it is the algorithms performance on larger datasets, like those to be expected from GMES that is of interest. The two deterministic methods show the lowest computation time by some margin, followed by the meta-heuristic genetic algorithm and lastly by the stochastic simulated annealing. In terms of problems that global optimisation methods are applied to, this is quite simplistic and as such the simpler deterministic methods perform better. Of the two deterministic methods, DIRECT was chosen for its slightly superior performance in large datasets as shown in the figures below.

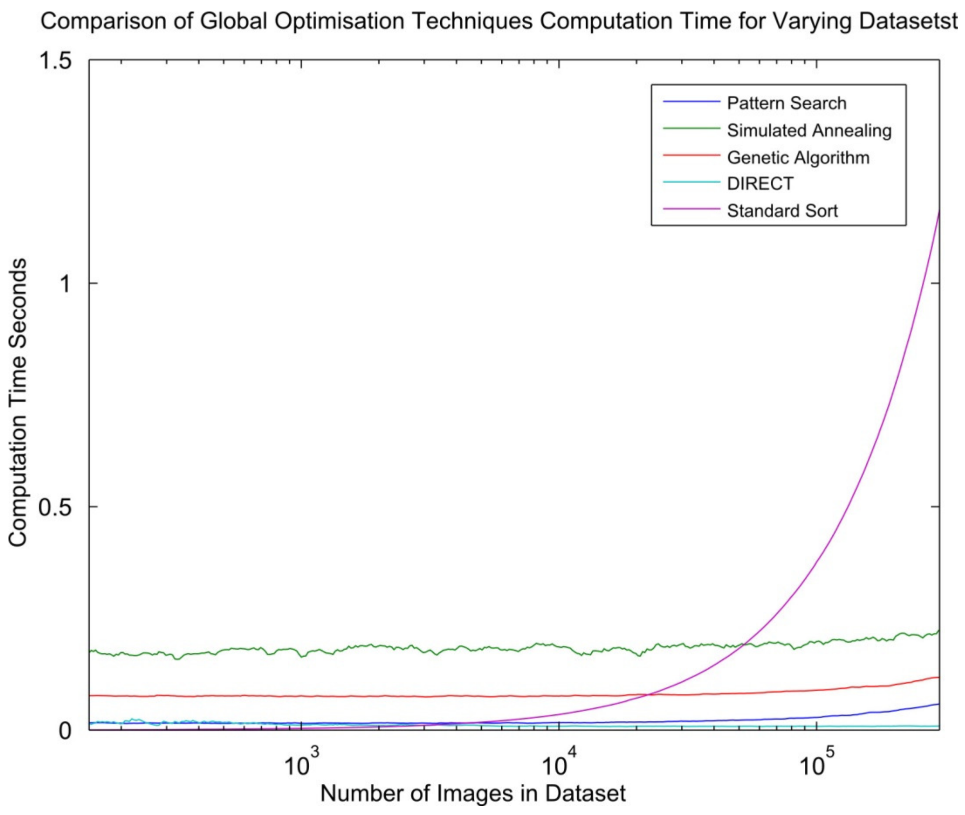


Figure 5-8 Comparison of global optimisaton techniques computation time for varying dataset size

For the sake of this comparison each global optimisation method was run five times to find the five optimal images within the set. The sort only had to be run once as the best five images can be accessed directly once the sort is complete. Each method was implemented as shown in Appendix D and without significant optimisation. Even though this is the case a clear performance improvement over the basic sort for large datasets is shown.

5.10.4 GUI

In this section the graphical user interface that was developed to demonstrate the image selection algorithm is presented.

The GUI itself allows the user to easily set the thresholds and variables required by the algorithm and to either see the top results or the suggestions for threshold corrections.

The GUI was built using Java and a number of other tools. These included the Google App Engine platform for running the server side tasks, the Google web toolkit for creating the browser based client side and the Google Maps API for generating the maps presented to the user.

The matrix manipulation of the sorting algorithm's first pass and especially the multi-dimensional global optimisation step can be computationally expensive depending on how many images are being sorted. In order to distribute the workload away from the end user a client server model is used as shown in Figure 5-9.

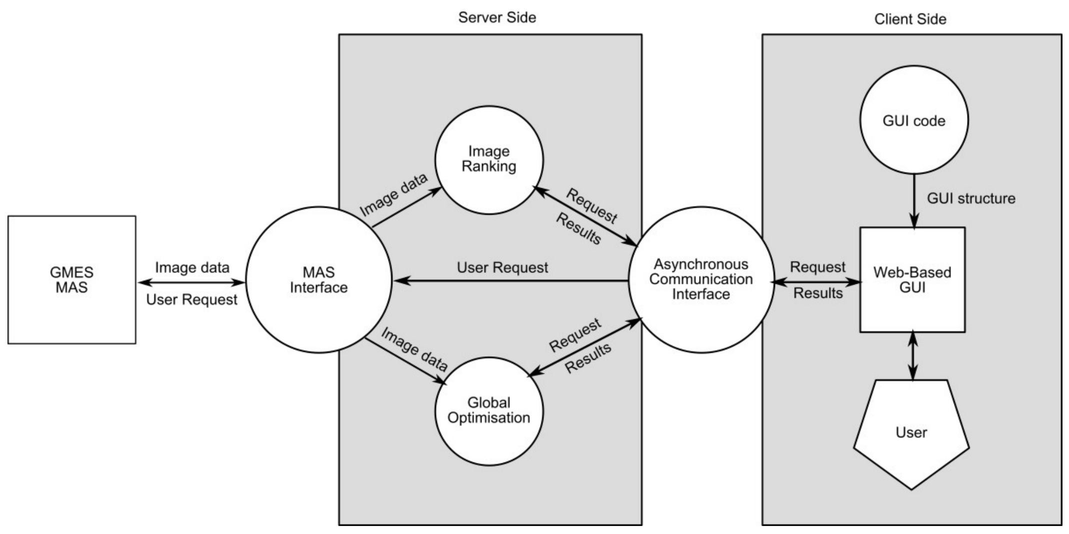


Figure 5-9 Client server interface structure betwee GMES MAS and GUI

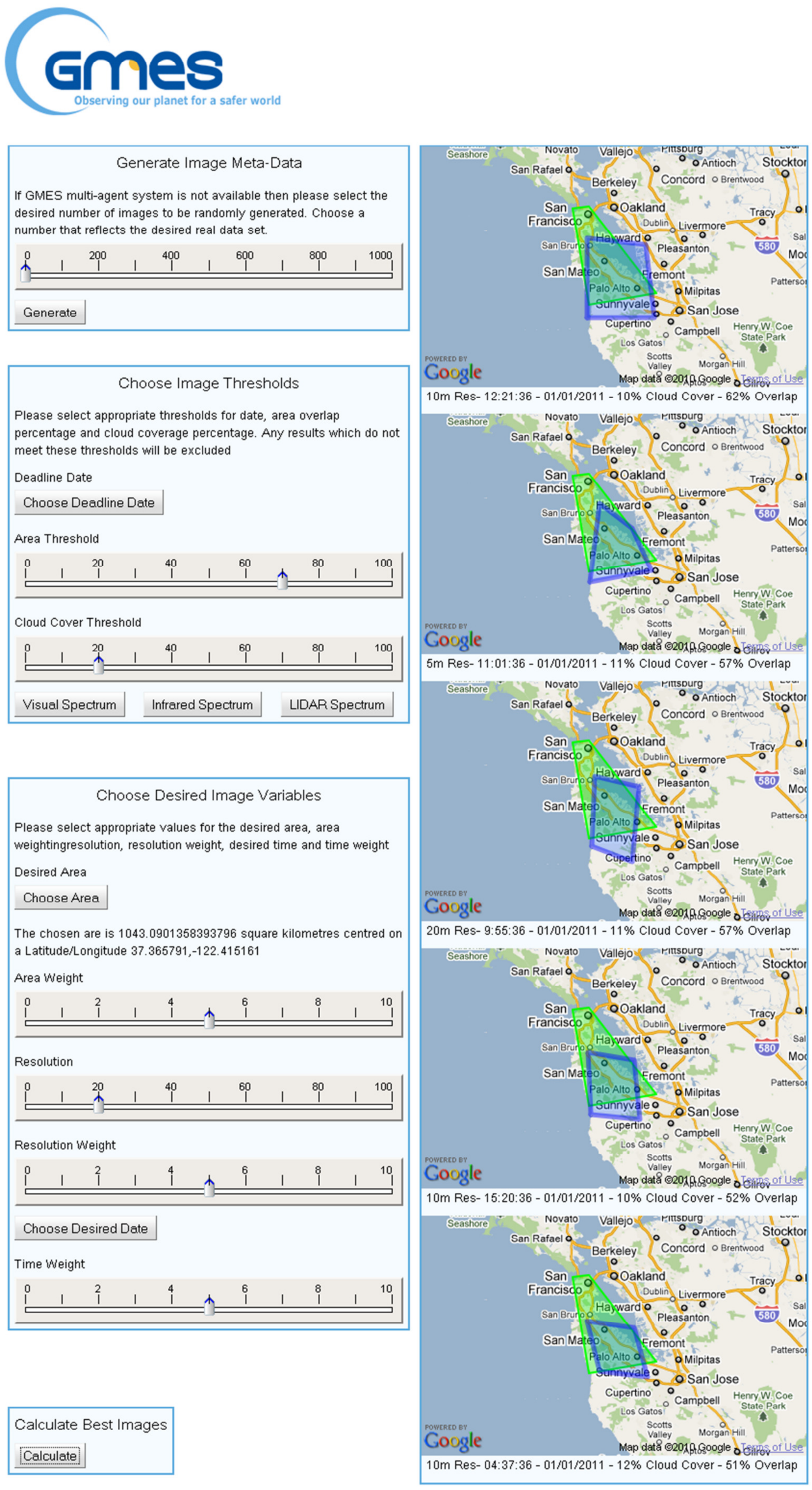
The Google Web Toolkit (GWT) [140] was employed, as it allows a lightweight web based front end to make calls to a server running the native code of the algorithm which is in turn coupled to the MAS. On the front end we wanted the interface to be as easy to use and as instantly accessible as possible. To this end all selection of geographical areas and the display of results is handled by using the GWT interface with the Google Maps application programming interface (API). This offers the users a familiar interface and allows us to easily acquire all latitude and longitude data from the user. To further improve the ease of use of the interface, variables are entered where possible through calendar pop ups and sliders to minimise the use of text boxes and to show the user realistic and sensible ranges for a given variable.

The web based GUI is delivered to the user entirely using standards compliant HTML, CSS and JavaScript. This will allow any user on any platform that runs a modern web browser to use the GUI while all of the computationally intensive calculations are carried out on the remote server. The use of Java and open frameworks will also allow easy integration with the GMES MAS.

In this test of the GUI and algorithm the image data is not supplied by the MAS but instead generated within the server. This allows us to test the algorithm easily for different data sets to ensure it is operating correctly. After the

generation of the image data set the user sets the hard thresholds for area coverage, maximum cloud cover and the deadline for the image.
Next, the user sets the variables for the desired image, this includes the longitude and latitude points of the desired area. This is done in a pop up showing a scalable map of the Earth and the points of the polygon are selected by simply clicking on the desired points on the map.

The user also sets the desired resolution of the image and the desired time of the image. All of these variables also have accompanying weighting sliders which will increase or decrease their influence on the overall ranking of resulting images. See Figure 5-10.


Generate Image Meta-Data

If GMES multi-agent system is not available then please select the desired number of images to be randomly generated. Choose a number that reflects the desired real data set.

Generate

Choose Image Thresholds

Please select appropriate thresholds for date, area overlap percentage and cloud coverage percentage. Any results which do not meet these thresholds will be excluded

Deadline Date

Choose Deadline Date

Area Threshold

Cloud Cover Threshold

Visual Spectrum | Infrared Spectrum | LIDAR Spectrum

Choose Desired Image Variables

Please select appropriate values for the desired area, area weightingresolution, resolution weight, desired time and time weight

Desired Area

Choose Area

The chosen are is 1043.0901358393796 square kilometres centred on a Latitude/Longitude 37.365791,-122.415161

Area Weight

Resolution

Resolution Weight

Choose Desired Date

Time Weight

Calculate Best Images

Calculate

10m Res- 12:21:36 - 01/01/2011 - 10% Cloud Cover - 62% Overlap

5m Res- 11:01:36 - 01/01/2011 - 11% Cloud Cover - 57% Overlap

20m Res- 9:55:36 - 01/01/2011 - 11% Cloud Cover - 57% Overlap

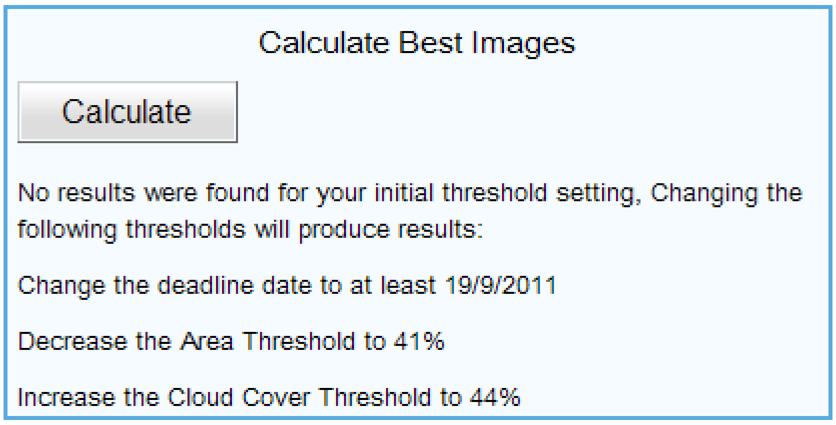
10m Res- 15:20:36 - 01/01/2011 - 10% Cloud Cover - 52% Overlap

10m Res- 04:37:36 - 01/01/2011 - 12% Cloud Cover - 51% Overlap

Figure 5-10 The complete GUI for image selection

Once all the thresholds, variables and weightings have been set, the user initiates the algorithm to calculate the ranking of the images. Any image that meets the threshold criteria it is displayed in ranked order on the right hand side of the GUI. The result shows the area covered by the image, the desired area and the overlap between the two (Figure 5). The values for the area, resolution and cloud cover are also shown. Cloud cover for future dates is currently modelled by a simple probability function but could be extended to encompass a more accurate meteorological model.

If an image is not available then no data is displayed in the results section. Instead a set of suggested threshold changes are displayed to the user as shown in Figure 5-11. These are generated by the algorithm's global optimisation step but are displayed to the user as suggestions. The user may choose which variable they would like to change and then re-run the algorithm knowing that at least one result will be presented as the new threshold values generated by the global optimisation step ensure this.



No results were found for your initial threshold setting, Changing the following thresholds will produce results:

Change the deadline date to at least 19/9/2011

Decrease the Area Threshold to 41%

Increase the Cloud Cover Threshold to 44%

Figure 5-11 Sample results screen showing suggestions to the user

The GUI that has been implemented here could further be extended in the future by incorporating more data from outside sources, for instance using semantic links to other GIS data sources [141] .

5.10.5 Test Cases

In the first example we will show how the system can be used to acquire data for monitoring crop coverage over a large area. In this example we will use the monitoring of rice crops in the Kunming area of China. Rice crops can be monitored in a number of ways including using the visual spectrum and by synthetic aperture radar (SAR). Accurate image data allows the user, in this case most likely a government agency, to predict future yields and thus supply and price of future crops.

In this case the deadline for the images can be easily defined as the end of the growing season as the crops growth cycle is well established and growing seasons known for each region. The Area threshold will be set at 70% as multiple images can be combined for full coverage. The cloud cover threshold will be relatively tight and set at a maximum of 25%. Any images taken with SAR will not have to meet this requirement as cloud cover is invisible to SAR sensors. Images taken in different spectral bands are distinguished by tags in the image metadata.

With the desired area chosen as shown in Figure 5-12 the desired resolution must be selected to give enough detail to identify areas where the crop is being grown. The desired date of the image is chosen but is weighted less strongly than the area and resolution variables as the exact day of the image is not important as long as it is close to the desired date.

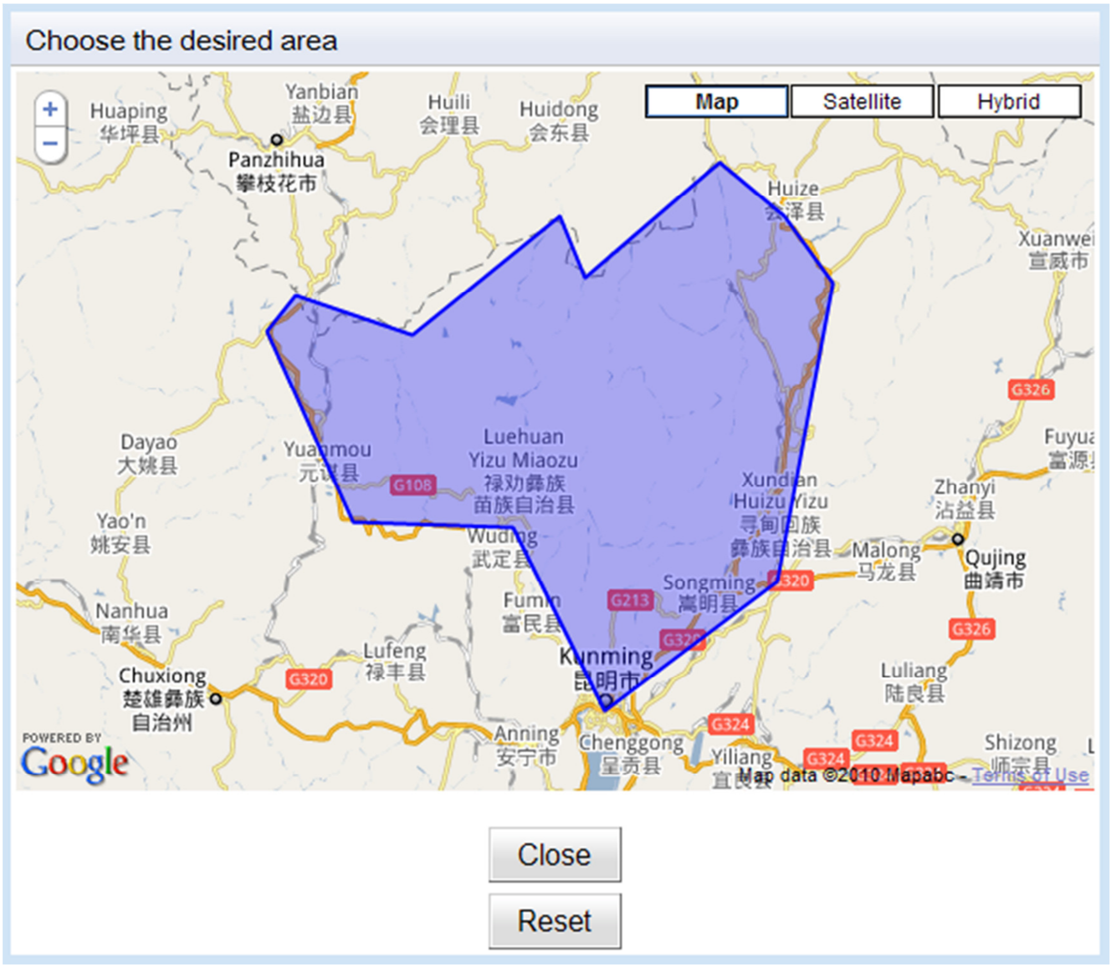


Figure 5-12 Area selection in the GUI

The data set supplied by GMES for this request would be large due to the relatively lax threshold values and thus probably not require anything more than the algorithms initial sort.

In the second example we will show how the GMES MAS could be used to provide images for a rapidly changing time critical such as that in a natural disaster or emergency. The response of systems to natural disasters is major area of research [142, 143, 144]

In a natural disaster scenario the deadline for images will be in the order of a few days, if not even a few hours. The further forward the deadline is brought the more chance that there will be no images available. At this point the global optimisation step is carried out and the best images selected, essentially telling the user when the first image that meets their criteria is available.

The area threshold will be set at 50% as any data on the area will be more useful if it can be supplied quickly, for the same reason the cloud cover threshold is at 25% as some occlusion can be accepted as long as there is data for the majority of the image.

The desired date variable will be given the highest weighting followed by the desired area and resolution. In this case the data set could be fairly small depending on the area and response time required. With a smaller data set the need for the global optimisation step is more likely. Scenarios where the algorithm cannot supply an image of the desired area but can supply an image of another affected area can be envisaged and the global optimisation step allows the system to suggest such opportunities to the user through suggested threshold settings. The results are shown to the user as in Figure 5-13.

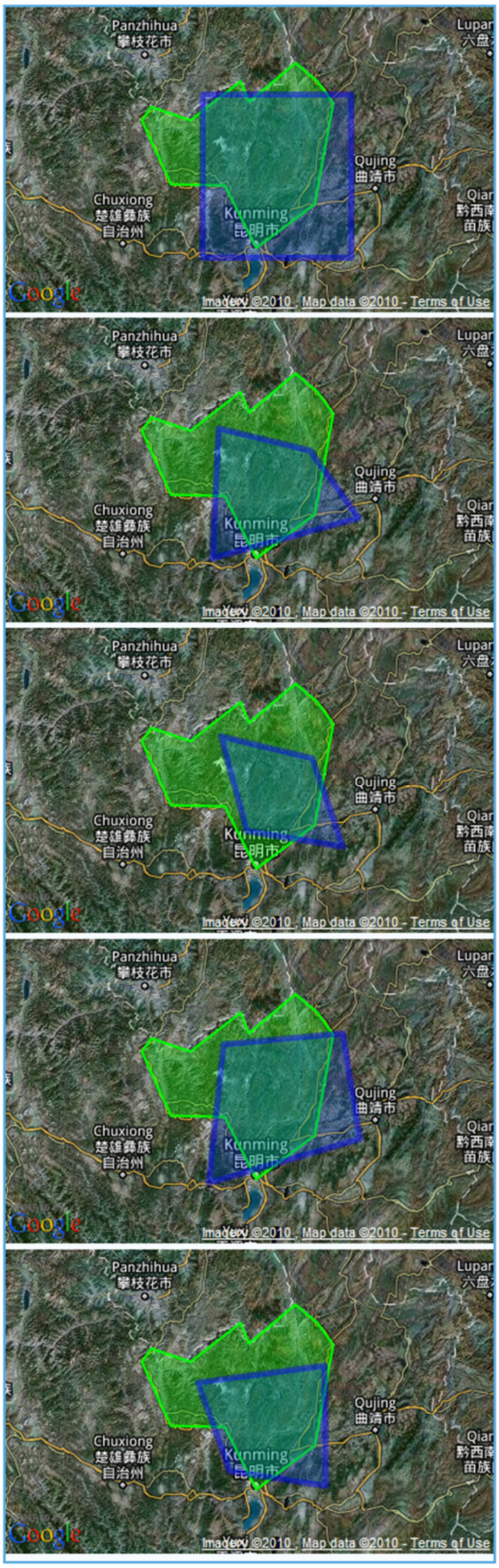


Figure 5-13 Display of results in the GUI

5.10.6 Algorithm Performance

The two part structure is important because if images are not found using an exhaustive search by means of the first algorithm then we must very quickly tell the user how to proceed. We could run a slightly modified version of the first algorithm to try to give us answers instead of the global optimisation algorithm but there are a number of issues with this. The global optimisation step gives us a different insight into the data set when compared to our initial sort and supplies us with revised threshold suggestions in a very simple and timely manner.

For our initial image sorting algorithm the time taken to sort the data increases roughly linearly with the size of the data set, this is shown for up to 5000 images in Figure 5-14.

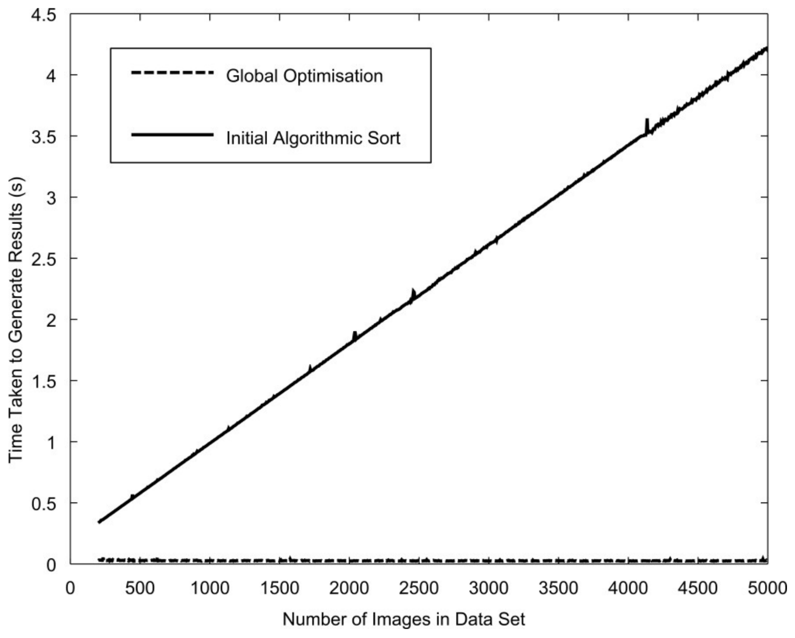


Figure 5-14 Global optimisation vs algorithmic sort, up to 5000 images

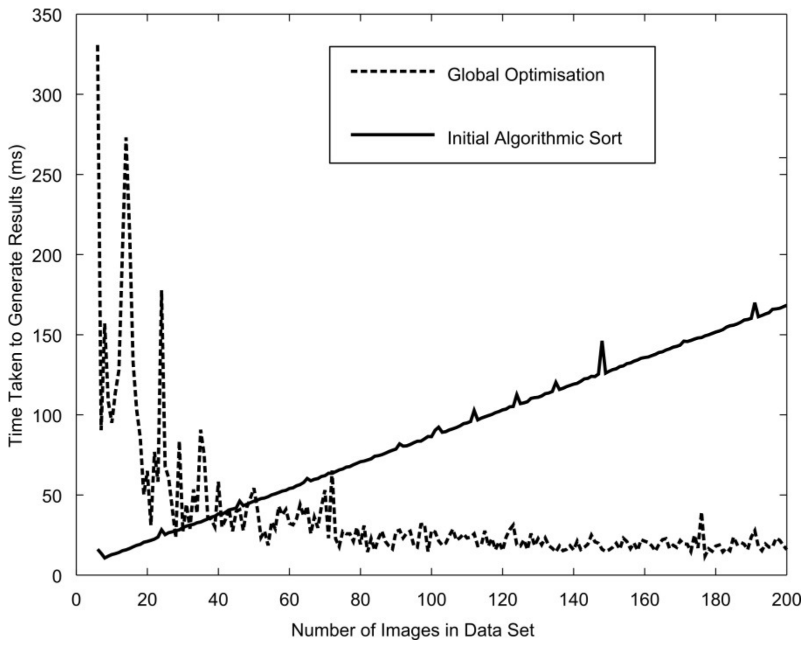


Figure 5-15 Global optimisation vs algorithmic sort, up to 200 images

The system can afford to take its time with the initial sorting as the user will expect this but to take the same time again just to make suggestions may force the user to wait for an unnecessarily long time depending on the size of the data set.

The time taken for the global optimisation step to compute is roughly constant for the datasets utilised in this case and of the order of 50ms as shown in figures 5-13 and 5-14. There is some increase in calculation time when performing the global optimisation step on very small datasets as shown in Figure 5-15. This is due to the algorithm being configured for the larger datasets it is much more likely to encounter.

As one of the aims is to have this system easily accessible by the end user the total response time of the system is important. It consists of the communication time between the client and the server and the time it takes to execute the algorithm or algorithms (depending on whether the first sort was successful).

Miller proposed [145] that there are 3 broad time frames to take into account when designing a user interface. If the response time is less than 0.1 seconds then the response is judged by the user to be immediate, if less than 1 second then the user will notice the delay but not require specific feedback to notify them of the delay, up to ten seconds the user will wait as long as adequate feedback is given but if the delay is over 10 seconds, whatever the feedback given the user will want to perform other tasks and will assume that the system has failed. As such we strive to keep our overall response time under 10 seconds and this is greatly helped by the fast response of the global optimisation step.

The time taken for the response to get back to the user is based both on the raw computational time needed as shown in figures 5-13 and 5-14 but also on the speed and more importantly latency of the users' internet connection. The client server structure allows all of the algorithm computation to be done on the server, thus putting very few hardware constraints on the user. The downside of this is that the requests and results must be sent to the server and this takes time. As the server acts as the link to the GMES MAS the full set of image data itself is not sent between client and server. Instead only the requests and the resulting images are transferred between the client and server. This means that the user will not require much bandwidth but the latency of the connection will dominate. Typical values of round trip latency are below 500ms [146]. The round trip latency however will be dwarfed by the time taken for the algorithm to run on the server in all but the most trivial of data sets.

5.10.7 Conclusions on the Image Selection Algorithm

In conclusion, an algorithm was developed to rank the images supplied by the GMES multi-agent system. This algorithm has been incorporated into a functional GUI that will allow users to easily set their desired image variables such as the desired area and hard thresholds such as the deadline for the image to be taken. The use of a global optimisation step allows the algorithm to quickly suggest changes to the user supplied thresholds in the event that the users desired images cannot be acquired. The algorithm is still in the early stages of development and much work can be done to further increase its functionality and speed. For instance both the simple sort and global optimisation steps can be optimised for the specific structure of the data as the data structure is set and relatively simple, this would further increase the speed of the algorithm. Different global optimisation techniques could be tested as they may offer increases in speed and accuracy over the current Lipschitzian Optimization. The prototype GUI is written in Java and presents a simple to understand web based interface to the user. Overall this work demonstrates that working algorithms and interfaces can be developed for complex multi-agent systems in order to hide that complexity from the user while still providing data or suggestions to the user rapidly.

5.11 Chapter Summary

In this chapter the GMES mission was introduced and the MAS developed with GMV outlined. This MAS was then converted into the HASA architecture for clarity. As the MAS was implemented by GMV an image selection algorithm as developed to complement this work and round out the system.

Chapter 6 Conclusions

6.1 Summary and findings of the thesis

This thesis addressed the hypothesis *"can multi-agent control enable more autonomous space missions?"*. This resulted in three main contributions. The first was the definition of a novel recursive multi-agent architecture for space missions named HASA. The second contribution was the design of a multi-agent control system based on the HASA architecture for the DARWIN mission as well as the design and implementation of a multi-agent simulation suite which was used to extensively test the formation flying capabilities of the DARWIN multi-agent system. The third was the design of a multi-agent control system for the GMES mission based on the HASA architecture which utilised both embodied agents and purely software agents. It also included a novel image selection algorithm.

In Chapter two the key concepts of autonomy, agents and agent architectures were introduced. The difference between deliberative and reactive architectures was defined and the idea that they can be combined in order to mitigate their respective shortcomings (in layered architectures for example) was discussed. The concept of distributed architectures was described along with how this approach addresses the needs of multi satellite space missions. The idea that a distributed architecture could also be recursive was discussed and this was taken as the starting point for the development of the HASA architecture in the following chapters. The design choices that need to be made were also outlined, specifically how knowledge, intelligence and decision making is distributed within a system. This led naturally on to the issue of how knowledge, intelligence and decisions are managed and communicated through the system and possible communication processes and architecture were outlined.

In Chapter two the basic concepts of a space based autonomous system are discussed. The ground station is where most of the current level of automation and autonomy is found and the difference between these current systems and an agent based approach would not be as great as in other mission components. Pure software agents, that is, agents with no physical basis, can also take over the data processing and product ordering tasks normally carried out for the missions and allow for parallel, robust processing of the data produced by the mission. An agent approach would also enable new ways of receiving requests and disseminating results to users, more fully described in Chapter five. The idea of replacing the monitoring and control components of traditional system with an autonomous agent based system is particularly applicable to formation flying missions where the control of all of the satellites from the ground may be prohibited by communication delays or the complexity and accuracy required by the particular mission. These ideas were further discussed in Chapter four with respect to the DARWIN mission and its formation flying requirements. The responsibilities of the attitude and orbital control system along with the failure detection, isolation and recovery components were discussed. These components would have to work in fundamentally different ways within a multi-agent system and monitor many more systems and the links between them.

In Chapter two the concept of agent system architectures, the way agents are organised and operate as a group, is introduced. Traditional agent system architectures are described including the blackboard, modular, layered and production architectures. The key concepts to be considered when designing a system architecture are then introduced. The difference between homoarchical (where the hierarchy of agents is fixed) and heterarchical (where the hierarchy of agents can change to suit the problems encountered) systems is outlined as well as the level of federation within a structure, that is the ability of a local group to solve a local problem.

In Chapter three the key concept introduced in this chapter is that of the holonic agent and system architecture. This is based on the idea that a system with many intermediate levels of abstraction leads more easily to a robust solution to a complex problem. This recursive nature allows the agents themselves and the system architecture to be defined in one unified language and is used to create the novel HASA architecture for distributed space missions. The ability of holonic agents to abstract away complexity allows each agent to have both physical and software agent components which makes it ideal for space missions where a common language for defining hardware and software components can be extremely useful.

The next part of Chapter three deals with the key problems which an agent system architecture must solve. Intelligent control and the difference between high level and low level control is discussed. The idea of designing with safety in mind from the outset is also put forward along with details of some applicable standards and discussion of safety in real time systems and which systems are critical to operation. The safety related themes of heterogeneous versus homogeneous redundancy were discussed as well as the verification and validation of a complex agent based control system.

The key contribution of this chapter is the description of HASA the holonic agent space architecture. Based upon the idea of multiple levels of abstraction allowing intermediate stages between agent architecture and agent system architecture while allowing all levels to be described using the same language. The HASA architecture is made up of 4 holons, units that can be combined or separated into more holons. They are the Executor, Planner, Resource and Product holons. More details are then given of how holons are aggregated to create the different levels of abstraction, how specialisation is handled and the data and functions of each holon.

Advantages of HASA for space applications

- Holonic architecture allows for the system to be described on many different levels of abstraction.
- Focused on the creation and maintenance of ‘products’ brings with it a production, safety and reliability centric view of multi-agent systems. Ideal for space applications.

- Holons specifically tasked with real time execution mean it can be used for complex control problems that are found in the space domain and not in the manufacturing domain.

Disadvantages of HASA for space applications

- Novelty of the architecture requires designers to rethink how their mission will be controlled.
- Has not yet been fully implemented for a space mission so no common modules to be reused.

Chapter four describes the development of a MAS for the DARWIN mission. The DARWIN MAS must be tested by using a physical simulation of the environment. This is due to the fact that the technical challenge in DARWIN is the fact that it is a formation flying mission with very tight constraints. The distributed nature of the hardware must also be utilised by any MAS and it was chosen because it would require a very specific structure in its multi-agent system. The ability of the HASA architecture to model a variety of missions control systems shows its versatility in the space domain.

The DARWIN mission itself comprises a number of space telescopes flying in formation, a central hub then collects the light collected by each telescope and through selective interferometry aims to stop the light from the star swamping the light of the exoplanet being imaged. Formation flying and formation control is covered and the MAS design for the DARWIN mission described. The design of the MAS consists of 5 different types of main agents, the planning agent, the formation flying command agent, the formation flying execution agent, the feedback agent and the negotiation agent. They are homoarchical on the spacecraft scale but on the formation scale the hierarchy can change to suit the needs of the mission as agents take turns in verifying others actions and certain craft take the lead. The MAS was then described using the HASA architecture for clarity. The issue of how to measure the performance of multi-agent systems is discussed and the outcome is that the best way to measure their performance is to simulate their operation in an environment as close to reality as possible. As such, a novel multi-agent testing suite was developed to see how the DARWIN MAS would operate in a number of operational scenarios. The simulation suite itself acts as an interface between the running MAS on its own hardware and models of the agents external environment, internal knowledge and their intelligent behaviours. Utilising these fundamental models in this way allows for them to be changed and compared at will without affecting the underlying structure of the MAS or rewriting the agent code. In all of the tests carried out on the DARWIN MAS using the multi-agent simulation suite the agents were running in Java and the simulation, internal knowledge and intelligence models were written in a combination of Matlab, C++ and Java. This ability to easily integrate models written in any language was the major innovation in the multi-agent simulation suite.

A number of test scenarios were developed to ascertain the performance and feasibility of the multi-agent formation flying design. All of the formation flying scenarios were carried out in predefined halo orbit at the second Lagrangian point. The dynamics of the system were modelled using a standard approach to modelling the circular restricted three body problem. The initial scenarios showed the station keeping requirements of a single satellite on the defined reference halo orbit. Other craft were then added to show how the relative distances between spacecraft changed over the course of an orbit. The collision avoidance mechanism of the MAS was then tested using scenarios where all of the craft used the CAM and a scenario where one of the crafts control system had failed and was not responding in the usual way. The primary formation that was tested was 12 satellites, one each on the vertices of an icosahedron. This formation had four satellites each on orthogonal planes and was complex enough to model the changes in formation and common problems found in formation flying missions. The underlying controller used in the formation flying scenarios was a proportional-integral-derivative feedback controller and this coupled with the information shared by each of the satellites was enough to hold the satellites in the icosahedron formation. A number of formation changes were also tested. A change in size of the formation in only one dimension was the simplest, followed by a change in formation size in all three dimensions. A change from the icosahedron formation to a ring was also tested as well as satellites changing positions within a formation without affecting the other satellites. Another simulation model was developed and while using the same equations of motion from the CRTBP it was implemented using the Matlab

Simulink tool which used a different interface to the test suite and allowed the use of Simulink specific tools, such as an automatic PID gain tuner. Another major benefit of this approach and this test suite is the user's ability to easily change intelligence models. To demonstrate this, the PID based avoidance algorithm was replaced with a model based on ORCA. This was implemented in a different language to that of the agents but as it used the standard interfaces that are part of the test suite this was not a problem. The ORCA based controller was tested using the same agents in the icosahedron and it was shown that it allowed for collision avoidance in otherwise dangerous scenarios. To demonstrate the scalability of the test suite another scenario was created involving 812 satellites all crossing through a single point, the test was successfully carried out and the results show that collision were avoided.

There are a number of steps that would be required in order to fully realise this architecture for a real DARWIN type mission. The recursive nature of the HASA architecture means that it is ideally suited to building up the system from sub-systems up. The logical place to start would be to fully implement the control aspects of the mission on a hardware test bed. It is much more likely that HASA subsystems will be flown before an entirely HASA space mission so the hardware test bed validated control system could be used on missions to prove the utility in the architecture.

In Chapter five the GMES mission is introduced. Special attention is paid to the services that GMES is designed to provide and the hardware and software components of the mission. The role of the customer is extremely important in this mission and the flow of customers requesting products and the mission delivering them is discussed. The hardware side of the mission is also discussed, the key issues being the heterogeneity of the constellation and the possibility for some satellites to have the capability for full autonomy. The possibilities offered by fully autonomous Earth observation satellites and how an autonomous science agent could operate, are outlined. What follows is the description of a multi-agent system for GMES and a definition of this MAS using the HASA architecture.

The MAS can be thought of as comprising four main components, a ground segment tasked with acquiring user requests and acting as a central negotiator, a multi-mission layer which acts as a proxy for the satellites combined, a mission layer which interacts with only one satellite and acts as its proxy during negotiations and finally the space segment which are the agents that reside on the satellites themselves. This proposed MAS for GMES is then implemented using the HASA architecture which illustrates how a complex MAS can be defined succinctly.

The final section of Chapter five describes the work on creating a novel image selection algorithm for GMES using global optimisation techniques. The algorithm itself is described and its key features are the fact that the user supplies a set of constraints with which to sort the images which can be generated by the constellation. If the constraints supplied by the customer return no images then a global optimisation step on the data set, which suggests changes to the customer's constraints, is performed. This relaxes the customer's constraints enough to return images while still maintaining their original intent. This global optimisation step returns results significantly faster than using conventional sorting algorithms on what would be an extremely large global dataset. Finally the user interface and client server infrastructure for using this algorithm is demonstrated with case studies showing its use in practice on a test dataset.

6.2 Fulfilled Objectives

The aim of this thesis was to see if the control of multi satellite space mission could be improved by the use of distributed multi-agent systems. This hypothesis was explored by the creation of a novel multi-agent architecture, HASA, which was then used to design multi-agent system for two contrasting multi satellite missions. The DARWIN MAS design had to be able to cope with the challenges presented by formation flying. In order to test the DARWIN MAS a multi-agent test suite was built and the MAS run through a series of increasingly challenging scenarios using a variety of simulation and intelligence models. The GMES MAS design needed to deal with complex user requests and did so using a novel image selection algorithm and interface. It also had to successfully bridge the gap between the ground and space segment and allow for a complex bidding and negotiation framework.

6.3 Limitations and Further Research

The clear next step for this work is to further implement both of the MAS designs and design MAS for other missions. This work would feedback valuable knowledge which can be used to improve the HASA architecture. As multi-agent systems are developed for more missions, be it using HASA or not, the more compatible intelligence and simulation models become available. This will help multi-agent systems make the leap from theory and technology demonstration missions to enabling wholly new types of mission that would be not be possible with existing technologies.

The creation of a set of standard HASA agents that are able to carry out most generic spacecraft functions would be worthwhile as this would allow more time to be dedicated to the tasks and subsystems that can benefit the most from more autonomy. In this work development effort had to be concentrated on a subset of possible MAS applications (autonomous order/request management and autonomous formation flying). One of the main limitations of this work was thus the fact that the full set of spacecraft sub-systems were not included in the simulations.

Larger scale simulations of both the GMES and DARWIN MAS would be of great benefit in both fleshing out the agents themselves and offering insights into the operation of these types of missions which will become more prevalent in the future.

Overall I believe this work best acts a foundation for future work on multi-agent systems and spacecraft autonomy by providing a flexible agent architecture, multi-agent system designs based on this architecture and the development of a multi-agent simulation suite specifically for space missions.

References

[1] M. A. Swartwout, “Engineering data summaries for space missions,” in *Proceedings of the IEEE Aerospace Conference*, 1998, vol. 2, pp. 391–401.

[2] J. Wyatt, R. Sherwood, M. Sue, and J. Szijjarto, “Flight validation of on-demand operations: the deep space one beacon monitor operations experiment,” *NASA JPL Report*, 1999.

[3] R. L. Ticker and D. McLennan, “NASA’s new millennium space technology 5 (ST5) project,” in *Proceedings of the IEEE Aerospace Conference*, 2000, vol. 7, pp. 609–617.

[4] A. S. Driesman, B. W. Ballard, D. E. Rodriguez, and S. J. Offenbacher, “STEREO observatory trade studies and resulting architecture,” in *Proceedings of the IEEE Aerospace Conference*, 2001, vol. 1, pp. 1–63.

[5] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, “NASA’s swarm missions: The challenge of building autonomous software,” *IT professional*, vol. 6, no. 5, pp. 47–52, 2004.

[6] J. J. Guzmán and A. Edery, “Mission design for the MMS tetrahedron formation,” in *Proceedings of the IEEE Aerospace Conference*, 2004, vol. 1, pp. 533–540.

[7] IBM, “An architectural blueprint for autonomic computing.,” *white paper*, 2004.

[8] S. Franklin and A. Graesser, “Is it an agent, or just a program? A taxonomy for autonomous agents,” in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Language*, 1996.

[9] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2009, p. 484.

[10] B. L. Tapscott, *Elementary Applied Symbolic Logic*. University Press of America, 1986, p. 532.

[11] D. Dennett, *The Intentional Stance*. MIT Press, 1989, p. 400.

[12] W. Brenner, R. Zarnekow, H. Wittig, and C. Schubert, *Intelligent Software Agents: Foundations and Applications*. Springer-Verlag, 1998, p. 326.

[13] P. Maes, “The agent network architecture (ANA),” *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 115–120, 1991.

[14] L. Strippoli and G. Radice, “Distributed agents for autonomy technical note 1: Identification of mission scenarios,” 2008.

[15] L. Strippoli, G. Radice, J. Ocon, and S. Grey, “Distributed agents for autonomy technical note 2: Identification of Distributed Agents Architecture and Selection of Reference Mission Scenario,” 2008.

[16] A. Cesta, J. Ocon, R. Rasconi, and A. Montero, “Simulating On-Board Autonomy in a Multi-Agent System with Planning and Scheduling,” in *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 2010, pp. 15–20.

[17] J. Ocon, E. Rivero, L. Strippoli, and M. Molina, “Agents for Space Operations,” in *Proceedings of the SpaceOps Conference, …*, 2008.

[18] J. Ocon, “DAFA - Distributed Agents for Autonomy - Final Report,” 2009.

[19] M. Genesereth and N. Nilsson, *Logical foundations of artificial intelligence*. Morgan Kaufmann, 1987.

[20] S. Russell and P. Norvig, “Artificial intelligence: a modern approach,” Pearson, 2009.

[21] C. Castelfranchi, F. Dignum, C. Jonker, and J. Treur, “Deliberative normative agents: Principles and architecture,” in *Proceedings of the 6th International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages*, 2000, pp. 364–378.

[22] H. Dreyfus, “From Socrates to expert systems The limits of calculative rationality,” *Technology in Society*, vol. 6, no. 3, pp. 217–233, 1984.

[23] M. Dastani, F. Dignum, and J. J. Meyer, “Autonomy and agent deliberation,” *Agents and Computational Autonomy*, pp. 114–127, 2003.

[24] E. H. Durfee and J. S. Rosenschein, “Distributed problem solving and multi-agent systems: Comparisons and examples,” *AAAI Technical Report*, 1994.

[25] M. E. Pollack and M. Ringuette, “Introducing the Tileworld: Experimentally evaluating agent architectures,” in *Proceedings of the eighth National conference on Artificial intelligence*, 1990, pp. 183–189.

[26] M. Wooldridge and N. R. Jennings, “Intelligent agents: Theory and practice,” *The knowledge engineering review*, vol. 10, no. 02, pp. 115-152, 2009.

[27] D. Kinny and M. Georgeff, “Commitment and effectiveness of situated agents,” in *Proceedings of the twelfth international joint conference on artificial intelligence*, 1991, pp. 82–88.

[28] R. A. Brooks, “Intelligence without representation,” *Artificial intelligence*, vol. 47, pp. 139–159, Oct. 1991.

[29] J. Müller, “The Design of Autonomous Agents: A Layered Approach,” *Lecture Notes in Artificial Intelligence*, vol. 1177, 1997.

[30] A. Giret and V. Botti, “Towards a recursive agent oriented methodology for large-scale MAS,” in *Agent-Oriented Software Engineering IV*, Springer, 2003, pp. 135–161.

[31] T. Finin, R. Fritzson, D. McKay, and R. McEntire, “KQML as an agent communication language,” *Proceedings of the third …*, pp. 456–463, 1994.

[32] Foundation for Intelligent Physical Agents, “FIPA ACL Message Structure Specification,” 2002.

[33] Foundation for Intelligent Physical Agents, “FIPA SL Content Language Specification,” 2002.

[34] Foundation for Intelligent Physical Agents, “FIPA KIF Content Language Specification,” 2003.

[35] R. Axelrod, *The complexity of cooperation: Agent-based models of conflict and cooperation*. Princeton University Press, 1997.

[36] P. Pasquier and B. Chaib-draa, “The cognitive coherence approach for agent communication pragmatics,” in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003, pp. 544–551.

[37] C. Castelfranchi, “Guarantees for autonomy in cognitive agent architecture,” in *Lecture Notes in Computer Science*, vol. 890, M. Wooldridge and N. R. Jennings, Eds. Springer, 1995, pp. 56–70.

[38] N. Chomsky, *Language and Problems of Knowledge: The Managua Lectures*. MIT Press, 1987.

[39] D. E. Bernard, E. B. Gamble, and N. F. Rouquette, “Remote Agent Experiment DS1 Technology Validation Report,” *NASA JPL Technical Report*, 2000.

[40] G. Rabideau, D. Tran, and S. Chien, “Mission Operations of Earth Observing-1 with Onboard Autonomy,” *NASA JPL Technical Report*, 2006.

[41] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, 2000.

[42] B. Hayes-Roth, K. Pfleger, and P. Lalanda, “A domain-specific software architecture for adaptive intelligent systems,” *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 288–301, Apr. 1995.

[43] T. Ishida, “Parallel, distributed and multi-agent production systems-A research foundation for distributed artificial intelligence,” in *Proceedings of the First International Conference on Multi-Agent Systems*, 1995, pp. 416–422.

[44] R. Brooks, “A robust layered control system for a mobile robot,” *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[45] P. H. Enslow, “What is a ‘Distributed’ Data Processing System?,” *Computer*, vol. 11, no. 1, pp. 13–21, Jan. 1978.

[46] T. Vamos, “Cooperative Systems- An Evolutionary Perspective,” *IEEE Control Systems Magazine*, vol. 3, no. 3, pp. 9–14, 1983.

[47] J. Hatvany, “Intelligence and cooperation in heterarchic manufacturing systems,” *Robotics and Computer-Integrated Manufacturing*, vol. 2, no. 2, pp. 101–104, 1985.

[48] R. G. Smith, “The Contract Net Protocol : High-Level Communication and Control in a Distributed Problem Solver,” *IEEE Transactions on computers*, vol. C–29, no. 12, pp. 1104–1113, 1980.

[49] M. Genesereth and S. Ketchpel, “Software agents,” *Stanford University*, 1994.

[50] J. G. Mcguire, D. R. Kuokka, and J. C. Weber, “SHADE : Technology for Knowledge-Based Collaborative Engineering,” *Journal of Concurrent Engineering: Applications and Research*, vol. 3, pp. 1–17, 1993.

[51] A. M. Sanchez-Montero, “DAFA Architecture Design Document,” 2008.

[52] “SeSam - Multi-Agent Simulation Environment.” [Online]. Available: http://www.simsesam.de/. [Accessed: 25-Apr-2012].

[53] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, “An architecture for autonomy,” *The International Journal of Robotics Research*, vol. 17, no. 4, p. 315, 1998.

[54] BBN-Technologies, “Cougaar Architecture Document,” 2004.

[55] N. Criado, E. Argente, and V. Botti, “A BDI architecture for normative decision making,” *… of the 9th International Conference on …*, pp. 1383–1384, 2010.

[56] V. Hilaire, A. Koukam, and S. Rodriguez, “An adaptative agent architecture for holonic multi-agent systems,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3, no. 1, pp. 1–24, Mar. 2008.

[57] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, “The CLARAty architecture for robotic autonomy,” *In Aerospace Conference*, vol. pages, pp. 121–132, 2001.

[58] M. Ghallab, F. Ingrand, S. Lemai, and F. Py, “Architecture and tools for autonomy in space,” *ISAIRAS, Montreal*, 2001.

[59] G. Verfaillie and M. C. Charmeau, “A generic modular architecture for the control of an autonomous spacecraft,” in *5th International Workshop on Planning and Scheduling For Space. USA, Baltimore*, 2006.

[60] M. Sierhuis and W. Clancey, “NASA’s OCA Mirroring System An application of multiagent systems in Mission Control,” … *Agents and Multi Agent* …, pp. 85–92, 2009.

[61] P. Leitao and F. Restivo, “ADACOR: A holonic architecture for agile and adaptive manufacturing control,” *Computers in Industry*, vol. 57, no. 2, pp. 121–130, Feb. 2006.

[62] H. Van Brussel, J. Wyns, and P. Valckenaers, “Reference architecture for holonic manufacturing systems: PROSA,” *Computers in industry*, vol. 37, no. 3, pp. 255–274, 1998.

[63] A. Koestler, *The ghost in the machine*. Macmillan, 1968.

[64] H. Simon, *Sciences of the Artificial*. MIT Press, 1996.

[65] M. Fletcher, E. Garcia-Herreros, and J. H. Christensen, “An open architecture for holonic cooperation and autonomy,” *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, pp. 224–230, 2000.

[66] S. M. Deen, *Agent-based manufacturing: advances in the holonic approach*. Springer, 2003.

[67] S. Balasubramanian and R. Brennan, “An architecture for metamorphic control of holonic manufacturing systems,” *Computers in Industry*, vol. 46, no. 1, pp. 13–31, Aug. 2001.

[68] A. Labeyrie, H. Coroller, and J. Dejonghe, “Luciola hypertelescope space observatory: versatile, upgradable high-resolution imaging, from stars to deep-field cosmology,” *Experimental Astronomy*, vol. 23, no. 1, pp. 463-490, Sep. 2008.

[69] J. Kramer, “Dynamic configuration for distributed systems,” *IEEE Transactions on Software Engineering*, no. 4, pp. 424–436, 1985.

[70] N. Leveson, *System safety engineering: Back to the future*. MIT Press, 2002.

[71] S. Grey, G. Radice, M. Vasile, and Q. Wijnands, “Design of a Multi-Agent System for Cost Reduction in Multi-Craft Space Missions,” in *Proceedings of the 60th International Astronautical Congress*, 2009, pp. 7072–7078.

[72] G. Lund and H. Bonnet, “Darwin-the Infrared Space Interferometer,” *Comptes Rendus de l’Académie des Sciences - Series IV - Physics*, vol. 2, no. 1, pp. 137–148, 2001.

[73] M. Ollivier, J. M. Mariotti, A. Léger, P. Sékulic, J. Brunaud, and G. Michel, “Nulling interferometry for the DARWIN space mission,” *Comptes Rendus de l’Académie des Sciences - Series IV - Physics*, vol. 2, no. 1, pp. 149–156, 2001.

[74] A. Léger, “Strategies for remote detection of life–DARWIN-IRSI and TPF missions-,” *Advances in Space Research*, vol. 25, no. 11, pp. 2209–2223, 2000.

[75] M. Hechler, M. Mora, M. Sancheznogales, and A. Yanez, “Orbit concepts at L2 for Soyuz launches from Kourou,” *Acta Astronautica*, vol. 62, no. 2–3, pp. 140–150, Jan. 2008.

[76] R. N. Bracewell, “Detecting nonsolar planets by spinning infrared interferometer,” *Nature*, vol. 274, pp. 780–781, 1978.

[77] D. Savransky, N. J. Kasdin, and R. J. Vanderbei, “An evaluation of the effects of non-uniform exo-zodiacal dust distributions on planetary observations,” *Proceedings of SPIE, Techniques and Instrumentation for Detection of Exoplanets IV*, vol. 7440, 2009.

[78] M. Ollivier, “Towards the spectroscopic analysis of Earthlike planets: the DARWIN/TPF project,” *Comptes Rendus Physique*, vol. 8, no. 3–4, pp. 408–414, Apr. 2007.

[79] J. Bermyn, “PROBA-project for on-board autonomy,” *Air & Space Europe*, vol. 2, no. 1, pp. 70–76, 2000.

[80] “ESA - Proba Missions - About Proba 3.” [Online]. Available: http://www.esa.int/SPECIALS/Proba/SEMG2R4PVFG_0.html.

[81] W. Ren, “On Consensus Algorithms for Double-Integrator Dynamics,” *IEEE Transactions on Automatic Control*, vol. 53, no. 6, pp. 1503–1509, Jul. 2008.

[82] R. W. Beard and E. M. Atkins, “A survey of consensus problems in multi-agent coordination,” *Proceedings of the American Control Conference*, pp. 1859–1864, 2005.

[83] J. Marshall and M. Brouke, “A pursuit strategy for wheeled-vehicle formations,” in *Proceedings of the IEEE conference on Decision and Control*, 2003, pp. 2555–2560.

[84] V. Gupta, “Stability analysis of stochastically varying formations of dynamic agents,” in *Proceedings of the IEEE Conference on Decision and Control*, 2003, pp. 504–509.

[85] J. R. Lawton and R. W. Beard, “Synchronized multiple spacecraft rotations,” *Automatica*, vol. 38, no. 8, pp. 1359–1364, 2002.

[86] P. Gurfil and E. Kivelevitch, “Flock properties effect on task assignment and formation flying of cooperating unmanned aerial vehicles,” *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 221, no. 3, pp. 401–416, Jan. 2007.

[87] C. V. M. Fridlund, “Darwin-the infrared space interferometry mission,” *ESA Bulletin 103*, pp. 20–63, 2000.

[88] A. Altman, A. Procaccia, and M. Tennenholtz, “Nonmanipulable selections from a tournament,” in *Proceedings of the international joint conference on Artifical intelligence*, 2009, pp. 27–32.

[89] P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra, “Multimode control attacks on elections,” in *Proceedings of the international joint conference on artifical intelligence*, 2009, pp. 128–133.

[90] A. Procaccia, “Thou shalt covet thy neighbor’s cake,” in *Proceedings of the international joint conference on artificial intelligence*, 2009, pp. 239–244.

[91] H. Hexmoor, “Autonomy in Spacecraft Software Architecture,” in *Proceedings of the international FLAIRS conference*, 1999, pp. 69–72.

[92] S. Boccaletti, C. Grebogi, and Y. Lai, “The control of chaos: theory and applications,” *Physics Reports*, vol. 329, pp. 103–197, 2000.

[93] J. Klein, “Breve: a 3d environment for the simulation of decentralized systems and artificial life,” *Proceedings of the 8th International Conference on the Simulation and Synthesis of Living Systems*, p. 329, 2003.

[94] S. Luke, “MASON: A Multiagent Simulation Environment,” *Simulation: Transactions of the society for Modeling and Simulation*, vol. 81, no. 7, pp. 517–527, Jul. 2005.

[95] N. Minar, R. Burkhart, and C. Langton, “The swarm simulation system: A toolkit for building multi-agent simulations,” *Santa Fe Institute Working Paper*, pp. 1–11, 1996.

[96] G. Sohl and J. L. Kellogg, “Distributed Simulation for Formation Flying Applications,” *NASA JPL Technical Report*, 2005.

[97] S. Grey and G. Radice, “Design and Testing of an Autonomous Multi-Agent Based Spacecraft Controller,” in *Proceedings of the 61st International Astronautical Congress*, 2010.

[98] R. Werninghaus, “The TerraSAR-X Mission,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 2, pp. 606–614, 2006.

[99] G. Krieger, A. Moreira, and H. Fiedler, “TanDEM-X: A Satellite Formation for High-Resolution SAR Interferometry,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 11, pp. 3317–3341, 2007.

[100] H. Hellman, S. Persson, and B. Larsson, “PRISMA – a formation flying mission on the lanchpad,” in *Proceedings of the International Astronautical Congress*, 2009.

[101] D. P. Scharf, F. Y. Hadaegh, and S. R. Ploen, “A survey of spacecraft formation flying guidance and control (part II): control,” *Proceedings of the American Control Conference*, pp. 2976–2985, 2004.

[102] K. Thanapalan and S. M. Veres, “Agent Based Controller for Satellite Formation Flying,” in *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2005, pp. 385–389.

[103] R. W. Beard, J. Lawton, and F. Y. Hadaegh, “A coordination architecture for spacecraft formation control,” *IEEE Transactions on Control Systems Technology*, vol. 9, no. 6, pp. 777–790, 2001.

[104] R. Olfati-saber and R. M. Murray, “Distributed cooperative control of multiple vehicle formations using structural potential functions,” *Proceedings of the World Congress of the International Federation of Automatic Control*, 2002.

[105] E. M. C. Kong, “Spacecraft formation flight exploiting potential fields,” Massachusetts Institute of Technology, 2002.

[106] F. E. Schneider and D. Wildermuth, “A potential field based approach to multi robot formation navigation,” *Proceedings of the IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, pp. 680–685, 2003.

[107] A. Pereira and L. Hsu, “Adaptive formation control using artificial potentials for Euler-Lagrange,” *Proceedings of the World Congress of the International Federation of Automatic Control*, pp. 10788–10793, 2008.

[108] D. J. Bennet and C. R. Mcinnes, “Pattern transition in spacecraft formation flying via the artificial potential field method and bifurcation theory,” *Proceedings of the International Symposium on Formation Flying, Missions and Technologies*, 2008.

[109] G. Wie, *Space Vehicle Dynamics and Control*. AIAA, 1998, p. 661.

[110] C. Simo, G. Gomez, and J. Llibre, “On the optimal station keeping control of halo orbits,” *Acta Astronautica*, vol. 15, no. 6–7, pp. 391–397, Jun. 1987.

[111] G. Gómez, M. Lo, J. Masdemont, and K. Museth, “Simulation of Formation Flight Near L2 for the TPF Mission,” *NASA Technical Report*, 2001.

[112] J. D. M. James, “Celestial Mechanics Notes Set 5: Symmetric Periodic Orbits of the Circular Restricted Three Body Problem and their Stable and Unstable Manifolds.” 2006.

[113] G. Gómez, À. Jorba, C. Simó, and J. Masdemont, *Dynamics And Mission Design Near Libration Points, Volume III: Advanced Methods for Collinear Points*. World Scientific, 2001.

[114] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, “JADE, a white paper,” *Telecom Italia report*, 2003.

[115] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-agent Systems with JADE*. Wiley-Blackwell, 2007.

[116] D. J. Wilkie, P. M. Johnson, and D. R. Katebi, *Control Engineering*. Palgrave Macmillan, 2001, pp. 277–311.

[117] J. R. Dormand, “A family of embedded Runge-Kutta formulae,” *Journal of computational and applied mathematics*, vol. 6, no. 1, pp. 19–26, Mar. 1980.

[118] L. Shampine and M. Gordon, *Computer solution of ordinary differential equations: the initial value problem*. W.H.Freeman & Co Ltd, 1975.

[119] J. Van Den Berg, S. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Proceedings of the International Symposium of Robot research*, 2009.

[120] E. Oks and M. Sharir, “Minkowski Sums of Monotone and General Simple Polygons,” *Discrete & Computational Geometry*, vol. 35, no. 2, pp. 223–240, Dec. 2005.

[121] “GMES Official Site,” 2011. [Online]. Available: http://www.gmes.info/. [Accessed: 13-Jan-2011].

[122] J. Schmetz, P. Pili, and S. Tjemkes, “An Introduction to Meteosat Second Generation (MSG),” *Bulletin of the American Meteorological Society*, vol. 83, no. 7, pp. 991–991, Jul. 2002.

[123] Y. Desnos, C. Buck, and J. Guijarro, “The ENVISAT advanced synthetic aperture radar system,” in *Proceedings of the international geoscience and remote sensing symposium*, 2000, no. 39, pp. 991–991.

[124] J. Aschbacher and M. P. Milagro-Pérez, “The European Earth monitoring (GMES) programme: Status and perspectives,” *Remote Sensing of Environment*, vol. 120, no. 2012, pp. 3–8, May 2012.

[125] P. F. Levelt and R. Noordhoek, “CAMELOT Final Report Issue 1,” *ESA*, 2009.

[126] H. J. Kramer, *Observation of the Earth and Its Environment: Survey of Missions and Sensors*. Springer, 2001.

[127] M. C. Hansen and T. R. Loveland, “A review of large area monitoring of land cover change using Landsat data,” *Remote Sensing of Environment*, vol. 122, pp. 66–74, Feb. 2012.

[128] “Charter on cooperation to achieve the coordinated use of space facilities in the event of natural or technological disaster.” [Online]. Available: http://www.disasterscharter.org/web/charter/charter.

[129] Astrium, “Astrium GEO-Information Services SPOT International Price List,” 2012.

[130] VEGA, “EOLI-SA 9.1.0 - User Guide,” 2011.

[131] “CREPAD: centro de recepción, proceso, archivo y distribución.” [Online]. Available: http://crepadweb.cec.inta.es/en/index-en.html. [Accessed: 03-Apr-2012].

[132] “EO Portal.” [Online]. Available: http://services.eoportal.org. [Accessed: 05-May-2012].

[133] S. Chien, R. Sherwood, and D. Tran, “The EO-1 Autonomous Science Agent,” in *Proceedings of the international conference on autonomous agents and multi-agent systems*, 2004.

[134] D. Tran, S. Chien, and R. Sherwood, “The autonomous sciencecraft experiment onboard the EO-1 spacecraft,” in *Proceedings of the conference on artifical intelligence*, 2004.

[135] R. Sherwood, S. Chien, D. Tran, and B. Cichy, “Intelligent systems in space: the EO-1 Autonomous Sciencecraft,” *NASA JPL Technical Report*, 2005.

[136] S. Grey, G. Radice, M. Vasile, and Q. Wijnands, “Image selection algorithm for GMES mission,” in *Proceedings of the 60th International Astronautical Congress*, 2009, pp. 2382–2387.

[137] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, “Lipschitzian optimization without the Lipschitz constant,” *Journal of optimization theory and application*, vol. 79, no. 1, pp. 157–181, 1993.

[138] Y. Brise, “Lipschitzian Optimization , DIRECT Algorithm and Applications,” *Author’s Presentation*, 2008.

[139] C. Audet and J. E. Dennis, “Analysis of Generalized Pattern Searches,” *SIAM Journal on Optimization*, vol. 13, no. 3, p. 889, 2002.

[140] “Google Web Toolkit.” [Online]. Available: http://code.google.com/webtoolkit/. [Accessed: 24-Mar-2009].

[141] N. Athanasis, K. Kalabokidis, M. Vaitis, and N. Soulakellis, “Towards a semantics-based approach in the development of geographic portals,” *Computers & Geosciences*, vol. 35, no. 2, pp. 301–308, Feb. 2009.

[142] D. Al-Khudhairy, “Geo-spatial information and technologies in support of EU crisis management,” *International Journal of Digital Earth*, vol. 3, no. 1, pp. 16–30, Mar. 2010.

[143] J. . Bessis, J. Bequignon, and A. Mahmood, “Three typical examples of activation of the International Charter ‘space and major disasters’,” *Advances in Space Research*, vol. 33, no. 3, pp. 244–248, 2004.

[144] A. Ito, “Issues in the implementation of the International Charter on Space and Major Disasters,” *Space Policy*, vol. 21, no. 2, pp. 141–149, May 2005.

[145] R. B. Miller, “Response time in man-computer conversational transactions,” *Proceedings of the fall joint computer conference*, pp. 267–277, 1968.

[146] M. E. Crovella and R. L. Carter, “Dynamic Server Selection in the Internet,” in *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, 1995.

[147] G. Calzolari, T. Beck, and Y. Doat, “From the EMS concept to operations: First usage of automated planning and scheduling at ESOC,” in *Proceedings of SpaceOps conference*, 2008.

[148] J. Spall, *Introduction to Stochastic Search and Optimization: Estimation*. Wiley, 2003.

Appendices

Appendix A - Traditional Spacecraft Control Structure

Ground station component

This component implements the interface between the ground and the space segments. It provides monitoring data for both systems and allows the user to execute any given plan or activity on the system. The ground station component also provides tracking data and computes ground station availability. It requires data on the antenna used for tracking as well as plans and activities from the users. Its tasks include the planning and execution of ground based activities, uplink and downlink from the space segment, processing and pre-processing of data and data archival. The traditional approach such as the European Space Agency's (ESA) ESTRACK Management System (EMS) [147] calculates the availability of different resources and schedules them accordingly. This component can benefit from an autonomous agent approach by increasing its ability to dynamically negotiate between components of the ground station network.

Data Processing Component

The data processing component processes all of the mission data, generates the mission products and then distributes these products to the users of the system. The services it provides are the mission products themselves and interfaces to allow for the generation of bespoke products to users specifications. The component requires monitoring data to allow it to create the relevant products with the relevant guarantees and is tasked with executing the appropriate processing algorithms to create the right product, calibration and quality control for these processing chains and consolidating and archiving the generated products. The traditional approach for this component is to have linear string processing steps, each step taking the output of the previous step as its input, newer systems instead use a data driven approach and the processing step is only carried out when all of its input data is available from all of its supplier components. The data processing steps are computationally very intensive and as such are usually distributed to a number of nodes within a larger computing cluster. The agent approach to the data processing component offers the possibility of increased performance due to dynamic resource utilisation and cooperation between agents. For example the ability of an agent to move within a heterogeneous computing environment will allow the data processing to dynamically move to where most computation resources can be found and this would greatly increase resource utilisation. Agents could also be involved at a higher level, for instance in the dynamic creation of new products for users by combining existing processing actions available to the agents.

Flight Dynamics Component

The flight dynamics component's main functionality is concerned with orbit determination, orbit control, attitude determination and attitude control. It provides the facility to execute actions singly or as part of some larger orbital determination procedure. It can make predictions of variables and states based on its current position and historical data. It can execute manoeuvres and manipulate manoeuvre data and plans. It also provides flight dynamics data such as current orbit, current attitude, alerts for significant events and data concerned with tracking and monitoring of the spacecraft to ensure communications. The flight dynamics component requires monitoring data of many of the other subsystems to aid in making the correct manoeuvres or acquiring the correct orbit. It requires tracking data from the two way communication link as, in order to achieve a given orbit or position, it must have reliable data on its current position. The component requires sensor data both when a manoeuvre is being executed and to confirm data and readings from the ground. The flight dynamics component must also be privy to the long term plan for the mission and the current operational plan in order to successfully meet its requirements. In order to successfully operate the flight dynamics agent must carry out a wide range of tasks. These include the determination of its current orbit and attitude both with help from the ground and without. It must also be able to predict a future orbit and compare orbital possibilities. It must also be able to predict any future events that may arise from a change in orbit or from the following the operational plan. The flight dynamics components must also generate its own plans and activities to acquire the desired orbit or position and also be able to generate the required flight dynamics data that may be required by other components.

In the traditional approach the orbital and attitude of the spacecraft is controlled through a combination of ground based and autonomous control on board the spacecraft. The flight dynamics system (FDS) is tasked with orbital determination and control with the on-board attitude and orbital control system (AOCS) executing these commands and monitoring their progress. The on-board AOCS is more autonomous than the ground based orbit determination and control. The FDS is charged with producing a range of data products which are used by many components including the AOCS such as restituted and predicted orbit and attitude.

When designing a future autonomous agent based flight dynamics components the logical conclusion would be to have a fully autonomous AOCS coupled with extra components that would allow the spacecraft to autonomously carry out on-board those tasks currently done on the ground. As we have seen before a more gradual introduction of autonomy is desirable and as such it can be envisaged that the services currently provided by the ground segment will continue to be provided by the ground. Instead of the current ground systems there will be a number of ground based agents who can interact more closely with the agents present on the spacecraft. The flight dynamics agents would have to closely work with mission planning, monitoring and control agents in order to operate effectively. In both the fully autonomous agent case and the component replacement case the flight dynamics agent would be given goals from the mission planner. As a precursor level to the ground based agents the current ground based components, instead of being replaced could just have some agent-like capability added. This would allow them to be more tightly integrated with the system and benefit from increased responsiveness and interlinking with the other agents.

Monitoring and Control Component

The Monitoring and control agent's primary task is that of controlling the mission and in order to do this it must also fully monitor the current status of the mission. This component works at a higher level than other executive components and can be thought of as taking the global view. This component also has the important task of supplying the mission control interface to the users to allow for ground based control. The monitoring and control component requires access to all of the services tasked with executing activities and monitoring activities in order to give it a complete view of the current state of the mission and to be able to make changes to this state as it sees fit. It must also have access to the operational plan and will use this plan to make decisions that it deems necessary in order to the complete the said plan. At a lower level the monitoring and control component must monitor individual space system elements and define their activities including any parameters needed for execution. It must also carry out pre-execution validation steps for the desired commands and procedures and then execute these activities based on the schedule supplied from the plan or on an ad hoc basis when certain events occur. Finally it must check that each action is successfully executed by the correct component.

The traditional monitoring and control component is there to support the telecommunication and telecommand loops between the ground based operations and the spacecraft. Some adaptation is possible in the activities to be carried out but any changes to the operations are usually carried out by the planning component. The traditional monitoring and control component does have some autonomy in carrying out closed loop actions and scheduling.

A fully autonomous agent based approach would see major changes to monitoring and control component as all of the activities currently carried out would instead be goals within the agent society and monitoring and control would be an inherent feature of the agent organisation rather than an external component. In a more conservative approach instead of replacing the monitoring and control components outright the level of automation and the number of automated procedures could be increased and agent layers of abilities be added in order to foster more efficient and reliable communication between the disparate components and any agents present in the system.

Mission planning component

The mission planning components main functions are to generate a mission plan and then execute this mission plan. The only outputs of this component are the mission plans it creates and the commands required to enable the plan to be executed. The mission planning component takes in planning requests as its inputs. These planning requests can originate from a number of sources, most commonly from the end user or client but also from sources internal to the system such as orbit determination, flight dynamics, etc. In order to create a useful plan the mission planning component requires planning data. This data will take the form of task descriptions linked with priorities, dependencies and costs in both time and other resources. The mission planning component also needs to have data regarding the successful completion or not of any task in order to re-plan or proceed with the current plan or re-plan. There are number of different plans that can be produced to enable the smooth and successful running of the space mission. The long term plan must be generated based on mission planning policies and this plan serves as a high level framework for the more detailed plans that will be produced later. Lower level plans such as operational plans must then be produced to allow the meeting of the higher levels plans objectives. The operational plans must bring together the required activities, their execution constraints and co-dependencies into a conflict free whole. The mission planning component must also be able to re-plan at any stage and this requires the execution of the plan to be closely monitored for any failures or divergences. The plans at all levels must also be monitored and controlled so that they continue to lead the mission within its constraints and performance envelope.

In the traditional approach the mission planning component undertakes all of the tasks related to processing the plan inputs and requests for action, creating plans for on-board and ground activities as well as resolving any conflicts that may arise in the system. The mission planning component must process a large amount of data including user requests, ground station visibility and data pertaining to the flight dynamics modules and the current state of the spacecraft and mission as a whole. The plan is executed either from the mission planning component directly or from the component tasked with carrying out the specific task. The two main components charged with executing the plan are the ground station component and the monitoring and control component. The data processing component tends to be more event driven and as such is less frequently called upon for execution by the plan.

The distributed autonomous agent approach has a radically different approach to the centralised structure in the traditional mission planning module. In the multi-agent system planning and the associated tasks of replanning are distributed throughout the system and instead of being generated and corrected by one component it treats planning as a societal problem and agents and components cooperate to generate the plans needed for the mission.

Ordering component

The ordering components primary function is to transmit the data orders or requests from the user to the system. The ordering component bridges the gap between the user and the mission. It must generate and communicate data on the current service level of the system including data availability and system availability. It must also deliver mission data to the users when requested and interface with external systems in order to publish the data for other users of systems. The ordering component also provides a data request plan as well as the results themselves from the mission planning component. The requirements of the ordering component are limited only to the mission data of the mission so that it can offer the correct products to the users and of course it needs inputs from users or external systems. The ordering component extends the processing capabilities of the data processing component and takes the processed data and creates products for the end users. Once these end products have been created the ordering component is also tasked with generating and maintaining the requisite data catalogues to store the products for future use.

In the traditional architecture the ordering component takes all of the user requests and passes them to the mission planner. It also determines what constraints are to be put on any given request and what products can be offered at any given time. The ordering component may have to bring together data from a number of sources including external sources to create the given project.

There is large scope for improvement when using a distributed autonomous agent approach for taking the duties of the ordering component. The main advantage would be that agents could negotiate on behalf of users with the system in order to better and more quickly provide the users their desired products. This approach could help to improve overall efficiency and the agents could be constrained to meet any relevant service level agreements or quality of service metrics. The ability of agents to replicate, be mobile and cooperate also offers the scope to increase service availability and further optimise resource allocation.

Sensor Web Component

The sensor web component is charged with providing the data from the systems sensors as a service to other components. As well as the data itself the sensor web component must also provide availability and sensor capability data. The sensor data must be supplied to its user components in the appropriate form be it real-time, near real-time or archived/retrieved data. The sensor web might also require some processing capability in order to give the other components the data in a format they can utilise. The sensor web component must also allow for the change in sensors in such that sensors may be added or removed for the system during development. Sensor data may also be required to be published in its relatively raw state to users or to the ground for checking of the system. As such the sensor web component requires data from its sensors, interfaces to allow for the operation and configuration of the sensors and monitoring data from other components.

The sensor web component must also determine the capabilities of all the sensors that are under its purview as well as undertaking simple data processing tasks that are required to provide any data correlation and data fusion services. This processing must also include some level of data quality analysis. This metric is required by other components that will use the sensor data so they can make informed decisions based on the perceived accuracy of the data they are given. In order to give components access to retrieved data as well as real-time and near real-time data the sensor web must instigate its own archiving procedure of the raw or processed sensor data, whichever is more appropriate, for the components that require it.

The sensor web component varies greatly from mission to mission and forms a key component in a system such as Global Monitoring for Environment and Security (GMES) which will be looked at in more detail in subsequent sections. The traditional approach for a GMES type mission where the aim is to bring multiple heterogeneous satellite sensing platforms together is to provide the sensor data through different channels depending on whether the data is real-time, near-real-time or retrieved.

The sensor web also provides multiple different mechanisms for users to access the data such as immediate data provision, subscription based mechanisms, sending out notification to users and allowing for data search and retrieval. In the traditional approach the sensor web component also supplies services that allow users to access data directly on the current service level which includes the current capabilities and characteristics of any sensors as well as service availability for the users and data availability which includes what data is being acquired at any given time and in the immediate future. There must also be services that allow for new sensors and platforms to be added to the system.

A distributed autonomous agent approach is a very good fit for the requirements of the sensor web component. It allows for an easy interface between the external user and system and a multi-agent system comes with inherent abilities to manage availability and the flow of information that is key to the operation of this component. This will be covered in much more detail where we outline our design for a multi-agent system in later chapters.

Mission and Vehicle Management Component

The mission and vehicle management (MVM) component's main function is to command any spacecraft and spacecraft subsystems according to the available mission plan, whether it is uploading from an external source or generated on the spacecraft. The MVM must also have the ability to trigger a number of subsystems such as payload and communications.

The MVM must provide commands to the AOCS in order to modify the spacecraft's physical state as well as supplying commands to the payload and communication subsystems in order to carry out the mission tasks. To do this the MVM needs an uploaded plan and access to the components charged with failure detection and monitoring in order to be able to trigger any contingency plan. Its typical tasks involve reading the plan uploaded from the ground or generated on-board and then parsing this and sending the appropriate commands to the AOCS, payload and communication components.

In the traditional approach, the plan is generated on the ground and then uploaded to the MVM. These plans are usually static and cannot be adapted once they have been uploaded. Often the planning is based on static look up tables that contain directions on which guidance navigation and control, communication or payload modes are to be triggered for any given event or on pre-defined schedule. There is usually a number of such look up tables for the different mission phases as well as tables for contingency operations.

In the distributed autonomous agent approach the MVM will be entirely replaced. The use of static plans and look up tables can be replaced by a dynamic set of autonomous agents which will enhance the capability of this component. Examples of the agents that could be used are:

Spacecraft manager agent: This agent is a high level deliberative agent and the main decisional component of the spacecraft control system. It provides plans to the other agents, notably the AOCS agent and the payload agents based on high level goals sent from the ground. The plan is dynamic and this allows the agent to re-plan in the case of an unexpected event or error in the plan's execution. The plan can also be updated autonomously if new goals are received from the ground or the scientific component in mid-execution. The spacecraft manager agent will also negotiate with any communication agents to try to optimise the data downlink and uplink process. In more complex systems the agent can also interact with other spacecraft manager agents on other spacecraft or multiple ground sites to try to optimise the overall mission plan for the good of the collective. This agent will also have a close relationship with the failure detection, isolation and recovery (FDIR) agent in order to manage any emergency situations and will also have access to the reactive component of the AOCS agent the same reason.

On board science agent: This agent monitors the payload of the spacecraft system and autonomously detects unplanned scientific events. In the event of a new science event being detected a goal is formulated, for example to observe the phenomenon, and this is then sent to the spacecraft manager agent. This agent will obviously be of most utility in scientific observation missions but may also allow for mission to gather scientific data outwith their original observational scope if the opportunity arises and they are able.

Data downlink agent: This agent is concerned with downloading data to the ground, as such it will generate a plan to downlink as much of its data as possible based on various priorities and then negotiate this plan with the spacecraft manager agent as this plan may well conflict with the requirements of the spacecraft as a whole. This method could help to improve the quality and response time of the data being downloaded by having a dedicated agent generating plans.

Resource manager agent: This agent builds a picture of all the available resources of the spacecraft system as whole and negotiates the use of these resources with the spacecraft manager agent in order to produce valid plans based on actual availability of any spacecraft resources.

Attitude and Orbit Control System (AOCS) component

The main functionality of the AOCS is to send commands to the actuators of the spacecraft in order to follow an orbital and attitude profile as supplied by the spacecraft plan. It may also have secondary functionality such as computing optimal attitude for communications and for proper alignment of the solar panels. In order to operate effectively the AOCS must have a number of inputs. It requires measurements from the spacecraft sensors in order to ascertain its position and current state. It must also have guidance navigation and control data such as the current GNC mode and other parameters. This is so that it knows what orbit and attitude it should try to achieve. In order for the spacecraft to recover from any failure the AOCS must have access to and understand the commands coming from the FDIR so that recovery actions can be undertaken.

The AOCS must know the spacecraft's current state in order to operate effectively and this is derived from the spacecraft's sensor data. Once this state is known, the AOCS must generate a mission profile which meets the requirements of the mission plan and is feasible given its current requirements. The AOCS must also be able to detect possible hazards to the space craft and when the hazard involves a possible collision be able to take direct action to avoid or mitigate the hazard.

In the traditional approach the AOCS is frequently non-autonomous. In this case all of the AOCS tasks are carried out on the ground and then uploaded for execution by the spacecraft. Thus the current state of the spacecraft is deduced on the ground from sensor readings and the appropriate mission profile is uploaded in order to make the spacecraft maintain or acquire the desired states/orbit. The attitude guidance system has been a fertile area for the development of autonomy and it is fairly common to have an autonomous guidance system that works in conjunction with target profiles supplied from the ground. Having both autonomous attitude guidance and autonomous orbital navigation has proved more difficult due the complexity of the orbital guidance algorithm and the lack of on-board knowledge of the spacecraft's full current state.

In fully autonomous missions the area with a lot to gain is the AOCS. In fully autonomous systems the guidance system autonomously generates an orbital profile and manoeuvres which adhere to the mission profile. The current attitude and trajectory are autonomously estimated based on the current sensor readings. The AOCS is then charged with generating commands which the actuators use to follow the reference profiles generated on the spacecraft. The collision and hazard avoidance abilities of the AOCS are also autonomous in many of today's systems and are able to compute their own navigation solution to avoid the hazard autonomously. .

Using autonomous agents for the AOCS would allow for improvements over the traditional systems by having a closer relationship with the spacecraft manager agent. An autonomous agent based AOCS will be able to transform the high level plan it receives and execute it robustly. The plan will be decomposed into steps and its execution monitored in real-time within the agent community. This close interaction between the planning and control could allow for more flexible operation, allowing the spacecraft to react more effectively and quicker to unknown events.

FDIR Component

The FDIR is tasked with monitoring all of the different sub-systems and components within the spacecraft and then detecting, isolating and recovering from these failures either by instigating action via some other system or triggering the use of redundant systems. If a failure that cannot be resolved satisfactorily is detected then the FDIR must be able to instruct the spacecraft to adopt a "safe mode", this is achieved by close contact with the mission vehicle manager component.

The FDIR must be able to provide commands to all of the components or system it monitors in order to effect a change, this includes commands to instigate any safe mode protocols. To do this it needs accurate status data for all of the monitored components and systems. This status data must be analysed and conclusions drawn about the health of any given system or component, even if it does not itself know it has failed. For cases such as these the FDIR must have the ability to forcibly shut down a unit and replace it with a redundant system, this gives the FDIR a lot of power within the system and any errors in the FDIR could lead to a total system shut-down. Likewise should FDIR send a safe mode command in error this will greatly reduce the effectiveness of the spacecraft to complete its mission as it may take a significant amount of time to recover from the safe mode.

The traditional approach to FDIR is to monitor the space segment and make any decisions on the ground. Sensor data and system state information is collected on the ground and analysed to check for any errors and then changes uploaded to the spacecraft to correct these errors. More autonomous FDIR systems have been developed and deployed and are present in many modern spacecraft. They have reactive components that allow for quicker and more decisive action when faced with certain types of failure. The trend is towards increasingly more sophisticated and autonomous FDIR systems which enable the spacecraft to monitor and analyse more subsystems and thus detect and recover from more failures.

An autonomous agent based approach is logical progression of this trend. An agent based system allows for robust communication and negotiation between the FDIR and the agents reporting systems status and this can reduce the time taken to find faults and then recover from them. In the multi-agent case the FDIR would take little action directly, instead it is envisaged that it would receive the state of the systems under its domain and then negotiate with the spacecraft control agent to undertake the specified recovery actions. In this case recovery options that would conflict with current mission parameters could be weighed up by the mission control agent and executed if prudent.

Appendix B – Code snippets

Below is part of a formation change instruction which is used in the testing of the multi-agent system. It shows the type of information which is needed by all of the agents to successfully change formation.

Formation change format example

```
<?xml version="1.0"?>
<!-- Formation definition for 12 craft in a ring in x-y plane
centred on reference orbit number 20 -->
<!-- Author: Stuart Grey -->
<formation>
 <formation_name>"Ring100km"</formation_name>
 <reference_orbit>L2Halo0020</reference_orbit>
 <number_of_craft>12</number_of_craft>

       <craft id=dw00>
             <desired_x_offset_km>0.00000<desired_x_offset_km>

 <desired_y_offset_km>100.00000</desired_y_offset_km>
             <desired_z_offset_km>0.00000</desired_z_offset_km>
             <neighbour>dw12<neighbour>
             <neighbour>dw01<neighbour>
             <child></child>
             <parent></parent>
       </craft>

       <craft id=dw01>
             <desired_x_offset_km>50.00000<desired_x_offset_km>

 <desired_y_offset_km>86.66025</desired_y_offset_km>
             <desired_z_offset_km>0.00000</desired_z_offset_km>
             <neighbour>dw00<neighbour>
             <neighbour>dw02<neighbour>
             <child></child>
             <parent></parent>
       </craft>

       <craft id=dw02>
```

```
                <desired_x_offset_km>86.66025<desired_x_offset_km>

 <desired_y_offset_km>50.00000</desired_y_offset_km>
                <desired_z_offset_km>0.00000</desired_z_offset_km>
                <neighbour>dw01<neighbour>
                <neighbour>dw03<neighbour>
                <child></child>
                <parent></parent>
          </craft>

          <craft id=dw03>

 <desired_x_offset_km>100.00000<desired_x_offset_km>
                <desired_y_offset_km>0.00000</desired_y_offset_km>
                <desired_z_offset_km>0.00000</desired_z_offset_km>
                <neighbour>dw02<neighbour>
                <neighbour>dw04<neighbour>
                <child></child>
                <parent></parent>
          </craft>
....
```

Agent declaration

This example shows how the individual agents are defined and gives examples of agent 'behaviours'. These behaviours are used to allow the agents to perform tasks. The first behaviour is used to create other agents within the simulation. In the simulation suite a single reference agent is created which then in turn creates the other agents based on a given scenario. This also demonstrates the ability of agents to create sub-agents or peers on demand.

Example Agent Declaration Code

```
/*Import required libraries to enable the agent to run on the
JADE platform import jade.core.Agent; etc etc*/

// Declare agents class
public class ReferenceAgentIcosahedron4 extends Agent
```

```
{
    //Set up initial values for variables
    Boolean ReceivedPrevious = false;
    int numberofagents = 13;
    int iterationnumber = 1;
    int totaliterations = 100;
    String messagecontent;
    int agentnumber = 0;

    //Start a connection between this JADE platform and Matlab
    MatlabClient connection = new MatlabClient();
    String result = "null";

    //Set up initial conditions for the reference orbit
    BigDecimal refX = new BigDecimal("1.008420601516730");
    BigDecimal refY = new BigDecimal("0.0");
    BigDecimal refZ = new BigDecimal("-0.00028");
    BigDecimal refXdot = new BigDecimal("0.0");
    BigDecimal refYdot = new BigDecimal("0.009835862759924");
    BigDecimal refZdot = new BigDecimal("0.0");

/*The setup() method is run when the agent is successfully
created and is used to describe what behaviours the agent is to
have.  In this case the agent's main tasks are to create the
other agents required for the simulation and and then listen in
to all of the communication between the agents and write out a
communication log to a data file.*/

        protected void setup()
        {

/*The CreateAgents behaviour is defined elsewhere but is here
attached as a behaviour to our reference agent.  It creates all
of the agents required by the simulation.  Automatically
creating the required agents for the scenarion reduces the
effort required to set up and change scenarios.*/

        addBehaviour( new CreateAgents() );

/*The EavesDropper behaviour repeats every 20 milliseconds and
records position data from all of the craft in the simulation*/

        addBehaviour( new EavesDropper(this, 20));
/*The log file is set up and title lines etc added.  The
eavesdropper
behaviour adds data to this file as it receives it.*/

    String file_name1 = "log.txt";
    final FileWrite referencedata = new FileWrite( file_name1 ,
false );
        try {
            referencedata.writeToFile("%reftest");
        }
        catch (IOException e)
        {
            System.out.println( e.getMessage());
        }
}
```

Communication between agents and Matlab

This example shows how the agents communicate with Matlab. This is done by passing of a defined set of strings to a given network port. This allows for variables to be passed back and forth and Matlab methods to be called. This example shows how the values for the satellites position and velocity are passed to Matlab to be propagated using its numerical integration routines. A key benefit of this string passing approach is that it is platform independent and the simple interface can be written to connect the agent simulation with any external resource for simulation, modelling or agent intelligence.

Communication between Matlab and the Agents

```
void propagatecraft(){

    try{
/*Using the connection to Matlab (connection) send the current
state of the craft to the method PropagationModel, assign the
resulting string to result.*/

result = connection.createJob(
 "PropagationModel" + " " + tempX + " " + tempY + " " + temp +
" " + tempXdot + " " + tempYdot + " " + tempZdot
);

/*Set up appropriate counter variables and arrays to hold the
series of strings after tokenization and the numerical values
of these strings*/

    int index=0; int tokenCount;
    String words[]=new String [100];
    BigDecimal numbers[] = new BigDecimal[100];

/*Tokenize the string, that is break up the single large string
returned by the Matlab method into the string array.*/

    String message=result;
    StringTokenizer st=new StringTokenizer(message);
    tokenCount=st.countTokens();
    while (st.hasMoreTokens())
        {words[index]=st.nextToken(); index++;}

/*For each element in the array os trings convert it into a
BigDecimal and place in the BigDecmimal array.*/

    for (index=0;index<tokenCount; index++)
    {
        numbers[index] = new BigDecimal(words[index]);
    }

/*Update the values for the crafts position and velocity using
the values computed in matlab */

    tempX = numbers[0];
    tempY = numbers[1];
      tempZ = numbers[2];
    tempXdot = numbers[3];
    tempYdot = numbers[4];
    tempZdot = numbers[5];

        }
        catch( Exception e ) { System.err.println( e ); }
}
```

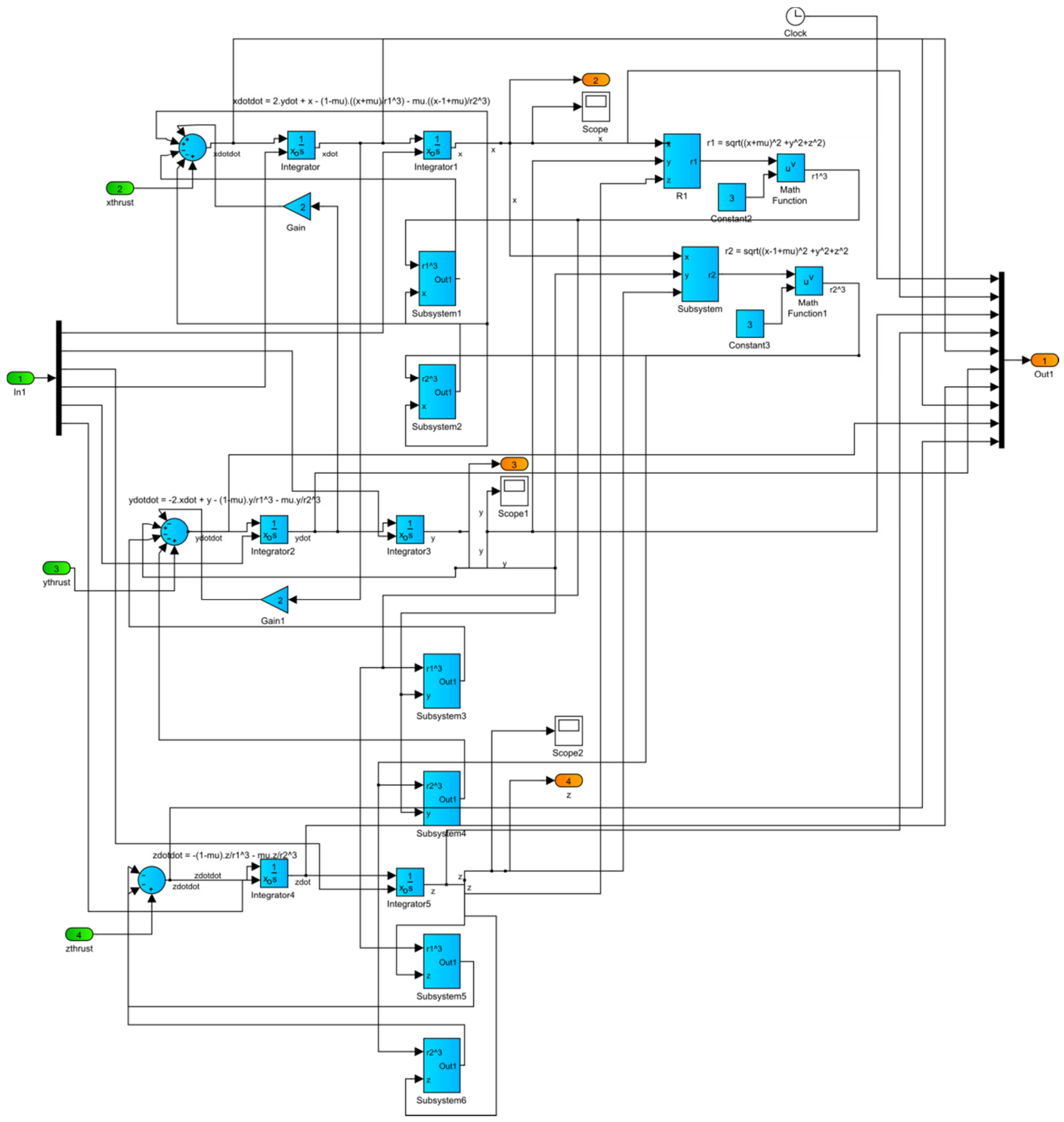
Matlab function that can be called by agents

This example shows the other side of the communication between the agent simulation and Matlab. This Matlab function is called by the simulation for each satellite trajectory that is to be propagated.

Communication between Agent and Simulation

```
function [output] = PropagationModel(a,b,c,d,e,f)

G=1;
GM_sun=1.327*10^(11);
GM_earth=4.053*10^(5);
mu=GM_earth/(GM_sun+GM_earth);
period = 3.102523281056765;
timeinterval = period/10000;

referencex0=[a b c d e f];

options=odeset('RelTol',2.5e-14,'AbsTol',1e-22);
[t,reference]=ode113('CRTBP',[0:timeinterval/5:timeinterval],re
ferencex0,options,[],G,mu);
result=reference(end,1:6);

s0=result;

x = sprintf('%0.15g',s0(1));
y = sprintf('%0.15g',s0(2));
z = sprintf('%0.15g',s0(3));
xdot = sprintf('%0.15g',s0(4));
ydot = sprintf('%0.15g',s0(5));
zdot = sprintf('%0.15g',s0(6));

global orbit;
global counter;
orbit(1,counter) = s0(1);
orbit(2,counter) = s0(2);
orbit(3,counter) = s0(3);
orbit(4,counter) = s0(4);
orbit(5,counter) = s0(5);
orbit(6,counter) = s0(6);
counter = counter+1

class  = 'java.lang.String';
text = [x ' ' y ' ' z ' ' xdot ' ' ydot ' ' zdot ];

output = javaObject(class, text);
```

Appendix C – Simulink dynamic model



Simulink block diagram showing the CRTBP equations of motion implemented using Simulink rather than written in Matlab code or Java.

Appendix D – Global optimisation methods

DIRECT

The DIRECT algorithm search works by subdividing the parameter hypercube into sub-rectangles and searching iteratively within those. The sub-algorithm for dividing any subsequent sub rectangles of the hypercube is described below:

Step 1: Identify the set I of dimensions with the maximum side length. Let ∂ equal one-third of this maximum side length.

Step 2: Sample the function at the points $c \pm \partial e_i$ for all $i \in I$, where c is the centre of the rectangle and e_i is the ith unit vector.

Step 3: Divide the rectangle containing c into thirds along the dimensions in I, starting with the dimension with the lowest value of $w_i = min\{f(c + \partial e_i), f(c - \partial e_i)\}$, and continuing to the dimension with the highest w_i.

The multivariate DIRECT algorithm can then be described as follows:

Step 1: Normalize the search space to be the unit hypercube. Let c_1 be the centre point of this hypercube and evaluate $f(c_1)$. Set $f_{min} = f(c_1)$, $m = 1$ and $t = 0$ (the iteration counter).

Step 2: Identify the set S of potentially optimal rectangles.

Step 3: Select any rectangle $j \in S$.

Step 4: Using the sub-algorithm described above determine where to sample within rectangle j and how to divide the rectangle into sub-rectangles. Update f_{min} and set $m = m + \Delta m$, where Δm is the number of new points sampled.

Step 5: Set $S = S - \{j\}$. If $S \neq \emptyset$ got to step 3.

Step 6: Set $t = t + 1$. If $t = T$, then stop; The iteration limit has been reached.

Simulated annealing

The simulated annealing algorithm implemented is as follows [148]:

Initialization: Set the initial temperature T and initial parameter vector $\hat{\theta}_0 = \theta_{curr} \in \Theta$; determine $L(\theta_{curr})$.

Step 1: Relative to the current value θ_{curr}, randomly determine anew value of θ,$\theta_{new} \in \Theta$ and determine $L(\theta_{new})$.

Step 2: Compare the two L values above using the metropolis criterion (0-1). Let $\delta = L(\theta_{new}) - L(\theta_{curr})$. If $\delta < 0$, accept θ_{new}. Alternatively, if $\delta \geq 0$, accept δ_{new} only if a uniform (0,1) random variable U satisfies $U \leq exp[-\delta/(c_b T)]$. If θ_{new} is accepted then θ_{curr} is replaced by θ_{new}. Otherwise θ_{curr} remains.

Step 3: Repeat steps 1 and 2 for some period until either the budget of function evaluations allocated for T has been used or the system reaches a state of equilibrium.

Step 4: Lower T according to the annealing schedule and return to step 1. Continue the process until the total budget for function evaluations has been used or some indication of convergence is satisfied. The final estimate is $\hat{\theta}_n$ (taken as the most recent θ_{curr}), representing the θ value after n iterations ($= n + 1$ loss evaluations).

Metropolis criterion:

$$\exp(-\frac{\varepsilon_{\text{new}} - \varepsilon_{curr}}{c_b T}) \qquad (0\text{-}1)$$

Where ε_{curr} is the current energy state of the system. ε_{new} is the new energy state of the system, c_b is the Boltzmann constant and T is the temperature of the system.

Genetic Algorithms

The genetic algorithm implemented was as follows [148]:

Initialisation: Randomly generate a population of N chromosomes and evaluate the fitness function (an inverted $L(\theta)$) for each of the chromosomes.

Step 1: (Parent Selection) Select the parents from the population. Those parents with a higher fitness based on their chromosomes are selected more often.

Step 2: (Crossover) For each pair of parents identified in step 1, perform crossover on the parents at randomly selected splice points with a probability P_c. If no crossover takes place then form two offspring that are exact copies of the parents.

Step 3: (Replacement and mutation) Replace the parent population with the offspring population. Perform a mutation on an element of the chromosome with probability P_m.

Step 4: (Fitness and end test) Compute fitness values for the new population of N chromosomes. Terminate the algorithm if the stopping criterion is met or if the budget of fitness function evaluations is exhausted; otherwise return to step 1.