



Software architectures for cloud robotics : the 5 view Hyperactive Transaction Meta-Model (HTM5)

Vineet Nagrath

► To cite this version:

Vineet Nagrath. Software architectures for cloud robotics : the 5 view Hyperactive Transaction Meta-Model (HTM5). Other. Université de Bourgogne; Université de Technologie de Malaisie, 2015. English. NNT : 2015DIJOS005 . tel-01202747

HAL Id: tel-01202747

<https://theses.hal.science/tel-01202747>

Submitted on 21 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



SPIM

Thèse de Doctorat



école doctorale **sciences pour l'ingénieur et microtechniques**
U N I V E R S I T É D E B O U R G O G N E

Software Architectures for Cloud Robotics

The 5 View Hyperactive Transaction Meta-Model (HTM5)



VINEET NAGRATH

SPIM

Thèse de Doctorat



école doctorale sciences pour l'ingénieur et microtechniques
UNIVERSITÉ DE BOURGOGNE

N° X X X

THÈSE présentée par

VINEET NAGRATH

pour obtenir le

Grade de Docteur de
l'Université de Bourgogne

Spécialité : **Informatique**

Software Architectures for Cloud Robotics

The 5 View Hyperactive Transaction Meta-Model (HTM5)

Soutenue publiquement le 15 January 2015 devant le Jury composé de :

TOMAS MAUL	Rapporteur	Professeur à l'University of Nottingham
FADZIL BIN HASSAN MOHD	Examineur	Professeur à l'Universiti Teknologi Petronas
OLIVIER MOREL	Co-directeur	Professeur à l'Université de Bourgogne
GILLES GESQUIERE	Examineur	Professeur à l'Université de Lyon
BRUNO SADEG	Rapporteur	Professeur à l'Université du Havre
FABRICE MERIAUDEAU	Directeur de thèse	Professeur à l'Université de Bourgogne

Software Architectures for Cloud Robotics

*The 5 View Hyperactive Transaction Meta-Model
(HTM5)*

A dissertation presented to the
Université De Bourgogne, FRANCE and
**Universiti Teknologi Petronas,
MALAYSIA** in fulfilment of the
requirements for the degree of
DOCTOR OF PHILOSOPHY

By:

NAGRATH, Vineet

Advisors:

Prof. Fabrice Meriaudeau

Dr. Aamir Saeed Malik

Dr. M Naufal B M Saad

Dr. Olivier Morel

**Le Creusot, FRANCE
January 2015**

Le Creusot, FRANCE
January 2015

This was a cotutelle (Double/Joint) doctoral study conducted at
Université De Bourgogne, FRANCE (Home)
and
Universiti Teknologi Petronas, MALAYSIA (Host)
during October 2011 and January 2015.

Le Creusot, FRANCE
January 2015

Le Creusot, FRANCE
January 2015

ABSTRACT

Software Architectures for Cloud Robotics

The 5 View Hyperactive Transaction Meta-Model (HTM5)

Software development for cloud connected robotic systems is a complex software engineering endeavour. These systems are often an amalgamation of one or more robotic platforms, standalone computers, mobile devices, server banks, virtual machines, cameras, network elements and ambient intelligence. An agent oriented approach represents robots and other auxiliary systems as agents in the system. The concept of agency preserves the autonomy on individual agents, which is essential in implementing cloud computing business logic in a commercial cloud robotic system. To enable flexible implementation of such systems we have introduced hyperactivity mechanism to selectively release an agent's autonomy to its associated agents.

Software development for distributed and diverse systems like cloud robotic systems require special software modelling processes and tools. Model driven software development for such complex systems will increase flexibility, reusability, cost effectiveness and overall quality of the end product. Currently there is no industry oriented meta-model for agent oriented development of cloud robotic systems. In this thesis we present a complete meta-model framework catering to all stakeholders in a cloud robotics ecosystem. The proposed 5-view meta-model has separate meta-models for specifying structure, relationships, trade, system behaviour and hyperactivity in a cloud robotic system. The thesis describes the anatomy of the 5-view Hyperactive Transaction Meta-Model (HTM5) in computation independent, platform independent and platform specific layers. The thesis also describes a domain specific language for computation independent modelling in HTM5.

Automatic transformations from the domain specific language to lower layer components are also explained with examples. The Model-to-Text transformations generate a Java class hierarchy for platform independent and platform specific components of the system. Model-to-Model transformation from HTM5 to UML is also explained in the thesis. Multiple case studies regarding the design and implementation of a cloud based multi-robot systems are also presented in the Thesis. The case studies present use cases where dynamic trade decisions are modelled using HTM5, validating feasibility of the proposed meta-model.

**Le Creusot, FRANCE
January 2015**

The thesis therefore has presented a complete meta-model for agent oriented cloud robotic systems and has several simulated and real experiment-projects justifying HTM5 as a feasible meta-model.

Le Creusot, FRANCE
January 2015

Acknowledgements

I would like to thank my supervisors Prof. Fabrice Meriaudeau at Université De Bourgogne, FRANCE and Dr. Aamir Saeed Malik at Universiti Teknologi Petronas, MALAYSIA for the opportunity to conduct my PHD studies in the two universities in a wonderfully managed cotutelle doctoral program. I would also like to thank my co-supervisors Dr. M Naufal B M Saad and Dr. Olivier Morel for their help and guidance during these years.

I thank my advisors for the freedom and unconditional support that I received. You were the coolest advisors any PHD student could have wished for. I hope my work do justice to the free hand given to me in design and implementation of my ideas. Your quick and constant assistance in bureaucratic affairs at the two universities and visa consulates saved me a lot of troubles, special thanks for that. I would also like to thank the examiners and reviewers of my thesis as their comments have helped me produce a better manuscript.

I would like to thank the colleagues and staff at Le2i, IUT, Condorcet and CISIR for making these years a memorable experience. I would like to thank the Bourgogne regional council for their financial support for my research. A special thanks to the people of Le Creusot. I am sure I will turn to Le Creusot time and again for the peace and happiness it has brought to me. I regret that I could not reciprocate by mastering the French language, but Le Creusot will remain in my memories as my village. People all around Malaysia will be remembered for their multicultural, multilingual and familiar culture. I would like to thank the monks and brothers of Taizé Community for letting an atheistic traveller into their world and rest for a while. The numerous friends and moments at Taizé will stay with me forever.

No words will be enough to thank Mom, Dad and Puneet for their support. Thanks to friends, uncles, aunts, cousins in India and Cheers to friends and nameless strangers who helped me on the streets. Thanks everyone ☺

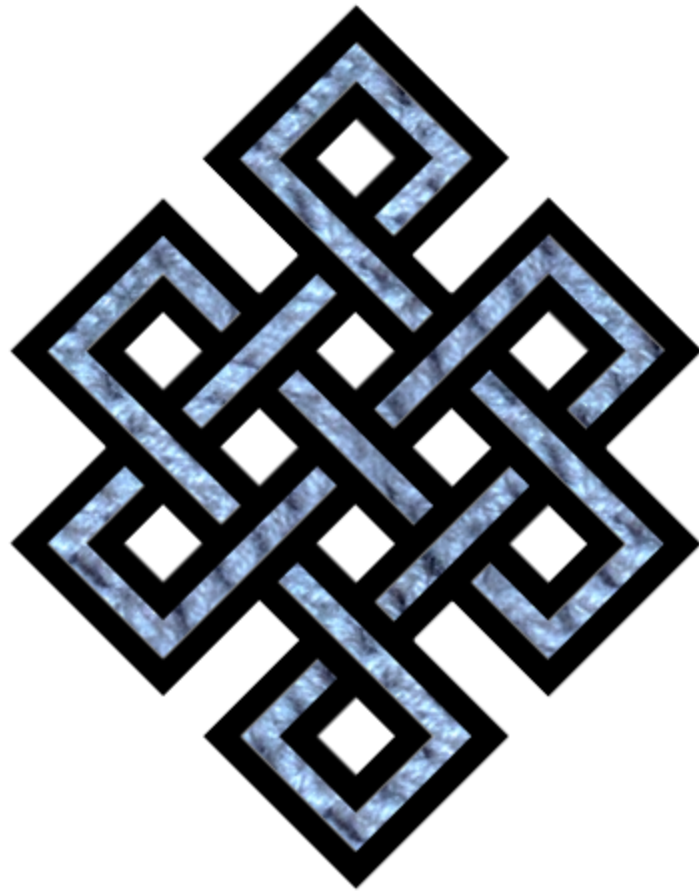
Le Creusot, FRANCE
January 2015

Le Creusot, FRANCE
January 2015

Dedicated to Teachers...

**Le Creusot, FRANCE
January 2015**

Le Creusot, FRANCE
January 2015



As Above, So Below
As Within, So Without

“Logic is immaturity weaving its nets of gossamer wherewith it aims to catch the behemoth of knowledge. Logic is a crutch for the cripple, but a burden for the swift of foot and a greater burden still for the wise.”

-The Book of Mirdad

Contents

1	Introduction	29
1.1	Overview	29
1.2	Objectives	35
1.3	Contributions	36
1.4	Outline of the Thesis	36
2	Background Information	39
2.1	The Cloud	39
2.1.1	Cloud Computing	39
2.1.2	The Cloud Business Model	40
2.1.3	Cloud Computing In Robotics	40
2.1.4	Cloud Robotics Ecosystem	43
2.2	Model Driven Engineering and Architecture	44
2.2.1	Component Based Software Engineering	44
2.2.2	Models	44
2.2.3	Meta-Models	44
2.2.4	Multi-View Modelling	45
2.2.5	OMG Model Driven Architecture	45
2.3	Agents and Cloud Robotics	45
2.3.1	Agent and Agency	45
2.3.2	Agent Metaphor and usage	46
2.3.3	Multi Agent Systems	46
2.3.4	Agent oriented cloud robotic systems	47
2.4	Hyperactivity	48
2.4.1	The Mechanism	49
2.4.2	Flexible Agency	51
2.4.3	Agent Classification in HTM5	52
3	5 View Hyperactive Transaction Meta Model	55
3.1	HTM5 Sub Models	56
3.1.1	HTM5 Computation Independent Model	56
3.1.2	HTM5 Platform Independent Model	58
3.1.3	HTM5 Platform Specific Model	59
3.1.4	HTM5 Machine Descriptor Model	60
3.2	HTM5 Views	62
3.2.1	HTM5 Structural View	62
3.2.2	HTM5 Relational View	63
3.2.3	HTM5 Trade View	64

3.2.4	HTM5 Hyperactivity View	65
3.2.5	HTM5 Behavioural View	67
3.2.6	Functionalities in PIC*, PIC and PSC Component Classes	69
3.3	HTM5 Agent Relation Charts (ARCs)	73
3.3.1	Agent Relation Charts (ARC)	74
3.3.2	Trade - Agent Relation Charts (T-ARC)	75
3.3.3	Hyperactivity - Agent Relation Charts (H-ARC)	75
3.3.4	Use Case - Agent Relation Charts (U-ARC)	75
3.3.5	Sequence - Agent Relation Charts (S-ARC)	76
3.4	HTM5 P2P	93
3.5	HTM5 Component	95
3.6	The HTM5 Domain Specific Language (HTM5-DSL)	96
3.7	HTM5 Domain Specific Language and Automated Model Trans- formations	97
3.8	DSL development	103
3.9	System development life-cycle	107
4	Case Study Projects	111
4.1	NAO Telerobotics	111
4.1.1	The Initial Ideas	114
4.1.2	Creation of CIM model using Agent Relation Charts	114
4.1.3	Encoding the CIM model in HTM5-DSL	115
4.1.4	Creation of PIM layer UML model for various compo- nents	116
4.1.5	Execution of DSL code to produce Java and UML com- ponent classes	119
4.1.6	Refinement of component classes	120
4.1.7	Independent development of Components respective Ma- chines	120
4.1.8	Component and system testing	124
4.2	Dynamic Electronic Institutions	125
4.2.1	Dynamic Electronic Institutions	125
4.2.2	Dynamic Electronic Institutions in Cloud Robotics	127
4.2.3	Dynamic Electronic Institutions in HTM5	128
4.2.4	The Experiments	134
4.2.5	Conclusion and future direction	138
4.3	Peer-to-Peer Cloud Trade Ecosystem	139
4.3.1	Peer-to-Peer Trade modelling in HTM5	144
4.3.2	Case Study Experiments	145
4.3.3	Conclusion	152
4.4	Industrial and Research Feasibility	153
4.5	A summary of all Feasibility and Viability studies	162
5	Conclusions	171
	Appendices	185

A	Full page version of intricate images.	187
B	A Detailed explanation for Object Miner project	205
C	Cloud Robotic Workshop at UTP	251

List of Figures

1.1	Elements of a cloud robotic system	30
1.2	Model Driven Architecture proposed by Object Management Group (OMG-MDA)	31
1.3	Schematic representation of the 3-View Component Meta-Model (V3CMM) views and relationships that exist among them . .	33
2.1	Comparison of popular approaches towards cloud robotics and cloud connected devices.	42
2.2	Agent Oriented Cloud Robotics environment that implements a Digital Business Ecosystem (DBE)	47
2.3	A conceptual representation of an HTM5 agent and its Hyperactivity mechanism	49
2.4	Classification of HTM5 components based on their functionality and level of Agency.	52
3.1	An anatomical overview of the 5 views Hyperactive Transaction Meta Model (HTM5)	56
3.2	An anatomical description of the 5 views Hyperactive Transaction Meta Model (HTM5)	57
3.3	Separation of PIM and PSM parts of HTM5-MDM	60
3.4	An ARC diagram specifying Structural and Relational elements of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.4 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i>	78
3.5	A Normalized version of the ARC diagram shown in Fig. 3.4	79
3.6	A Trade-ARC diagram specifying Trade elements of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.6 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i>	81
3.7	A Hyperactivity-ARC diagram specifying Hyperactivity elements of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.8 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i>	84
3.8	A Use Case-ARC diagram specifying Use Case 'Trader is Allocated Miners' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i> .	87

3.9	A Use Case-ARC diagram specifying Use Case 'Mining Process' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i>	88
3.10	A Sequence-ARC diagram specifying Use Case 'Trader is Allocated Miners' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i>	89
3.11	A Sequence-ARC diagram specifying Use Case 'Mining Process' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). <i>To Readers: Objects are referenced in the thesis text using 2 axis references.</i>	90
3.12	Graphical representation of a section of HTM5-DSL grammar specification for 'Component' keyword.	99
3.13	Excerpts of code from HTM5-DSL Grammar specification in Xtext and Model Transformations in Xtend	100
3.14	Excerpt of code written in HTM5-DSL for the example project 'Object Miner'	101
3.15	automated 'Model-to-Text' and 'Model-to-Model' transformations from the HTM5-DSL code presented in Fig. 3.14	102
4.1	Structural Overview of 'Telerobotics On Nao' System.	113
4.2	ARC and Normalized ARC Diagrams for 'Telerobotics On Nao' System.	115
4.3	TARC, HARC, UARC002 and SARC005 Diagrams for 'Telerobotics On Nao' System.	117
4.4	Excerpt of code written in HTM5-DSL for 'Telerobotics On Nao' System	118
4.5	Excerpt of UML Class Components generated by execution of HTM5-DSL code (Model-to-Model Transformation).	119
4.6	Excerpt of Component classes developed independently at various Machines in 'Telerobotics On Nao' System (Part 1)	121
4.7	Excerpt of Component classes developed independently at various Machines in 'Telerobotics On Nao' System (Part 2)	122
4.8	3F Life cycle of a Dynamic Electronic Institution. The three phases are in order: Formation, Foundation and Fulfilment. Re-Formation and Re-Foundation processes are also within the 3F life cycle.	126
4.9	Example of Digital Business Ecosystem (DBE) in an Agent Oriented Cloud Robotics environment	131
4.10	An example of Dynamic Electronic Institutions implemented using HTM5 methodology	132
4.11	Instants from Dynamic Electronic Institutions based Digital Business Ecosystem simulations	133
4.12	Case Study Experiments	135
4.13	Results of Case Study Experiments	136
4.14	Cloud entities in the "Mine Cloud" cloud robotic system	141

4.15	ARCs specifying the structural and relational aspects of "Mine Cloud" cloud robotic system	142
4.16	Trade-Agent Relation Chart (T-ARC) for the "Mine Cloud" cloud robotic system	143
4.17	A set of screen shots from the simulated experiments conducted on the "Mine Cloud" cloud robotic system	148
4.18	The physical robot colony of 5 TurtleBOT robots that was used to implement a scaled down version of the "Mine Cloud" case study	149
4.19	Test cases and corresponding productivity and profit data for "peer-to-peer" and "fixed-teams" trade models. Test cases 1 till 12 (marked red) are scenarios with bottleneck where one part of the mining process lacks resources (e.g. Availability of <i>Digger</i> or <i>Transporter Robots</i>).	150
4.20	Summary of the key recommended practices and their inter-relationships in IEEE Std 1471-2000, IEEE Recommended practice for architectural description of software-intensive systems. Source: IEEE Std 1471-2000 [1]	158
4.21	Model Driven Architecture proposed by Object Management Group (OMG-MDA)	159
4.22	An anatomical overview of the 5 views Hyperactive Transaction Meta Model (HTM5)	160
4.23	An anatomical description of the 5 views Hyperactive Transaction Meta Model (HTM5)	160
4.24	Separation of PIM and PSM parts of HTM5-MDM	161
4.25	A summary of the case studies and a workshop conducted during the incremental development of HTM5 meta-model and HTM5-DSL	162
4.26	Top: Structural Overview of 'Telerobotics On Nao' System. Centre: Development of <i>Nao</i> Agent using choreograph, NaoQi and OpenNao platforms. Bottom: The graphical user interface at the <i>Client</i> Agent. Location of the <i>Nao</i> Robot is updated to the human user as it moves towards a target location specified by the user.	163
4.27	Above are some instants from Dynamic Electronic Institutions based Digital Business Ecosystem simulations. Top: Formation of eight Institutions of different cardinality and type. Some of the groups are in Formation or Re-Formation phase. Centre: Two institutions have moved to Re-Foundation phase while one of the institutes has finalized freeing its member BOTs. Bottom: TurtleBOT setup for a scaled down version of Dynamic Electronic Institutions based Digital Business Ecosystem.	164

- 4.28 Above are a set of screen shots from the simulated experiments conducted on the "Mine Cloud" P2P cloud robotic system. The Cloud in these experiments is simulated by the inter-agent message passing mechanism of the simulator. Left: The mining action. *Miner Robots* search for their respective target mineral ores while *Digger* and *Transporter Robots* are hired by the *Miner Robots* at different stages of the mining process. The *Transporter Robots* loads the mineral ore from a mined plot and delivers the load to the market selected by the *Miner Robot* (Through the matchmaking mechanism). Right: The physical robot colony of 5 TurtleBOT robots that was used to implement a scaled down version of the P2P "Mine Cloud" case study. Out of the five robots, three were implemented as *Miner Robots* while the other two were implemented as *Digger Robots*. No *Transporter robots* or *Physical Market locations* were implemented. There were in all five mine plots (named A, B C, D and E) which were both mining locations and parking locations for *Digger Robots*. 165
- 4.29 A project based workshop on cloud connected multi-agent systems. 166
- 4.30 A list of inspection cases (checks) for the case studies. 166
- 4.31 Pairwise analysis of inspection cases (checks) presented in Fig. 4.30 with the case studies and the workshop presented in Fig. 4.25. The legends rate the studies based on the extent to which an inspection case was tested. The combined result of each of the studies for an inspection case gives a measure of the extent to which an inspection case was tested in these studies. The combined result of each of the checks for a study gives a measure of the extent to which a study contributed in testing the HTM5 methodology. The combined sum of all contributions tests the overall satisfaction measure 341 (67.7 per cent of 504=3x6x28). A perfect 504 would mean each of the implementation cases was rigorously proven while any score above 336 (2x6x28) would mean that at an average all inspection cases were tested with satisfaction. Further studies are planned to test the inspection cases with scores lower or equal to 50 per cent. 167

List of Tables

3.1	A summary of Functionalities associated with the 5 views and PIC*/PIC/PSC Classes of HTM5 (PART 1).	70
3.2	A summary of Functionalities associated with the 5 views and PIC*/PIC/PSC Classes of HTM5 (PART 2).	71
3.3	A summary of Functionalities associated with the 5 views and PIC*/PIC/PSC Classes of HTM5 (PART 3).	72
3.4	Concerns in Structural and Relational Views of HTM5 and corresponding Design Elements in ARC.(Part 1)	77
3.5	Concerns in Structural and Relational Views of HTM5 and corresponding Design Elements in ARC. (Part 2)	80
3.6	Concerns in Trade View of HTM5 and corresponding Design Elements in Trade-ARC. (Part 1)	82
3.7	Concerns in Trade View of HTM5 and corresponding Design Elements in Trade-ARC. (Part 2)	83
3.8	Concerns in Hyperactivity View of HTM5 and corresponding Design Elements in Hyperactivity-ARC. (Part 1)	85
3.9	Concerns in Hyperactivity View of HTM5 and corresponding Design Elements in Hyperactivity-ARC. (Part 2)	86
3.10	Concerns in Behavioural View of HTM5 and corresponding Design Elements in Use Case and Sequence ARCs. (Part 1)	91
3.11	Concerns in Behavioural View of HTM5 and corresponding Design Elements in Use Case and Sequence ARCs. (Part 2)	92
4.1	A description of various <i>Machines</i> in the 'Teleroobotics On Nao' cloud robotic system.	112

Chapter 1

Introduction

To propose a meta-model [2] for agent oriented development of cloud robotic systems is the main objective of this thesis. The proposed meta-model is built following the guidelines laid out by object management group's model driven architecture document (MDA) [3]. Development of a domain specific language for supporting the proposed meta-model and several case studies to justify its usability are principle secondary objectives of this thesis. This chapter provides an introduction to the work presented in this thesis. The organization of this thesis is presented at the end of this chapter.

1.1 Overview

Cloud robotics is an emerging domain in distributed intelligent systems. The domain is derived from the cloud computing business logic, cloud based services and other internet technologies. Cloud robotics enables robots to offer their resources as a service to other robots. The advantage of this service oriented architecture is that it enables robots to carry just a minimum set of hardware resources on board, and access all knowledge and resources available on the cloud. Another advantage of cloud robotics is that it gives a wider, richer canvas for designers to propose multi robot systems. The maintenance overheads are also reduced as updates are required where a service is hosted and not at all the places where there are utilized. Rapidly increasing data transfer rates and cheaper connectivity also increase the scope of cloud robotic applications. A typical cloud robotic ecosystem (The "environment" created by a cloud robotic system) has robots as well as various other entities which enable the cloud infrastructure and services to function (see Fig. 1.1). The cloud itself could be built of more than one kind of computer networks. Server banks, cloud platform, mobile devices, ambient intelligence and various other auxiliary systems could all be part of a cloud robotic system even though they are not exactly robotic entities. There are a number of systems and practices which are closely linked with the idea of cloud robotics such as Internet enabled robots, Internet of things, web of things, camera (and sensor) networks, RoboEarth project and robot app stores.

Software development for cloud connected robotic systems is a complex software engineering endeavour. These systems have different kinds of soft-

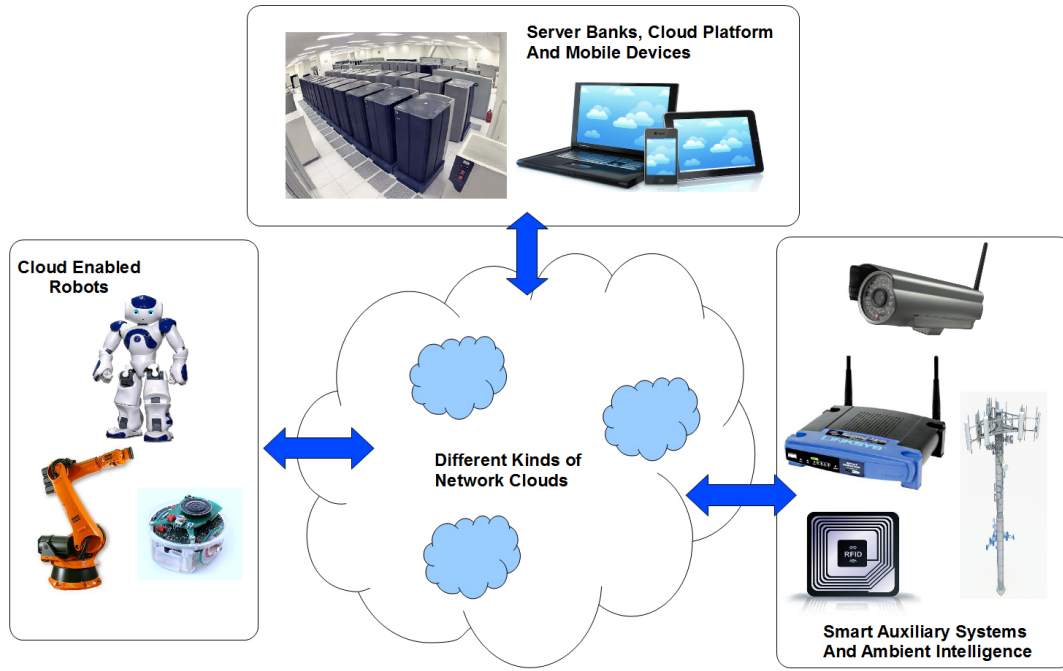


Figure 1.1: Elements of a cloud robotic system

ware and hardware platforms, and several independently running units of code. Every code unit may be built by a different set of developers and at different times. The code units could be as simple as camera drivers on an IP camera (or even a simple light switch) and as complex as an AI based server bank owned by a big corporation like Google. The companies developing the auxiliary systems may want to maintain the autonomy of their code units, propose their own terms for services offered by them, and may want to dynamically change the kind and cost of services offered by them. On the other hand, there could be developers working on simple robotic platforms who are just interested in a send-receive pay-per-use kind of cloud service with flexibility of changing the service provider or the quality and scale of the service as and when required. The complexities involved in making an open access cloud robotic system involves publishing information about the setup of the system without compromising the autonomy of the involved entities. It is important to have a system where an individual's business logic is kept autonomous, while the cloud computing based trade of services can take place dynamically. USA's National Institute of Standards and Technology defines Cloud Computing as: *"Cloud computing is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction."* An approach that is based on cloud computing phenomenon and supports independence and heterogeneity would have a good impact on the cloud robotic ecosystem. Furthermore, the approach should be assisted with tools that help design such systems. To enable rapid absorption into the ecosystem, the proposed tools and methodologies should require minimum changes in the way robots and other auxiliary systems are developed individually.

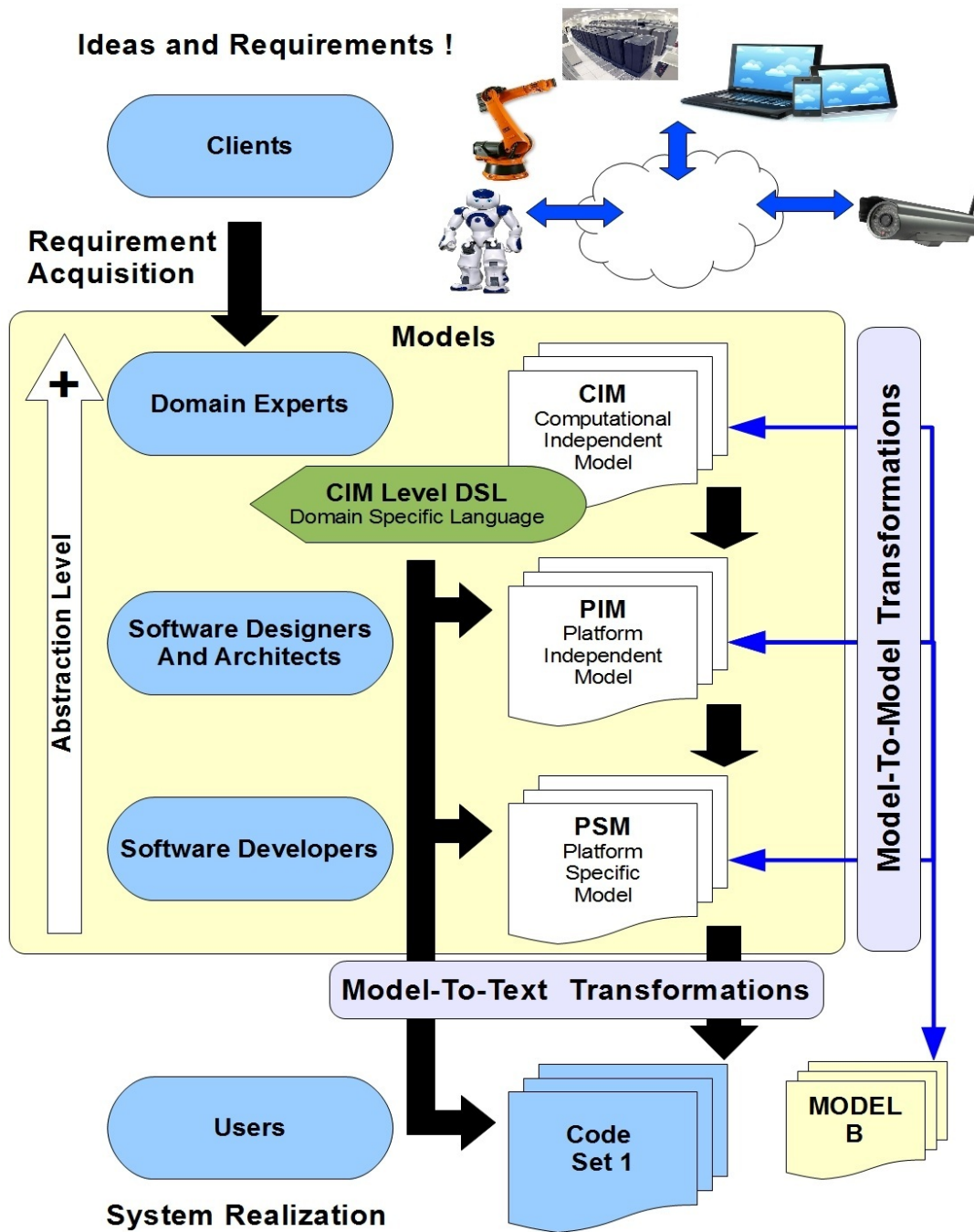


Figure 1.2: Model Driven Architecture proposed by Object Management Group (OMG-MDA). The three layers of MDA cater to different actors in the software development life cycle. "Model to model" transformations are between the three layers of MDA, or to an external model as the target. "Model to Text" transformation has a general purpose programming language as target. Models are executed independently or via a domain specific language for model transformations.

Software Agents are used in various application domains and are specified in different ways for different usage scenarios. The potential of agents as a programming concept varies with the way they are specified and used in

different domains. One generic definition thus cannot be used to explain the concept of agents. [4, 5, 6] are some of the most quoted definitions where agents are seen as members of systems of autonomous computers where each member has a sense of autonomy and flexibility over its actions. Some features that distinguish such systems are a general unpredictability of their behaviour, dynamism in their activity and openness. Open systems are systems which allow third party products to plug in or interoperate with them [7]. Agents in such systems have personal goals, goals which concern a set of agents and goals which are common to the whole system. In order to resolve conflicts, there may be a set of protocols and semantics in the system. To summarize, Software Agents are computational entities with specific roles and personal objectives working in a visible environment with other entities which may have dissimilar roles and objectives [8]. Distributed artificial intelligence (DAI) [9] has two key elements. (1) Distributed problem solving or DPS [10] which deals with the distribution of DAI's problem solving process and (2) Multi agent systems (MAS) which deals with interactions and behaviour complexities in a DAI system. Multi Agent Systems are a collection of Agents interacting using complex but flexible protocols. The Agents give equal weightage to personal and the collective goals of the system. The intelligence in a multi agent system results from several simple competitions and collaborations between agents [11]. Multi agent systems are capable of solving problems which are normally beyond the capability of any one of its member agents. We believe that an agent oriented methodology has the essential elements for developing cloud robotic systems. The concept of agency maintains autonomy of individual agents. The robots and other auxiliary systems in a cloud robotic ecosystem could be represented as their respective agents in the cloud. Later Chapters of this thesis will further explain this approach in the 5-View Hyperactive Transaction Meta-Model (HTM5). We will also discuss the adjustments that were required.

A model is a set of true statements about a system being studied [2]. A specification model species the behaviour of a system under study while a descriptive model describes the way a system works. A meta-model is a descriptive model where the entity being described is a model itself. A meta-model describes the way a particular model has to be made [2]. Model driven engineering or MDE is the engineering methodology where models with high abstraction are built to specify a system, without including implementational details. How a model will be implemented is described in lower layer models, but this is not a priority at the early stages of the design process. MDE speeds up the prototyping process and increases client involvement in requirement acquisition and design. This is possible as top layer models are extremely abstract and are syntactically specific to the domain. MDE has become a popular mode of development in many industries including the software industry. Cloud robotic systems with all their complexities are an ideal candidate for model driven engineering.

Object management group (OMG) is a consortium of standards in computer industry. OMG released a guide to model driven architecture (MDA) [3] in the year 2003. OMG-MDA has gained popularity in the software industry in the past decade. Many domain specific meta-models based on MDA

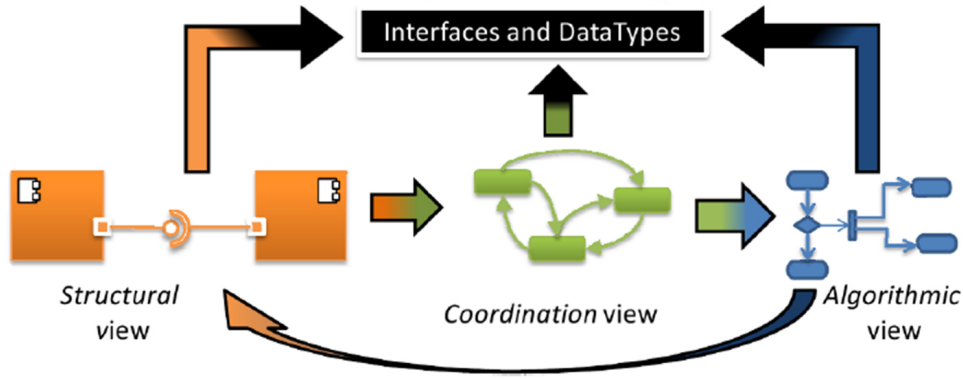


Figure 1.3: Schematic representation of the 3-View Component Meta-Model (V3CMM) views and relationships that exist among them

are now available for development of software for specific domains. There is a strong emphasis on the development of system models [2] in MDA. Software product line engineering or (SPLE) [12, 13, 14] is a software manufacturing technique on the lines of automotive or other component based product manufacturers. The key idea in SPLE is to develop reusable components that can be replaced when they are updated or damaged. The methodology promotes independent development and sale of components by one or several manufacturers that are compatible with one another. Software product line engineering encourages creation of reusable software entities. These entities could be individual code units, components, classes, libraries, models or components within a model. It is thus necessary that a meta-model should support a modular design, with clear boundaries between components, and between different functionalities within a component. MDA is a three layer design where every layer has a different degree of abstraction and is meant to be used at different stages of the product development and by different people (see Fig. 1.2). Platform specific model (PSM), platform independent model (PIM) and computation independent model (CIM) are designed for software developers, software designers and domain experts respectively.

It is common to support a meta-model by a domain specific language (DSL). DSL is a programming language based on a meta-model, which is used to write solutions in a particular domain. Common examples of domain specific languages are Structured Query Language (SQL) [15] for database management, Very High Speed Integrated Circuit Hardware Description Language (VHDL) [16] for FPGA/IC design, Hyper-Text Markup Language (HTML) [17] for web page design and Matrix Laboratory (MATLAB) [18] for matrix based rapid prototyping. These languages provide a syntax which is readable to those working in a particular domain. Although a domain specific language can be written for any of the three layers of MDA, a domain specific language with computation independent abstraction is an ideal one for domain experts as well as for clients. The effort is also to build executable models and DSLs which can be executed to make lower layer models and executable code units (see Fig. 1.2). These "model to model" (M2M) and "model to text" (M2T) (code/script) transformations are key objectives

in Model-driven software engineering [19, 20].

Due to greater complexities in some domains, a single model is not sufficient to specify a system. When more than one model is made to represent/specify a system, it is called multi-view modelling. A multi-view system is clearer as separate models/views are used to separate concerns for stakeholders [21]. V3CMM (3-View Component Meta-Model) [22] is an example where multi-view modelling as well as OMG-MDA is used to develop a Meta-model for development of software in robotic systems. V3CMM is a 3-View Component Meta-Model for Model-Driven Robotic Software Development. The three views of V3CMM (see Fig. 1.3) capture different aspects of a robotic software system, and it is supported by a platform independent DSL and model transformations. The V3CMM structural view captures the structure and component placement within the software architecture. The coordination and interactions between the components is captured in the Coordination view in V3CMM. The individual components might have one or more algorithms running within them. The V3CMM algorithmic view is a graphical representation of these algorithms.

Current cloud robotic systems are structured around the client server methodology. One or more robotic clients access cloud resources through one or many web servers. This is a useful approach since services such as algorithms, maps, text and image based search engines are easily available as a free service on the internet. A client server system provides a simple migration of cloud computing applications to robotic platforms. Although client server based cloud robotic systems are useful, they are centralized systems with a limited scope of services. The authors of this thesis are supporters of a peer-to-peer cloud robotic methodology that will enable all members of the cloud robotic ecosystem to act like a server and a client as and when required. A peer-to-peer cloud robotic system will enable individual robots to offer and avail services from other robots and auxiliary systems. These services could be a simple sharing of sensor data or computational resources or data concerning learnability, Belief, Desire and Intentions (BDI agents) in a distributed artificial intelligent (DAI) application. The traditional client server based system can be part of the proposed peer-to-peer systems and the end result will be a flexible, dynamic exchange of services over the cloud.

At present there is no UML like design methodology or meta-model present for agent oriented development of cloud robotic systems. The use of UML and other traditional software engineering models for agent oriented cloud robotic systems are not beneficial since those tools were not designed at a time when cloud and distributed computing was widespread. Since the domain is new, there is currently no domain specific language to write computation independent designs for such systems. In our work we have targeted these gaps in the software development for agent oriented cloud robotic systems which will be found of use when imminently cloud robotics and cloud connected multi agent systems mature into industrial and business reality.

1.2 Objectives

Following are the research objectives presented in this thesis:

- **To develop an OMG-MDA based Meta-Model for agent oriented development of cloud robotic systems.**
- To specify the 5 views (Structural, Relational, Trade, Hyperactivity and Behavioural views) and their scope in the development of cloud robotic system.
- To provide provisions in HTM5 to have a graphical representation (Agent Relation Charts) for inter-agent interactions specific to each of the 5 views.
- To provide a mechanism (Hyperactivity) by which an agent's autonomy can be released for specific agents. This is to give flexibility to system designers by inducing an object-like character to some agents, without fully dissolving their agency characteristics.
- To enable Components built in PIM layer of HTM5 be extendable to PSM layer.
- To provide provisions in HTM5 to support a relationship based, peer-to-peer exchange of services. HTM5 should propose mechanisms to implement cloud computing business logic.
- To develop a domain specific language HTM5-DSL with CIM layer abstraction.
- To automatically translate HTM5-DSL code to Java class hierarchy and equivalent UML class diagrams.
- To conduct diverse case studies on real and simulated robot colonies justifying the usability of HTM5 in cloud robotic systems. The case studies should incorporate the following test cases for HTM5 usability:
 - Incorporating multiple robotic platforms in one study
 - Incorporating internet based agents in at least one case study
 - Incorporating internet based commercial servers in at least one case study
 - Incorporating non robotic elements of a cloud robotic system
 - Realtime execution response for all case studies
 - Simulations for cloud robotic agent colonies in at least two case studies
 - At least 5 cloud robotic entities in all physical case studies
 - Up to 1000 cloud robotic entities in simulated case studies
 - Incorporation of Dynamic Electronic Institutions in at least one case study

- Incorporation of Digital Business Ecosystem in at least two case studies
- Incorporation of Peer-to-Peer trade dynamics in at least one of the case study
- A scaled down version of simulated case studies to be implemented on physical robots
- Incorporation of cloud robotic business model in at least two case studies

1.3 Contributions

The key contributions of this doctoral research are as follows:

- Development of a 5 View Hyperactive Transaction Meta-Model (HTM5) for development of agent oriented cloud robotic systems. The Meta-model will address the absence of a unified metamodel for agent oriented development of cloud robotic systems.
- Incorporation of OMG-MDA guidelines for development of HTM5, an industrial standard for development of MDA meta-models.
- Development of a computation independent Domain Specific Language (HTM5-DSL) for HTM5.
- "Model to Model" and "Model to Text" automated transformations from HTM5-DSL to UML and Java class hierarchy.
- Numerous case study experiments to test and demonstrate the usability of HTM5 in real projects and with complicated cloud robotic implementation scenarios.

1.4 Outline of the Thesis

Following is general description of the content of this thesis:

Chapter 1 presents a motivational introduction and overview of the thesis, its motivation, objectives and contributions.

Chapter 2 provides a general overview of background information and current industrial practices regarding cloud computing, cloud robotics, model driven engineering, agents and agent driven cloud robotics. This chapter also describes the concept of *Hyperactivity*, flexible agency and agent classification based on *Hyperactivity*.

Chapter 3 presents in detail the anatomy of HTM5 meta model for agent oriented development of cloud robotic systems. This chapter also presents the design elements and grammar for HTM5-Domain specific language (HTM5-DSL), a computation independent DSL for HTM5.

Chapter 4 presents various case studies conducted during the incremental development of HTM5 meta-model and its DSL. This chapter discusses in detail the feasibility and viability analysis for the proposed model and the elements of HTM5 and HTM5-DSL that makes it a feasible meta-modelling methodology for agent oriented cloud robotic systems in research and industrial systems.

Chapter 5 discusses and analyses the presented work, summarizes the conclusions of this thesis and outlines future directions.

Chapter 2

Background Information

This chapter provides a general overview of background information and current industrial practices regarding cloud computing, cloud robotics, model driven engineering, agents and agent driven cloud robotics.

2.1 The Cloud

"I don't need a hard disk in my computer if I can get to the server faster... carrying around these non-connected computers is byzantine by comparison."
-Steve Jobs

2.1.1 Cloud Computing

In the past decade we have seen the emergence of cloud computing as a new business model for internet based service industry. The dot-com bubble bursted in the year 2000 after which businesses moved towards virtualization and cloud computing. Cloud computing business model allows small and medium businesses to use enterprise level resources without actually buying and maintaining the hardware and human-resource. This *pay-per-use* and *scale as and when required* methodology of cloud computing made it a popular industry model. Banking, security and standardization in cloud computing increased confidence of businesses while affordable internet connected devices and mobile connectivity exponentially increased the number of users of such services. Cloud based Applications on mobile devices and integration of traditional services in the cloud brought us today to age of cloud driven mobile business ecology. The robotic community adopted the concepts and ideas proposed by cloud computing and the phenomenon of utilizing cloud services for internet enabled robots gave birth to the domain of Cloud Robotics.

Cloud computing is the current generation of internet computing which is currently evolving to become a peer-to-peer system where every cloud entity is a potential service provider. The next generation peer-to-peer cloud computing will give rise to a business ecosystem where entities in a cloud could freely share their resources as services without a centralized cloud server. Peers provide resources to other peers and reduce cost and dependency on

original service provider. Addition of new peers increase the demand of existing resources but they also contribute to the pool of resources shared between all the peers. Most robots are entities that are capable of performing a physical action. In essence every robot is a potential service provider and a peer-to-peer cloud robotic framework is a more suitable methodology for robots working in a common physical environment. The shift from the current client-server like cloud robotics to peer-to-peer cloud robotics will enable robotic ecosystems to avail cloud resources as well as contribute to the pool of cloud based services (Robot's sensor data, images, gathered knowledge, physical action through its actuators, power or communication relay for other robots/devices). Service oriented peer-to-peer cloud robotic methodologies could emerge to a whole new sub-domain in multi-robot systems and Distributed Artificial Intelligence (DAI) [9] systems. Design and development of these systems will require special design tools, meta-models [2] and development methodologies.

2.1.2 The Cloud Business Model

Cloud computing is a business methodology and is a remoulded use of existing computing and internet technologies. There are two levels at which cloud robotics could evolve from cloud computing. Cloud services like cloud storage, software and platform as a service (IaaS, PaaS, SaaS) could be readily adopted to work for robots by treating robots as any other clients to these cloud services. In this form, cloud robotics is cloud computing with a robot's computer as client and the underlying tools and mechanisms for the two remain the same. In other adaptations, cloud robotics could mean that robots should be made capable to provide their physical and functional capabilities as a service to other robots and computers across the cloud. In this second form, the ideas of cloud computing are adopted as such to cloud robotics, but special tools and mechanisms will be required to implement these ideas on traditional robots. Tele-operated robotics is an example for the second form, where a robot acts as a server offering its functionalities to a remote user. In both form, there is a robotic/non-robotic server that provides its functionalities as a service to other non-robotic/robotic clients. This matches with the traditional client-server mechanism in cloud computing and could be identified as client-server like cloud robotics.

2.1.3 Cloud Computing In Robotics

In Chapter 1 we discussed cloud robotics as an upcoming domain in robotics. In the past few years, a number of projects have taken initiatives towards cloud robotics. These are in addition to the developments in the cloud computing domain, which too contribute to the cloud robotic ecosystem. Following are descriptions of some of the key initiatives in cloud robotics:

- 1999: Internet of Things (IoT): Internet of things [23] is a system wherein objects, animals, people and services have the ability to transfer useful data over internet without the requirement of human inter-

action and help. A thing in IoT can refer to an animal with a biochip transponder, a human with heart monitor, or an even automobile with sensors to send alerts about low tire pressure. It includes any natural or manmade object that can have an IP address and the ability to transfer data using internet. IoT is often associated with machine to machine communication (M2M); for example in case of manufacturing. The products that are built with the capability of machine to machine communication are referred to as being smart like smart grid. More and more objects are embedded with sensors which provide them with the ability to communicate. This has led to information networks that improve efficiency in business processes, create novel business models resulting in reduced costs.

- 2008: Web of Things (WoT): Web of Things [24] is a concept to incorporate day to day physical objects into the World Wide Web by providing them with API (application programming interface). This will help create a virtual profile for all the objects which can be used for various applications. WoT goal is to build a web of devices that is open, scalable, and flexible. Objects are made smart by providing them the ability to connect to the web and these objects are capable of storing and sharing data. These smart devices are programmed to make decisions based on data that they have created and the data from other sources. The resulting infrastructure is a mix of equipment, machineries, systems, methods, structures and devices that require monitoring and management when added to the web.
- 2010: Rosbridge [25]: This is a specification for a network layer protocol that enables communication between a ROS environment (hosted on the cloud) and a robot. ROS [26] is abbreviation for Robotic Operating System that contains tools and software libraries for robotic application development.
- 2010: DAVinCi [27]: This is a cloud computing framework for service robots. The project brings the parallelism and scalability aspects of cloud computing to the robotics domain. This project is not publically available.
- 2011: Rosjava [28]: A Java based library that allows Android devices to utilize cloud services through ROS.
- 2011: RoboEarth [29]: This is a framework that allows robots to utilize database and services hosted on a WWW style server. The robots can share their behaviours with other robots through the database leading to collective learning. RoboEarth project was started in 1999 in collaboration of robotic researchers and the industry. The purpose of this project was to prove that connection to an information network repository will catalyze the process of learning in robots that allows robotic system to perform complex tasks. The purpose was also to show that network connected to information repository will autonomously be

	HTM5	Rosbridge	DAVinCi	Rosjava	RoboEarth	GostaiNet	Cloud Based Robot Grasping	Internet of things	Web of things	V3CMM
Is a methodology?	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes
Is a Model driven methodology?	Yes	No	No	No	No	No	No	Yes	Yes	Yes
Follows OMG MDA guidelines?	Yes	No	No	No	No	No	No	No	No	Yes
Is specifically for cloud robotic applications?	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	
Includes auxiliary devices in the design domain?	Yes	Yes	No	Yes	No	No	No	Yes	Yes	
Is used for a holistic development of cloud robotic ecosystem design?	Yes	No	No	No	Yes	No	No	Yes	Yes	
Is not specific to a particular application?	Yes	No	Yes	No	Yes	No	No	Yes	Yes	Yes
Supports peer-to-peer operation?	Yes	Yes	No	Yes	No	No	No	Yes	Yes	
Is multi-view?	Yes	No	No	No	No	No	No	No	No	Yes
Is industry oriented?	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Has special constructs to embed business logic?	Yes	No	Yes	No	Yes	Yes	No	Yes	Yes	
Supports independent component development?	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Is Agent oriented?	Yes	No	No	No	No	No	No	No	No	
Has a machine descriptor model?	Yes	No	No	No	No	No	Yes	No	No	
is open to public use?	Yes	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes
Supports Dynamic Business Ecosystems?	Yes	No	No	No	No	No	No	No	No	
Supports Digital Electronic institutions?	Yes	No	No	No	No	No	No	No	No	
It is possible to use this as a Distributed Artificial Intelligence (DAI) platform?	Yes	No	No	No	No	No	No	No	No	
Has a supporting Domain specific language (DSL)?	Yes	No	No	No	No	No	No	No	No	Yes
Is the supporting DSL is computation independent?	Yes	No	No	No	No	No	No	No	No	Yes
Supports automated model to text (M2T) transformations?	Yes	No	No	No	No	No	No	No	No	Yes
Supports automated model to model (M2M) transformations?	Yes	No	No	No	No	No	No	No	No	Yes
The methodology is not dependent on associated platform and tools?	Yes	No	No	No	No	No	No	No	No	Yes
Has associated feasibility study ?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Figure 2.1: Comparison of popular approaches towards cloud robotics and cloud connected devices.

able to perform tasks that were not exclusively laid out at the time of designing the system. RoboEarth project received funding from the European Commission's Cognitive Systems to develop the open source network database platform of RoboEarth project. RoboEarth platform is used to create and execute action recipes. It is also used to perform localization and mapping, 3D sensing and to track objects dynamically. It is also a rich source of data. This database can be further used to research and analyze to develop standard language protocol and modular designing of cloud robotic system.

- 2011: GostaiNet [30]: This is a private project that allows sharing of vision and algorithmic behaviours amongst compatible robots.
- 2013: Cloud Based Robot Grasping [31]: A cloud robotics system for recognizing and grasping common household objects. The system utilizes Google Goggle [32] and the Point Cloud Library (PCL) [33] to estimate the ideal orientation for grasping common household items. This project is not publically available.

Cloud robotics as a domain is fairly new. The projects mentioned above are tools and frameworks that are an extension of traditional client server methodology. Scalability and parallelism concepts are borrowed from the cloud computing systems with robots as the clients. Projects like RoboEarth [29] and GostaiNet [30] promote contributions from robots to the cloud

server. These are extensions of cloud storage services widely used in cloud computing systems. A comparison of the popular approaches towards cloud robotic ecosystems with the Meta-Model HTM5 is presented in Fig. 2.1. The parameters chosen for this comparison are inspired by common needs from Industry, research and business personals. The comparison being made has several components. We first see if the discussed entity is a methodology, a software tool or a just a commercial product. If it is a methodology, then we compare the entities on being model driven, multi-view, OMG-MDA based, domain specificity, scope and its support for peer to peer transactions. We also compare the entities on the extent to which they are in sync with the popular industrial practices and their ability to include complex research and business concepts. More refined comparisons on the existence and kind of domain specific language and M2M/T transformations that the methodology proposes. In general all of these entities have some level of feasibility study associated with them. We did not compare the extent to which their feasibility was checked by the developers and tools any kind of feasibility study as a valid one.

2.1.4 Cloud Robotics Ecosystem

Development of cloud robotics as a business model will require new tools and methodologies. It is essential to develop methodologies that are industry and business oriented. The cloud robotic methodologies should go one step further to include models that incorporate concepts like Distributed Artificial Intelligence (DAI) [9], registry based service discovery and automated match-making mechanism. Many of the services offered by a robot to other robots will have a physical world component. A robot's physical reach will determine the scope of the physical services offered by it and any business model for a robotic ecosystem should include provisions to model factors that codify physical world interactions. A cloud robotic ecosystem will also include many non-robotic entities. These entities could range from ambient intelligence to server banks. In theory any device that can communicate through a network could be included as a working component of a cloud robotic system. The communication networks that collectively build the cloud could be of different kinds and visible in selective physical regions. A methodology that allows modelling of these non-robotic devices, networks and interfaces will give a complete design toolset to designers of cloud robotic systems. Fig. 1.1 shows a typical cloud robotic ecosystem with robotic and non-robotic entities. A design methodology for cloud robotic ecosystem should provide tools to model all physical and theoretical aspects of these systems. Key theoretical elements of a cloud robotic ecosystem would be its network structure, event driven behaviour, social interactions, norm driven peer-to-peer trade, micro level competitions and dynamically regulating collaboration [11].

2.2 Model Driven Engineering and Architecture

"And at the end of the day the Internet is still all about Software."

–Marc Andreessen, from an interview in the NY Times

2.2.1 Component Based Software Engineering

The ratio of time spent reading code versus writing is well over 10 to 1 ... making it easy to read makes it easier to write.

–Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

When writing software for the industry, managerial concerns such as reusability have to be taken into account in the design process. Products based on multi agent systems are generally very dynamic with newer versions being produced very often and parts of software being used in other products. In most cases, people working on robots and other hardware components of the multi agent system are not trained in software engineering. Hence the Ad-hoc software development at the time of research is not very useful for the product development. To solve the above mentioned problems, the complete system is split into components which are loosely connected and functionally independent. Component-based software engineering (CBSE) [34] is a reuse-based approach to software development. Once implemented and tested, the components can be reused with ease in other products saving time and the cost of software development. It is common practice in the software industry to implement CBSE using objects.

2.2.2 Models

A set of comments about the behaviour or structure of a system under study is a model of the system [2]. Both specification and descriptive models are documents that define a system and are the starting points in a model driven approach towards system development. In Model Driven Engineering (MDE), models are used to specify a system at a high level of abstraction. Implementational details of lower layers are later added to the system model and are not given much attention while building the model of a system in early stages of system design.

2.2.3 Meta-Models

A meta-model is a descriptive model for models [2]. A meta-model specifies how models for systems in a particular domain should be built. Complicated systems where lower layer implementation details are less important than the overall idea of a system are ideal candidates for MDE. Rapid prototyping and participation of clients in model development has made MDE popular in several industries.

2.2.4 Multi-View Modelling

It is common to build more than one model for the same system. This multi-model approach targets to capture different aspects of the system under study in separate models (called Views) bringing clarity to the model document. For systems with several stakeholders, the multi-view modelling methodology helps in separation of concerns [21].

2.2.5 OMG Model Driven Architecture

Object Management Group's Model Driven Architecture (OMG-MDA) guidelines [3] have become a popular standard in development of software models for complicated systems. MDA promotes development of system models [2] and Software Product Line Engineering (SPLE) [12, 13, 14] encourages reuse of software components and models. Any proposed model for cloud robotics will have greater acceptance and industrial feasibility if it is in accordance with MDA and SPLE standards. MDA has a three layered design with separate models for Computation Independent, Platform Independent and Platform Specific stages in system development (see Fig. 1.2). Each layer of MDA serves design needs of stakeholders at various stages of development.

2.3 Agents and Cloud Robotics

"Never send a human to do a machine's job."
 –Agent Smith, The Matrix, 1999

2.3.1 Agent and Agency

"Software Agents are computational entities with specific roles and personal objectives working in a visible environment with other entities which may have dissimilar roles and objectives" [8].

The concept of agents is essentially an extension to the concept of objects in object oriented methodology. An object is specified by its class attributes and operations which can be accessed from other objects. This open access to an object's interior functionality compromises its autonomy. What makes agents different from objects is the autonomy an agent has over its own operations. Agents are deployed in various domains with different functionalities and thus there is no consensus on the definition of an agent. In general, an autonomous entity in a system of computing entities interacting with other autonomous entities with a mandate to complete their personal and shared goals is known as an agent [35, 4, 5].

The concept of agency [36] however explains a wider set of abstractions associated with an agent. Apart from autonomy, an agent should be heterogeneous in design giving its designer independence to design it in any manner irrespective of the other agents and the network administration. Its interactions must protect its autonomy and the communication protocol should

not reveal its internal design. The commitments that an agent makes to other agents should be based on a social concept with a debtor, a creditor, action and a context. Unlike components, the receiving party (agent) takes the ownership/responsibility for an action taken when a message is received and not the party (agent) that sends the message. An agent should have mental states, explicit goals and knowledge and none of these should be in public domain. The above characteristics of an agent make it ideal for a business logic implementation where it preserves the autonomy of the entity it is representing. Agents present an excellent methodology for an open and peer-to-peer implementation of cloud computing enabled robotics.

2.3.2 Agent Metaphor and usage

Agent technology is a well-defined methodology when seen as a design metaphor of definitive approaches for designing and implementing software intensive systems. The agent concept provides elegant tools/methods for abstraction and encapsulation.

"Currently, agents provide software designers and developers an appropriate way of structuring software tools and applications around autonomous, communicative, situated and problemsolving entities to achieve the required design goals." [8]

"The agent concept offers a promising route to the development of computational systems, especially in open and dynamics environments of several real-world domains." [4]

2.3.3 Multi Agent Systems

Software agents are used in various scenarios and there is no single programming concept that completely explains the idea of agents in all domains, roles and scenarios where they are used. In other common definitions, agents are described as members of a system of autonomous computers where every member has a sense of flexibility and autonomy [4, 5, 6]. These systems are known as Multi-Agent Systems and are known for inherent dynamism, activity, heterogeneity and openness. An open system [7] is a system where third party products can interoperate by plugging in. Protocols and semantics resolve conflicts in such systems bringing a sense of order in an otherwise chaotic system. The studies in Multi-Agent Systems are concerned with interactional and behavioural complexities of a distributed Artificial Intelligence System [9]. The intelligence in Multi-Agent Systems is a combined result of several simple competitions and alliances among its member agents [11]. For their inherent autonomy, flexibility in development, and as platforms to implement intelligent trade models, agents are ideal for representing cloud robotic entities in a cloud robotic business ecosystem.

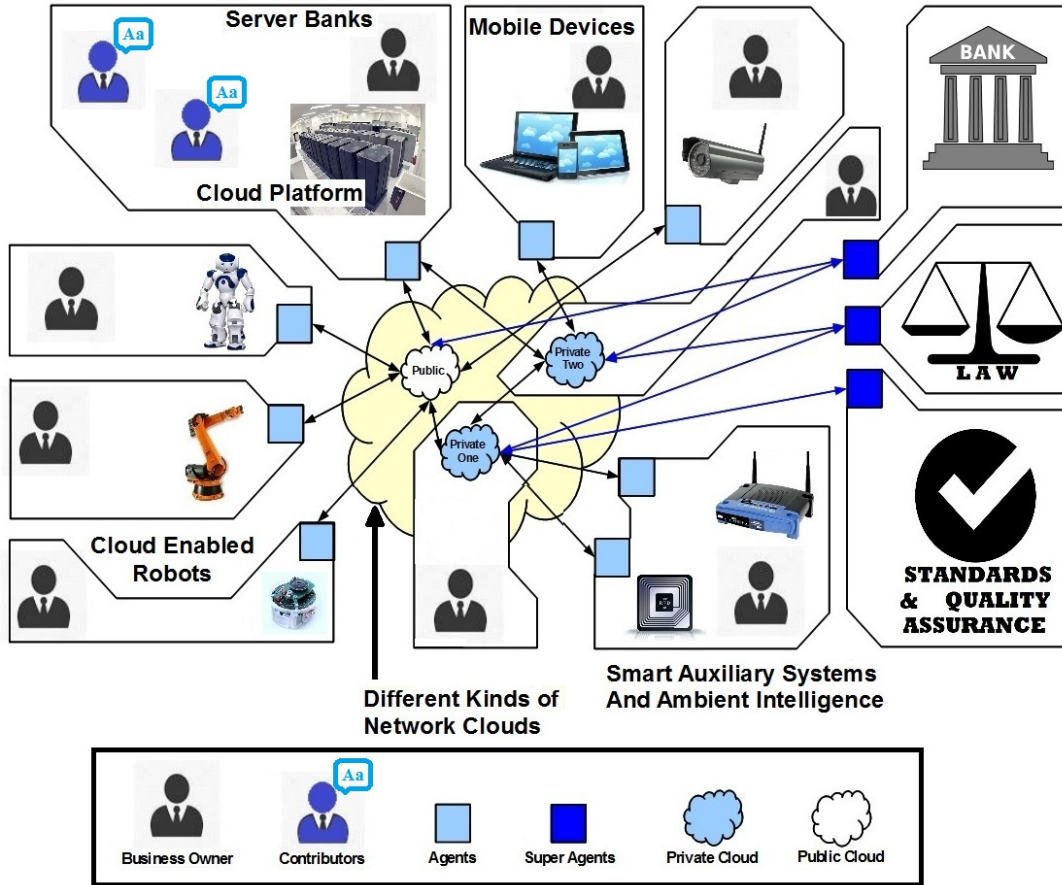


Figure 2.2: Above is an example of an Agent Oriented Cloud Robotics environment that implements a Digital Business Ecosystem (DBE).

2.3.4 Agent oriented cloud robotic systems

Distributed Artificial Intelligence (DAI) [9] and Multi-Agent systems (MAS) [4, 5, 6] are closely related domains. The challenge in DAI is distribution of a complicated problem between multiple entities. MAS on the other hand deal with the behavioural and transactional complexities that arise in implementation of DAI ideas. Problem formulation and distribution in multi-robot systems resembles DAI applications and thus an agent oriented approach towards design and development of multi-robot systems has some distinct advantages. Agent oriented development of cloud robotic systems enables easy transfer on DAI solutions in a cloud robotic ecosystem.

A typical cloud robotic ecosystem may have several robotic as well as non-robotic entities. Fig. 1.1 shows a typical agent oriented cloud robotic ecosystem where several robotic and non-robotic entities collaborate to establish digital business ecology. Representing robotic/non-robotic entities in a cloud robotic ecosystem by representative agents enables developers of those cloud entities to have independent product development life cycles. Unlike objects, agents are autonomous closed systems and they do not release their interior structure to the outside world. Their functionality is controlled by an internal operating logic and the communication between agents is through messages.

This is different than objects since they communicate through function calls. Agents are by design better suited to implement dynamically evolving business logic and thus representing business interests of cloud robotic entities through representative agents is an attractive proposition. Agent oriented cloud robotic systems also promote inclusion of Dynamic Electronic Institutions [37, 38, 39, 35] and Digital Business Ecosystem (See Fig. 2.2) [40, 41] in a cloud robotic ecosystem further enhancing the usability of the approach. In agent oriented cloud robotic system, all robotic/non-robotic entities are represented by their respective agents. The cloud entities represented by these agents may have different hardware and software configuration and may be owned by different businesses. The agents advertise the services that are offered by the entities they represent. Some cloud servers may have more than one contributor that collectively builds up a pool of resources on the cloud server. Existence of an open [7] service registry and match-making mechanisms enable agents advertise and enrol to services available on the cloud ecosystem. Portions of the cloud network infrastructure may be owned by private businesses and their usage may also be provided as a service in the cloud ecosystem. Special agents may be present in the system which represent the banking and administrative entities. These special agents help enforce standards (Trade, industry or legal rules) and enables transfer of money between cloud entities. Each of the entities in the cloud robotic system may have their internal developmental life cycle and the businesses that deploy these entities may have dynamically evolving business models. Entities may freely join or leave the ecosystem at will and their operating logics may change with time. An agent oriented approach enables individual entities to have their independent and dynamic operation. This ensures heterogeneity in business logic, design methodology and implementation of these entities.

2.4 Hyperactivity

Hyperactive transaction model is an attempt to provide a model for designing multi agent systems with a different "thought process". A similar example can be seen in object oriented software modeling. Object oriented programming is not only a different way to write code, but a new "thought process" to design software where we think in terms of objects in real life before making their equivalent class in the code. Likewise in Hyperactive Transaction Model, the attempt is to think of agents in terms of humans working together in a team.

HTM5 is a meta-model for agent oriented development of cloud robotic systems. In Chapter 1 we supported use of agents to represent robots and auxiliary systems in a cloud robotic system. In this section we aim to provide elements which make agent orientation a suitable design choice for such systems.

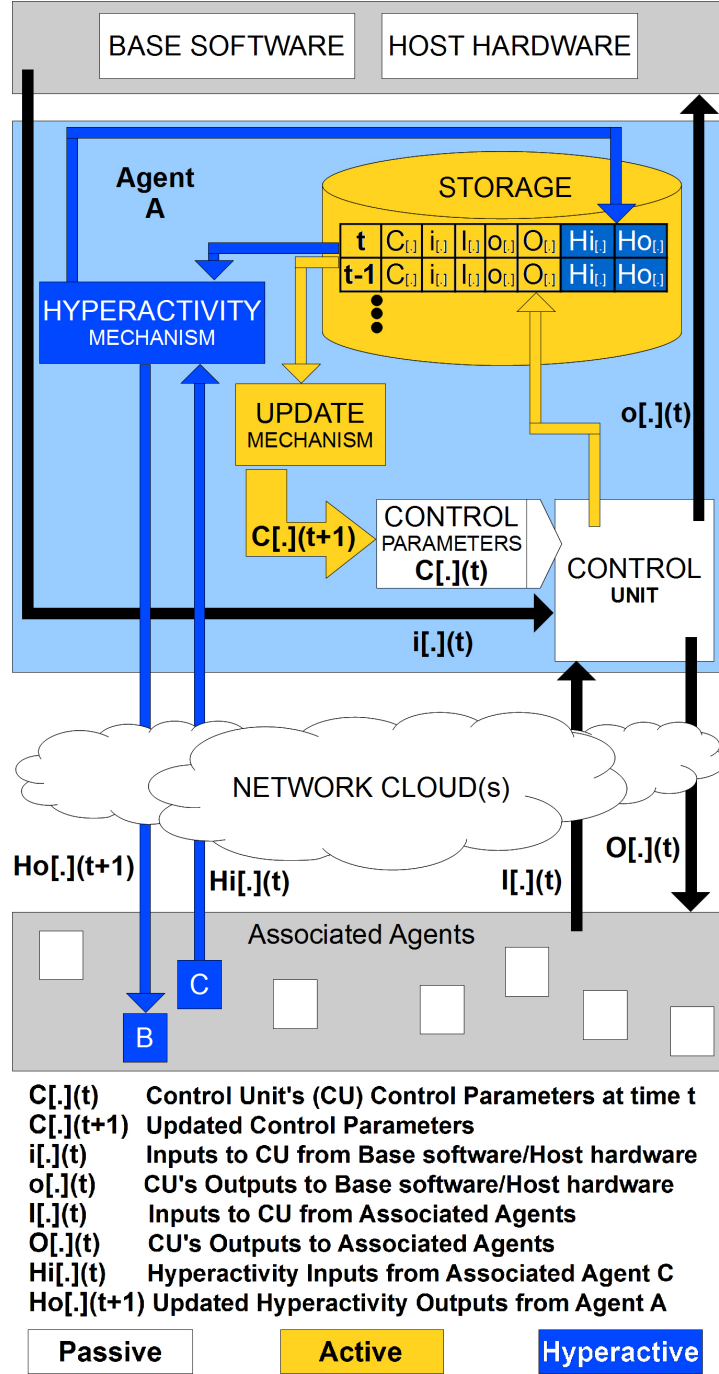


Figure 2.3: A conceptual representation of an HTM5 agent. The agent is a representative of base software and host hardware in the cloud ecosystem. Agent communicates with associated agents through two separate channels. An active agent is closest to the idea of agency [36]. HTM5 provides mechanisms to relax the idea of agency in two ways. Hyperactive agents are agents with relaxed autonomy towards specific agents, while passive agents are agents with reduced variability in their working logic.

2.4.1 The Mechanism

The agency concept is closely related to the concept of objects. In object oriented methodology [42] objects are specified by their data and operations.

The visibility of these data and operations can be controlled by visibility rules (Private, public or protected). Objects have data and functions which can be accessed from outside. Although an object may define some of its member data variables and functions as private, there will still be at least one public member data by which the outside world makes use of the object. The access to an internal function or data of an object from outside compromises its autonomy as such an access is not a communication that object can decide upon. Agents are different from objects since agents have full autonomy over their data and operations. Agents are generally understood as autonomous entities which interact within a system of other autonomous entities with a set of personal and shared goals [6, 5, 4]. However, the concept of agency [36] is specified by a wider sphere of ideas:

- Independence to act (Autonomy): Agents are independent to act on their own, even if their actions appear irrational to an external agent.
- Independence of design (Heterogeneity): A designer should be free to design internal structure of an agent and thus a system of agents could be heterogeneous and open [7].
- Independence from a central administration (Dynamism): There may be no central administration in a system of agents. System is managed by a distributed administration.
- Interactions that preserve autonomy (Communication): Agent to agent interactions may not give any hints to the internal working of an agent. It is possible that an agent interacts randomly with other agents without any belief or intention attached to those interactions.
- Protocols: The only protocols in a system of agents are when and how an agent may communicate with another agent. The protocols should not specify why a communication is needed.
- A Social concept: Agents work with a social concept of commitments. Every interaction has a debtor, a creditor, an action and a context.

The above ideas find relevance as a development methodology for cloud robotic systems. Autonomy, heterogeneity, dynamism and interactions that preserve autonomy are essential for a business oriented, open development of cloud robotic systems. Robots and auxiliary systems in a cloud robotic ecosystem may be developed by different manufacturers and deployed by various businesses or individuals. The systems could be opened or closed, the interactions may be standard or custom, and there could be high variability in terms of software engineering practices followed by organizations. Agents are representative entity for a larger hardware or software framework (see Fig. 2.3). The legacy designs of various products could be adapted to the cloud robotic ecosystem by representing them via agents in the cloud. The ideas of agency mentioned above are for idealistic scenarios. In practical usage, some relaxations and adjustments (Discussed in the next section "Flexible Agency") may be required for flexibility in system development.

An example of one such relaxation in the idea of agency is when a managerial agent needs to "force-stop" or "boot/initiate" an agent. Multi agent systems can be chaotic and unstable. The idea is to create a sense of order in these systems by including mechanisms that relax the idea of agency as and when required.

2.4.2 Flexible Agency

Fig. 2.3 is a conceptual representation of a HTM5 agent. A traditional agent have one channel for communicating with other agents. The agent have some internal control parameters which will change with time. Control parameters could be parameters for Belief, Desires and Intentions of a BDI agent [43], or any other data entity that governs control logic of an agent. For a simplistic representation, let us call the internal mechanism for update of an agent's control parameter as *Update mechanism* (Fig. 2.3) which generates a new set of control parameters $C[.](t+1)$ at time (t) based on some or all stored data items. The data items could be control parameters from time (t) ($C[.](t)$), parameters exchanged with the base software ($i[.](t)$, $o[.](t)$) and parameters exchanged with associated agents ($I[.](t)$, $O[.](t)$). In HTM5 methodology, we call an agent with such internal structure as an *Active agent*. An active agent is closest to the idea of agency [36]. Some device manufacturers or software developers may demand flexibility in the agency concept. The flexibility may be required to bring an object like character to an agent, e.g. an agent that acts as a static software entity with no variability in its control logic. In such cases a software developer could choose to remove the *Update mechanism*, and thus design a *Passive agent* in HTM5.

Passive agents are deviated from the agency concept since there is no variability in their behaviour, and are not intelligent agents in the traditional sense. We earlier discussed that autonomy is the essential idea in the concept of agency. A software designer could find it unnecessary to make agents with full autonomy, in various scenarios. HTM5 allows the designers to relax an agent's autonomy for specific agents by allowing a secondary channel to expose internal parameters of an agent. The *Hyperactivity mechanism* (Fig. 2.3) is a back channel communication mechanism that releases stored parameters of an agent to select associated agents ($Ho[.](t+1)$). Although the outside agents can access internal parameters, the hyperactivity mechanism could be designed to implement access control for individual agents, and for individual data items. The *Update mechanism* in a hyperactive agent includes the hyperactivity parameters ($Hi[.](t)$, $Ho[.](t)$) to update the control parameters $C[.](t+1)$. A software designer working on an agent oriented methodology using HTM5 has the flexibility to relax the ideal concepts of agency in a control way. This makes HTM5 a more useful methodology for heterogeneous development of cloud robotic systems, where some agents could have partial object like characteristics.

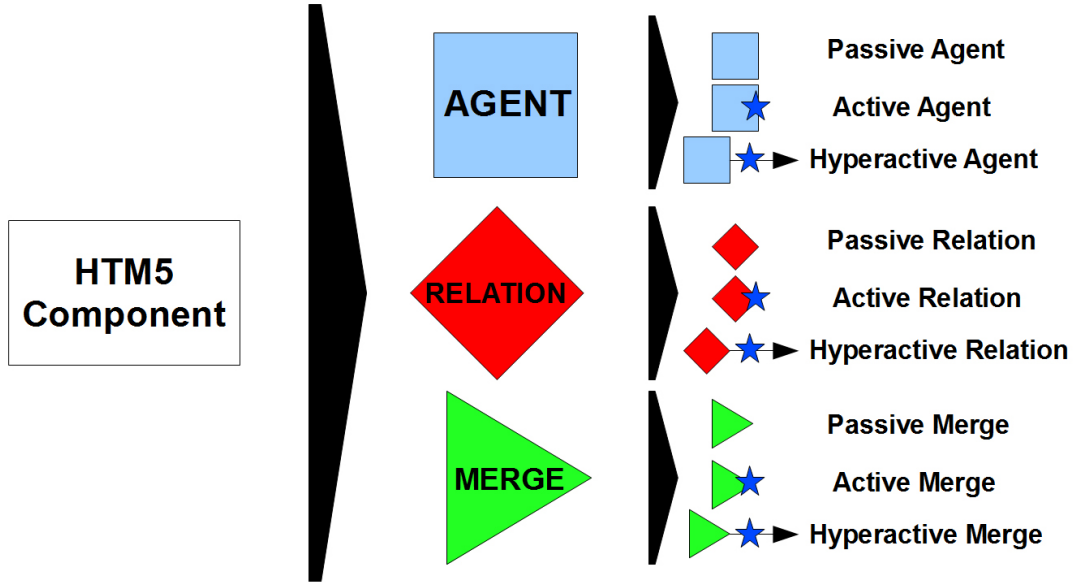


Figure 2.4: Classification of HTM5 components based on their functionality and level of Agency.

2.4.3 Agent Classification in HTM5

We will discuss later in this thesis the five views of HTM5 and their scope. It is common in distributed systems to have entities that enable the system to function in a managed manner. Elements of this distributed management may be hosted on various agents across the system, but some agents may be placed in the system for this purpose alone. There could be agents who have management of trade, relationships and message multiplexing as their primary objective. There could be agents which are ports at which other agents could dynamically join or leave a system. HTM5 has not placed any functional or modelling limitation for any agent to have these capabilities but to increase the readability of the model, agents are classified into three classes based on their primary functionality. (1) *Merges* are agents which manage the flow and merger of messages where there is a fixed number of lines on one side (usually one), and a variable number of lines on the other. *Merges* are also ideal for modelling ports where agents can join or leave the system. (2) *Relations* are agents which manage a relationship between other agents. (3) HTM5 components other than those classified as *Merges* and *Relations* are called *Agents*.

In ideal agency, the management of relationships between agents by a third agent is a violation of their autonomy. The *Relations* are special agents which host the data and functional requirements necessary for sustaining collaboration between agents. A managed collaboration between specific agents undermines their autonomy to some extent as relations enforce protocols on an agent's behaviour with other agents in a relationship. The advantage of this external management of interactions is the flexibility and control a designer has while modelling the social concepts of commitments (*Debtor*, *Creditor*, *Action and Context*). *Relations* are also ideal to model a business

logic, digital institutions [5, 40] and other human concepts (e.g. Trust [44], Ethics [45]) in a cloud robotic system. Like any agent in HTM5, a *Merge* or a *Relation* could be *Active*, *Passive* or *Hyperactive* (Fig. 2.4). *Agents*, *Merges* and *Relations* are collectively called Components in HTM5 methodology.

In this chapter we presented a general overview of the current state of the art in cloud robotics. We discussed popular attempts towards development of a cloud connected ecosystem for robots, humans and things. We discussed the new trends towards model driven architectures and agent orientations. We presented the concepts like Hyperactivity and flexible agency which are of great importance to the meta-model HTM5 presented in the next chapter. The next chapter will present the central product of our work, the HTM5 meta-model for agent oriented development of cloud connected robots.

Chapter 3

5 View Hyperactive Transaction Meta Model

This chapter explains different anatomical elements of HTM5 and the ideas that they actualize. As discussed in previous sections, HTM5 is a MDA based 5 view meta-model for agent oriented development of cloud robotic system. Fig. 3.1 and Fig. 3.2 gives an overview of the structure of the HTM5 meta-model. In accordance with the MDA guide [3], HTM5 has a three layered structure (Three layers of MDA: CIM, PIM and PSM Fig. 1.2). The three layers are named as:

- HTM5-CIM: Computation Independent Model
- HTM5-PIM: Platform Independent Model
- HTM5-PSM: Platform Specific Model

The five views of MDA are designed to separate concerns of different stakeholders. The *Hyperactivity* view is further divided in 4 sub-views to separate hyperactivity in individual views. In HTM5-CIM, views are separating concerns at the macro-level. Agent Relation Charts (ARCs) are a graphical representation for inter-agent interactions specific to each of the 5 views. In HTM5-PIM and HTM5-PSM layers, the views separate concerns within an HTM5 component (*Agent*, *Merge* or *Relation*). The HTM5-components may be hosted at various machines. HTM5-MDM (Machine Descriptor Model) is a model to describe these Machines. Agents are representing host hardware or base software in the cloud ecosystem (Fig. 2.3). HTM5-MDM is used to model the hardware or software framework on which an HTM5 component is representing in the cloud. The cardinality of HTM5 is as follows:

- Number of Layers = 3
- Number of Views = 5
- Number of ARC diagrams ≥ 5
- Number of HTM5-PIM models = Number of HTM5 components in the system

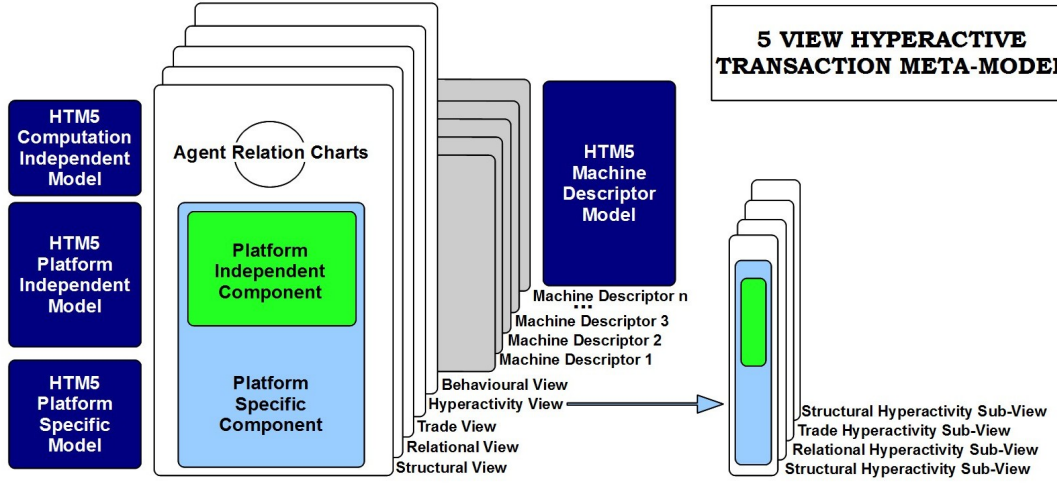


Figure 3.1: An anatomical overview of the 5 views Hyperactive Transaction Meta Model (HTM5). HTM5 is structured following the three layers of OMG-MDA and has 5 views for separating concerns for stakeholders. The Hyperactivity view is subdivided into 4 views, separating hyperactivity in different views. The machine descriptor model is present for describing hardware that hosts various agents. HTM5-MDM is spread over the bottom two layers of HTM5 Meta Model. HTM5-CIM layer consists of Agent Relation Charts, which are abstract representations of view specific concerns for various stakeholders. An HTM5 component (*Agent*, *Merge* or *Relation*) has two part design. The component is first specified in HTM5-PIM layer. The platform specific parts of a component are declared as abstract classes [46], but not completely defined in HTM5-PIM. In HTM5-PSM, the abstract entities are extended and defined specific to the platform

- Number of HTM5-PSM models = Number of HTM5-PIM components with abstract elements (Parts specific to a platform)
- Number of HTM5-MDM = Number of Host Machines (Hardware or Software Framework)
- Number of Hyperactivity sub-views $\leq (4 \times 2 \times \text{Number of Hyperactivity Links})$

3.1 HTM5 Sub Models

3.1.1 HTM5 Computation Independent Model

Fig. 3.1 and Fig. 3.2 shows the three layered structure of HTM5 with HTM-CIM layer on the top. While lower layers of HTM5 are separately specifying the individual HTM5 components (*Agent*, *Merge* or *Relation*) and Machines, the CIM layer in HTM5 is a specification of the cloud robotic system as a whole. As shown in Fig. 1.2, this layer is the most abstract layer and is developed in the early stages of the project design.

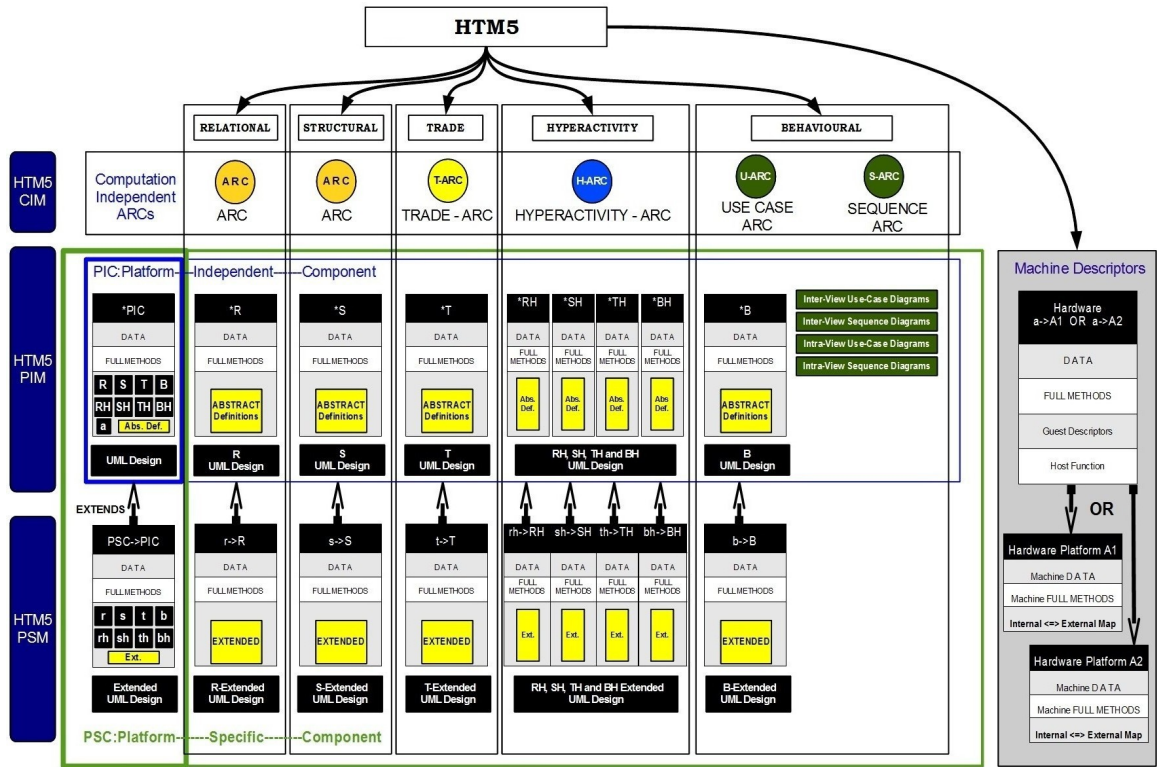


Figure 3.2: An anatomical description of the 5 views Hyperactive Transaction Meta Model (HTM5). In CIM layer, there are 5 different Agent Relation Charts for different views. The Behavioural view has two kinds of ARCs namely Use case and Sequence ARCs while Relational and Structural views have one common ARC. The PIM layer specifies the Platform Independent parts of all HTM5 components. Every HTM5 component (*Agent*, *Merge* or *Relation*) has an internal structure as shown above. The classes in PIM layer are abstract classes [46] (Represented by an asterisk *) which are extended by the classes in PSM layer. The core class of a component (*PIC or PSC) is a class that envelopes objects of other view classes. The *PIC class may contain an object of the PIM Machine Descriptor(MD) Class ('a' in the above figure). Machine Descriptor Classes are used to model host hardware or a base software (see Fig. 2.3) that an HTM5 component (*Agent*, *Merge* or *Relation*) represents in the cloud. PIM MD Classes extends PSM Machine Descriptor Classes. When a Machine is changed, modification is required in PSM layer MD classes alone. The Internal to external map in PSM layer MD classes ('A1' and 'A2' in the above image) maps the new machine names to the internal names used in PIM layer MD classes. Since an HTM5 *PIC class has an object of the PIM MD classes, no modification in HTM5 is required when a Machine is changed.

In HTM5, this layer consists of Agent Relation Charts (ARCs). ARC is a graphical CIM model and its variations are suited for different views and ideas in HTM5. The *Structural* and *Relational* elements of the design are specified using graphical representations in the ARC diagram. *Trade* and *Hyperactivity* views have their own ARC diagrams (*Trade* and *Hyperactivity*

ARCs. Fig. 3.2). In systems where the complexities in *Trade* view are dense, more than one *Trade* ARC may be made. The same applies to ARCs in other views as well. The *Behavioural* view in this layer consists of two kinds of ARC diagrams. *Use-Case* ARCs (U-ARCs) are made to specify various use case scenarios for the system while *Sequence* ARCs (S-ARCs) are used to specify sequence of key events in those scenarios. As there could be several number of use cases for a system, there could be any number of ARCs in the *Behavioural* view. Fig. 3.2 shows the placement of different ARCs with respect to different views. The structure and ideas behind individual ARCs are explained later in Section 3.3. The cardinality of ARC diagrams is as follows:

- Number of ARCs ≥ 1
- Number of T-ARCs ≥ 1
- Number of H-ARCs ≥ 1
- Number of U-ARCs ≥ 1
- Number of S-ARCs = Number of U-ARCs

Total: ≥ 5

3.1.2 HTM5 Platform Independent Model

As discussed in Chapter 1, the concept of Platform in MDA is vaguely defined. This comes as an advantage in software development as the definition of a platform is always with respect to a frame of reference. Different software frameworks are structured differently, and a strict definition of platform will have several operational difficulties. A platform could simply be a hardware entity, or a software one. It could be an operating system in one component and server bank in another. In HTM5, platform is understood as a *point of separation* in the code of a component. HTM5 splits the PIM and PSM parts of an HTM5 component without specifying the meaning of this separation. Within the same cloud robotic system, different HTM5 components (*Agent*, *Merge* or *Relation*) may have different meanings for this line of separation between platform independence and platform specificity. The idea is just to have a separable PIM layer which could be modified without affecting the above layers.

Fig. 3.2 shows a description of HTM5-PIM component model. The separation between PIM and PSM parts is implemented using abstract classes [46], represented by asterisk '*' in Fig. 3.2. The abstract classes are classes where some elements are not completely defined. Abstract classes are extended by other classes which complete the missing parts, and these extending classes are then used to make objects. It is important to note that this is just one mechanism to separate the platform specific concerns. Software developers are not restricted by HTM5 methodology to use some other mechanisms. The PSM layer classes extend the abstract classes of PIM layer and complete the platform specific elements. When a platform changes, changes

are required only in the PSM layer classes leaving PIM layer classes unchanged. The mechanism is further explained in Section 3.1.3.

In general, since *PIC is an abstract class, it cannot be used to form objects. In a scenario where PSM layer is not required, *PIC class will not have any abstract elements and can be represented as PIC. The PIC class can then have nested objects of the view specific classes (R: Relational, S: Structural, T: Trade, RH: Relational Hyperactivity, SH: Structural Hyperactivity, TH: Trade Hyperactivity, BH: Behavioural Hyperactivity, B: Behavioural) as shown in Fig. 3.2 and can be used to make objects. Apart from view specific classes, the model also includes UML diagrams for individual view classes. The interactions within a particular view, and between different views are specified in *Behavioural* view using UML behavioural diagrams. This closeness with UML on internal levels of HTM5 makes it easy for individual manufacturers to adapt their products for an HTM5 based cloud ecosystem.

The separation between PIM and PSM parts also exists in HTM5-MDM. The meaning of Platform in MDM is clearer than in the 5 views. The separation in MDM is between physical hardware (or base software that an agent is representing in the cloud ecosystem) and the internal Hardware model used by HTM5 components. The *PIC class contains a nested object of the MDM Hardware class ('a' in Fig. 3.2) and the view classes access the machine elements using this class object. The separation mechanism in MDM will be explained later in Section 3.1.4.

3.1.3 HTM5 Platform Specific Model

HTM5-PSM layer is a platform specific model for HTM5 components. As explained in Section 3.1.2, the view classes in PSM layer extend the view classes in PIM layer. The abstract elements of abstract PIC classes are extended in these classes to form complete component classes. The PSC class contains nested objects of all view classes. Since PSM class is extending the *PIC class, the MDM object is also available in objects of PSC class (See Fig. 3.2)). Like PIC, PSC too have UML diagrams to specify their functionality. The diagrams here specify the functionality that is extended in the PSM layer. No behavioural diagrams are present in this layer since at the PIM layer, the behavioural diagrams specify inter and intra view behaviour of HTM5 components. The missing abstract elements in PIC do have their declarations in the PIM view classes, thus their references can be made in the PIM level behavioural diagrams. Once the model is specified, it is used by making an object of the PSC class. The scenario where the PSM layer is missing, an object of the PIC class is made to utilize the model as explained in Section 3.1.2. PSC is implementation oriented and closest to the platform thus the separation of concerns at this layer may be diluted by specifying common routines as members of the PSC class and not within the view classes (r, s, t, rh, sh, th, bh, b). The extension of PIM layer components to PSM layer components is further explained with an example in Section 3.5.

In HTM5-MDM, the PSM layer contains classes to specify a particular machine. These classes have a mapping functionality which maps the names

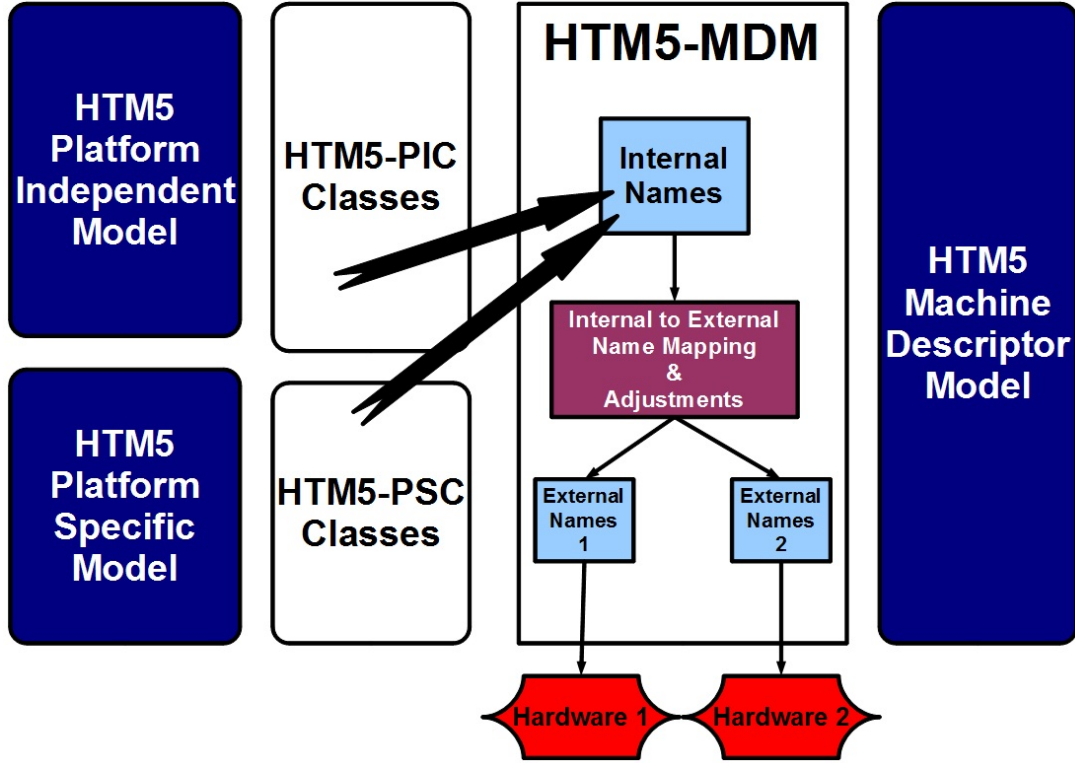


Figure 3.3: Separation of PIM and PSM parts of HTM5-MDM. HTM5 component classes utilize the internal names of the hardware entities by nesting an object of MDM PIC class. MDM PIC class extends the abstract hardware platform class in the PSM layer of MDM. When hardware is changed; a mapping mechanism reconfigures the internal names to the new hardware settings. Adjustments and calibrations to the new hardware are included in PSM layer model for the new hardware, thus no change in MDM PIC layer, or HTM5 component classes is required.

used in PIM classes of MDM. Adjustments or calibrations for the new machine are also specified in the PSM classes of MDM. The mechanism is further explained in the next section (Section 3.1.4).

3.1.4 HTM5 Machine Descriptor Model

We discussed in Chapter 1 that in agent oriented cloud robotics, HTM5 components (*Agents*, *Merges* or *Relations*) represent host hardware or base software in the cloud robotic ecosystem. The 5 views of HTM5 are for modelling of interactions between these HTM5 components. HTM5-MDM is included in HTM5 for modelling of the hardware or software framework that a HTM5 component is representing in the cloud. These entities could be any of the following:

- A robot with or without an operating system onboard.
- A computer or mobile device where an agent is hosted as an application program.

- A server bank, database system or an existing cloud service.
- Cameras, sensors and ambient intelligence devices.
- A *Close* system, hosting an agent as a controlled interface.
- Virtual devices.
- A subsystem within a hardware or software framework. Note that one machine can host more than one HTM5 components.
- Human actors, Interactive environments and practically any entity that needs a representation in the modelling of cloud robotic ecosystem.

HTM5-MDM is spread across bottom two layers of HTM5 (See Fig. 3.2). The separation between PIM and PSM parts of HTM5-MDM is required to avoid alterations in HTM5 components when a machine is modified or replaced. PIM layer MDM model is an internal hardware model that is utilized by HTM5 components to access the machine. This model extends from the PSM layer model, which is a model for the actual hardware platform (actual machine that an agent is representing). As HTM5 component classes utilize the internal model (By making an object of the MDM PIC class, see 'a' in Fig. 3.2), their namespace requires no change when the hardware platform is changed. A mapping mechanism exists in the hardware platform model to map the machine names and configurations to the corresponding internal names in the PIM layer hardware model (See Fig. 3.3). Using abstract classes and their extension to separate PIM and PSM layers of MDM is one of the mechanisms that can be used to implement the idea presented in Fig. 3.3. HTM5 does not restrict software developers from using some other mechanisms (e.g. Inheritance without the use of abstract classes or using nested classes or member class objects) to implement this idea. We have seen in this section the range of Machines that can be represented in the cloud. In a commercial scenario, these machines are bound to be modified and replaced. A machine description model separated across the platform line will ease software change management in both sides of the platform line. Changes in HTM5 components will not induce major changes in machine software and vice versa.

3.2 HTM5 Views

A view by definition is a representation of the system with concerns specific to a particular stakeholder. In a multi view model it is essential that the designer should have a clear idea about the scope of a particular view. At HTM5-CIM layer, the views have separate ARC diagrams. At lower layers, the views have separate view classes. Every design element actualize some ideas. In a multi view model, the design ideas cannot be specified at random. The ideas that a design element represents, has an associated view and that is where the design element should be specified. In Sections 3.2.1, 3.2.2, 3.2.3, 3.2.4 and 3.2.5 we will present the ideas that are represented by 5 views of the HTM5 meta-model.

3.2.1 HTM5 Structural View

In software engineering, the structural view is generally understood as the structure of a system being modelled. In standalone software systems, structure normally means the internal arrangement of a software product. UML [47] has several diagrams that are used to specify the structure of a system. *Class* and *Component diagrams* in UML specify the internal structure of entities that constitute a system. *Composite structure diagrams* and *Package diagrams* showcase the logical groupings and collaborations between the classes. *Deployment diagrams* specify the hardware used in the system and the execution environments that different entities operate in. HTM5 structural view provides structural details for distributed and heterogeneous cloud robotic systems. Unlike a standalone system, a cloud robotic system is built on entities which are autonomous and intelligent in design. The structural view of HTM5 aims to capture the following design elements of an agent oriented cloud robotic ecosystem:

1. **Names, Physical and Relative Locations** of the following entities in a cloud robotic ecosystem:
 - (a) Machines (as defined in Section 3.1.4)
 - (b) Components (Agents, Merges and Relations)
 - (c) Network Clouds
2. The following **Information** about the above entities:
 - (a) Associations between Components and their host Machines.
 - (b) Connections between Components via Network Clouds.
 - (c) Cardinality of connections when *Merges* are involved.
3. Following **Functionalities** should exclusively go in the Structural view classes of various components:
 - (a) Localization: Locating one's position in the cloud ecosystem.
 - (b) Communication interfaces with associated clouds and agents.

- (c) Maintaining dynamic connections with other components (Especially if the component is a *Merge*).
- (d) Communication interfaces with the Host Machine.
- (e) Representation of the host machine in the cloud ecosystem.
- (f) Generating triggers for Structural Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).

3.2.2 HTM5 Relational View

Previously we have discussed the classification of agents in HTM5 based on their role in the cloud robotic ecosystem. We discussed that although management of agent relationships by a special agent is a violation of their autonomy but presence of *Relations* in HTM5 provides flexibility to the system designer. Agents are an extension to the concept of objects oriented technology, and it is thus normal to see them as a system of entities with some relationships. The relationships are any kind of meaningful associations amongst relatives and thus relationship modelling is one important concept in agent systems. *Relations* are ideal for modelling *Social Concepts* (*Debtor*, *Creditor*, *Action* and *Context*), *Business logic*, *Institutions* [5, 40] and other *Human Concepts* [44, 45]. The relational view of HTM5 separates relationship concerns in a cloud robotic system. The relational view of HTM5 aims to capture the following design elements of an agent oriented cloud robotic ecosystem:

1. **Names** and **Relative Locations** of *Relation* Components in a cloud robotic ecosystem.
2. The following **Information** about the Relationships:
 - (a) Names of Components in Relationships.
 - (b) The role different components play in their relationships.
 - (c) Cardinality of connections around a *Relation*.
 - (d) Trade associated to a particular relationship.
3. Following **Functionalities** should exclusively go in the Relational view classes of various components:
 - (a) Localization: Locating one's position in different relationships.
 - (b) Maintaining Communication with *Relations* and *Relatives*.
 - (c) Establishing, Maintenance and finalization of Relationships.
 - (d) Communication and Enforcement of Relationship norms on Trade logic.
 - (e) Receiving suggestions from Trade logic on readjustment of Relationship norms.
 - (f) Communication and Negotiations (for adjustments in norms of relationships) with *Relations* and *Relatives*.

- (g) Implementation of Social logic of a Component (Functionality related to social concerns of a particular HTM5 Component).
- (h) Implementation of Social logic of the system (Functionality related to social concerns of the cloud robotic system).
- (i) Generating triggers for Relational Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).

3.2.3 HTM5 Trade View

Trade view of HTM5 is for modeling the trade logic in a cloud robotic ecosystem. The meta-model proposes a Peer-to-peer service oriented trade mechanism managed by *Relation* agents. For a system designer, it is important to specify the norms and default trade relationships in these systems. Agent transactions in such systems follow the cloud computing business logic, but unlike current cloud robotic systems, these transactions are driven by decentralized *Relation* agents. In Chapter 1 we discussed how a peer-to-peer cloud robotic system will enable individual robots to offer and avail services from other robots and auxiliary systems. The traditional client server based systems can also be modelled using HTM5 but the meta-model is principally designed to provide tools for an agent driven, peer-to-peer transaction modelling of these systems. In Chapter 2 we discussed that the proposed meta-model also enables incorporation of cloud computing business logic. Pay-per-use, dynamic service and demand discovery, actual transfer of money through banking agents, trust control are all elements of cloud computing business model. The Trade view is for separating trade concerns. In Section 3.4 we will discuss the modelling of per-to-peer trade using the proposed meta-model. The Trade view of HTM5 aims to capture the following design elements of an agent oriented cloud robotic ecosystem:

1. **Names** and **Relative Locations** of the following *Trade* elements in a cloud robotic ecosystem.
 - (a) Components (*Agent*, *Relation* or *Merge*) that are involved in Trade.
 - (b) Items in Trade.
 - (c) Data entities associated with Trade items.
2. The following **Information** about the above entities:
 - (a) Associations between Trade items and Components.
 - (b) Nature of association between a Trade item and a Component:
 - i. Item is a Demand by a Component.
 - ii. Item is a Service provided by a Component.
 - (c) Entities associated with a Trade item:
 - i. Components that provide the item as a service.
 - ii. Components which demand the item.
 - iii. Data entities which are associated with Trade of the item.

- iv. The Components (Generally *Relations*) that are hosting and managing those data entities.
- (d) Nature of various Data entities:
 - i. Is it a Lookup table?
 - ii. Is it a cost metric?
 - iii. Is it a management variable?
- 3. Following **Functionalities** should exclusively go in the Trade view classes of various components:
 - (a) Localization: Locating one's position in different transactions.
 - (b) Identifying relationships associated with a particular trade item.
 - (c) Implementing relationship norms associated with a trade item.
 - (d) Implementation of Business logic of a Component (Functionality related to business concerns of a particular HTM5 Component).
 - (e) Implementation of Business logic of the system (Functionality related to business concerns of the cloud robotic system).
 - (f) Calculating readjustments in relationship norms based on business logic.
 - (g) Communicating desired readjustments to relational view classes.
 - (h) Maintaining data entities associated with a trade item.
 - (i) Reading and updating of remotely hosted data entities associated with a trade item.
 - (j) Generating triggers for Trade Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).

3.2.4 HTM5 Hyperactivity View

We discussed in Chapter 2 the adjustments in the concept of agency that may be required for flexible development of cloud robotic system. We discussed *Hyperactivity* as a mechanism to provide a back channel to access the internal parameters of an agent. In simple terms, *Hyperactivity* allows direct modifications in an agent's internal data, or a forced execution of an agent's internal method (Functionality). This is against the autonomy concept in Agency but allows a simple and controlled way to release an agent's (Component in general) autonomy to specific agents (Components). The *Hyperactivity* view of HTM5 is for modeling of these adjustments in a cloud robotic ecosystem. The *Hyperactivity* view in HTM5 is subdivided into 4 sub-views. The separation into sub-views occur at the bottom two layers of the HTM5 (See Fig. 3.1 and Fig. 3.2). The reasons to separate hyperactivity concerns in a separate view, and subdivision of this view into 4 sub-views are as follows:

- To identify design and code segments that are implementing the hyperactivity mechanism.

- To separate design and code segments that are implementing hyperactivity in individual views.
- Hyperactivity gives special outside access to internal variables of a Component. The mechanism to implement this should be separate from the regular code to avoid inconsistencies.
- From security point of view, a separate view ensures a managed implementation of access control mechanism. This is even more important in agents representing a business interest.
- Subdivision is necessary since the views have their own namespaces, and there is a chance that a variable name is repeated in different views.
- Subdivision will also make it easier to implement separate hyperactivity mechanisms (Chapter 2) for the 4 views.
- Subdivision enhances reusability in view specific hyperactivity classes, and makes it easier to manage change.
- Subdivision is not essential at the CIM layer. But in systems with complicated hyperactivity logic, a designer can make separate *Hyperactivity* ARCs for the sub-views (Section 3.3).

The Hyperactivity view of HTM5 aims to capture the following design elements of an agent oriented cloud robotic ecosystem:

1. **Names and Relative Locations** of the following *Hyperactivity* elements in a cloud robotic ecosystem.
 - (a) Components with Hyperactivity (*Hyperactive* Components, Chapter 2).
 - (b) Components with Activity (*Active* Components, Chapter 2).
 - (c) Components with No Activity (*Passive* Components, Chapter 2).
2. The following **Information** about the above entities:
 - (a) Hyperactive Associations between Components (*Hyperactive Links*).
 - (b) Role of Components in a Hyperactive association (Master or Slave)
 - (c) Nature of Hyperactive Associations:
 - i. One Way Hyperactivity: Active Master and Hyperactive Slave. The hyperactivity mechanism in Active Master does not release its autonomy.
 - ii. Two Way Hyperactivity: Two separate one way hyperactive links exists between a pair of Components. Both are Hyperactive masters and Hyperactive slaves at the same time.

- iii. **Passive Hyperactivity: Passive Master and Hyperactive Slave.**
This is the rare scenario when a hyperactive link originates from a Passive Component. Passive Components do not have an update mechanism for their internal control logic but logic may exist in the hyperactivity mechanism to send hyperactive inputs across hyperactive link. Hyperactivity mechanism in a Passive component does not have stored variables as input (see Fig. 2.3). A simple example of this could be a Boolean hyperactive variable sent across a hyperactive link when a passive component boots up (Powered on).
 - (d) **Goal of a Hyperactive Association:** At CIM layer, the exact names and locations of variables accessed by a hyperactive mechanism are not available. These are specified in the bottom two layers. At CIM layer, the purpose of a hyperactive association is specified as an abstract statement (Text).
- 3. Following **Functionalities** should exclusively go in the Hyperactivity view classes of various components:
 - (a) **Localization:** Locating one's position in different Hyperactive associations.
 - (b) **Localization:** Locating one's activity status (*Passive*, *Active* or *Hyperactive*).
 - (c) Reading Hyperactive inputs from associated components.
 - (d) Implementing Hyperactive inputs from associated components.
 - (e) Generating Hyperactive outputs for associated components.
 - (f) Sending Hyperactive outputs for associated components.
 - (g) Implementing Hyperactivity logic.
 - (h) Initiation, management and finalization of Hyperactive links (Triggered by the other view classes).

3.2.5 HTM5 Behavioural View

Behaviour modelling in software engineering is one of the principle components of system design. A behaviour model of a system specifies the valid behaviour patterns. In UML [47], behaviour modelling is done through *Interaction*, *Activity*, *Use Case* and *State Machine* diagrams. *Use Case* diagrams describe a system's functionality in terms of actors, their goals and functional dependencies. *Sequence* diagrams are a kind of *Activity* diagrams in UML which specifies how objects communicate with each other. *Sequence* diagrams depict the sequence of messages that translate into a usage scenario for the system. HTM5's structural, relational, trade and hyperactivity views specify the functionalities related to the view concerns. HTM5's behaviour view specifies system behaviours that emerge as a result of functionalities specified in the other 4 views. The event scenarios modelled in HTM5's behavioural view are the net result of individual functionalities specified across

various views and in different HTML5 Components (*Agents*, *Merges* and *Relations*). At the CIM layer, the behavioural view specifies the behaviour of the complete cloud robotic system, while at PIM layer it specifies behavioural patterns within a Component.

The CIM layer in Behavioural view consists of Use Case and Sequence ARCs. At PIM layer, there are UML based Use case and sequence diagrams to specify behaviour patterns within a component. The Behaviour view class holds functionalities which give shape to behaviour patterns specified in behavioural view. Functionalities that are not part of any other view, but are necessary for implementation of behaviour are placed in the behavioural view classes. The Behavioural view of HTML5 aims to capture the following design elements of an agent oriented cloud robotic ecosystem:

1. The following **Information** about **System's** behaviour (Inter-Component Behaviour):
 - (a) Identifying various Use Cases of the system by **Name**.
 - (b) External *Actors* to a **System** in various Use Case scenarios.
 - (c) Internal *Actors* (Components) associated with Use Cases.
 - (d) Roles of External and Internal *Actors* in each Use Case scenario.
 - (e) The sequence of events, and messages passed between *Actors* in all Use Case scenarios.
2. The following **Information** about a **Component's** behaviour (Inter-View Behaviour):
 - (a) Identifying various Use Cases of the Component by **Name**.
 - (b) External *Actors* to a **View** in various Use Case scenarios.
 - (c) Internal *Actors* (*Views*) associated with Use Cases.
 - (d) Roles of External and Internal *Actors* in each Use Case scenario.
 - (e) The sequence of events, and messages passed between *Actors* in all Use Case scenarios.
3. The following **Information** about a **View's** behaviour (Intra-View Behaviour):
 - (a) Identifying various Use Cases of the View by **Name**.
 - (b) External *Actors* to a **Method** (Code Unit) in various Use Case scenarios.
 - (c) Internal *Actors* (*Method*) associated with Use Cases.
 - (d) Roles of External and Internal *Actors* in each Use Case scenario.
 - (e) The sequence of events, and messages passed between *Actors* in all Use Case scenarios.
4. Following **Functionalities** should exclusively go in the Behavioural view classes of various components:

- (a) Functionalities that are not directly concerned with any of the other 4 views.
- (b) Functionalities which manage a sequence of events, unless it's already specified in any of the 4 other views.
- (c) The 'Main' routine implementation and 'Booting-up' Views (Behavioural view could act as the *System Routine*, in case the PIC* or PSC class is not implemented as the *System Routine*)
- (d) Generating triggers for Behavioural Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).

3.2.6 Functionalities in PIC*, PIC and PSC Component Classes

The view classes for a HTM5 component is enveloped by PIC* or PSC class when PIC class is *Abstract* and by PIC class when there is no platform level separation (See Sections 3.1.2, 3.1.3 and Fig. 3.2). Following are the functionalities which can exist in these classes apart from functionalities that they inherit by enveloping the objects of the view classes:

1. The 'Main' routine implementation and 'Booting-up' Views.
2. Management of the Host hardware when there is no software layer between the HTM5 Component and the Hardware.
3. Management of threads when more than one thread are initiated on the Component.
4. Functionalities that are not modelled at the CIM layer, but were required to be added at the time of implementation.
5. Code to implement 'Quick Fix' solutions at the time of testing (or deployment), without creating disorder in the code for view classes.

A summary of functionalities concerned with the 5 views and the Component classes is presented in the Tables 3.1, 3.2 and 3.3 .

Table 3.1: A summary of Functionalities associated with the 5 views and PIC*/PIC/PSC Classes of HTM5 (PART 1).

Structural View 3.2.1
<ol style="list-style-type: none"> 1. Localization: Locating one's position in the cloud ecosystem. 2. Communication interfaces with associated clouds and agents. 3. Maintaining dynamic connections with other components (Especially if the component is a <i>Merge</i>). 4. Communication interfaces with the Host Machine. 5. Representation of the host machine in the cloud ecosystem. 6. Generating triggers for Structural Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).
Relational View 3.2.2
<ol style="list-style-type: none"> 1. Localization: Locating one's position in different relationships. 2. Maintaining Communication with <i>Relations</i> and <i>Relatives</i>. 3. Establishing, Maintenance and finalization of Relationships. 4. Communication and Enforcement of Relationship norms on Trade logic. 5. Receiving suggestions from Trade logic on readjustment of Relationship norms. 6. Communication and Negotiations (for adjustments in norms of relationships) with Relations and Relatives. 7. Implementation of Social logic of a Component (Functionality related to social concerns of a particular HTM5 Component). 8. Implementation of Social logic of the system (Functionality related to social concerns of the cloud robotic system). 9. Generating triggers for Relational Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).

Table 3.2: A summary of Functionalities associated with the 5 views and PIC*/PIC/PSC Classes of HTM5 (PART 2).

Trade View 3.2.3
<ol style="list-style-type: none"> 1. Localization: Locating one's position in different transactions. 2. Identifying relationships associated with a particular trade item. 3. Implementing relationship norms associated with a trade item. 4. Implementation of Business logic of a Component (Functionality related to business concerns of a particular HTM5 Component). 5. Implementation of Business logic of the system (Functionality related to business concerns of the cloud robotic system). 6. Calculating readjustments in relationship norms based on business logic. 7. Communicating desired readjustments to relational view classes. 8. Maintaining data entities associated with a trade item. 9. Reading and updating of remotely hosted data entities associated with a trade item. 10. Generating triggers for Trade Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).
Hyperactive View 3.2.4
<ol style="list-style-type: none"> 1. Localization: Locating one's position in different Hyperactive associations. 2. Localization: Locating one's activity status (<i>Passive</i>, <i>Active</i> or <i>Hyperactive</i>). 3. Reading Hyperactive inputs from associated components. 4. Implementing Hyperactive inputs from associated components. 5. Generating Hyperactive outputs for associated components. 6. Sending Hyperactive outputs for associated components. 7. Implementing Hyperactivity logic. 8. Initiation, management and finalization of Hyperactive links (Triggered by the other view classes).

Table 3.3: A summary of Functionalities associated with the 5 views and PIC*/PIC/PSC Classes of HTM5 (PART 3).

Behavioural View 3.2.5
1. Functionalities that are not directly concerned with any of the other 4 views.
2. Functionalities which Manages a sequence of events, unless it's already specified in any of the 4 other views.
3. The 'Main' routine implementation and 'Booting-up' Views (Behavioural view could act as the <i>System Routine</i> , in case the PIC* or PSC class is not implemented as the <i>System Routine</i>)
4. Generating triggers for Behavioural Hyperactivity sub-view class. (Initiation, management or finalization of a Hyperactive link).
Enveloping Component Classes 3.2.6
1. The 'Main' routine implementation and 'Booting-up' Views.
2. Management of the Host hardware when there is no software layer between the HTM5 Component and the Hardware.
3. Management of threads when more than one thread are initiated on the Component.
4. Functionalities that are not modelled at the CIM layer, but were implementational layer obligation.
5. Code to implement 'Quick Fix' solutions at the time of testing (or deployment), without creating disorder in the code for view classes.

3.3 HTM5 Agent Relation Charts (ARCs)

Agent Relation Charts are a set of graphical models which constitute the CIM layer of HTM5 (See Fig. 3.2 and Fig. 3.3). In 3.1.1 we introduced different kinds of ARC diagrams and the views associated with each of them. In Sections 3.2.1, 3.2.2, 3.2.3, 3.2.4 and 3.2.5 we have discussed the ideas that are represented by the 5 views of HTM5. The ARCs that are associated with these views are used to specify the design elements that implement those ideas. We have taken a '**Object Miner**' Cloud robotic system as an example to showcase various elements of the ARC diagrams. Attributes of '**Object Miner**':

- **Machines:**

1. Trader Machines: [Mobile Computers] [Count: Na]
2. Miner: [Mobile Robots] [Count: $N = N_c + (N_{b1} + N_{b2} + N_{b3} + \dots + N_{bNa})$]
3. Collector Server: [Web Server] [Count: 1]
4. Agency Server: [Local Server] [Count: 1]

- **Network Clouds:**

1. Cloud-1: [Local Wireless Network]
2. Cloud-2: [WWW]

- **Component Types:**

1. Agents: [Miner] [Trader][Collector][Agency]
2. Relations: [R1] [R2] [R3] [R4]
3. Merges: [M1] [M2] [M3] [M4]

- **Trade Items:** *Item:* [Available At] \Rightarrow [Demanded At]

1. Mine Coordinates: [Collector] \Rightarrow [M1]
2. Cargo: [Trader] \Rightarrow [Collector]
3. Search Space: [M1] \Rightarrow [Trader]; [Trader] \Rightarrow [M4]
4. Cargo Payment: [Collector] \Rightarrow [Trader]
5. Sub Search Space: [Trader] \Rightarrow [Miner]
6. Trader ID: [Agency] \Rightarrow [Miner]
7. Miner ID: [Agency] \Rightarrow [Trader]
8. Initial Location: [Agency] \Rightarrow [Miner]
9. Initial Coordinates: [M4] \Rightarrow [Trader]
10. Miner Salary: [Trader] \Rightarrow [Miner]
11. Target Minerals: [M4] \Rightarrow [Miner]
12. Sub Space Hit: [Miner] \Rightarrow [M4]

- **External Actor:** Mine Field
- **Trade Data:** *Item: [Type] [Location]*
 1. Mineral l Price: [Demand Lookup Table] [R1]
 2. Trader ID l Cargo: [Service Lookup Table] [R1]
 3. Miner ID l Salary: [Demand Lookup Table] [R4]
 4. Mineral ID: [Data entity] [R4]
 5. Location l Mineral ID: [Service Lookup Table] [R4]
 6. Mined Area % :[Data entity] [R4]
 7. Mineral ID l Hz:[Service Cost Metric] [R4]
 8. Trader ID l NumMiner:[Demand Lookup Table] [R2]
 9. Trader ID l Coordinates: [Demand Cost Metric] [R2]
 10. Trader ID l Salary: [Demand Cost Metric] [R2]
 11. Miner ID: [Data entity] [R3]
 12. Miner ID l Job Status: [Service Cost Metric] [R3]
 13. Miner ID l Min. Salary: [Service Cost Metric] [R3]
 14. Pipeline Trader ID l Salary: [Demand Cost Metric] [R3]
- **See APPENDIX B for a detailed explanation of 'Object Miner' Project**

In this section we will describe different Agent Relation Charts, and the way they are used to specify design elements as a graphical model. The text based attributes mentioned above is a abstract and computation independent description of the '**Object Miner**' Cloud robotic system. These attributes can be separated in various views using ARC diagrams.

3.3.1 Agent Relation Charts (ARC)

ARC diagrams are for specification of *Structural* and *Relational* concerns of a cloud robotic system. *Machines* are located and named as *Rectilinear Polygon* enveloping the *Components* that it is hosting. In case a *Machine* is hosting only one *Component*, the enveloping polygon is not drawn. *Agents* are represented as *Rectangles*, *Relations* as *Rhombuses* and *Merges* are represented as *Isosceles triangles*. *Network Clouds* are represented as *Circles* and are placed on the *Links* that are utilising that network(See Fig. 3.4). Similar *Components* which do not have complex interactions amongst themselves can be represented as one *Component* in **Normalized versions** of ARC diagrams (See Fig. 3.5). *Components* in a relationship are attached to different ports of a *Relation Component*. A comparison of ARC elements with view concerns is summarized in Table 3.4.

3.3.2 Trade - Agent Relation Charts (T-ARC)

T-ARC diagrams are for specification of *Trade* concerns of a cloud robotic system. *Components*, *Services*, *Demands*, *Data entities* and *Trade links* are Named and Located in T-ARC diagrams (See Fig. 3.6). Trade items are associated to various components as a *Service* provided by them, or a *Demand*. Data entities associated with a *Trade* item can be of different types. Trade Data could be implemented as lookup tables, cost metrics or trade variable. Trade variables are data entities associated to a trade item as demand or service management variables. The nature of association between a Component and the trade item is depicted by the shape of the Item (Angular Box: Service; Box with one corner truncated: Demand; Parallelogram: Data item See Fig. 3.6). Data entities associated with Trade items are hosted by the Relations managing their trade. A comparison of T-ARC elements with view concerns is summarized in Table 3.6. The implementation of peer-to-peer trade logic in a cloud robotic system is discussed in the Section 3.4 of this thesis.

3.3.3 Hyperactivity - Agent Relation Charts (H-ARC)

H-ARC diagrams are for specification of *Hyperactivity* concerns of a cloud robotic system. *Components*, Their *Hyperactivity Status*, *Hyperactivity Links* and *Statements about Hyperactivity* are named and located in H-ARC diagrams. A *Component* is *Hyperactive* if there is at least one *Hyperactivity link* directed towards it. *Active Components* are depicted by a *Star* attached to them in H-ARC diagrams (See Fig. 3.7). Components with no star attached to them are *Passive Components*. There are three kinds of *Hyperactive Associations* between HTM5 components. When a one directional Hyperactive link exists between two components, the starting point of the link is the Master component and the end point of the link is the hyperactive Slave. In *One Way Hyperactive association*, the link is from an *Active Component* towards a *Hyperactive Component*. In *Two Way Hyperactive association*, two separate hyperactive links exist between a pair of Components. The two links are in opposite directions and thus both Components are Hyperactive Slaves and masters at the same time. The third kind of Hyperactive association is called *Passive Hyperactive association*. In *Passive hyperactive association*, the hyperactivity link originates from a *Passive Component*, towards a *Hyperactive Component*. Goal of a Hyperactive Association is specified by *Abstract statements (Text)* on the hyperactive links. A comparison of H-ARC elements with view concerns is summarized in Table 3.8.

3.3.4 Use Case - Agent Relation Charts (U-ARC)

U-ARC diagrams are for specification of *Behavioural* concerns of a cloud robotic system. Use case Scenarios of System's behaviour are Named and Specified in Use Case-ARCs. External Actors to a System are named and Located and behaviour of Internal Actors are specified in Use Case-ARC dia-

grams. For every Use Case Scenario, a different U-ARC is built (See Fig. 3.8 and 3.9). *Action bubbles* are drawn around every interaction link in a Use Case ARC to specify the intended action between a pair of Actors. U-ARCs are similar to UML's Use Case diagrams, but unlike Use Case diagrams, the Actors are agents representing different machines and not modules within a software product. A comparison of U-ARC elements with view concerns is summarized in Table 3.10.

3.3.5 Sequence - Agent Relation Charts (S-ARC)

S-ARC diagrams are for specification of *Behavioural* concerns of a cloud robotic system. Use case Scenarios of System's behaviour are Named and Specified in Use Case-ARCs. For every Use Case, a Sequence ARC is drawn to specify the sequence of messages and events associated to the scenario. *Action sequences* are drawn around every Actor in a Sequence ARC to specify the intended action between a set of Actors (See Fig. 3.10 and 3.11). S-ARCs are similar to UML's sequence diagrams, but unlike Sequence diagrams, the Actors are agents representing different machines and not modules within a software product. A comparison of S-ARC elements with view concerns is summarized in Table 3.10.

In this section we presented various ARC diagrams in CIM layer of HTM5 meta-model. A graphical representation of a system's attributes and separation of concerns in 5 views makes ARC diagrams a useful model for agent oriented development of cloud robotic systems.

Table 3.4: Concerns in Structural and Relational Views of HTM5 and corresponding Design Elements in ARC.(Part 1)

Structural and Relational View Concerns 3.2.1, 3.2.2
<p>Structural View</p> <ol style="list-style-type: none"> 1. Names, Physical and Relative Locations of the following entities in a cloud robotic ecosystem: <ol style="list-style-type: none"> (a) Machines (as defined in Section 3.1.4) (b) Components (Agents, Merges and Relations) (c) Network Clouds 2. The following Information about the above entities: <ol style="list-style-type: none"> (a) Associations between Components and their host Machines. (b) Connections between Components via Network Clouds. (c) Cardinality of connections when <i>Merges</i> are involved.
Equivalence In ARC Diagrams 3.3.1
<ol style="list-style-type: none"> 1. Machines are located and named as Rectilinear Polygons enveloping the Components that they are hosting (e.g. Fig. 3.4E9...E10: <i>'Collector Server'</i>). 2. Components are located and named as follows: <ol style="list-style-type: none"> (a) Rectangles: Agents e.g. Fig. 3.4E1: <i>'Miner' Agent</i> (b) Rhombuses: Relations e.g. Fig. 3.4A5: <i>'R4' Relation</i> (c) Triangles: Merges e.g. Fig. 3.4A4: <i>'M4' Merge</i> 3. Network Clouds are located and named as Circles (e.g. Fig. 3.4A3,D8: <i>'1' and '2' are separate Network Clouds</i>). 4. Components associated to a machine are enveloped by the polygon representing the machine (e.g. Fig. 3.4E9...E10: <i>'Collector Server' hosts 3 Components</i>). In case the Machine hosts just one Component, the Component itself represents the Machine. 5. Connections are Lines between Components, marked by the Circle representing the Network Used (e.g. Fig. 3.4B9: <i>'Miner' Agent is connected to 'M3' Merge via '1' Network</i>).

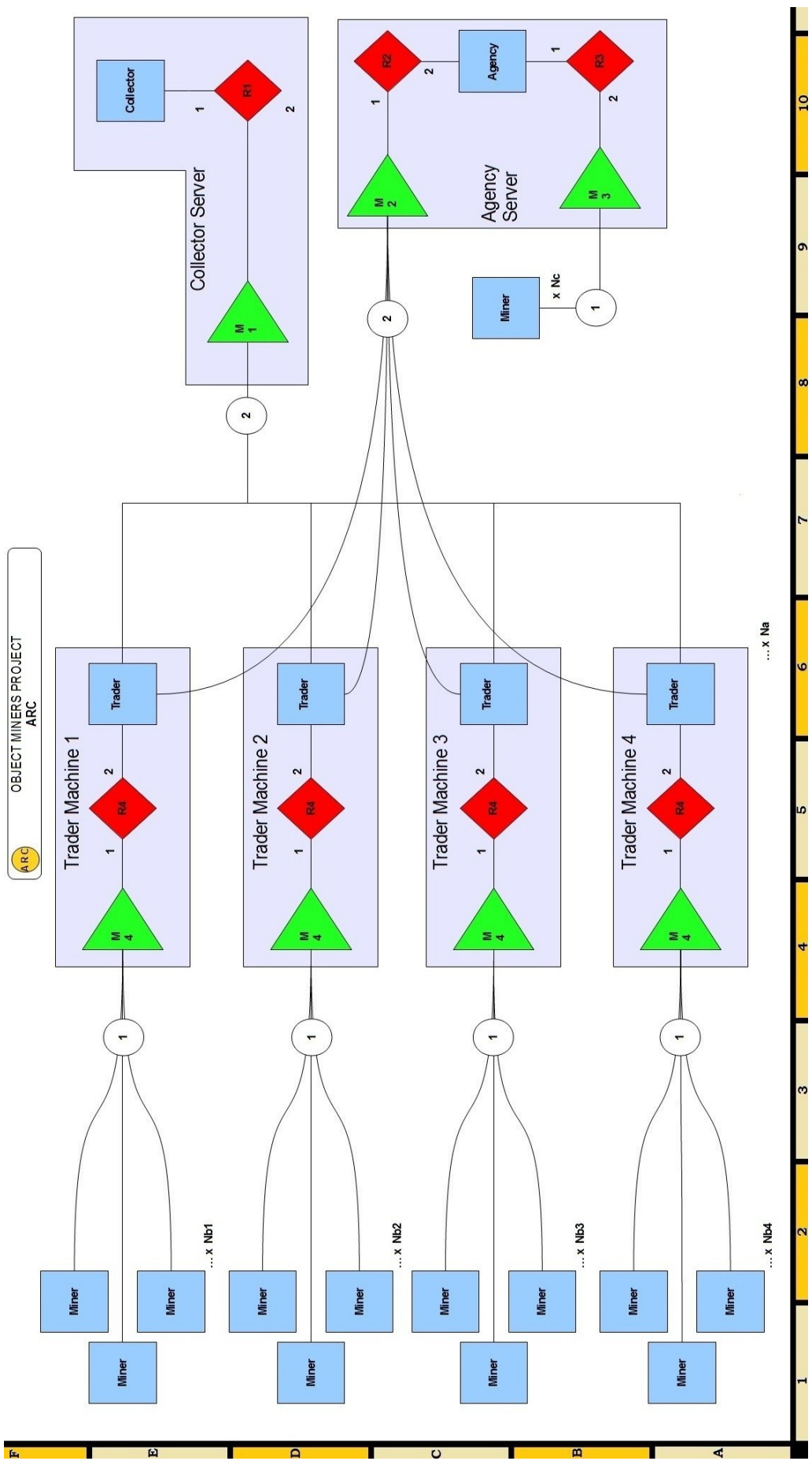


Figure 3.4: An ARC diagram specifying Structural and Relational elements of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.4 For Description). To Readers: Objects are referenced in the thesis text using 2 axis references.

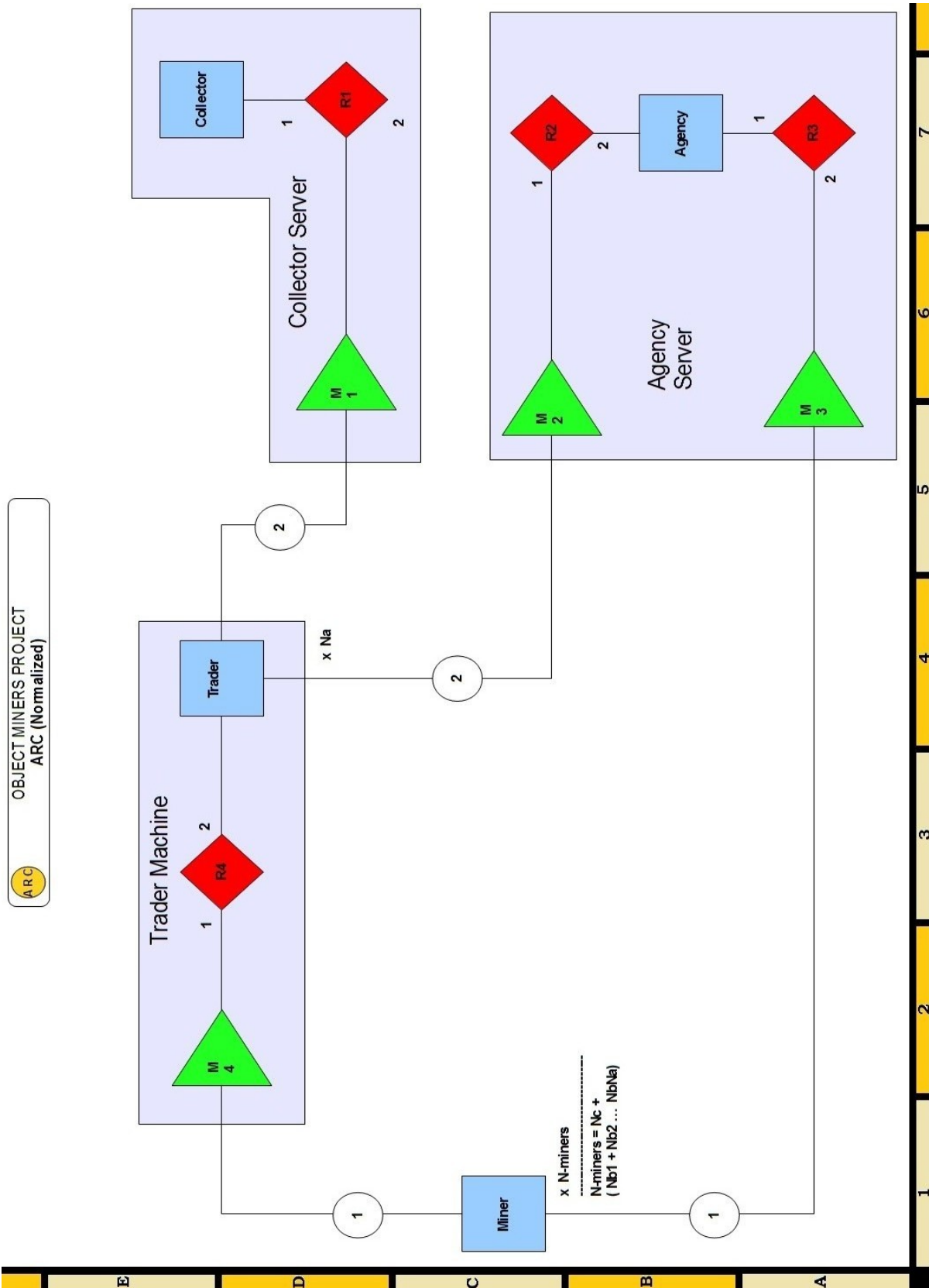


Figure 3.5: A Normalized version of the ARC diagram shown in Fig. 3.4. The Normalized version of an ARC diagram represents a simplified version of the cloud robotic system. Multiple instances of similar Components and Machines are removed in a normalized version. This version is ideal for representing the system when components with more than one instance ('Miner', 'Trader Machine' in the above example) have **Identical Behaviour**. In case multiple instances of a Component Interact extensively with each other, a normalized ARC will prove less useful. To Readers: Objects are referenced in the thesis text using 2 axis references.

Table 3.5: Concerns in Structural and Relational Views of HTM5 and corresponding Design Elements in ARC. (Part 2)

Structural and Relational View Concerns 3.2.1, 3.2.2
<p>Relational View</p> <ol style="list-style-type: none"> 1. Names and Relative Locations of <i>Relation</i> Components in a cloud robotic ecosystem. 2. The following Information about the Relationships: <ol style="list-style-type: none"> (a) Names of Components in Relationships. (b) The role different components play in their relationships. (c) Cardinality of connections around a <i>Relation</i>.
Equivalence In ARC Diagrams 3.3.1
<ol style="list-style-type: none"> 6. Cardinality of connections: (e.g. Fig. 3.4B9: <i>'Miner' Represents 'Nc' similar Entities attached to Merge 'M3'</i>; Fig. 3.4C: <i>Column 4,5,6: 'Na' Trader Machines attached to Merge 'M2'</i>). 7. Components attached to a Relation are in a relationship (e.g. Fig. 3.4B10). 8. A Component's role in a relationship depends on the port at which it is attached to a relation (e.g. Fig. 3.4A4...A6: <i>'M4' Merge is attached to port '1' of Relation 'R4', and 'Trader' Agent to port '2'</i>). 9. Cardinality in Relations: More than 4 links can be attached to a relation provided they are uniquely identified by their port numbers. Alternatively, to avoid confusions, the designer may Merge all relatives to a single port (e.g. Fig. 3.4E8: <i>Merge 'M1' Merges 'Na' instances of 'Trader' Agent to port '2' or Relation 'R1'</i>).

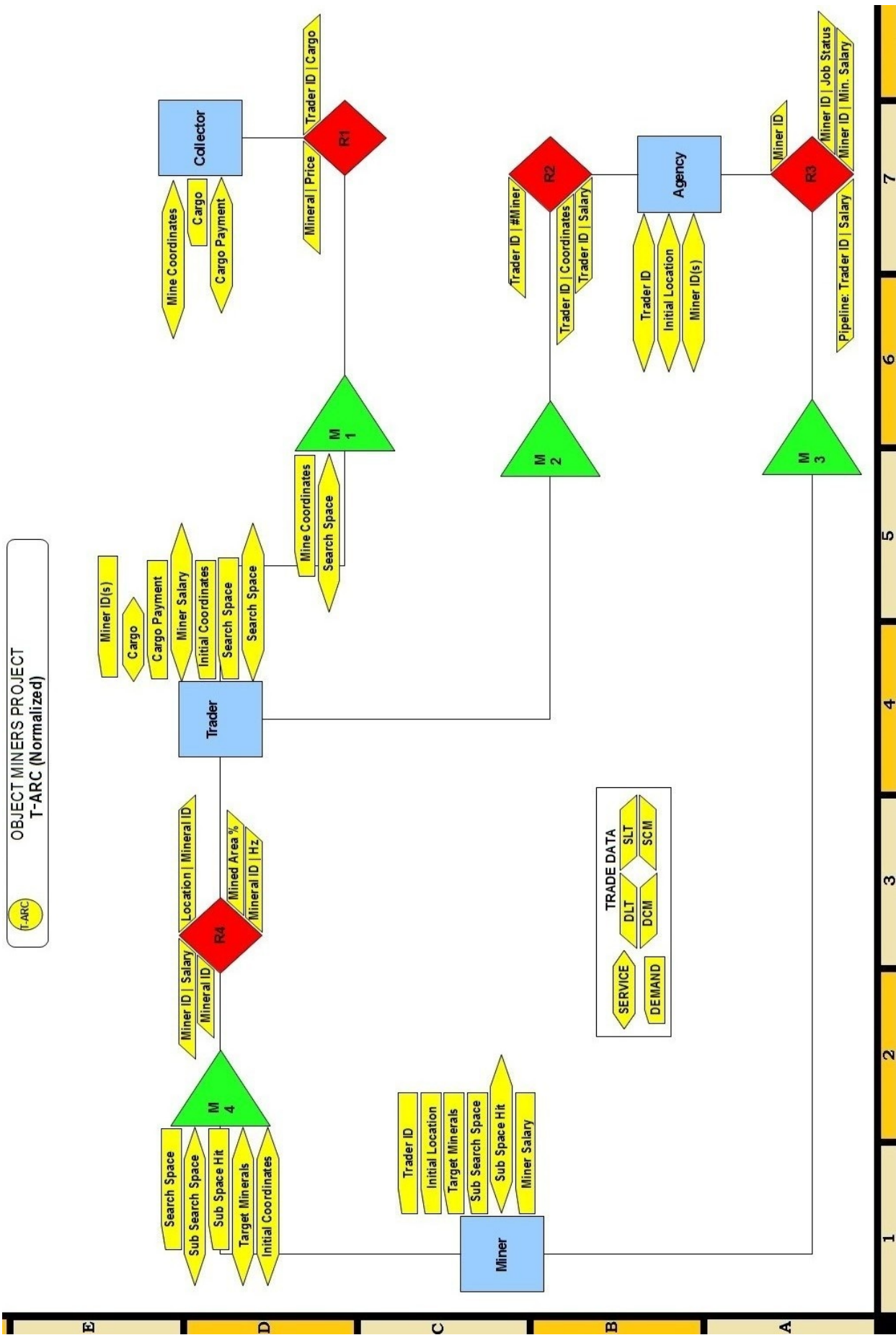


Figure 3.6: A Trade-ARC diagram specifying Trade elements of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.6 For Description). To Readers: Objects are referenced in the thesis text using 2 axis references.

Table 3.6: Concerns in Trade View of HTM5 and corresponding Design Elements in Trade-ARC. (Part 1)

Trade View Concerns 3.2.3
<p>1. Names and Relative Locations of the following <i>Trade</i> elements in a cloud robotic ecosystem.</p> <ul style="list-style-type: none"> (a) Components (<i>Agent</i>, <i>Relation</i> or <i>Merge</i>) that are involved in Trade. (b) Items in Trade. (c) Data entities associated with Trade items.
Equivalence In Trade-ARC Diagrams 3.3.2
<p>1. Components which are involved in Trade are located in the Trade-ARC (e.g. Fig. 3.6 <i>Components, Services, Demands, Data entities and trade links are Named and Located</i>).</p> <p>2. Items in Trade are located as Services (or Demands) in Trade-ARC (e.g. Fig. 3.6C1: <i>Item 'Trader ID' is a demand while Item 'Sub Space Hit' is a service associated with the 'Miner' Agent</i>).</p> <p>3. Data Entities are located as Lookup Tables, Cost Metrics or Trade Variables (e.g. Fig. 3.6D3, B3: <i>'Miner ID 1 Salary' is a Demand Lookup Table; 'Mineral ID 1 Hz' is a Service Cost Metric and 'Mined Area %' is a Data Entity</i>).</p> <p>4. Trade items associated with a Component are placed attached to them.</p>

Table 3.7: Concerns in Trade View of HTM5 and corresponding Design Elements in Trade-ARC. (Part 2)

Trade View Concerns 3.2.3
<p>2. The following Information about the above entities:</p> <ul style="list-style-type: none"> (a) Associations between Trade items and Components. (b) Nature of association between a Trade item and a Component: <ul style="list-style-type: none"> i. Item is a Demand by a Component. ii. Item is a Service provided by a Component. (c) Entities associated with a Trade item: <ul style="list-style-type: none"> i. Components that provide the item as a service. ii. Components which demand the item. iii. Data entities those are associated with Trade of the item. iv. The Components (Generally <i>Relations</i>) that are hosting and managing those data entities. (d) Nature of various Data entities: <ul style="list-style-type: none"> i. Is it a Lookup table? ii. Is it a cost metric? iii. Is it a management variable?
Equivalence In Trade-ARC Diagrams 3.3.2
<p>5. The nature of association between a Component and the trade item is depicted by the shape of the Item (Fig. 3.6B3).</p> <p>6. Locations of Data Entities associated with a Trade Item is depicted by their placement. Data entities associated with Trade items are hosted by the Relations managing their trade (e.g. Fig. 3.6D7: <i>Data entities associated to 'Cargo' and 'Cargo Payment' trade items are hosted at 'R1' Relation</i>).</p> <p>7. Nature of a data entity may be depicted by their placement around a Relation. This however can also be achieved by the way they are named (e.g. Fig. 3.6D3, B3: <i>'Miner ID 1 Salary' is a Demand Lookup Table; 'Mineral ID 1 Hz' is a Service Cost Metric and 'Mined Area %' is a Data Entity</i>).</p>

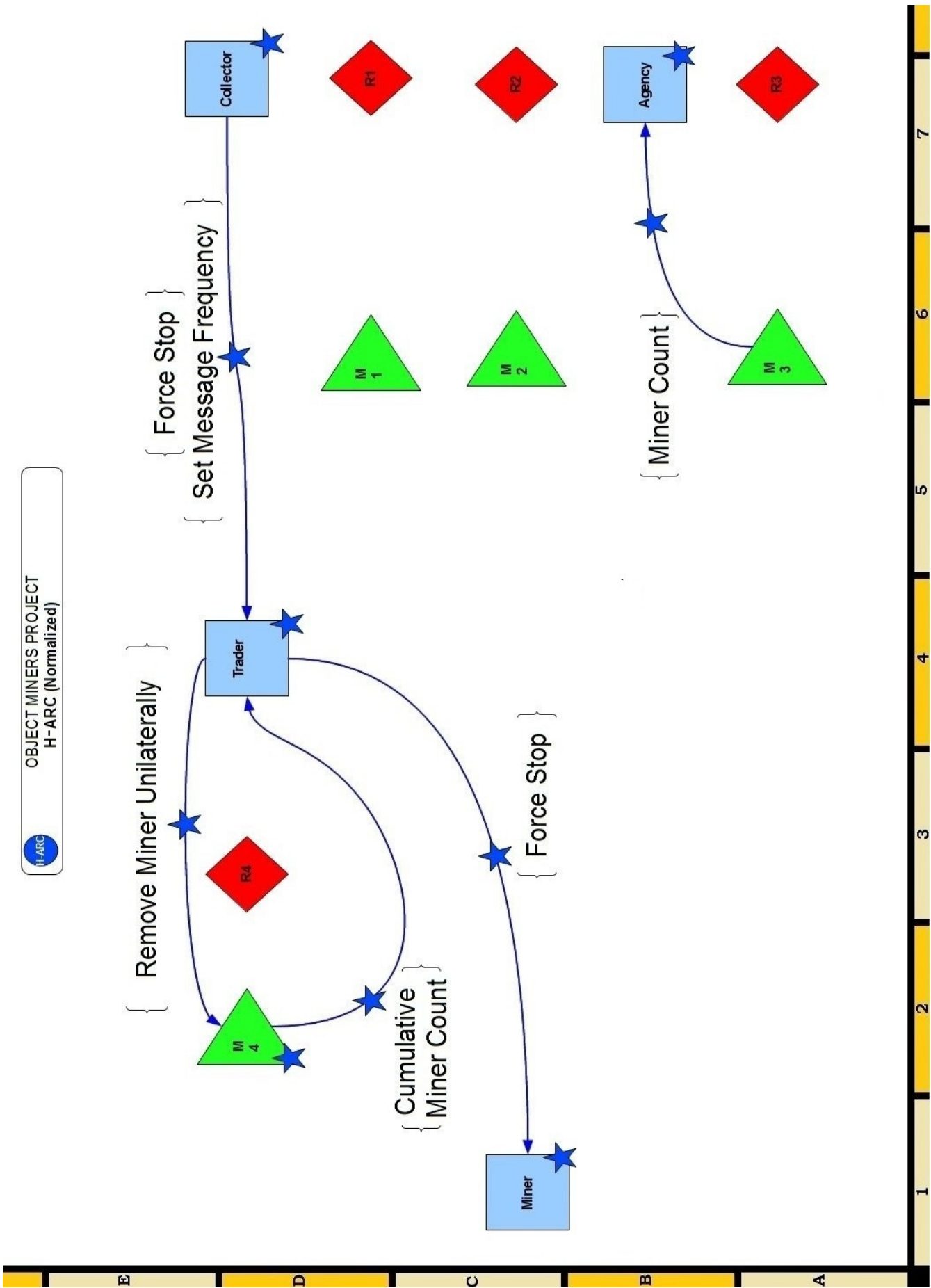


Figure 3.7: A Hyperactivity-ARC diagram specifying Hyperactivity elements of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.8 For Description). To Readers: *Objects* are referenced in the thesis text using 2 axis references.

Table 3.8: Concerns in Hyperactivity View of HTM5 and corresponding Design Elements in Hyperactivity-ARC. (Part 1)

Hyperactivity View Concerns 3.2.4
<p>1. Names and Relative Locations of the following <i>Hyperactivity</i> elements in a cloud robotic ecosystem.</p> <ul style="list-style-type: none"> (a) Components with Hyperactivity (<i>Hyperactive</i> Components, Chapter 2). (b) Components with Activity (<i>Active</i> Components, Chapter 2). (c) Components with No Activity (<i>Passive</i> Components, Chapter 2).
Equivalence In Hyperactivity-ARC Diagrams 3.3.3
<ol style="list-style-type: none"> 1. Components with their Hyperactivity status are located in the Hyperactivity-ARC (e.g. Fig. 3.7 <i>Components, Their Hyperactivity Status, Hyperactivity Links and Statements about Hyperactivity links are Named and Located</i>). 2. Hyperactive Components are those attached to a Hyperactive Link directed towards them (e.g. Fig. 3.7 <i>C1: 'Miner' is a Hyperactive Agent</i>). 3. Active Components are depicted by a Star attached to them (e.g. Fig. 3.7 <i>E7: 'Collector' is a Active Agent</i>). 4. Passive Components are depicted by NO Star attached to them (e.g. Fig. 3.7 <i>D6,D7: 'M1' is a Passive Merge; 'R1' is a Passive Relation</i>). 5. Hyperactive Associations are depicted by Hyperactive links. Hyperactive links are links with a Star on them (e.g. Fig. 3.7 <i>D5...D7: A Hyperactivity link between 'Collector' Agent and 'Trader' Agent</i>).

Table 3.9: Concerns in Hyperactivity View of HTM5 and corresponding Design Elements in Hyperactivity-ARC. (Part 2)

Hyperactivity View Concerns 3.2.4
<p>2. The following Information about the above entities:</p> <ul style="list-style-type: none"> (a) Hyperactive Associations between Components (<i>Hyperactive Links</i>). (b) Role of Components in a Hyperactive association (Master or Slave) (c) Nature of Hyperactive Associations: <ul style="list-style-type: none"> i. One Way Hyperactivity: Active Master and Hyperactive Slave. The hyperactivity mechanism in Active Master does not release its autonomy. ii. Two Way Hyperactivity: Two separate one way hyperactive links exists between a pair of Components. Both are Hyperactive masters and Hyperactive slaves at the same time. iii. Passive Hyperactivity: Passive Master and Hyperactive Slave. This is the rare scenario when a hyperactive link originates from a Passive Component. Passive Components do not have an update mechanism for their internal control logic but logic may exist in the hyperactivity mechanism to send hyperactive inputs across hyperactive link. Hyperactivity mechanism in a Passive component does not have stored variables as input (see Fig. 2.3). A simple example of this could be a Boolean hyperactive variable sent across a hyperactive link when a passive components boots up (Powered on). (d) Goal of a Hyperactive Association: At CIM layer, the exact names and locations of variables accessed by a hyperactive mechanism are not available. These are specified in the bottom two layers. At CIM layer, the purpose of a hyperactive association is specified as an abstract statement (Text).
Equivalence In Hyperactivity-ARC Diagrams 3.3.3
<p>6. Role of a Component in a Hyperactive association is depicted by the Arrow Head of the link. The link originates from Master, and ends at the Slave Component (e.g. Fig. 3.7 D5...D7: <i>'Collector' Agent is Master while 'Trader' Agent is Slave</i>).</p> <p>7. Nature of Hyperactive Associations:</p> <ul style="list-style-type: none"> (a) One Way Hyperactivity: Fig. 3.7 D4...D7: <i>'Collector' is Active, 'Trader' is Hyperactive</i>. (b) Two Way Hyperactivity: Fig. 3.7 D2...D4: <i>'M4' is Hyperactive, 'Trader' is Hyperactive and a pair of hyperactivity links exists between them</i>. (c) Passive Hyperactivity: Fig. 3.7 A6...B7: <i>'M3' is Passive, 'Agency' is Hyperactive</i>. <p>8. Goal of a Hyperactive Association is specified by abstract statements (Text) on the hyperactive links e.g. (Fig. 3.7 E2...E4: <i>Hyperactivity link from 'M4' to 'Trader' is associated to the goal of 'Remove Miner Unilaterally'</i>).</p>

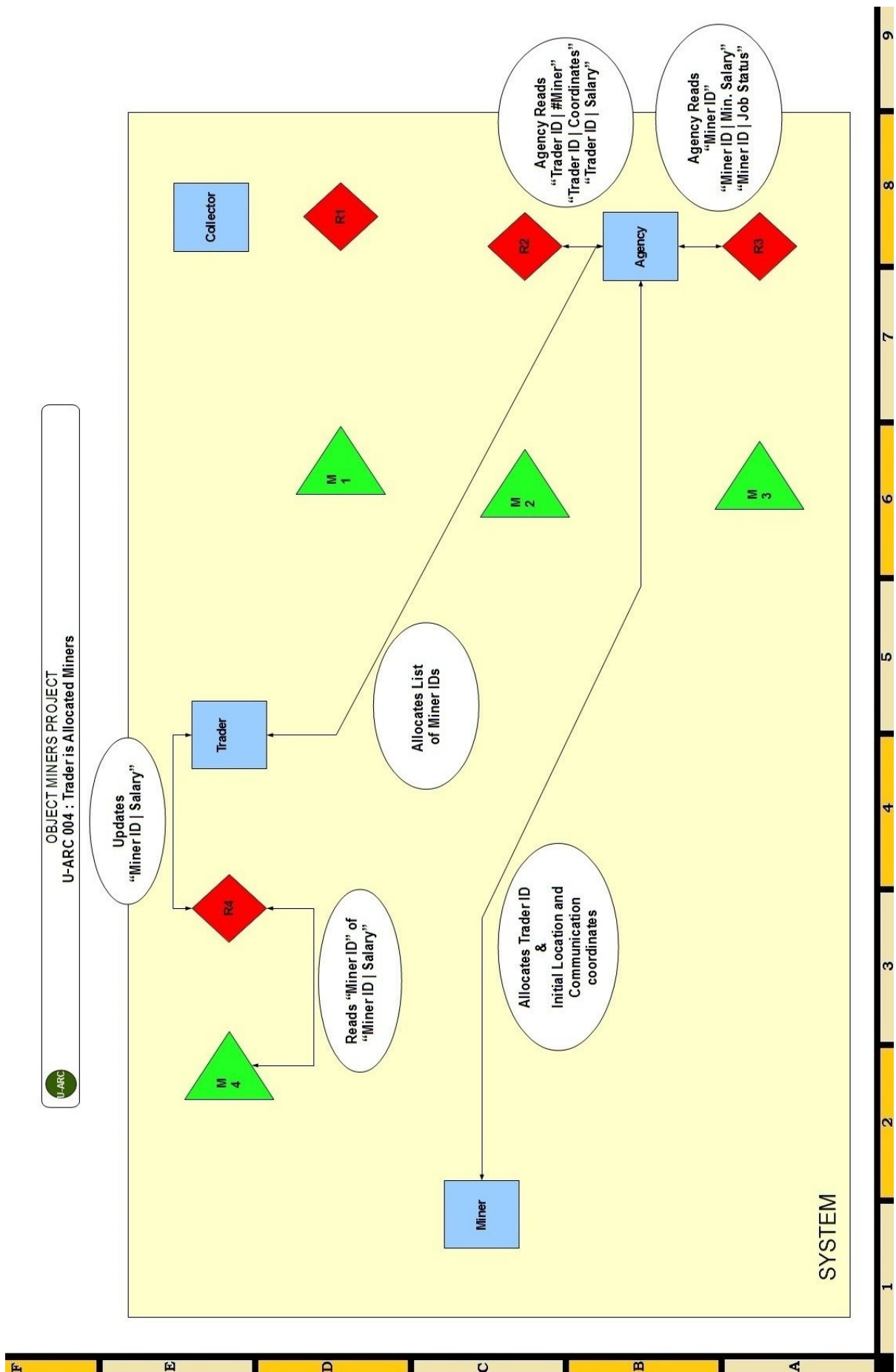


Figure 3.8: A Use Case-ARC diagram specifying Use Case 'Trader is Allocated Miners' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). To Readers: Objects are referenced in the thesis text using 2 axis references.

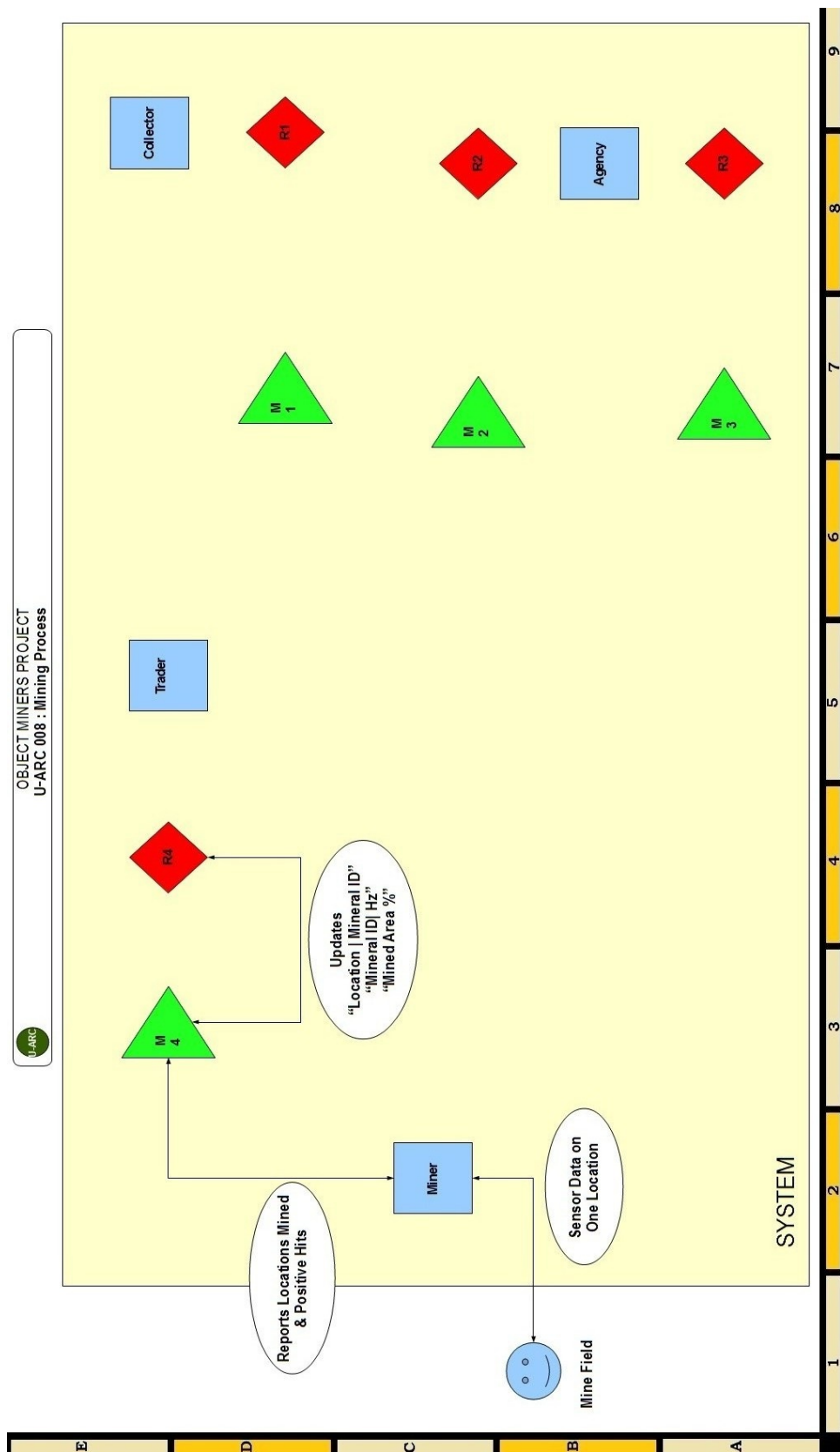


Figure 3.9: A Use Case-ARC diagram specifying Use Case 'Mining Process' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). To Readers: Objects are referenced in the thesis text using 2 axis references.

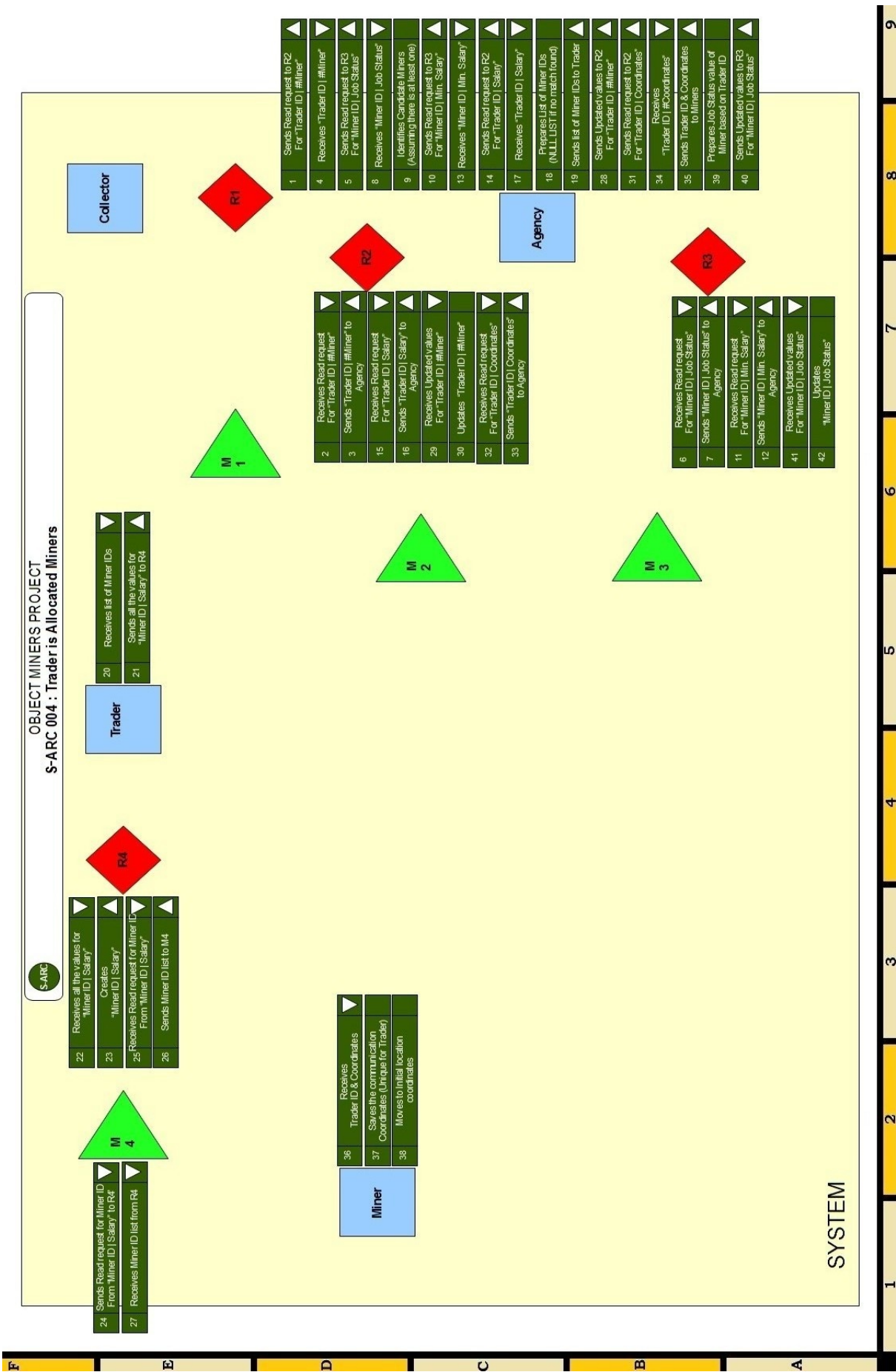


Figure 3.10: A Sequence-ARC diagram specifying Use Case 'Trader is Allocated Miners' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). To Readers: Objects are referenced in the thesis text using 2 axis references.

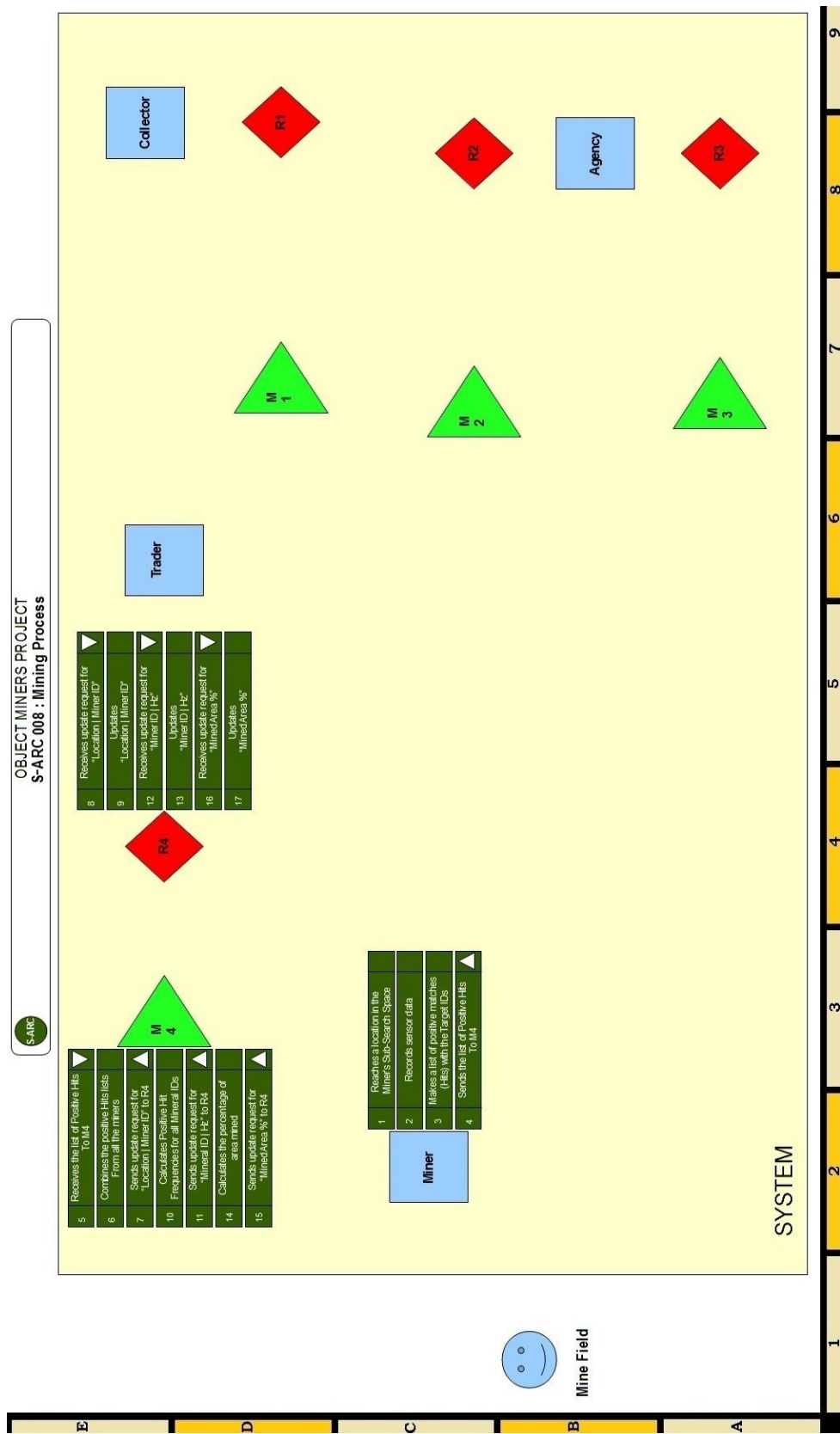


Figure 3.11: A Sequence-ARC diagram specifying Use Case 'Mining Process' of a Cloud Robotic System named 'Object Miner Project'. (See Table. 3.10 For Description). To Readers: Objects are referenced in the thesis text using 2 axis references.

Table 3.10: Concerns in Behavioural View of HTM5 and corresponding Design Elements in Use Case and Sequence ARCs. (Part 1)

Behavioural View Concerns 3.2.5
<ol style="list-style-type: none"> 1. The following Information about System's behaviour (Inter-Component Behaviour): <ol style="list-style-type: none"> (a) Identifying various Use Cases of the system by Name. (b) External <i>Actors</i> to a System in various Use Case scenarios. (c) Internal <i>Actors</i> (Components) associated with Use Cases. (d) Roles of External and Internal <i>Actors</i> in each Use Case scenario. (e) The sequence of events, and messages passed between <i>Actors</i> in all Use Case scenarios. 2. The following Information about a Component's behaviour (Inter-View Behaviour): <ol style="list-style-type: none"> (a) Identifying various Use Cases of the Component by Name. (b) External <i>Actors</i> to a View in various Use Case scenarios. (c) Internal <i>Actors</i> (<i>Views</i>) associated with Use Cases. (d) Roles of External and Internal <i>Actors</i> in each Use Case scenario. (e) The sequence of events, and messages passed between <i>Actors</i> in all Use Case scenarios.
Equivalence In Behavioural-ARC Diagrams 3.3.4, 3.3.5
<p>Use Case ARC</p> <ol style="list-style-type: none"> 1. Use case Scenarios of System's behaviour are Named and Specified in Use Case-ARCs (Fig. 3.8, Fig. 3.9). 2. External Actors to a System are Named and Located (e.g. Fig. 3.9B1: <i>'Mine Field' is an external Actor to the 'Object Miners' System</i>). 3. Internal Actors in a Use Case scenario are Specified in the Use Case-ARC of the scenario. For every Use Case Scenario, a different U-ARC is built (e.g. Fig. 3.8 is U-ARC for the scenario <i>'Trader is Allocated Miners'</i> while Fig. 3.9 is a U-ARC for the scenario <i>'Mining Process'</i>). 4. Roles of External and Internal Actors in a Use Case scenario are depicted with Action Bubbles similar to UML's Use Case Diagrams (e.g. Fig. 3.8 D3: <i>An Action Bubble associated to the Components 'M4' and 'R4'</i>).

Table 3.11: Concerns in Behavioural View of HTM5 and corresponding Design Elements in Use Case and Sequence ARCs. (Part 2)

Behavioural View Concerns 3.2.5
<p>3. The following Information about a View's behaviour (Intra-View Behaviour):</p> <ul style="list-style-type: none"> (a) Identifying various Use Cases of the View by Name. (b) External <i>Actors</i> to a Method (Code Unit) in various Use Case scenarios. (c) Internal <i>Actors</i> (<i>Method</i>) associated with Use Cases. (d) Roles of External and Internal <i>Actors</i> in each Use Case scenario. (e) The sequence of events, and messages passed between <i>Actors</i> in all Use Case scenarios.
Equivalence In Behavioural-ARC Diagrams 3.3.4, 3.3.5
<p>Sequence ARC</p> <ol style="list-style-type: none"> 1. Message Sequences in various Use case Scenarios are Named and Specified in Sequence-ARCs (Fig. 3.10, Fig. 3.11). 2. External Actors to a System are Named and Located (e.g. Fig. 3.11B1: <i>'Mine Field' is an external Actor to the 'Object Miners' System</i>). 3. For every Use Case Scenario, a different Sequence-ARC is built (e.g. Fig. 3.10 is <i>S-ARC for the scenario 'Trader is Allocated Miners'</i> while Fig. 3.11 is a <i>S-ARC for the scenario 'Mining Process'</i>). 4. The sequence of events, and messages passed between Actors are depicted by Action Sequences similar to UML's Sequence Diagrams (e.g. Fig. 3.10 E5: <i>Action Sequence shows that the step '20' and '21' of the Scenario 'Trader is Allocated Miners' takes place at the 'Trader' Agent</i>). <p>Inter-View and Intra-View Behaviour: These are PIM and PSM layer entities which are specified using UML's Use Case and Sequence diagrams within HTM5's PIC model.</p>

3.4 HTM5 P2P

In Chapters 1 and 2 we discussed the benefits of agent oriented, peer-to-peer cloud robotic system. In section 3.2.3 we saw Trade-ARC as a CIM layer model to specify service oriented trade in a cloud robotic ecosystem. We saw an example of a cloud robotic system with placement of relations and merges to manage trade and dynamism in a system of trading robots. In this section we present a discussion on some mechanisms that need to be placed in such a system to enable dynamic service discovery, relationship based trade, contracts and digital institutions. This section discusses these ideas as possible usage scenarios for HTM5 based meta-modelling. No claims are being made to show HTM5 as the only implementational methodology for such ideas. The attempt is more towards understanding technical feasibility of HTM5 meta-model as a methodology to implement the upcoming ideas in cloud robotic systems.

We believe most peer-to-peer cloud robotic systems will be dynamic open systems. The robots and auxiliary systems will trade services based on short term contracts and their economy will have a concept of money as well. Robots and auxiliary systems will publish the services they offer to the ecosystem. These services will be discovered by other robots and auxiliary systems which based on their business logic will choose a service from a list of service providers. Like other service oriented open systems, a service discovery mechanism will have to be placed to allow service providers to publish service advertisements and the quality and cost parameters associated to a service. Unlike a centralized service exchange, a peer-to-peer system allows a distributed system to advertise services and demands. In the example from Section 3.3 (the 'Object Miner' system), the 'Collector' Agent is in a trade relationship with 'Na' number of 'Trader' Agents (Fig. 3.6D4. . .D7). The trade relationship is managed by a relation 'R1' which hosts a service registry (Implemented as a lookup table) by the name of 'Trader ID l Cargo' where traders publish the information about the service 'Cargo'. Similarly, in the same relation 'R1', the 'Collector' agent publishes in a demand registry (Implemented as a lookup table) to display the list of minerals and the offered prices. Traders can modify their mining operations when 'Collector' agent updates prices of certain minerals. A similar logic exists all over the 'Object Miner' system, for various services managed by various relations. The traditional cloud robotic and cloud computing services can be included in a peer-to-peer system by including their service advertisements in distributed service registries.

The service registry and discovery mechanism is followed by a match-making mechanism. The matchmaking mechanism could be hosted by the relation that is managing the registry for the trade item. Special agents could exist in these systems which are primarily concerned with making matches between merchants. In the 'Object Miner' system, the 'Agency' Agent (Fig. 3.6B7) is an agent managing the matchmaking and registry services between 'Na' 'Trader' Agents and 'N' 'Miner' Agents. A matchmaking mechanism also exists at 'R1' Relation (Fig. 3.6D7) for management of 'Cargo' Service. We discussed in Chapter 2 how management of services between two agents by

an outside entity is a violation of an Agent's autonomy. An agent, by coming in a relationship releases this autonomy to the 'Relation' Agent. The advantage of letting a 'Relation' manage the matchmaking service is the visibility a 'Relation' has in the service ecosystem. An agent may be designed to retain its autonomy by initiating its own negotiations with other merchants. An agent may also be designed to use the registry service provided by a 'Relation' and use that registry to connect to merchants bypassing the matchmaking mechanism hosted by the 'Relation'. HTM5 is a meta-model that enables modelling all such scenarios without restrictions. System designers thus are free to develop their own idea of economy using HTM5 meta-modeling tools. The registries can be updated by merchants as and when required, thus a dynamic system can be designed in a cloud ecosystem. The flexibility and openness in implementation together with the *Hyperactivity* mechanism and flexible agency makes HTM5 a methodology that supports heterogeneity in development of open cloud robotic systems.

In multi agent systems, relationships are relative positioning of agents in a predefined social construct. An agent joining a relation knows its role in the relationship. This brings a sense of order in an otherwise chaotic system. Agents representing various 'Machines' and business interests have their personal goals. Being attached to a relation, the agents enter a contract of assisting in a collective effort towards personal and system goals of all relatives. A 'Relation' implements a matchmaking mechanism which can also work as an entity that ensures enforcement of terms in a service exchange. The mechanisms to monitor or regulate quality of service in a relationship can be implemented in the 'Relation' that is managing a service. This is another advantage of using external entities like 'Relation' agents to manage service exchanges between two agents. Cloud computing business logic like pay per use can be implemented at 'Relation' agents. Like commercial cloud computing scenario, Banking and Administrative agents could make cloud robotic systems a viable business model for commercial use. Banking, Administration, Law enforcement and Heterogeneity in design can bring cloud robotics in the commercial domain just the way they did for cloud computing. Relation based trade of services and contracts bring a level of trust and reliability in operations which will be essential in popularizing cloud robotics as a business possibility. We would like to see HTM5 methodology as a step in that direction.

Another step towards commercialisation of peer-to-peer cloud robotics is implementation of dynamic electronic institutions [40, 5]. Institutions represent a norm based collection of entities. Social, administrative and business institution shape the way human societies work. Electronic institutions are norm based multi agent systems that are ideal for modelling social concepts in agent systems. Institutions may be seen as factors that limit agent and system functionality, but their constraining effects decrease system randomness and make a system more productive in long run. One proposed life cycle for dynamic electronic institutions proposes a three phase life cycle (3F life cycle) [40]. The three phases named *Formation*, *Foundation* and *Fulfillment* take a collection of agents from 'Coalition' to a norm based 'Institution'. HTM5 can be used to model dynamic electronic institutions by

implementing the lifecycle in individual 'Relations'. An institutional framework will enable trade relations over an extended period bringing stability in otherwise disordered and greedy behaviour that emerges between trading parties.

3.5 HTM5 Component

A reusable software component is defined as "*A logical cohesive, loosely coupled module that denotes a single abstraction*" [48]. Components are deployable entities which can be composed into larger systems. Component Based Software Engineering (CBSE) [34] is a component based approach to software development. For past few decades, object oriented software engineering promotes direct or inherited reuse of classes. Object orientation is a good practice in software engineering but it doesn't guarantee reuse. The principle objective in object oriented software engineering is appropriate domain representation and not how the objects and classes will be reused. In CBSE, the goal is to develop connected components at more than one level which utilize services of runtime infrastructure provided by *Component Model*. Following are the levels at which a component driven approach is implemented:

1. **Model Component:** A specification or description model for development of components.
2. **Component:** A conceptual entity that can be independently identified in a system.
3. **Class Component:** An abstract template for creation of Object Component.
4. **Object Component:** Actual implemented instance of a component in a running system.

HTM5 has PIC and PSC component models in PIM and PSM layers (Sections 3.1.2 and 3.1.3, Fig. 3.2). A component in HTM5 is separated at the platform line. The definition of platform is subject to interpretation making this separation open to managerial decisions. In HTM5, the view classes in PIM and PSM layers are also component models. Individual views can be reused and redeveloped without interfering with the other view classes. Within a view class, object orientation ensures some level of modularity and reuse. The CIM layer models *Inter-Component* interactions while behavioural view in PIM and PSM layers models *Inter-View* and *Intra-View* interactions. One mechanism to implement separation between PIM and PSM layers of components, view classes and modules within a view class is to declare platform specific parts as abstract classes. The abstract elements of the abstract classes can be defined when classes in PSM layer extend the PIM classes. This however is one of many techniques to implement the separation across platform lines. Similar techniques can be used in separation of HTM5-MDM component model (Section 3.1.4).

Software Components are also defined as *a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* [49]. Interfaces are groups of related methods that facilitate a connection between components. These methods specify a contract, semantics, non-functional properties, quality of service and negotiation mechanism. 'HTM5 Components' (*Agents, Merges and Relations*) can be interlinked in standardized interfaces defined by 'Relation' agents. An interface can be described by standardized interface descriptors (e.g. Microsoft Interface Definition Language, M-IDL). Interface definition languages are language independent specification models for interfaces which can be compiled into language independent skeleton code.

3.6 The HTM5 Domain Specific Language (HTM5-DSL)

Cloud robotic systems are complicated and heterogeneous systems consisting of several computational entities. Cloud entities are developed by various vendors which have their internal developmental methodologies. Cloud robotics is an extension of cloud computing ideas of sharing resources as a service across network cloud. To maintain autonomy of cloud entities and to allow heterogeneity in their design and development, cloud entities may be linked to cloud network via a software agent. These agents are also responsible for implementing social and business logic of cloud entities that they represent. Agent oriented development of cloud robotic system is a distinct domain and requires tools and methodologies that are suited for its design and development. For systems with high complexities, a model driven methodology can be used to simplify the design process. HTM5 is a meta-model for designing models for agent oriented cloud robotic systems. The key design drivers behind HTM5 are agent orientation, industrial acceptability, cloud computing business model and reusability. General purpose programming languages like C++ and Java are not designed to contain features to aid software development for a particular domain. The current work was motivated by the need for a domain specific language for software development and system design using HTM5 meta-model. HTM5-Domain Specific Language can be used to codify design made using HTM5 methodology. The language can be executed to automatically generate code and design templates for developers of individual cloud entities. These model transformations generate results in Java and UML which are widely used in software industry. Availability of a domain specific language and its execution to generate automated model transformations promote industrial acceptability of a new development methodology like HTM5.

A General Purpose programming Language (GPL) like Java or C++ does not have a domain specific syntax. General purpose programming languages are often supported by function libraries that contain functionalities specific to a particular domain. Although function libraries bring reusability in the

development process, their reliance on the general purpose language inhibits specialized development of applications in a particular domain. A programming language based on a particular meta-model allows developers to encode models in a syntax specific to their domain. A Domain Specific Language (DSL) has grammar and syntax that are suited to model based development in a particular domain. HTML [17] for web page development, VHDL [16] for electronic hardware, MATLAB [18] for Matrix based computing and SQL [15] for database systems are popular examples of languages designed for domain specific development. In context of model driven development, Domain specific languages are also documentation tools for designs made following a particular modelling methodology.

In this Section we present a computation independent, Domain Specific Meta Modelling Language based on HTM5 meta model. The presented domain specific language (HTM5-DSL) is designed for codifying HTM5 based designs for agent oriented development of cloud robotic systems. An executable model or DSL is one that can be executed to generate automated model transformations (see Fig. 1.2). HTM5 DSL code is executable and generates automated "Model to Model" and "Model to Text" transformations from HTM5 to UML [47] and Java class hierarchy. Model transformations help in integration of a domain specific model with standardized development practices and are important components of Model-driven software engineering [19, 20].

3.7 HTM5 Domain Specific Language and Automated Model Transformations

In Chapter 1 we discussed Domain Specific Languages (DSL) and their association with meta-modelling methodologies. The aim in development of a DSL for HTM5 meta-model was to first give domain experts a language to code according to the meta-model, and later convert the written code to a general purpose language (GPL, e.g. Java) or a more commonly used software modelling language (like UML). In cloud robotic systems, individual components may be designed and manufactured by different vendors. In these systems, it is necessary that the component models should not impose any restrictions on individual development of functional elements of a model. Automated Model-to-Text transformations from HTM5-DSL generate a Java class hierarchy with class components based on HTM5's PIC and PSC component models. The Model-to-Model transformations generate UML class diagrams for these component classes with their inheritance and object dependencies in place. These template components (Java or UML) can be independently developed by vendors of different robots and auxiliary systems.

The benefits of component based approach were discussed in previous chapters. In this section we present the tool chain developed for HTM5-DSL and automated model transformations using the example project 'Object Miner'. ARC diagrams for this example were presented in Chapter 3 of this

thesis. HTM5-DSL is a CIM layer language which gives a grammar to codify information and design elements that are presented in the ARC diagrams. The grammar specification for any language is a set of rules that sets its syntax. Fig. 3.12 represents syntax rule for specifying a 'HTM5 Component'. The grammar of the language is written in Xtext [50] framework (Fig. 3.13A). HTM5-DSL is a collection of rules to codify 'Machines' (Fig. 3.14D), 'Components' (Fig. 3.14E), 'Clouds' (Fig. 3.14A), 'Trade Items' (Fig. 3.14B, E), 'Connections' (Fig. 3.14C), 'Hyperactivity' (Fig. 3.14E), 'Cardinality' (Fig. 3.14C, D, E), 'Use Cases' (Fig. 3.14F) and 'Sequence' ARCs (Fig. 3.14G). Fig. 3.14 is an example code written in HTM5-DSL. This code is a representation of the ARC diagrams presented as an example project 'Object Miner' in Chapter 3.

HTM5-DSL is an executable language. The HTM5-DSL 'Code Generator' is written in Xtend [51] (Fig. 3.13B) and it executes the code written in HTM5-DSL for automated Model-to-Text (Java GPL) and Model-to-Model (UML) transformations. The 'Code Generator' generates PlantUML [52] script and uses Graphviz [53] graphical visualizer framework to generate UML diagrams. Fig. 3.15 shows the Java class hierarchy, PIC and PSC component classes for various elements and UML class diagrams for the 'Object Miner' example project. These are results of automated model transformations by executing the HTM5-DSL code given in Fig. 3.14. Java GPL was used as a target for Model-to-Text transformation as Java is an object oriented language and is widely used in the industry. UML was chosen as the target for Model-to-Model transformation as UML is an OMG standard widely accepted and used in the software industry. HTM5-DSL and its model transformations are still a work in process. Refinements, wide-ranging documentation and support from open source community are essential to popularize HTM5-DSL and associated tool chain.

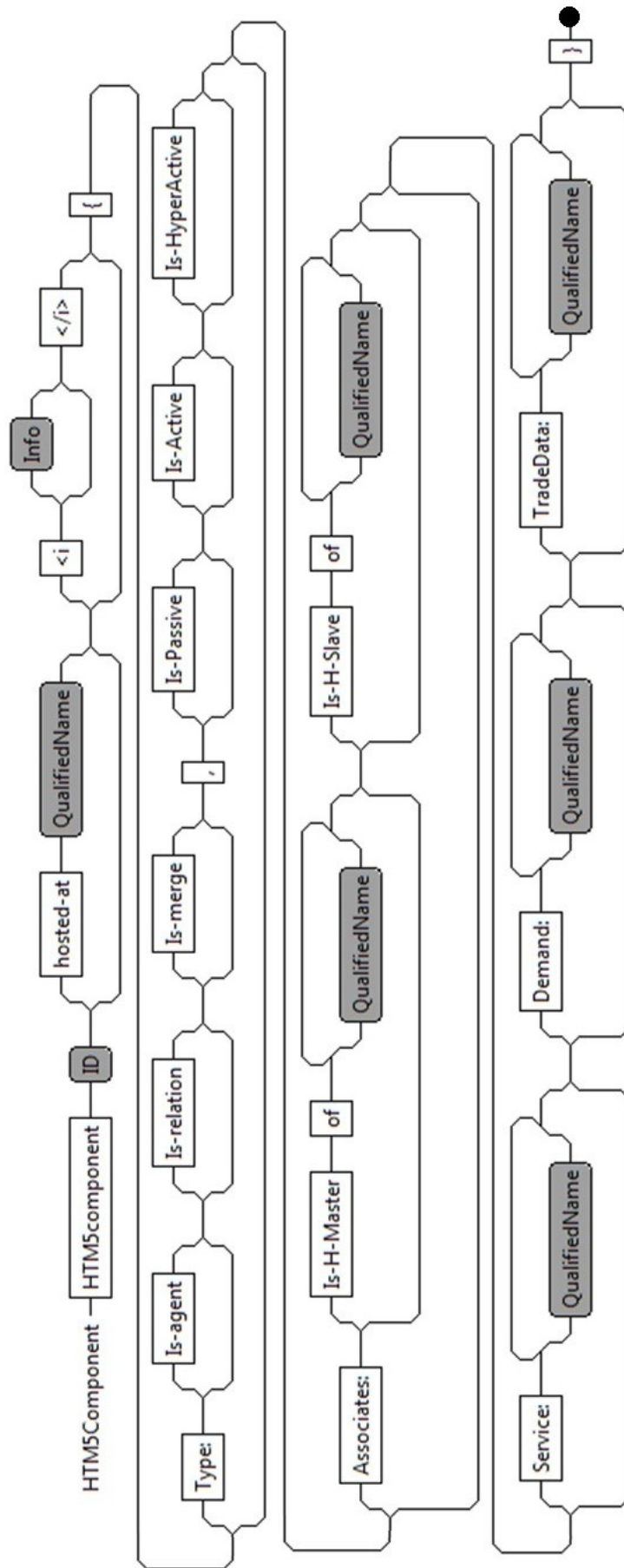


Figure 3.12: Graphical representation of a section of HTM5-DSL grammar specification for 'Component' keyword.

```

313 def compileTradeDatum(TradeDatum td) '''
314
315 «IF td.eContainer.fullyQualifiedName != null»
316 package «td.eContainer.fullyQualifiedName»;
317 «ENDIF»
318
319 public class «td.name» {
320     «FOR s:td.service»
321     struct «s.name» {
322         «IF td.isLookup»
323         struct Lookup {};
324         «ENDIF»
325         «IF td.isVariable»
326         struct Variable {};
327         «ENDIF»
328     };
329     «ENDIF»
330     «ENDFOR»
331 };
332
333 struct Demand {
334     «FOR d:td.demand»
335     struct «d.name» {
336         «IF td.isLookup»
337         struct Lookup {};
338         «ENDIF»
339         «IF td.isVariable»
340         struct Variable {};
341         «ENDIF»
342     };
343     «ENDIF»
344 };
345
346 struct Trade {
347     «FOR t:td.trade»
348     struct «t.name» {
349         «IF td.isLookup»
350         struct Lookup {};
351         «ENDIF»
352         «IF td.isVariable»
353         struct Variable {};
354         «ENDIF»
355     };
356     «ENDIF»
357 };
358
359
377 def compileHTM5C(HTM5Component htm5c) '''
378
379 «IF htm5c.eContainer.fullyQualifiedName != null»
380 package «htm5c.eContainer.fullyQualifiedName»; «htm5c.name»;
381 «ENDIF»
382
383 public class «htm5c.name» {
384     /* Creating View Class Objects */
385     «htm5c.name» _PSC_RH = new «htm5c.name» _PSC_R();
386     «htm5c.name» _PSC_SH = new «htm5c.name» _PSC_SH();
387     «htm5c.name» _PSC_TH = new «htm5c.name» _PSC_TH();
388     «htm5c.name» _PSC_BH = new «htm5c.name» _PSC_BH();
389     «htm5c.name» _PSC_B = new «htm5c.name» _PSC_B();
390
391     «IF htm5c.isHyperActive»
392     «htm5c.name» _PSC_RH = new «htm5c.name» _PSC_RH();
393     «htm5c.name» _PSC_SH = new «htm5c.name» _PSC_SH();
394     «htm5c.name» _PSC_TH = new «htm5c.name» _PSC_TH();
395     «htm5c.name» _PSC_BH = new «htm5c.name» _PSC_BH();
396     «ENDIF»
397
398     ... //EndOf compileHTM5C
399
400 def compileHTM5C_PIC_R(HTM5Component htm5c) '''
401
402 «IF htm5c.eContainer.fullyQualifiedName != null»
403 package «htm5c.eContainer.fullyQualifiedName»; «htm5c.name»;
404 «ENDIF»
405
406 public abstract class «htm5c.name» _PIC_R {
407
408     ... //EndOf compileHTM5C_PIC_R
409
410
519 def compileHTM5C_PSC_B(HTM5Component htm5c) '''
520
521 «IF htm5c.eContainer.fullyQualifiedName != null»
522 package «htm5c.eContainer.fullyQualifiedName»; «htm5c.name»;
523 «ENDIF»
524
525 public class «htm5c.name» _PSC_B extends «htm5c.name» _PIC_B {
526
527

```

A

```

76 Trade:
79
80 TradeDatum:
91
92 HTM5Component:
93 'HTM5Component' name = ID ('hosted-at'
94   superType = [ComponentHost | QualifiedName]
95 )?
96 '<i' (info += Info)? '</i'>'
97 '{'
98 ('Type'
99   (isAgent ?= 'Is-agent')?
100   (isRelation ?= 'Is-relation')?
101   (isMerge ?= 'Is-merge')?
102   ,
103   (isPassive ?= 'Is-Passive')?
104   (isActive ?= 'Is-Active')?
105   (isHyperActive ?= 'Is-HyperActive')?
106   )?
107   'Associates:'
108   (isMaster ?= 'Is-H-Master') 'of'
109   (isSlave ?= [HTM5Component | QualifiedName])?
110   )?
111   (isSlave ?= 'Is-H-Slave') 'of'
112   (isMaster ?= [HTM5Component | QualifiedName])?
113   )?
114   )?
115   )?
116
117 ('Service:'
118   (services += [Trade | QualifiedName])?
119   )?
120   ('Demand:'
121     (demands += [Trade | QualifiedName])?
122     )?
123   ('TradeData:'
124     (tradeData += [TradeDatum | QualifiedName])?
125     )?
126     // (features += Feature)*
127   '}'
128 ;

```

B

```

def compileHTM5C(HTM5Component htm5c) '''
...
package «htm5c.eContainer.fullyQualifiedName»; «htm5c.name»;
«ENDIF»

public class «htm5c.name» {
    /* Creating View Class Objects */
    «htm5c.name» _PSC_RH = new «htm5c.name» _PSC_R();
    «htm5c.name» _PSC_SH = new «htm5c.name» _PSC_SH();
    «htm5c.name» _PSC_TH = new «htm5c.name» _PSC_TH();
    «htm5c.name» _PSC_BH = new «htm5c.name» _PSC_BH();
    «htm5c.name» _PSC_B = new «htm5c.name» _PSC_B();

    «IF htm5c.isHyperActive»
    «htm5c.name» _PSC_RH = new «htm5c.name» _PSC_RH();
    «htm5c.name» _PSC_SH = new «htm5c.name» _PSC_SH();
    «htm5c.name» _PSC_TH = new «htm5c.name» _PSC_TH();
    «htm5c.name» _PSC_BH = new «htm5c.name» _PSC_BH();
    «ENDIF»

    ... //EndOf compileHTM5C
}

def compileHTM5C_PIC_R(HTM5Component htm5c) '''
«IF htm5c.eContainer.fullyQualifiedName != null»
package «htm5c.eContainer.fullyQualifiedName»; «htm5c.name»;
«ENDIF»

public abstract class «htm5c.name» _PIC_R {
    ... //EndOf compileHTM5C_PIC_R
}

def compileHTM5C_PSC_B(HTM5Component htm5c) '''
«IF htm5c.eContainer.fullyQualifiedName != null»
package «htm5c.eContainer.fullyQualifiedName»; «htm5c.name»;
«ENDIF»

public class «htm5c.name» _PSC_B extends «htm5c.name» _PIC_B {
    ... //EndOf compileHTM5C_PSC_B
}

```

HTM5Grammar.Xtext

HTM5CodeGen.Xtext

Figure 3.13: Above are Excerpts of code from *HTM5-DSL Grammar specification in Xtext (A)* and *Model Transformations in Xtend (B)*. Xtext is used to create a domain specific language (named HTM5-DSL) for CIM layer HTM5. The Xtend based Code Generator is the background processor that generates model transformations from a code written in HTM5-DSL. These automated transformations generate PIM and PSM layer *Java Class Components* (Model-to-Text) and *UML* (Model-to-Model).


```

1 package HTM5.DSL.ObjMiner.NetworkClouds
2 {
3   /*Specifying Different Clouds */
4   Cloud-Network Local_WiFi
5   {
6     Is-WiFi,
7     Is-Multiplex
8   }
9 }
10 Cloud-Network Wm[]
11 Cloud-Network Trader_Machine_Inter_Com[]
12 Cloud-Network C_Server_Inter_Com[]
13 Cloud-Network A_Server_Inter_Com[]
14 }

1 package HTM5.DSL.ObjMiner.TradeItems
2 {
3   /*Specifying Trade items */
4   Trade Mine_Coordinates
5   Trade Cargo
6   Trade Search_Space
7   Trade Cargo_Payment
8   Trade Sub_Search_Space
9   Trade Trader_ID
10  Trade Miner_ID
11  Trade Initial_Location
12  Trade Initial_Coordinates
13  Trade Miner_Salary
14  Trade Target_Minerals
15  Trade Sub_Space_Hit
16 }
17 TradeDatum Mineral_X_Price
18 {
19   Is-LookUpTable,
20   For-Demand: Cargo_Payment
21 }
22 TradeDatum MineralID_X_Hz
23 {
24   Is-CostMetric,
25   For-Demand: Target_Minerals
26 }
27 TradeDatum MineralID
28 {
29   Is-Variable,
30   For-Demand: Target_Minerals
31 }
32 TradeDatum TraderID_X_Cargo[]
33 TradeDatum MinerID_X_Salary[]
34 TradeDatum Location_X_MineralID[]
35 TradeDatum MinedAreaPc[]
36 TradeDatum TraderID_X_NumMiner[]
37 TradeDatum TraderID_X_Coordinates[]
38 TradeDatum TraderID_X_Salary[]
39 TradeDatum MinerID[]
40 TradeDatum MinerID_X_JobStatus[]
41 TradeDatum MinerID_X_MinSalary[]
42 TradeDatum Pi_TraderID_X_Salary[]
43 }

package HTM5.DSL.ObjMiner.Connections
{
import HTM5.DSL.ObjMiner.NetworkClouds.*
import HTM5.DSL.ObjMiner.HTM5Components.*
/*
 * Specifying Connections between HTM5 components
 * via Various Cloud Networks */
Connection L01 Miner [Nb] <- Local_WiFi --> M4
Connection L02 Miner [Nc] <- Local_WiFi --> M3
Connection L03 M4 <- Trader_Machine_Inter_Com --> #r1# R4
Connection L04 R4 #r2# <- Trader_Machine_Inter_Com --> Trader
Connection L05 Trader [Na] <- Wm[] --> M1
Connection L06 M1 <- C_Server_Inter_Com --> #r2# R1
Connection L07 R1 #r1# <- C_Server_Inter_Com --> Collector
Connection L08 Trader [Na] <- Wm[] --> M2
Connection L09 M2 <- A_Server_Inter_Com --> #r1# R2
Connection L10 R2 #r2# <- A_Server_Inter_Com --> Agency
Connection L11 Agency <- A_Server_Inter_Com --> #r1# R3
Connection L12 R3 #r2# <- A_Server_Inter_Com --> M3
}

1 package HTM5.DSL.ObjMiner.Machines
2 {
3   import HTM5.DSL.ObjMiner.TradeItems.*
4   import HTM5.DSL.ObjMiner.HTM5Components.*
5   import HTM5.DSL.ObjMiner.Connections.*
6 }
7 Cardinality Na Cardinality Nb Cardinality Nc
8 Cardinality N Cardinality r1 Cardinality r2
9 }
10 /*Specifying Machines*/
11 many [N] Component-Host MinerBot
12 <i>Mobile_Robot </i>
13 {
14   ROS
15   many Camera
16   many Movement
17   IRSensor
18   many ProximitySensor
19   WiFiCommunicator
20 }
21 many[Na] Component-Host TraderMachine[]
22 Component-Host CollectorServer
23 <i>Web_Server </i>
24 {
25   LinuxOperatingSystem
26   WiredInternet
27   SocketCommunicator
28 }
29 Component-Host AgencyServer[]
30 Component-Host External[]
31 }

1 package HTM5.DSL.ObjMiner.UseCases
2 {
3   import HTM5.DSL.ObjMiner.HTM5Components.*
4   import HTM5.DSL.ObjMiner.NetworkClouds.*
5 }
6 package UARC004Trader_Is_Allocated_Miners{
7   Act a001 Between M4, R4;
8   <i>Reads_MinerID_of_MinerID_x_Salary</i>
9   Act a002 Between R4, Trader;
10  <i>Updates_MinerID_x_Salary</i>
11  Act a003 Between Trader, Agency;
12  <i>Allocates_List_of_MinerIDs</i>
13  Act a004 Between Agency, R2;
14  <i>Agency_Reads_TraderID_x_Miner_and_TraderID_x_Coordinates_and_TraderID_x_Salary</i>
15  Act a005 Between Agency, R3;
16  <i>Agency_Reads_MinerID_and_MinerID_x_MinSalary_and_MinerID_x_JobStatus</i>
17  Act a006 Between Agency, Miner;
18  <i>Allocates_TraderID_and_Initial_Location_and_Communication_Coordinates</i>
19 }

21 package SARC008Mining_Process {
22   Action ID001 [Miner] [A]<i>Reaches_A_Location_In_The_Miners_Sub_Search_Space</i>
23   Action ID002 [Miner] [A]<i>Records_Sensor_Data</i>
24   Action ID003 [Miner] [A]<i>Makes_A_List_Of_Positive_Matches_HITS_with_the_Target_IDs</i>
25   Action ID004 [Miner] [O]<i>Sends_the_List_of_Positive_HITS_to_M4</i>
26   Action ID005 [M4] [I]<i>Recieves_the_list_of_positive_hits</i>
27   Action ID006 [M4] [A]<i>Combines_positive_hits_from_all_Miners</i>
28   Action ID007 [M4] [O]<i>Sends_update_request_for_Location_x_MinerID_to_R4</i>
29   Action ID008 [R4] [I]<i>Recieves_update_request_for_Location_x_MinerID</i>
30   Action ID009 [R4] [A]<i>Updates_Location_x_MinerID</i>
31   Action ID010 [M4] [A]<i>Calculates_Positive_Hit_Frequencies_for_all_MineralIDs</i>
32   Action ID011 [M4] [O]<i>Sends_update_request_for_MinerID_x_Hz_to_R4</i>
33   Action ID012 [R4] [I]<i>Recieves_update_request_for_MinerID_x_Hz</i>
34   Action ID013 [R4] [A]<i>Updates_MinerID_x_Hz</i>
35   Action ID014 [M4] [A]<i>Calculates_the_percentage_of_area_mined</i>
36   Action ID015 [M4] [A]<i>Sends_update_request_for_MinedAreaPc_to_R4</i>
37   Action ID016 [R4] [O]<i>Recieves_update_request_for_MinedAreaPc</i>
38   Action ID017 [R4] [I]<i>Updates_MinerAreaPc</i>
39 }
40 package SARC009Trader_Initiates_Trade_With_Collector {}
41 package SARC010Collector_Accepts_Cargo {}

1 package HTM5.DSL.ObjMiner.HTM5Components
2 {
3   import HTM5.DSL.ObjMiner.Machines.*
4   import HTM5.DSL.ObjMiner.TradeItems.*
5   /*External Actor */
6   HTM5Component MineField hosted-at External[]
7   /*Specifying HTM5 Components */
8   /*At MinerBot */
9   many [N] HTM5Component Miner hosted-at MinerBot
10  {
11    Type: Is-agent,
12    Is-HyperActive
13  }
14  Associates:
15  Is-H-Slave of Trader
16  Service: Sub_Space_Hit
17  Demand: Trader_ID
18  Initial_Location
19  Target_Minerals
20  Sub_Search_Space
21  Miner_Salary
22 }
23 /*At Trader Machine */
24 many [Na] HTM5Component R4 hosted-at TraderMachine
25 {
26  Type: Is-relation,
27  Is-Passive
28  Tradedata: MinerID_X_Salary
29  MineralID
30  Location_X_MineralID
31  MinedAreaPc
32  MineralID_X_Hz
33 }
34 many [Na] HTM5Component M4 hosted-at TraderMachine[]
35 many [Na] HTM5Component Trader hosted-at TraderMachine[]
36 /*At Collector Server */
37 HTM5Component Collector hosted-at CollectorServer[]
38 HTM5Component M1 hosted-at CollectorServer[]
39 HTM5Component R1 hosted-at CollectorServer[]
40 }

1 package HTM5.DSL.ObjMiner.Sequence
2 {
3   import HTM5.DSL.ObjMiner.HTM5Components.*
4   import HTM5.DSL.ObjMiner.NetworkClouds.*
5 }
6 package SARC001Trader_Gets_Search_Space {}
7 package SARC002Trader_Requests_Miners {}
8 package SARC003Miners_Register_With_Agency {}
9 package SARC004Trader_Is_Allocated_Miners {}
10 package SARC005Collector_Publishes_Mineral_Prices {}
11 package SARC006Miners_Registers_With_Trader {}
12 package SARC007Active_Miner_Looking_For_Switching_Jobs {}

```

Figure 3.14: Excerpt of code written in HTM5-DSL for the example project 'Object Miner'. Section 'A' specifies the cloud networks, section 'B' specifies the trade information while connections between components are coded in section 'C'. The syntax provides cardinality specification for 'Merges' and 'Relations'. HTM5 components are specified in section E while the machines which are represented by the components; they are specified in section 'D'. The behavioural elements of the system are specified by conversion of Use Case and Sequence ARCs to DSL code (section 'F', 'G'). The above figure marks the sections with labels specifying the ARCs (Fig. 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10 and 3.11) that they represent.



Figure 3.15: The above figure presents automated 'Model-to-Text' and 'Model-to-Model' transformations from the HTM5-DSL code presented in Fig. 3.14. The transformations generate a HTM5-PSC or PIC Java class hierarchy with class model templates. Class models have modules where Java code for individual functionalities can be placed. The target model for 'Model-To-Model' transformations is UML's class diagrams for individual class components with inheritance and object interdependencies. Both transformations generate template components which can be developed independently by stakeholders representing various machines in the cloud robotic ecosystem.

3.8 DSL development

In previous sections of this chapter we have presented the HTM5 domain specific language (HTM5-DSL) that was developed to support an OMG-MDA [3] based meta-model (HTM5) for agent oriented development of cloud robotic systems. During the incremental development of the meta-model and the DSL, several design choices were made based on the requirements of their end-users. In this section we present a post-development analysis of those choices and discuss general lesson that were learned with respect to developing a domain specific language.

The development of a domain specific language can be seen as a 5 step process: decision, analysis, design, implementation and deployment [54, 55]. The decision stage is an analysis of the reasons why a DSL should be developed. In analysis stage, the domain requirements are analysed and domain specific methodologies are identified that could act as a guide for the development of the DSL. The design stage conceptualizes the overall structure, syntax and execution semantics for the DSL. The DSL is then constructed in the implementation stage followed by its deployment in the field. In an incremental development model, the DSL features are added and recalibrated following multiple rounds of the 5 step process. We now go through these steps one by one with respect to the incremental development of HTM5-DSL.

The decision stage: As mentioned in previous sections, the HTM5-DSL is founded on HTM5 metamodel which is an OMD-MDA [3] based meta-model for agent oriented cloud robotic systems. The two most common ways in which a meta-model may be objectified are through an application library or a domain specific language. In the case of multi-vendor, multi-device and distributed systems like cloud robotic ecosystems, a domain specific language serves as a better option. The standardised DSL gives a common communication language for all stakeholders which may have their own independent software development methodologies. Individual components of a cloud robotic system are developed and deployed by different vendors. An executable DSL supported by appropriate model to text (M2T) and model to model (M2M) transformations is more suited compared to development of application libraries for every software development methodology.

Another benefit of using a DSL is unlike an application library, a DSL also works as a design document. This is important since DSL gives a concrete structure to the designs represented by the meta-model and is the end-document resulting from the multi-layer development of a model for the cloud robotic ecosystem. The DSL thus gives a sense of order to the multi-party distributed development of cloud robotic system components and can be executed to generate template components in another model (e.g. UML) or a general purpose language (e.g. Java). The importance of a DSL is even more when the meta-model is multi-view with interlinking inter and intra-view dependencies. For

the benefits mentioned above, it was decided that a DSL should be developed to support the HTM5 meta-model.

The other important decision regarding the development of the DSL was to choose where in the three layers of OMG-MDA (Computation independent, Platform Independent and Platform specific) the DSL should be placed. A Computation independent layer DSL was understood to be the ideal choice since it is the most abstract of the three layers and at this layer, the components are treated as closed entities. A computation independent DSL design aims to capture the component to component dependencies and functionalities in the code, which is later individually developed by the vendors responsible for the development of those components.

The analysis stage: After the decision to develop a computation independent DSL was taken, the next step was to identify the elements of the HTM5 meta-model that need to be included in the DSL. The computation independent model in HTM5 is a set of ARC diagrams. The computation independent DSL thus had a clear guide to follow. Furthermore, HTM5 is a 5 view meta-model with clearly identified view specifications. For the development of the DSL, these view specific specifications acted as a list of functionalities that needs to be included in the DSL. In general, the DSL was developed alongside the development (and refinement) of the ARCs and followed the identical entity relationships, parameterization and nomenclature.

The language being developed should be compact, expressive and efficient since it was meant to be used by domain experts, business strategists and non-technical personals that may not have any experience in programming. The language at computation independent layer should be expressive enough to include all design elements of the followed meta-model with syntax that resembles a non-programming language like English. The design of the DSL should be such that the non-programming language like syntax should not diminish the overall computational power of the code upon compilation.

The design stage: The computation independent DSL required an abstract syntax and structure. Like in the HTM5 computation independent model, the design of the DSL was based on the view specific elements. Individual elements of the ARC diagrams were identified as entities in an object oriented methodology. These entities were treated as classes and every unique instance of these entities as class objects. The individual entity classes were designed in the DSL followed by the containers that bring the linkage and hierarchy relationships between the entities.

The syntax for several design elements (of ARC diagrams) was intended to be a visual representation of their act. An example of this visual representation can be seen in the shorthand notation for *Connection* element which is represented as:

"*Connection C1 Comp1 #n#* \leftarrow *Net1* \rightarrow *#m# Comp2*".

This one line is the syntax to represent a connection named *C1* between *n* components of type *Comp1* and *m* components of type *Comp2* through a cloud network named *Net1*. The grammar rules to implement such syntax features were designed keeping in mind the end users and the decisions made at the analysis stage.

At this stage, it was also necessary to formulate a strategy by which the DSL code written for different views can be compatible with one another. Although the entities necessary to codify the ARCs of a particular view require to be placed together, the project code should have one common namespace. The use of one common namespace will also integrate the view specific ARC diagrams into one all-inclusive model. This is necessary to omit ambiguities especially in projects with several ARC diagrams built at various iteration of the development process.

Before the DSL syntax and grammar rules are implemented, it was essential to identify the appropriate execution targets for the language. Executability is not an essential feature for a DSL [55] but for reasons identified at the decision stage, it was essential to develop an executable DSL. A DSL may be executed to produce documents, scripts, code in another language (DSL or GPL) or design in another meta-model. Such executions are broadly classified into two categories namely model to text (M2T) and model to model (M2M) transformations. The targets for the automated transformations were chosen to be *Java* general purpose programming language (GPL) and *Unified Modeling Language (UML)*. *Java* is widely used in the industry and is object oriented programming language. UML is the most common modeling language used for standalone applications and thus is an ideal candidate as a target for M2M transformation. Since UML is widely used, automated transformations from UML to other common GPLs like *C++* and *Python* are easily available. Thus a M2T transformation to *Java* and a M2M transformation to *UML* cover a vast spectrum of software development methodologies that may be used by vendors of individual components of the cloud robotic ecosystem.

The implementation stage: This is the stage when the DSL is actually constructed using customized language development tools. In many cases, these tools are special DSLs that help define the syntax and compiler for other DSLs. For HTM5-DSL, we used Xtext [50], which is an open-source framework for development of domain-specific languages. For an executable DSL like HTM5-DSL, a complete set of compiler functions are required to be written. We used *Xtend* [51] based code generator to write the compiler functions for HTM5-DSL. These functions are executed while compilation of the DSL code and generate the M2T and M2M transformations from the DSL code to *Java* and *UML* respectively.

The compiler function for HTM5-DSL is distributed openly with the language. Keeping the compilation functions open enables end-users to modify these functions to generate more specific resultant transforma-

tions in *Java* and *UML*. As the compiler is open, new functions may be added to generate supporting documents or scripts to suit to a vendors software development requirements. The existence of an executable DSL and open compiler enhances the interoperability amongst designers of the cloud robotic systems and individual vendors (component developers). Such a DSL helps increase the overall speed of software construction as well as system integration. Since it is simpler to prototype a design decision in the DSL code, the executable DSL reduces the cost and time required for rapid prototyping of system components and the cloud ecosystem as whole. The design and implementation of HTM5-DSL thus encourages *generative* and *end-user* programming methodologies.

The deployment stage: The HTM5-DSL developed alongside the HTM5 meta-model was deployed in several case study projects. The feedback obtained from these studies was used to improve the meta-model and the DSL. The case studies were designed to test various features of the meta-model and through them DSL proved its usability as a computation independent design capture media.

3.9 System development life-cycle

This work so far has covered the background, HTM5 anatomy and elements of HTM5-DSL grammar and implementation. The authors understand that some parts of the thesis so far were very descriptive and might have appeared unappetizing to a non-practitioner. To present a clearer perspective to all readers, this section will go through all important steps in development of an agent oriented cloud robotic system using HTM5 methodology.

This section is a walk-through the system development life-cycle using HTM5 design methodology, HTM5 Domain Specific Language and automated model transformations.

- (1) **Initiation and system concept development:** The initial ideas and abstract concepts are identified. The system in the idea space is a verbal concept and often has very limited or no descriptive documentation. The initiator of the concept brings together the stakeholders (cloud businesses or product vendors) and technology practitioners at a forum and shares the concept and the benefits it has for the stakeholders. After the initial brainstorming, a team with representatives (managing technological and business interests) from all stakeholders is founded which together work on interdependencies and polishes the initial concepts.
- (2) **Identifying structural dependencies:** Once the system in the idea space is well established, the team then moves on to identify dependencies with respect to hardware, physical location, communication network and scale. Important decision on the scale of operation, the type of agents (software) that will be hosted at various hardware units, the relationships that these agents have with other agents are conceptualized. At this stage the emphasis is on what is to be made without going into the mechanics of how these ideas will be implemented.
- (3) **Identifying trade dependencies:** Items and services that will be provided by various stakeholders are identified and reward mechanisms for these services are envisioned. Services are also classified as those which will be backed by contracts and others that will remain open to runtime negotiations (automated or supervised). Physical locations and scope of service and demand registries (special agents that manage service-demand matchmaking) are identified together with open ports where stakeholders through their respective agents (software) can attach or detach from a size invariant system (open systems where members can join or leave dynamically).
- (4) **Identifying Hyperactivity:** The concept of agency in HTM5 is kept flexible. Along with regular agents, there are passive and hyperactive agents that deviate from one or more elements of ideal agency. The hyperactivity associations between agents and the control that an agent releases to an external agent are identified at this stage. This stage requires some insights from the engineers as a deviation from ideal agency

is often on account of ease of implementation and generalization of unnecessary bottlenecks. In some development cycles, the hyperactivity controls may be added in the second design iteration (after the first version of the system is ready) for implementing quick-fix solutions to implementation complexities.

- (5) **Identifying use cases:** Envisioning the system as a black-box, the stakeholders enlist all supported use-cases for the complete system. These use cases takes the entire system as one unit and external agencies (other systems or humans) as foreign actors. The use-cases for individual components of these systems however are not enlisted at this stage. The component level behaviours are defined later in component level design.
- (6) **Computation independent design:** The steps above were all part of the requirement elicitation and analysis for the first iteration of system development. At this point, the agent oriented design and requirements are made into the computation independent HTM5 model. At computation independent layer, the HTM5 has agent relation charts (ARCs) which are a visual representation of the ideas and design semantics identified in the previous steps. The initial ARC diagrams may be refined over several brainstorming sessions within the team and later in second and later iterations of the system development process. The model (ARC drawings) is design specification as well as requirement documentation model and thus legal contracts (if any) between the business stakeholders and product vendors can be made based on the specifications documented in computation independent ARC designs.
- (7) **Hardware independent machine description:** After the ARC designs are made, the individual stakeholders can design their respective agent components. In order to separate platform independent and specific parts of the component design, the component is built in two stages and with clear separation using abstract class inheritance (or any other feasible mechanism chosen by individual component designers). The agent host machine may also have some hardware dependencies and thus the design elements have to be separated in parts that are dependent on the actual hardware and the names that are used in the agent components. The hardware independent elements of the machine (agent host) are identified and built into a class.
- (8) **Platform independent component design:** The platform independent parts of an agent component are separated from the platform dependent parts by design teams of different stakeholders. HTM5 methodology gives independence to individual stakeholders to design their component with independence. A stakeholder may decide to keep bulk of the functionality in platform specific partition if it suits their internal software development life cycle. The components developed at this stage are UML based class diagrams (or executable code in

HTM5-DSL) which can later be developed as working code segments in a general purpose programming language (GPL).

- (9) **Platform specific component designs:** The platform specific designs may be described with little or no implementation details at the first design iteration. These designs need to be modified as and when there is a platform level change. The platform specific component design is the most frequently altered items in the complete agent oriented cloud system.
- (10) **Hardware specific machine description:** The machines that an agent is representing in the cloud ecosystem may change with time or may have some firmware updates. To manage these changes with limited effect on the overall design of the system, the machine are separately represented in HTM5 as hardware independent and dependent model classes. This part of the design is also updates very frequently like the platform specific component designs mentioned above.
- (11) **Writing executable code in HTM5-DSL:** As mentioned above, a stakeholder may decide to develop its agent component directly in a general purpose language or UML. Otherwise, the system level design team may take the computation independent HTM5 model (ARC diagrams) and using a domain specific language (HTM5-DSL) writes a corresponding computation independent code for that. The executable code written by the system design team can now be executed to generate template classes and designs for individual components via model to model and model to text transformations.
- (12) **Automated model transformations:** As mentioned above, the executable code written in HTM5-domain specific language can be executed to generate template classes in Java programming language or equivalent UML design classes. Java is a popular general purpose programming language used in the industry hence a template class in java that already has interconnections and elements to handle cloud interdependencies could serve as a good starting point for development teams of individual stakeholders. Stakeholders who do not use Java for their internal development processes can utilize the UML template classes at their starting point. The UML template classes can be processed manually (or via model to code transformation) to generate an object oriented code in several commonly used general purpose programming languages. Such UML to code transformations are commonly available in the industry since UML is a worldwide standard as a component description and specification model.
- (13) **Vendor side product development:** Starting from the template classes (Java or UML), the development teams at various stakeholders independently develop their agent components. As the development process has a template class and ARC designs to start with, the development teams are free to choose their internal development methodology. This

is beneficial since involvement of a business or product vendor in a cloud project should require minimal changes in its internal software development apparatuses. This flexibility to add/delete agency concepts and independence in internal component design process makes HTM5 comparatively an easier model to adapt to.

- (14) **System integration and testing:** Once the components are individually developed and tested at various development sites, the components are pairwise tested against the design and behaviour test-cases specified in the AR charts. Once individual demand-supply pairings are tested, the system can run several dependencies in parallel and start the system testing against the system use-cases specified in the ARC model.

The development of systems of such scale and diversity is an incremental process. The second and further iterations for system development build upon the system designed after the initial iteration.

In the next section of this thesis we present some of the case study experiments and a workshop conducted over the last 3 years to test the feasibility and viability of the HTM5 meta model and its corresponding domain specific language (HTM5-DSL). The objective of these studies was to improve the proposed model through feedback and usage and to present a defense to the major claimed benefits of the HTM5-DSL and the HTM5 methodology.

Chapter 4

Case Study Projects

4.1 NAO Telerobotics

This case study illustrates the usage of the proposed HTM5 methodology for a typical cloud robotic system. This is a simple telepresence application for a humanoid robot supported by a number of auxiliary devices. The humanoid robot in our system does not have localization capability. Three roof mounted cameras provide their feed as a service to camera reading computers. These computers then locate the robot in their respective camera feeds and offer the location as a service to a manager computer. The manager computer combines the information coming from these sources and provides direction and location as a service to the humanoid robot. The manager computer is connected to a web server across the internet cloud which runs a secure access telepresence service for the humanoid robot. Computers and Mobile devices around the world can register to the service and access the robot. The robot can be directed to a particular location in its workspace by the remote users. Once reached, the robot performs a set of actions as directed by the remote user and sends images or information as a service to the remote clients. The whole system is location independent and is a simple but adequate example of a cloud robotic system. Table 4.1 describes various machines in the 'Telerobotics On Nao' cloud robotic system while Fig. 4.1 gives a structural overview of the system without using any HTM5 representations.

Table 4.1: A description of various *Machines* in the 'Telerobotics On Nao' cloud robotic system.

Machine	Count	Location	Development Platform	Operating System
NAO Humanoid Robot	$>=1$	UTP, Malaysia	NaoQi	OpenNao
IP Camera	3	UTP, Malaysia	Embedded	Embedded
Camera Reader PC	2	UTP, Malaysia	Python GPL, Open CV	Windows
Manager PC	1	UTP, Malaysia	Python GPL, NaoQi, XML	Linux
Web Server	1	Google, USA	Google App Engine, AJAX	Customized Linux
Mobile Phone	m	New Delhi, India	AJAX, HTML	Android, Symbian, Ios
Mobile Device	N-m	UB, France	AJAX, HTML	Windows, Linux, MacOS

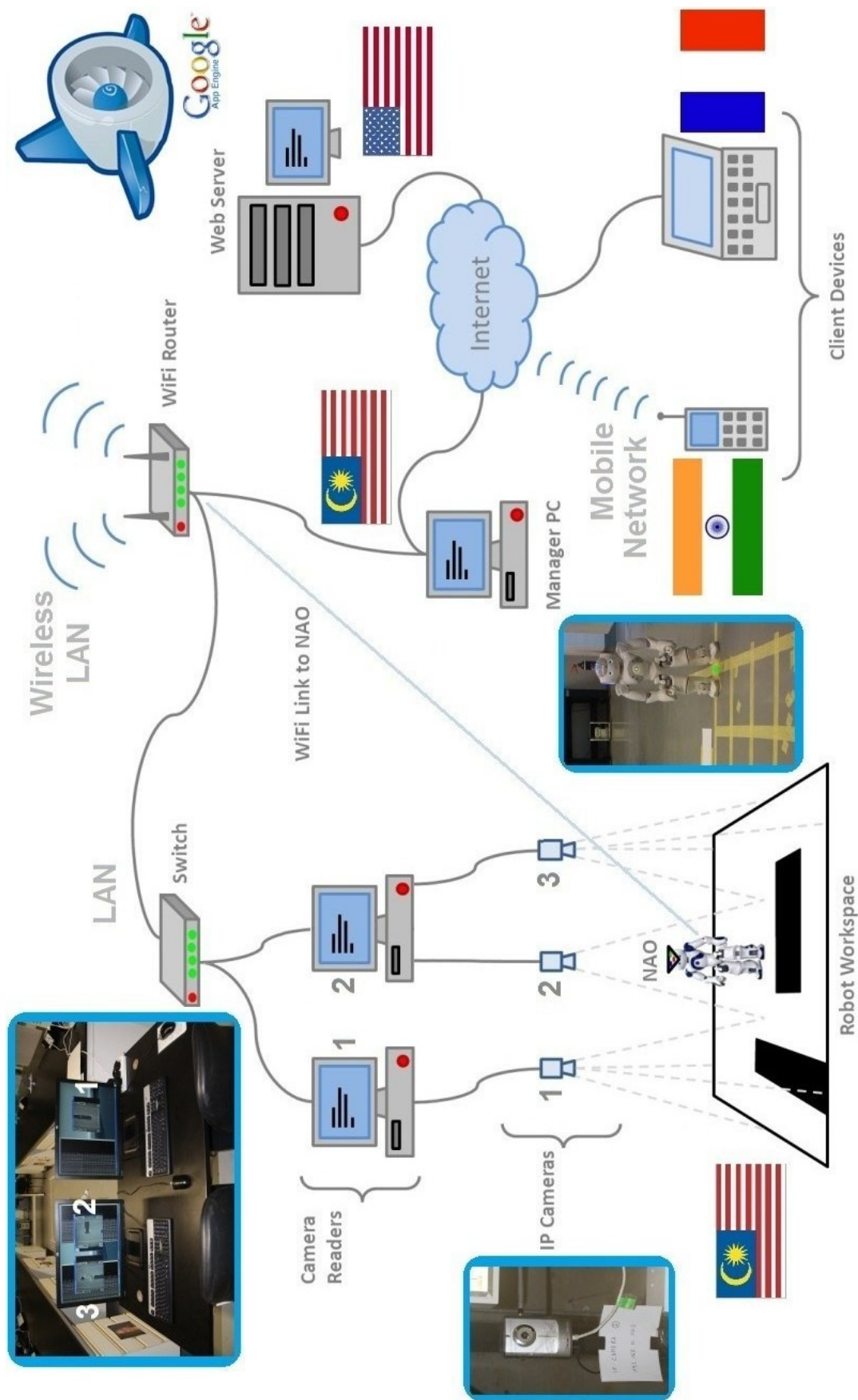


Figure 4.1: Structural Overview of 'Telerobotics On Nao' System.

The case study comprises of three main steps.

1. Designing the system using HTM5 Methodology.
2. Executing the HTM5-DSL code to generate a UML model for various *Components* in the system.
3. Independent development of system *Components* at various *Machines*.

The peer-to-peer trade in the 'Telerobotics On Nao' system is not based on predefined contracts or institutionalization. There is no actual transfer of money in the system and the services are not governed by measurement of *Trust* or *Quality of Service*. This is a scalable system with provisions to include any number of NAO robots, cameras and clients in the system. Automatic initialization of new members however is implemented only for clients and not for robots or cameras. In view of the cloud robotic concepts presented in this work, this case study is a simplified one. The Machines in this case study (see Table 4.1) have different development platforms, operating systems and locations. A variety of services dependencies exist in the system between various peers. Hyperactivity is extensively used to suit easy implementation. The heterogeneity and practical use of hyperactivity in the system makes this a simple but representative case study for HTM5 and cloud robotic systems. Following are details of step by step development of 'Telerobotics On Nao' system.

4.1.1 The Initial Ideas

The first step towards development of any system is a collection of ideas for an application. The ideas for a system need a representation in form of a textual or graphical sketch. The ideas that emerged for a 'Telerobotics On Nao' system were roughly jotted down in a set of graphical representations like the one shown in Fig. 4.1. The ideas in early brainstorming are abstract and crude. The aim at this stage is to understand the overall theme of the system. The next step is to migrate from the idea domain to a model for the system.

4.1.2 Creation of CIM model using Agent Relation Charts

The rough ideas about the system were now converted to a computation independent model using ARC diagrams. In Model based development, the ideas are represented using guidelines set by a meta-model. A good model not only helps in representation of ideas but it is a tool to assist in development of ideas about the system. The most essential anatomical elements of the 'Telerobotics On Nao' system design are ready by the end of this step. Fig. 4.2 and Fig. 4.3 wrt the ARC diagrams developed for 'Telerobotics On Nao' system. ARCs can be drawn using most generic drawing software. We used 'Open Office Draw software [56] for drawing these ARC diagrams. Open Office Draw is an open source software for making drawings and charts.

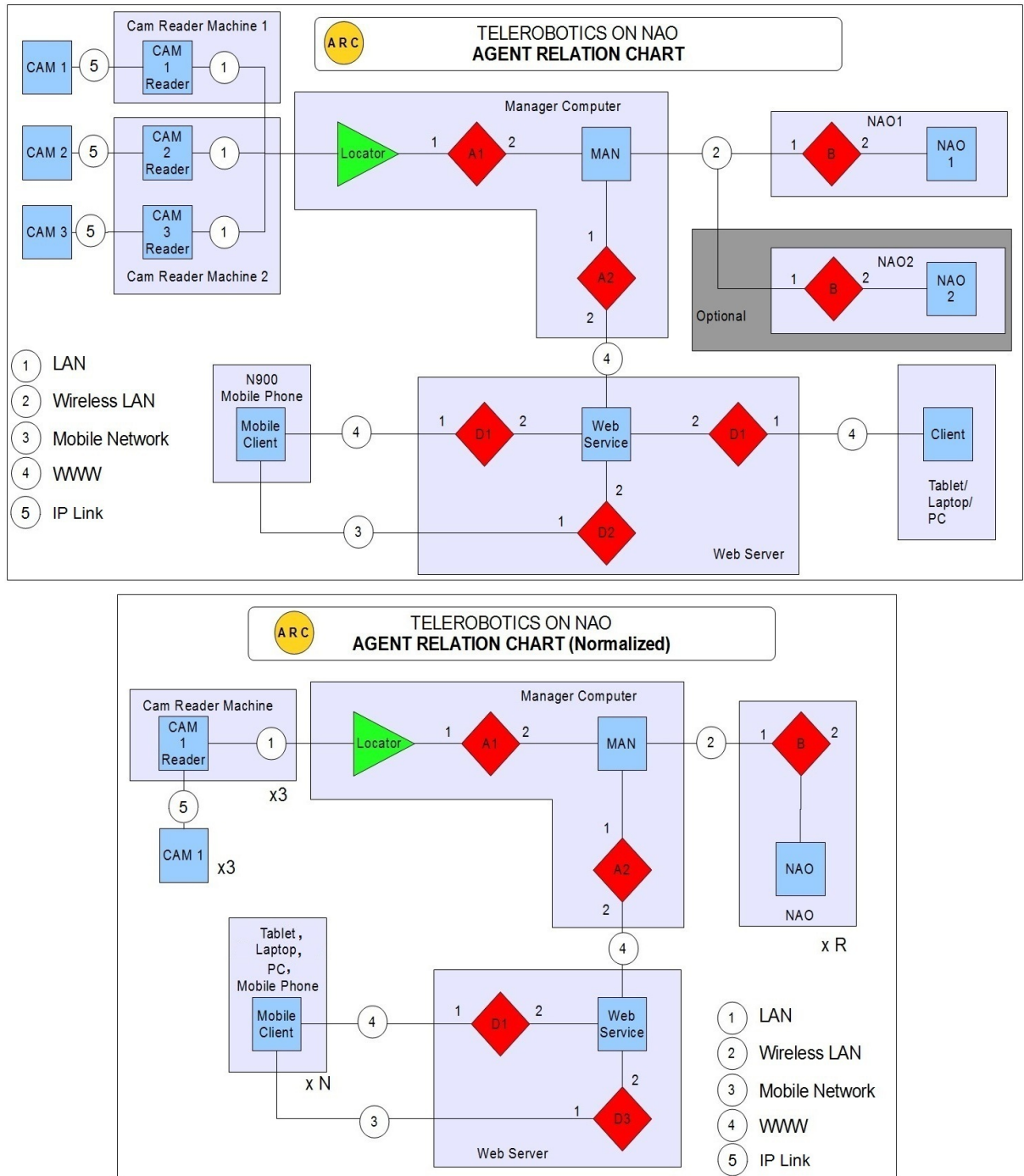


Figure 4.2: ARC and Normalized ARC Diagrams for 'Telerobotics On Nao' System.

4.1.3 Encoding the CIM model in HTM5-DSL

Once the CIM layer model is ready, HTM5-DSL is used to encode the model in an executable form. The domain specific language HTM5-DSL has CIM layer abstraction. Fig. 4.4 shows excerpts from the code written in HTM5-DSL for 'Telerobotics On Nao' system.

4.1.4 Creation of PIM layer UML model for various components

In previous steps we specified system level views and identified components. The functional and behavioural specification within a component is done in two phases. In first phase UML use case and sequence diagrams for inter and intra view interactions are specified (see Fig. 3.2). Platform independent components of a component are specified at this stage using UML. Platform specific elements of view classes are only partially specified in the first phase. In the second phase, extended specification of platform specific elements is done when component classes are independently developed by various vendors.

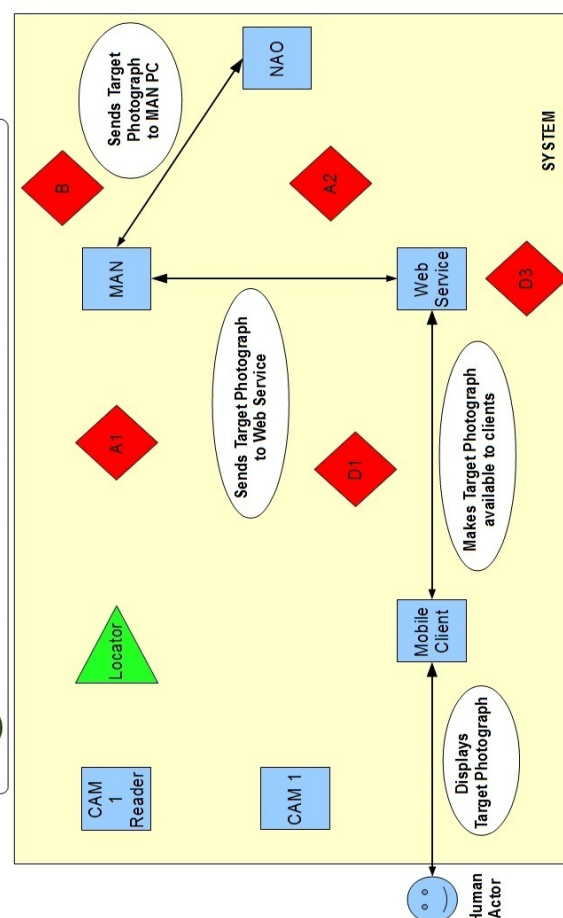
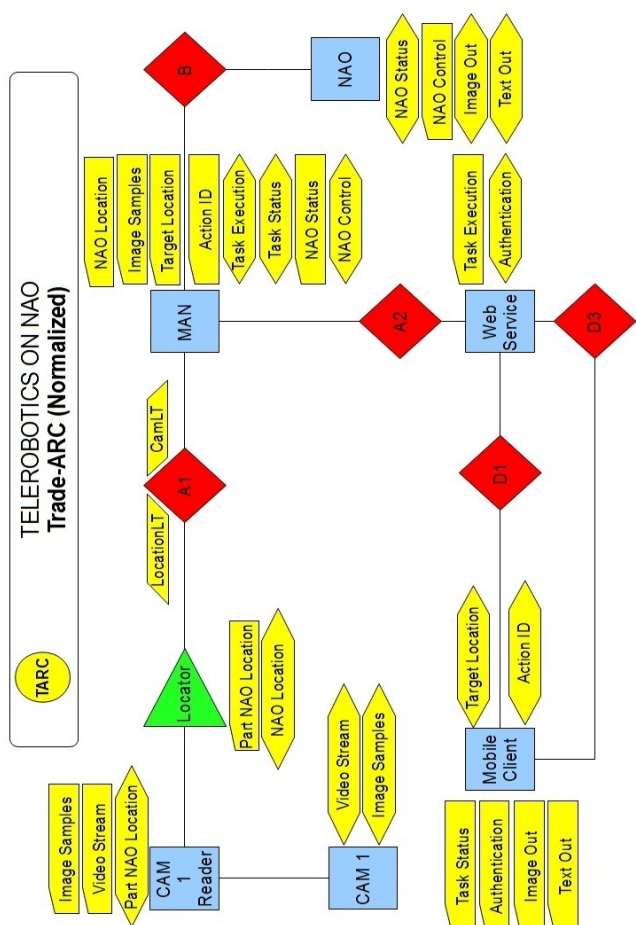
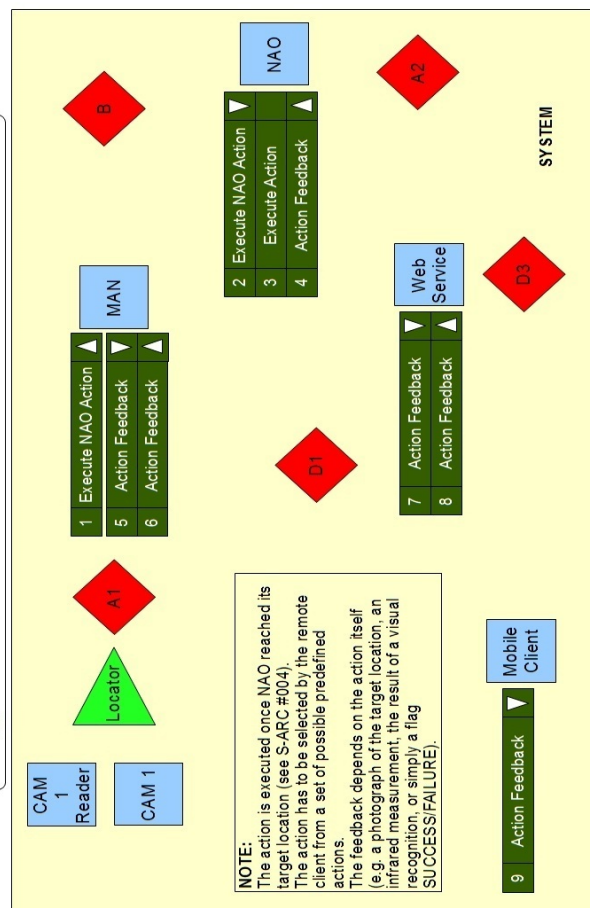
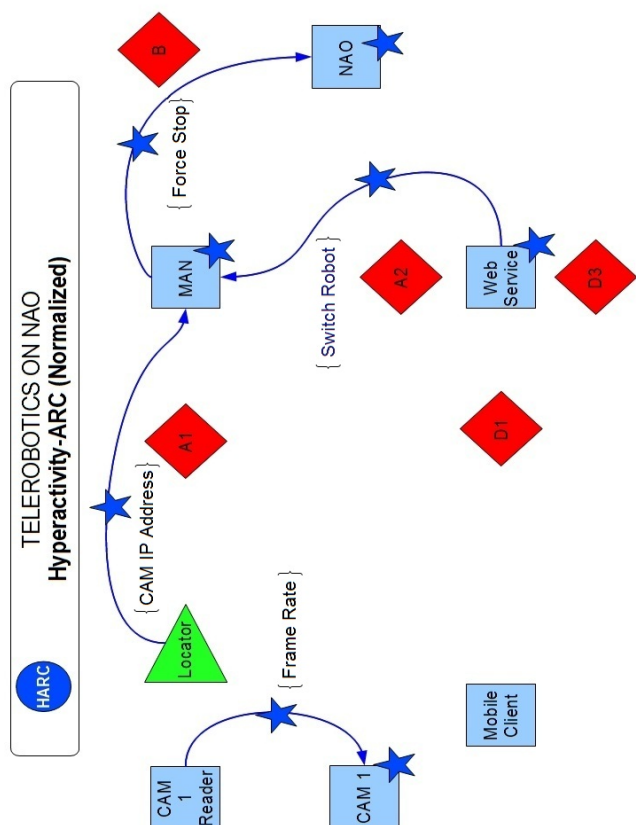


Figure 4.3: TARC, HARC, UARC002 and SARC005 Diagrams for 'Telerobotics On Nao' System.



Figure 4.4: Excerpt of code written in HTM5-DSL for 'Teleroobotics On Nao' System. Section 'A': Cloud networks, Section 'B': Trade Information, Section 'C': Machines, Sections 'D' and 'E': Behavioural Elements and Section 'F': HTM5 Components. The code is executed to generate Java Class components (Model-to-Text) and UML class model (Model-to-Model). Section 'G' is the component directory structure generated as a result of code execution.

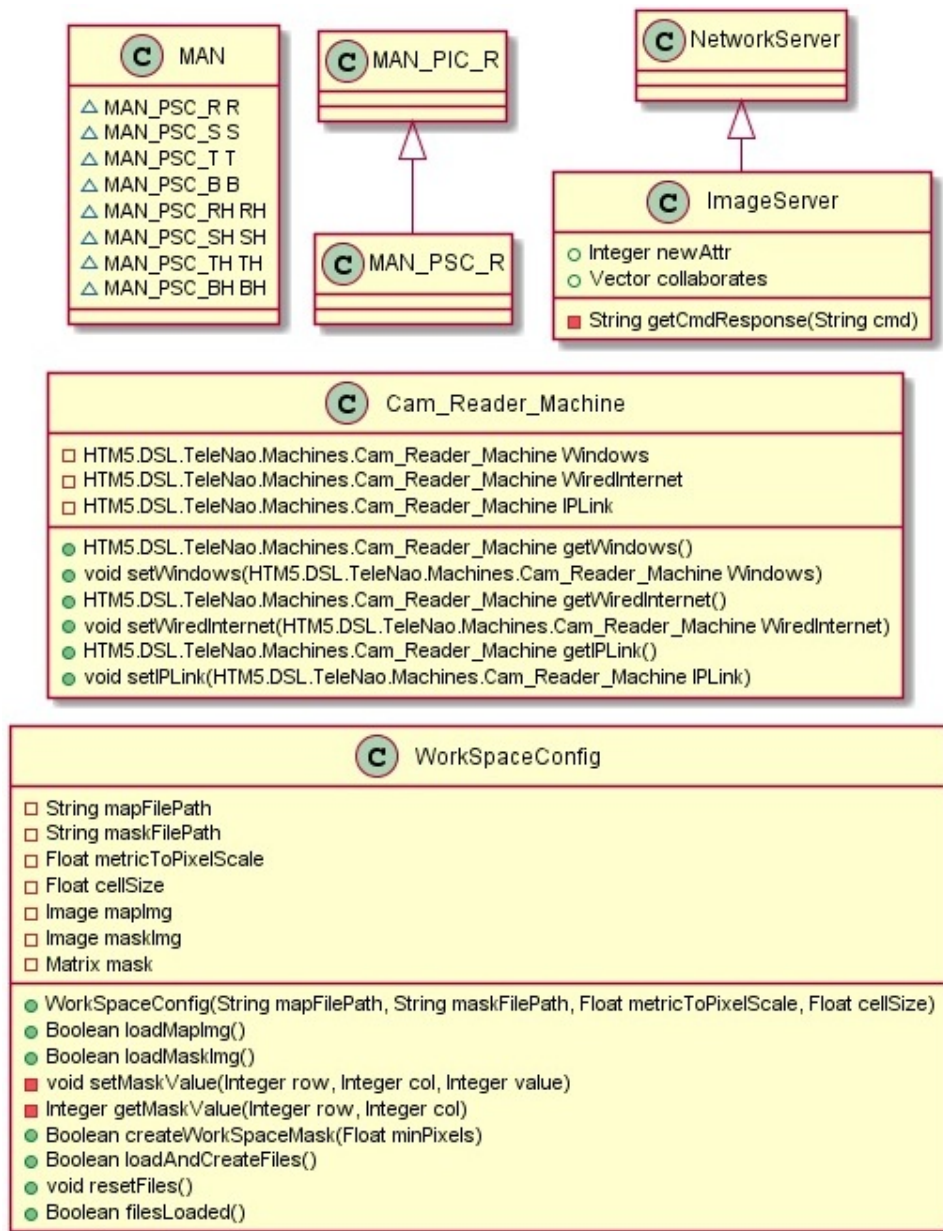


Figure 4.5: Excerpt of UML Class Components generated by execution of HTM5-DSL code (Model-to-Model Transformation).

4.1.5 Execution of DSL code to produce Java and UML component classes

The DSL code for the system is executed to generate Java class hierarchy and UML class components. Fig. 4.4G and Fig. 4.5 show results of these model transformations.

4.1.6 Refinement of component classes

The component classes generated by executing a HTM5-DSL code contains specification for only the essential framework. This is because HTM5-DSL is at the computation independent layer and does not include specifications made at the PIM layer. Before the component classes are dispatched to various vendors, the classes may be refined by adding some of the PIM functionalities. The classes may also be refined by adding appropriate comments in the code making them more readable. Some class diagrams in Fig. 4.5 are examples of refined components which include elements from the PIM layer model. Once the component classes (Java or UML) are finalized, the development enters its second phase where the component classes are independently developed for the machines they represent.

4.1.7 Independent development of Components respective Machines

The automated transformations and refinements generate Java class components or UML design for view and component classes. As various machines have different development platforms and operating systems, the Java and UML designs may be converted to a language more suited for machine's development platform.

1. **Cam Reader Machine:** The *Cam Reader* Agents were hosted on a windows machine and a Python programming platform. Python is a widely used GPL and it is designed to enable programmers to express concepts in fewer lines of code. The *Cam Reader* Agent reads a video stream from the cameras and extracts location of the Nao Robot. Every instance of *Cam Reader* Agent generates a part of the information about the location and offers it as a service to the *Location Merge*. For processing of the incoming video stream, OpenCV library was utilized. The component class generated in first phase of the development was available in both UML and Java. Like Java, Python is an object oriented high level language. One possibility to utilize the first phase designs could be to start from scratch and use UML design to build the Cam Reader component. Since automated conversion from Java component classes to Python is readily available, we converted the Java component classes to an equivalent Python code (Text to Text transformation). Python platform is more present in research domain while Java is more towards Industry. Python is a good balance between a high and a low level language and is ideal for video and image processing functionality required in Cam Reader component. Java component classes will be more usable in an industry setup, this was the reason Java and not Python was chosen as the target for Model-to-Text transformations from HTM5-DSL. Fig. 4.6A shows a segment of the developed *Cam Reader* class in Python. Development of components is an independent software engineering process. Vendors develop internal documentation to manage their product, which in this case is

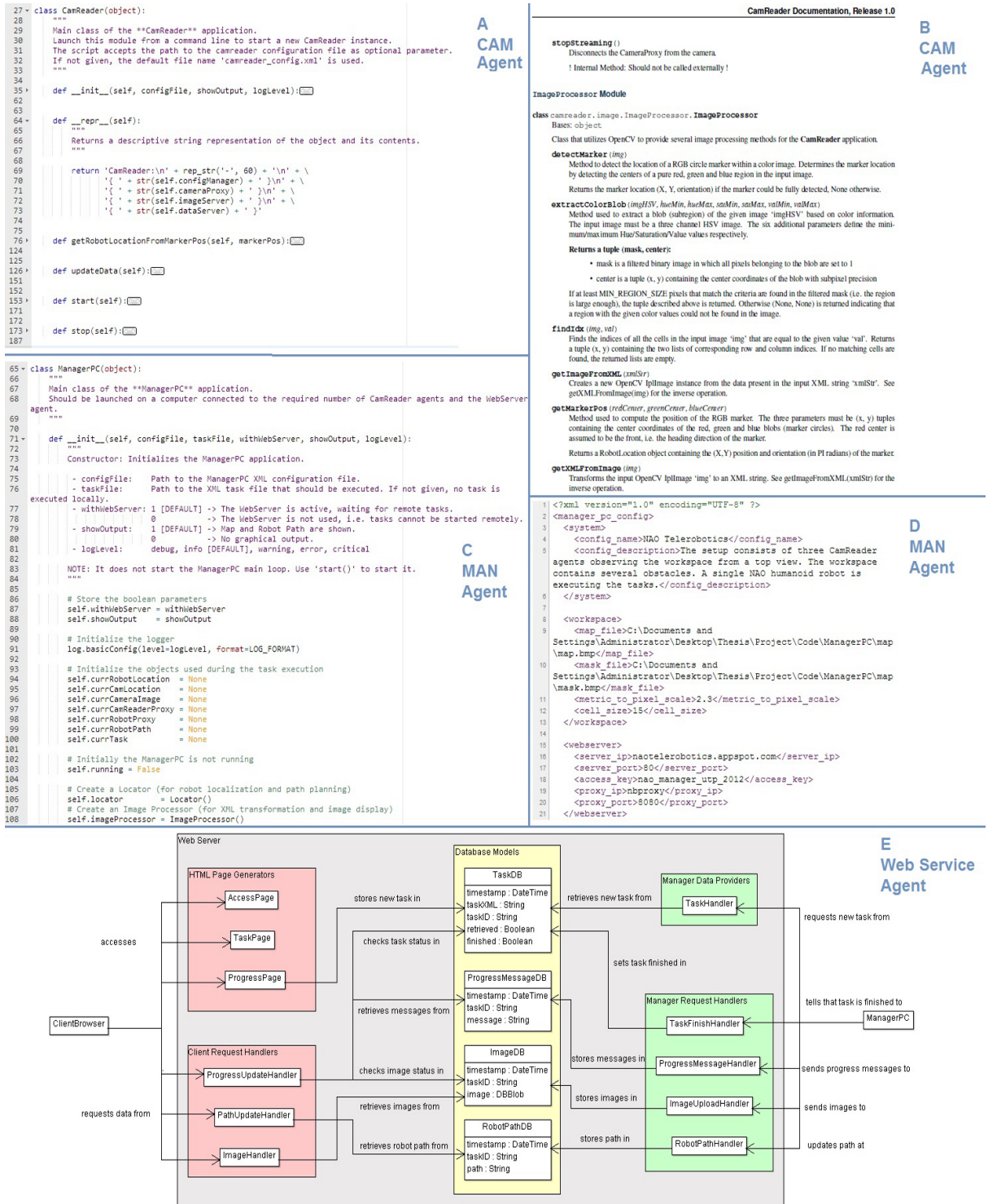
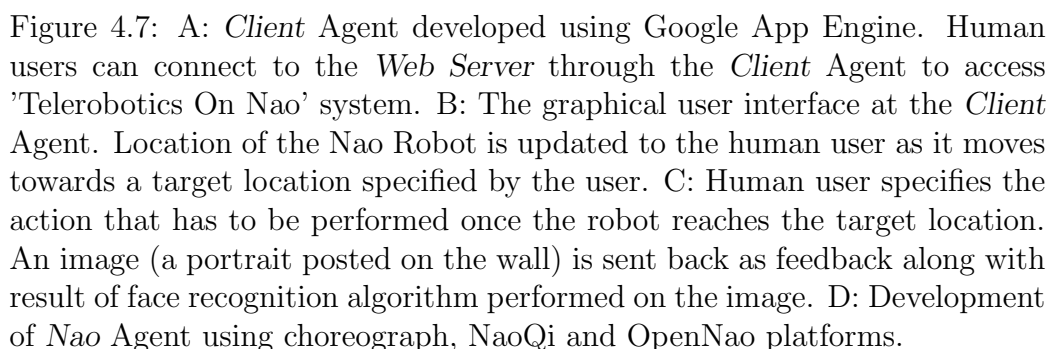


Figure 4.6: Excerpt of Component classes developed independently at various Machines. A: Cam Reader Agent developed in Python. B: Software specification documents for Image processing functionalities at Cam Reader Computer. C: MAN Agent developed at the Manager Computer. D: Configuration file written in XML. The file contains network and interface configurations for MAN Agent. E: Represents interactions of Web Service Agent hosted on Google's Infrastructure with other Agents and Database Servers.



a component class and supporting functionalities. Fig. 4.6B shows a section of the documentation for the image processing functionalities that support the *Cam Reader* component class.

2. **Manager Computer:** Like in the case of *Cam Reader* Agent, the *MAN* Agent was developed by automatically converting the Java component into equivalent Python code. The class was then developed to add functionalities and configurations. XML based configuration files were used to specify communication interfaces with *Web Server*, *Cam Reader computer* and *Nao Robot*. XML is a markup language that defines a set of rules in a format suited for Internet applications. Fig. 4.6C shows a section from the Python code for the *MAN* agent and Fig. 4.6D is part of the configuration file written in XML.
3. **Web Server:** The *Web Server* for 'Telerobotics On Nao' system was implemented using Google's App Engine. App Engine is a service provided by Google to develop cloud applications on Google's Infrastructure. Fig. 4.6E is the functional representation of the *Web Server* and how it connects to other components and Database infrastructure hosted by Google. The *Web Server* represents Google's server banks in the 'Telerobotics On Nao' system. It also represents the server side of the AJAX application that the *Client* Agent utilizes to connect to the 'Telerobotics On Nao' system.
4. **Client Agent:** The *Web Server* agent connects to the human users through *Client* Agents. Users can connect to the *Web Server* by logging on to the system and accessing the graphical user interface provided by the *Client* Agent. Fig. 4.7A shows a section of the code written on Google App engine to implement the *Client* Agent. A user first specifies the target location by clicking on the workspace map and specifies the action a robot should perform once it reaches the location. Image frames from the three cameras locates the robot and its path is updated on the workspace map (Fig. 4.7B). Once the robot reaches the target location, it performs the action specified by the user. The user is updated with action logs on robot's status, the robot may be ordered to take an image of the target and use its face recognition algorithm to identify the person in the image. In robot's workspace, we posted portraits on the walls. Fig. 4.7C shows the image sent to the *Client* Agent as the action feedback. The system can be accessed from anywhere since it is connected to the internet. At one time only one user can access the system, though system can be scaled up to add more robots and more instances of the *Client* Agent.
5. **NAO Humanoid Robot:** *NAO* Agent is developed using *Choreograph*, *NaoQi* and *OpenNao* development platforms provided by Aldebaran Robotics (Developers of Nao Humanoid robotic platform). *MAN* Agent sends instructions to the *NAO* Agent, guiding it through the workspace (avoiding obstacles) towards the target location specified by the human user (through *Client Agent*). Fig. 4.7D shows one of the

sequenced on *Choreograph* that develops the functionality associated with the NAO Agent.

4.1.8 Component and system testing

As components are developed independently, they are first tested at the vendor site. Test cases for components and system are made based on the behavioural designs made in CIM and PIM layers. Once the components are debugged and tested for all test case scenarios, components are tested in pairs. The testing is done in increments and in the end the whole system is tested against test cases based on behavioural ARC diagrams.

The above case study is a good usage scenario for the HTM5 methodology. We saw that various machines in the system had different developmental processes. This is true for any cloud robotic system since the robots and other auxiliary systems are different in terms of their hardware and software constructs. We saw that HTM5 methodology unifies the top layer design of a heterogeneous cloud robotic system without imposing constraints on development of individual components. We believe that the proposed HTM5 methodology can be scaled up from this case study to more complex and extensive cloud robotic systems.

4.2 Dynamic Electronic Institutions

Cloud computing is a business model for the internet. A typical scenario of cloud computing has a serving party that offers its infrastructure, platform or software resources to one or many clients across the network cloud. Cloud service businesses charge their clients based on the quality and volume parameters chosen as and when required by the client. Service contracts, banking and administrative mechanism created the trust envelop that made cloud computing business model a success. When we move the ideas of cloud computing to robotics, there are two kinds of adaptations that will take place. The first kind of adaptation will involve direct modification of cloud services to suit robotic applications while the second kind of adaptation will be on the lines of social and business ideas represented by cloud computing. We believe that this second kind of adaptation will require special tools and development methodologies. Cloud robotic entities include all robotic and non-robotic entities that collectively build a cloud robotic service ecosystem. Using software agents to represent cloud robotic entities will require minimal changes in the way those entities are independently developed by various vendors. Agents are also ideal for implementing social and business concerns of a cloud robotic entity. HTM5 is a 5-view meta-model for model driven development of agent oriented cloud robotic systems. The trade view of HTM5 promotes peer-to-peer exchange of services based on relationships and contracts between participating agents. Agents are autonomous entities with personal goals that may make them greedy in their interactions with other agents. Dynamic Electronic Institutions are modelled on the ideas of Institutions in Human societies. Norms based on trade contracts, social relationships and institutions bring a sense of order in multi agent systems. The aim of this study is to test feasibility of HTM5 methodology in implementing Dynamic Electronic Institutions.

4.2.1 Dynamic Electronic Institutions

Human societies are amalgamation of several norm based institutions that give order to otherwise random interactions. Institutions are structures based on mutual incentives based on predefined contracts. Social, political and economic institutions represent the norms of a society and interactions of its members. Institutes establish standardization in response from a member entity which, in absence of an institution, is free to act solely for its own benefit. Institution helps in controlling the greediness of individual entities and brings order to a system [61]. Electronic Institutions are a relatively new field where the concept of human institutions is extended to Multi Agent Systems (MAS) [4, 5, 6] and Distributed Artificial Intelligence (DAI) [9]. Some early attempts towards the use of organizational metaphors for system modelling systems were presented in [62, 63]. The first approach towards electronic institutions was given in [35] where an abstract notion of *agent-mediated electronic institution* was introduced for the first time. These institutions are described as environments where agents are interacting with other agents under predefined restrictions. An institution is specified by a

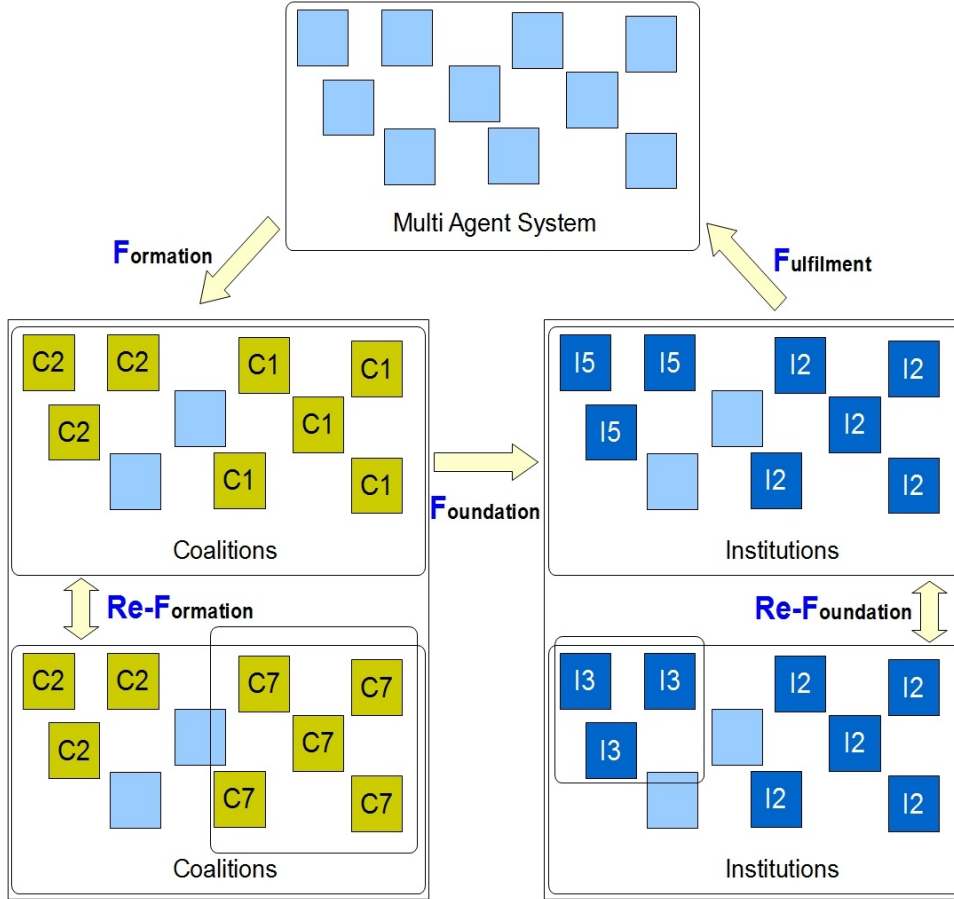


Figure 4.8: 3F Life cycle of a Dynamic Electronic Institution. The three phases are in order: Formation, Foundation and Fulfilment. Re-Formation and Re-Foundation processes are also within the 3F life cycle.

set of pre-defined norms that restrict actions of its member agents. The idea of an electronic institution is very open and various groups [37, 38, 39] are working on this problem with different perspectives. Electronic institutions require limited human intervention for institution design phase. In open agent systems, it is necessary to automate the design phase of institutions.

The term *Dynamic Electronic Institutions* first appeared in the roadmap for agent technology [5]. Dynamic Electronic Institutions are Electronic Institutions where formation, reformation and dissolution of institutions are automated processes. The norms and objectives of institutionalization are dynamically adapting to the needs of its member agents. Fig. 4.8 shows the 3-F life cycle of an institution proposed in recent works [40, 41]. Re-Formation and Re-Foundation processes are also incorporated in the 3-F life cycle.

- **Formation:** Agents with similar objectives come together to form a coalition. A coalition is usually not governed by a set of norms, but trust between agents may play a part in the coalition formation phase. Any logic that governs coalition formation between a set of agents is their *Formation logic*.

- **Re-Formation:** Re-formation is the process of reconfiguring a coalition. A reformation may be triggered by change in coalition membership or a change in parameters responsible for coalition formation.
- **Foundation:** The member agents in a coalition choose a candidate Institution to form. The norms of the candidate institution are based on collective views of individual members of the coalition. Once the target institution is selected, the coalition goes through institutionalization to form an institute of the selected type. This step presents a lot of challenges as selection of the kind of institution and maintenance of institution-base requires a well-defined strategy. In theory, any strategy that assigns an institute to a group of agents (based on their collective decision parameters) is qualified as a *Foundation Logic*.
- **Re-Foundation:** Re-foundation is the process of reconfiguring an institution. A reconfiguration may be triggered by a change in environment variables or a foundation-timeout value set by *Foundation logic* (at the time of institution foundation).
- **Fulfilment:** The member agents in an institute dissolve into individual free agents when the institute completes all its objectives. An institute may also fulfil when triggered by a fulfilment-timeout value set by *Foundation logic* (at the time of institution foundation). Like foundation, fulfilment is also a challenging logic to device. The decision may be based on weighted percentage of collective goals of member agents, or time elapsed since institution foundation / re-foundation. *Fulfilment logic* is usually devised at foundation state when the institute candidate is selected by *Foundation logic*.

4.2.2 Dynamic Electronic Institutions in Cloud Robotics

There are many application domains where Dynamic Electronic Institutions (DEIs) are applicable. In agent oriented cloud robotics, DEIs are applicable in tasks involving a contract based cooperation amongst ad-hoc teams. Collaboration between agents which were not originally programmed to work together is examples of Ad-Hoc teams. Common scenarios where DEIs are used are Ad-Hoc mobile networks, B2B electronic commerce and OOTW (operations other than war) scenario. For the current thesis we will concentrate on B2B electronic commerce as a scenario for agent oriented cloud robotic system. Following the 3F life cycle in B2B electronic commerce, following analogy was proposed by [40, 41].

- A Digital B2B electronic commerce ecosystem: *Agent Community*
- A Business Opportunity: *Coalition*
- Digital Business Ecosystem (DBE): *Dynamic Institution*

The aim of the DEI scenario mentioned above is to find new opportunities to exchange services amongst member agents. Formation of a coalition represents a viable service exchange (a business opportunity). When the member agents agree upon a set of norms to execute the service exchange, it is represented as foundation of an institution. Fig. 4.9 shows an example where DBE is applied to an agent oriented cloud robotic ecosystem. Individual cloud entities may have different hardware and software setup. Each of these entities may have a business owner and set of offered services. An approach based on representative agents ensures that heterogeneity in business logic, design methodology and implementation of these products is respected. In a real life scenario, it may be required to have a minimal human involvement in the 3F life cycle (see Fig. 4.8). A proposal by [40, 41] advocates a 7 step life cycle where human decision makers are involved at two stages (when a business opportunity is detected and when a contract needs acceptance) to validate business opportunities detected by the DBE system.

Steps of the DBE lifecycle [40, 41] :

1. Search of opportunities (Formation Logic)
2. Analysis of opportunity by business owner (Validation I, Optional)
3. Coalition establishment (Formation and Re-Formation Phase, Re-formation will require re-validation)
4. DBE selection (Foundation Logic)
5. Acceptance by business owner (Validation II, Optional)
6. DBE establishment (Foundation and Re-Foundation Phase, Re-foundation will require re-validation)
7. DBE Finalization (Fulfilment Phase)

4.2.3 Dynamic Electronic Institutions in HTM5

In the previous section we saw the steps in a DBE lifecycle and its applicability in the cloud robotic domain. HTM5 [58, 59, 60] is OMG-MDA [3] based meta-model for development of agent oriented peer-to-peer cloud robotic systems (See Chapter 1). The meta-model is designed to provide tools to implement advance distributed Artificial Intelligence (DAI) designs in a cloud robotic ecosystem. In this section we discuss the tools and anatomical elements of HTM5 that are utilized to implement a Digital Business Ecosystem based on Dynamic Electronic Institution.

Fig. 4.10 shows an example of a Dynamic Electronic Institution with two institutions. The HTM5 methodology allows for special agents called *Relations* that maintain the relationships between groups of agents. For the example shown in Fig. 4.10, the relations are designed to act as Institution seeds. No anatomical change is required in HTM5 to use *HTM5 relation* construct as Institution seeds. An institution between groups of agents can

be seen as a special kind of relationship. In HTM5, the norms and relationship variables of a relationship are hosted and managed by the *Relation* agent. When HTM5 is used to implement Dynamic Electronic Institutions, the variables, formation and foundation logic may be hosted in institution seeds (which are relation agents). For ease of implementation, an institution may be implemented as two separate agents. In Fig. 4.10, the *Manager agent* along with *InsA relation* and *M4 merge* are all hosted at one machine. It is possible to implement the logic of all three components (Manager, InsA and M4) onto one agent but separation and placement of machine functionalities into agent, merge and relation specific parts is encouraged in HTM5. In Fig. 4.10 Part II is a normalized version of the ARC diagram shown in Part I. Part III is the Trade-Agent Relation Chart (Trade-ARC) that defines the following trade relationships and dependencies between members of *Sandbox* cloud robotic ecosystem.

Trade relationships and dependencies:

1. Trade **Search Space** is a service provided by *Trader* to *Manager* and assigns a search space in the mine for the *Manager*.
2. Trade **Sub Search Space** is a service provided by *Manager* to *Miners* and assigns a section of the search space (allocated to the *Manager*) to the *Miner*.
3. Trade **Sub Space Hit** is the notification service from the *Miner* agent when it detects the target mineral. This service is a demand at *Manager* agent.
4. Trade **Initial coordinates** locates the found mineral in the mine field. This service is a demand at *Trader* agent.
5. Trade **Miner Salary** is the payment that a *Miner* agent receives in exchange of the mineral locations it delivers to the *Trader* agent. The service is a demand at the *Bank* agent which transfers the amount from *Trader's* to *Miner's* account.
6. Trade **Cargo** is a service by the *Trader* agent to the *Market* where it sells the acquired mineral locations.
7. Trade **Cargo Payment** is the payment that a *Trader* agent receives in exchange of the mineral locations (Cargo) it delivers to the *Market* agent. The service is a demand at the *Bank* agent which transfers the amount from *Market's* to *Trader's* account.
8. Demand Lookup Table **Miner ID — Salary** is a lookup table to get the desired salaries (Trade: **Miner Salary**) by individual *Miner* agents.
9. Demand Lookup Table **Mineral — Price** is a lookup table to get the prices of different minerals (Trade: **Sub Space Hit**) by individual *Trader* agents. A *Miner* agent chooses its target mineral based on the current price of minerals.

10. Demand Cost Metrice **Mined Area Pc**. Is a trade variable to check the percentage of mine's area that is already explored. This is a metrice to know when to allocate a new search space (Trade: **Sub-Search Space**) to individual *Miners*.
11. Service Lookup Table **Trader ID — Cargo** is for the trade **Cargo Payment** and is used by the *Market* agent to initiate cargo payments.
12. Institution **InsA** is an institution between **one** *Manager* agent, **one** relation *InsB*, **one** *Bank* agent and **Nm** *Miner* agents. The institution manages the allocation, reallocation and management of mine spaces for individual *Miners* and allows for dynamic updating of a *Miner's* asking salary based on inputs from *Bank* (*Miner's* current bank balance) and *Market Server* (updated prices of different minerals).
13. Institution **InsB** is an institution between **one** *Market* agent, **one** relation *InsA*, **one** *Bank* agent and **Nt** *Trader* agents. The institution manages the allocation, reallocation and management of Cargo items for individual *Traders* and allows for dynamic updating of a *Trader's* asking price for individual minerals based on inputs from *Bank* (*Trader's* current bank balance) and *President Machine* (updated percentage of Mined area).

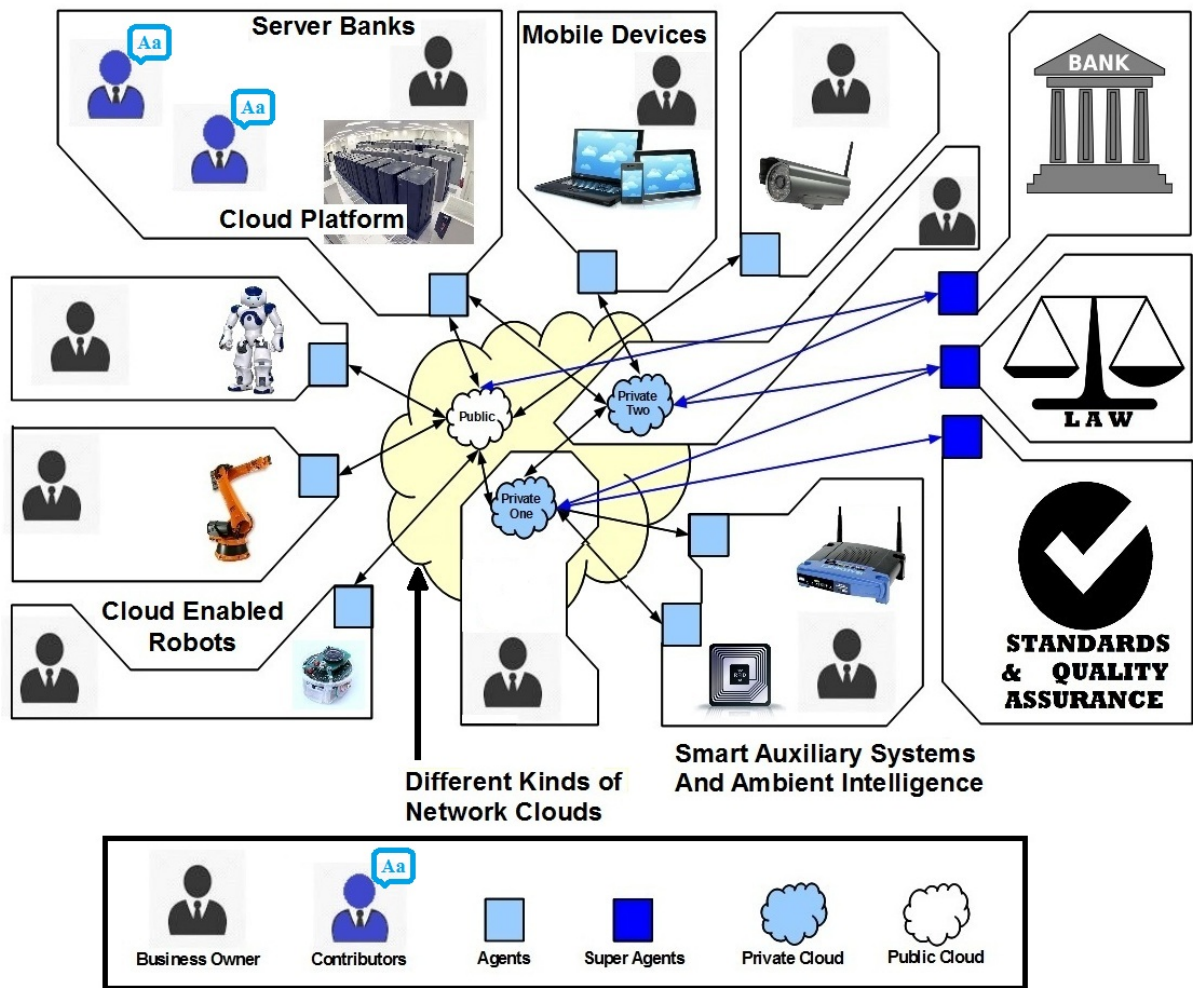


Figure 4.9: Example of Digital Business Ecosystem (DBE) in an Agent Oriented Cloud Robotics environment. Agents are independent software entities that represent a robotic/non-robotic entities in cloud robotic ecosystem. Individual cloud entities may have different hardware and software setup. Each of these entities may have a business owner and set of offered services. Some online server banks may have a number of contributors which build up a resource (like algorithms and other internet resources) which is then offered as a service to other entities through cloud. An open registry and matchmaking service allows peer-to-peer trade of these services in cloud robotic ecosystem. The network cloud itself may have many private and public networks, some of which may be owned by businesses that allow their use as a service. Banking super agents (Agents with special access norms) allow actual transfer of money between agents. Other super agents may be present to enforce law, quality, communication and trade standard over the agent community. Development life cycle of cloud robotic products may differ from vendor to vendor. Business model of organisations that deploy these products may also differ. Individual entities may enter or exit the ecosystem dynamically and their behaviour may change with time. An approach based on representative agents ensures that heterogeneity in business logic, design methodology and implementation of these products is respected.

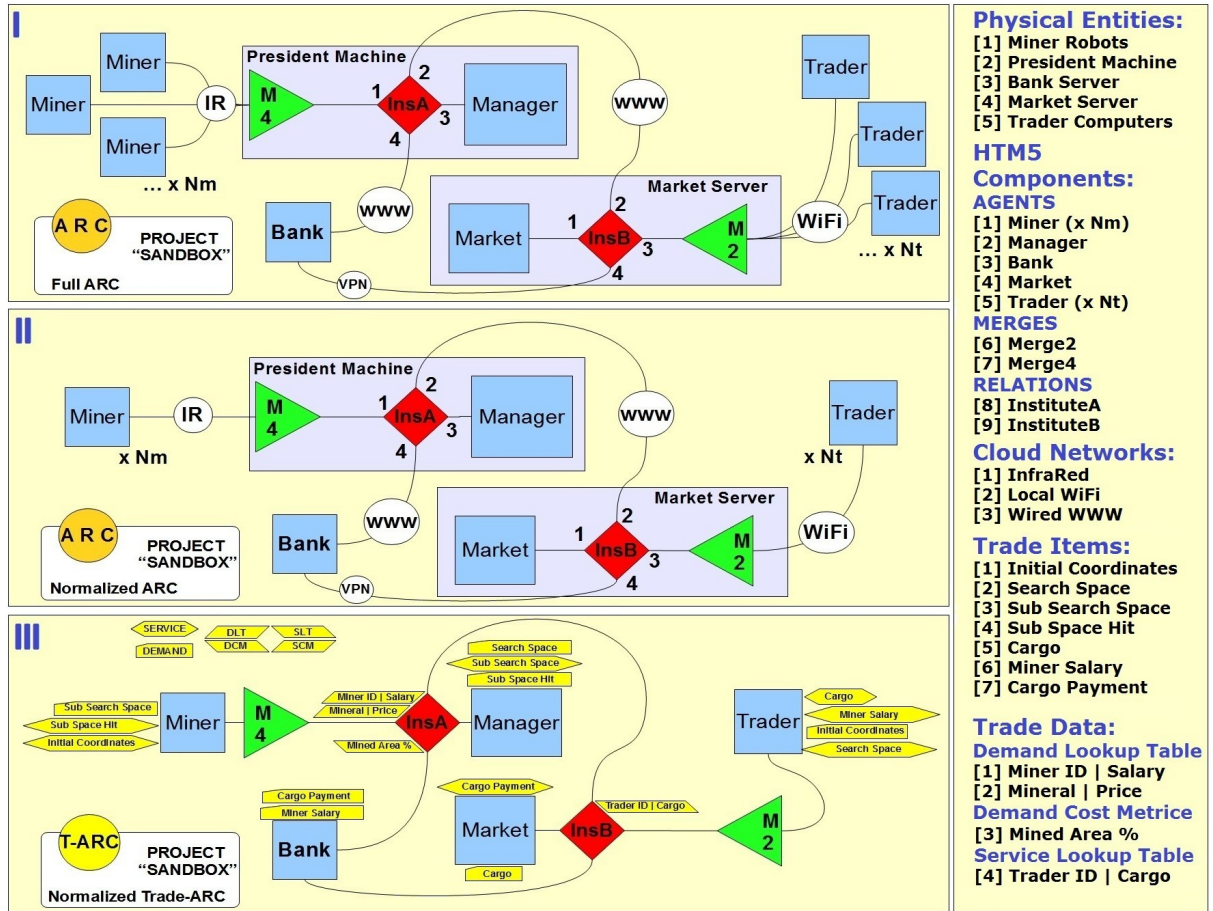


Figure 4.10: An example of Dynamic Electronic Institutions implemented using HTM5 methodology. Above are Agent Relation Charts (Computation Independent Layer of HTM5) for structural, relational and trade views of HTM5. The sample project *Sandbox* is a Digital Business Ecosystem example with two institution seeds. Part I of Figure is a full ARC diagram of the *Sandbox* system while part II is its normalized version. There is multiple numbers of Miner and Trader agents in the system. Part III models trade dependencies in *Sandbox* cloud ecosystem. Peer-to-Peer trade relationships exist between members of a relationship (here relationships are modelled as Institutions). Institutes (Relation Agents) manage trade within an institute and host service and cost lookup tables along with other trade data items. Social and business logic of an Institute is implemented at relation agents. Detailed description of ARCs and HTM5 anatomy can be found at [57, 58, 59, 60]. The above example locates elements of Digital Business Ecosystem in a HTM5 based implementation.

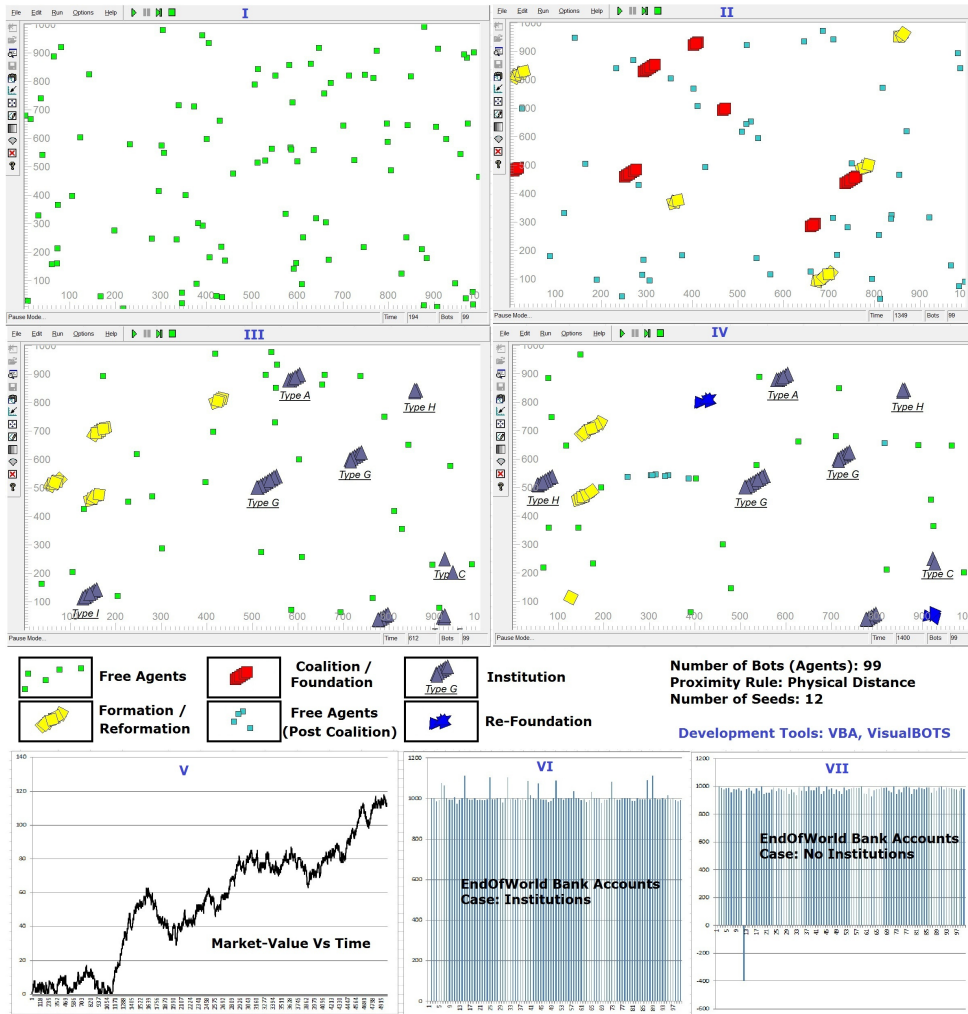


Figure 4.11: Above are some instants from Dynamic Electronic Institutions based Digital Business Ecosystem simulations. Part I shows a colony of BOTs (Agents) randomly moving in an environment. There are seed BOTs which are not moving and they act as the open ended joining point for other BOTs to form a coalition or an institution. Part II shows some of the seeds in Formation phase (3F life cycle) while some of the groups have formed a coalition and are now in Foundation phase. Part III shows formation of eight Institutions of different cardinality and type. Some of the groups are in Formation or Re-Formation phase. In going from Part III to Part IV two institutions have moved to Re-Formation phase while one of the institutes (of Type I in Part III) has finalized freeing its member BOTs. All simulations are run with different experimental conditions and against different Market trends. When institutions are not allowed to be formed, the groups come together without setting up a set of norms and are dissolve as soon as a minimal number of member BOTs are unhappy with the current market direction/trend. Part V shows sample graphs of market values over a period of 5000 event steps. The results of running the experiment without institution formation are matched to the scenario where institutions can be formed. The profits of all BOTs are visually inspected at the end of the experiment in graphs shown in Part VI and VII. ARC designs, analysis of experimental results and a scaled down version of simulation experiments on physical TurtleBOTs is presented in Fig. 4.12 and Fig. 4.13.

4.2.4 The Experiments

The case study for the use of HTM5 methodology in developing a Dynamic Electronic Institution based Digital Business Ecosystem was executed in two phases. In the first phase computer simulations using VisuaBOT [64] and VBA [65] were developed for conducting various experiments on the agent colony. Economic comparisons were made between agent colonies working with and without dynamic electronic institutions. In the second phase, a scaled down version of the experiments performed on the simulations were implemented on five physical robots (TurtleBOTS [66]). The principle motivation for these case study experiments is to show HTM5's feasibility in implementation of a Dynamic Electronic Institution based cloud robotic ecosystem. This section discusses the system design, experiments, results and key observations. HTM5 is proved to be a usable methodology for this implementation since the experiments were performed using anatomical and design constructs prescribed by HTM5. Fig. 4.11 shows some instances from the simulated experiments. Fig. 4.12 Part I and II show the physical TurtleBOT robots on which the scaled down versions of the experiments were performed. The Agent Relation Charts for simulated and physical experimental setup is shown in Fig. 4.12 Part III, IV. Due to a lower number of physical robots, the physical experiments were based on location based institution seeds. This is unlike the simulated experiments where institution seeds are assigned to agents and not their parking locations. Some run time videos of the simulation experiments and experiments (as part of HTM5 feasibility studies) on the physical TurtleBOT robots are available at [67].

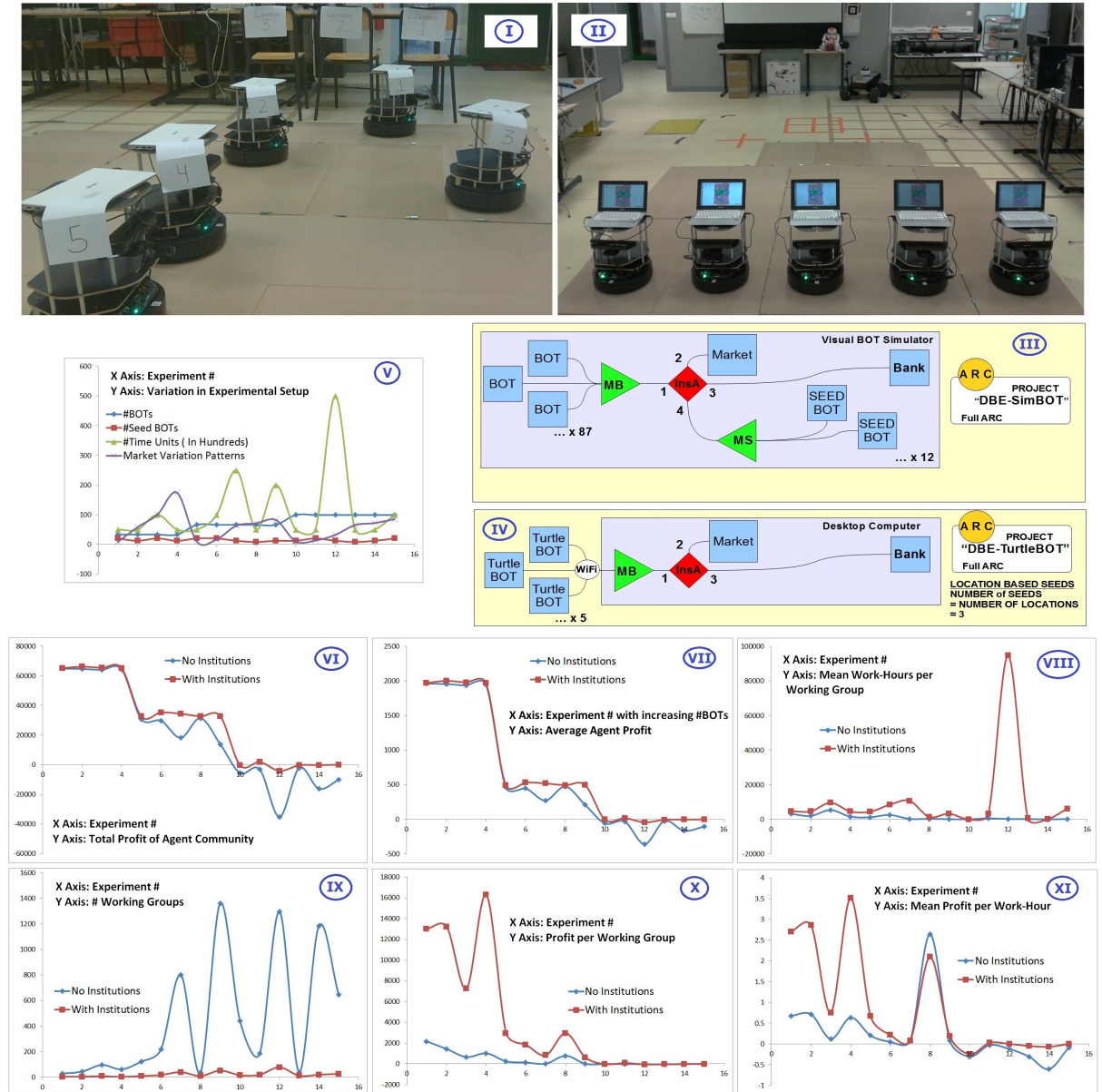


Figure 4.12: Part I and Part II of the above Figure shows the TurtleBOT setup for a scaled down (5 BOTs 3 location based seeds) version of Dynamic Electronic Institutions based Digital Business Ecosystem. Part III shows an Agent Relation Chart for the experiments on the VisualBOT simulator. All agents are running in the simulator environment (no real hardware) and the seeds are labelled by default. Part III shows ARC for a scenario with 99 BOTs of which 12 are seed BOTs. For experiments on real robots (TurtleBOTs), the seeds were based on the parking location of a robot. TurtleBOT Agents were running on individual TurtleBOT robots while Merge MB, Relation InSA, Market and Bank agents were running on a Desktop computer connected to TurtleBOTs through wireless LAN (WiFi). Part IV shows the modified ARC diagram from Part III for real life experiments. Control variables for the simulator experimental setup were varied to get a mix set of usage scenarios. Part V shows values for number of BOTs, number of Seed BOTs (Seed bots are designated BOTs that initiate institution formation. Other BOTs attach themselves to a seed BOT when the process of coalition/institution formation starts), length of Experiment and the kind of market variation pattern (generated based on a random integer as a seed input) used for all 15 experiments. The detailed textual outputs of these 15 experiments are presented in Fig. 4.13. Parts VI, VII, VIII, IX, X and X of the above image are graphical representations of some pair of result variables.

						NO Institute			With Institute			
#	#BOT	#Seed	EoW	Rand#	Bnk t0	Bnk EoW	Profit	Profit/BOT	Bnk EoW	Profit	Profit/BOT	Advantage
1	33	20	5000	8	33000	97798.1	64798.1	1963.578	98086.1	65086.09	1972.3058	288
2	33	12	5000	58	33000	97579.1	64579.1	1956.942	99102.1	66102.09	2003.0936	1523
3	33	20	10000	100	33000	96960.1	63960.1	1938.185	98335.1	65335.09	1979.8512	1375
4	33	12	5000	174	33000	97319.1	64319.1	1949.063	98164.1	65164.09	1974.6694	845
5	66	20	5000	9	66000	96736	30736	465.697	98515	32515	492.65152	1779
6	66	20	10000	17	66000	95675	29675	449.6212	101192	35192	533.21212	5517
7	66	12	25000	63	66000	84109	18109	274.3788	100344	34343.73	520.35955	16234.73
8	66	8	5000	71	66000	97347.1	31347.1	474.9559	98593.1	32593.09	493.8347	1246
9	66	12	20000	82	66000	79902	13902	210.6364	98739.9	32739.89	496.05894	18837.89
10	99	12	5000	10	99000	93377	-5623	-56.79798	98651	-349	-3.525253	5274
11	99	20	5000	13	99000	95820	-3180	-32.12121	100781	1781	17.989899	4961
12	99	12	50000	32	99000	63743	-35257	-356.1313	94826	-4174	-42.16162	31083
13	99	8	5000	65	99000	96732	-2268	-22.90909	98570	-430	-4.343434	1838
14	99	12	5000	72	99000	82738	-16262	-164.2626	98680	-320	-3.232323	15942
15	99	20	10000	85	99000	88810	-10190	-102.9293	98960.1	-39.875	-0.402778	10150.13
1056	220	2E+05					308645			425539.2		116893.7

NO Institute					With Institute				
#Set	Time/Set	Time	Profit/Set	Profit/Time	#Set	Time/Set	Time	Profit/Set	Profit/Time
30	3184.88	95546.4	2159.94	0.6781845	5	4806.85	24034.3	13017.2	2.7080558
45	2007.1	90319.5	1435.09	0.7150072	5	4616.99	23085	13220.4	2.8634279
97	5432.29	526932	659.382	0.121382	9	9627.67	86649	7259.45	0.75401987
63	1607.5	101273	1020.94	0.6351091	4	4637.79	18551.2	16291	3.51266929
124	1218.76	151126	247.871	0.2033796	11	4383.83	48222.1	2955.91	0.67427548
220	2580.58	567728	134.886	0.0522698	19	8482.91	161175	1852.21	0.21834612
803	288.59	231738	22.5517	0.0781444	41	10760.9	441197	837.652	0.07784218
40	296.117	11844.7	783.677	2.6465122	11	1413.73	15551	2963.01	2.09587982
1362	128.332	174788	10.207	0.0795363	53	3285.38	174125	617.734	0.18802506
442	41.4437	18318.1	-12.7217	-0.306964	17	82.6364	1404.82	-20.5294	-0.24843062
187	590.862	110491	-17.0053	-0.028781	21	3340.16	70143.3	84.8095	0.02539089
1297	223.237	289538	-27.1835	-0.12177	79	94826	7491254	-52.8354	-0.00055718
41	180.29	7391.89	-55.3171	-0.306823	13	664.417	8637.42	-33.0769	-0.04978338
1185	22.671	26865.1	-13.7232	-0.60532	21	224.765	4720.07	-15.2381	-0.06779568
647	185.93	120297	-15.7496	-0.084707	28	6237.08	174638	-1.42411	-0.00022833
6583	17988.6	2524196	6332.84	3.7551611	337	157391	8743388	58976.3	12.7511372

Figure 4.13: A Figure showing the table of aggregated results from the 15 experiments performed with varying controls on simulated Agent ecosystems. *EoW*: End Of World in event units of time. *Bnk t0* and *Bnk EoW* are combined Bank balance of all agents in an agent ecosystem. Advantage is the additional profit that the use of institutions brought to an agent ecosystem (All Positive means it is always advantageous to use institutions in current environmental setting). Number of *sets* (working groups) is the number of coalitions established during the experiment when institutes cannot be formed. When institutes can be formed, the number of Sets is the number of unique institutions formed (Re-Foundation counts as a new institute).

Institution Election and Maintenance of Institution Database :

In the foundation phase of the 3F life cycle of a Dynamic Electronic Institution, member agents of a coalition negotiate amongst themselves to choose a type of institution. The mechanism can be implemented in many ways of which Case Based Reasoning and Meta-Institute approach are most common [40]. For our experiments we have adopted a mixed approach where every Institute in the institute-base (a database of institutions) has a feature vector (a sequence of variables) that specifies the financial and social parameters of a particular institution (e.g. Institution size, Measure of institution's intent to stay in a falling market, Measure of an institution's will to not sell at low profit and aim for a higher profit in a raising market). When an agent is defined, it has its own feature vector that specifies financial and social parameters that an agent is happy with. In the foundation (or re-foundation) phase, the feature vectors of all the member agents of a collation are combined to form a mean feature vector. The mean feature vector is compared with the feature vectors of all the institutes in the institution base and the institute whose feature vector is closest to the mean feature vector is chosen for institution formations. Any institution with high performance is given additional weightage in the institution database. The feature vectors of institutions and the list of institutions in the database are thus dynamically adapting to the market values and profit trends (One element in the feature vector is the reputation of a particular institution type).

Variability in experiment control variables : We performed a number of different experiments on the agent colony of which 15 are presented as this case study. Fig. 4.12 Part V shows the variation in control parameters for the 15 experiments. Fig. 4.13 shows the control variables and results of these experiments in a tabular form. The control variables are: (1) Number of BOTs (agents) in the agent colony, (2) number of seed BOTs, (3) the time for which an experiment is allowed to run and (3) variation in market values (Fig. 4.11 Part V shows one of the 15 different market variation patterns used for the experiments). Fig. 4.12 Part VI to XI and Fig. 4.13 are graphical and tabular representations of outputs of these experiments.

Key Observations :

1. Fig. 4.12 Part VI shows that profit of the agent community is always greater when they operate with institutions. This is also visible by absence of any negative value in the *Advantage* column in Fig. 4.13.
2. Fig. 4.12 Part VII shows that average profits of agents are always greater when they operate within institutions irrespective of the size of the agent community.
3. Fig. 4.12 Part IX shows that the number of working groups is very high when agents do not operate with institutions. This is due to greedy nature on individual agents which force a working group to dissolve when their personal goals are not fulfilled. Institutions on the other hand are very low in number as they are dissolved when a collective decision is made.

4. Another observation from Fig. 4.12 Part IX shows that the number of working groups with institutions have fluctuations of greater magnitude with changing market patterns (as institutes are less sensitive to minor fluctuations in market trend).
5. Profit per working group is mostly higher in the case of institutions (Fig. 4.12 Part X and columns *Profit per Set* in Fig. 4.12).
6. Fig. 4.12 Part VIII shows that the working groups sustain for longer duration when they operate as institutions (More work hours per working group). The peak in Experiment 12 suggests that the man hours per working group go higher as experiments run for longer duration (Experiment 12 is the longest experiment).
7. With a few exceptions, an institution based ecosystem gives more profit per work hour (Fig. 4.12 Part XI).

Profits tend to be greater in general with institutions as unlike coalitions, institutions decide on the dissolution rules at the beginning of institution formation. In coalition, whenever an agent is not happy with the current market trend, it can leave the coalition leading to the dissolution of the complete coalition. In institutions, none of the member agents has any power over when an institute will dissolve (after the institution starts to function) thus the institutions are in general not susceptible to greedy behaviours of individual agents. Another factor that leads to greater profits in institution based economy is since the popularity of an institution is also a factor in its selection; a profitable institution structure is more probable to be chosen for institution formation.

4.2.5 Conclusion and future direction

In this work we presented Dynamic Electronic Institutions (DEI) implementations in HTM5 meta-model for agent oriented development of cloud robotic systems. Digital Business Ecosystem (DBE) is one application domain of dynamic electronic institutions in cloud robotic colonies. HTM5 meta-model is designed for including distributed artificial intelligence designs on cloud robotic ecosystems. Peer to peer trade based on relationships between agents, representing heterogeneous cloud entities in the cloud using agents, an OMG-MDA based three layered design and its domain specificity makes HTM5 a suitable methodology for development of agent oriented cloud robotic systems. The case study, examples and discussions presented in the current thesis give sufficient evidence that HTM5 is a feasible methodology for implementing complex trade and institution logics on a cloud robotic system. The complete HTM5 model, a domain specific language supporting HTM5 and a case study specific for peer to peer trade variability in HTM5 is currently submitted to well-known journals. The next step in this direction is to test the methodology in real life industrial projects and to improve the model by industrial feedback. A detailed user guide and a graphical design interface for HTM5 based designing is also currently under development.

4.3 Peer-to-Peer Cloud Trade Ecosystem

Cloud robotics is a general term to specify the use of cloud in robotic applications. Cloud is a term used to represent computer networks and thus any application where robotics utilizes a computer network to connect with other network entities is a cloud driven robotic application. It is important to remember that cloud computing is not a new computing or network technology. Cloud computing is a business model and a methodology that utilized existing technologies in a particular manner. Cloud computing is the business of offering one's resources to entities across the cloud at a price that is regulated by quality and quantity of utilized resource. When cloud robotics emerges as a new domain, it not only adapts existing cloud computing services but also adopts the cloud computing business model. A methodology for development of cloud robotic systems should have provisions to express and implement business ideas represented by cloud businesses. HTM5 is a meta-model that is designed for agent oriented development of cloud robotic systems. The HTM5 methodology has 5 views of which the Trade-view has models for specifying designs for peer-to-peer services oriented trade in cloud robotic ecosystems. In this work we first explain the anatomical elements in HTM5 that support peer-to-peer trade in cloud robotic systems. We present a case study implemented using HTM5 methodology that studies behaviour of simulated robot colonies deploying peer-to-peer trade. A scaled down version of these experiments were repeated on physical robots. The motivation behind the current work is to showcase HTM5 as a feasible meta-model for design of peer-to-peer, service oriented trade in agent oriented cloud robotic systems.

Cloud computing is the current generation of internet computing which is currently evolving to become a peer-to-peer system where every cloud entity is a potential service provider [68, 69]. The next generation peer-to-peer cloud computing will give rise to a business ecosystem where entities in a cloud could freely share their resources as services without a centralized cloud server. Peers provide resources to other peers and reduce cost and dependency on original service provider. Addition of new peers increase the demand of existing resources but they also contribute to the pool of resources shared between all the peers. Most robots are entities that are capable of performing a physical action. In essence every robot is a potential service provider and a peer-to-peer cloud robotic framework is a more suitable methodology for robots working in a common physical environment. The shift from the current client-server like cloud robotics to peer-to-peer cloud robotics will enable robotic ecosystems to avail cloud resources as well as contribute to the pool of cloud based services. Service oriented peer-to-peer cloud robotic methodologies could emerge to a whole new sub-domain in multi-robot systems and Distributed Artificial Intelligence (DAI) [9] systems. Design and development of these systems will require special design tools, meta-models [2] and development methodologies.

"Software Agents are computational entities with specific roles and personal objectives working in a visible environment with other entities which may have dissimilar roles and objectives" [8]. Agents are by design better

suited to implement dynamically evolving business logic and thus representing business interests of cloud robotic entities through representative agents is an attractive proposition. Agent oriented cloud robotic systems also promote inclusion of Dynamic Electronic Institutions [37, 38, 39, 35] and Digital Business Ecosystem (See Fig. 4.9) [40, 41] in a cloud robotic ecosystem further enhancing the usability of the approach.

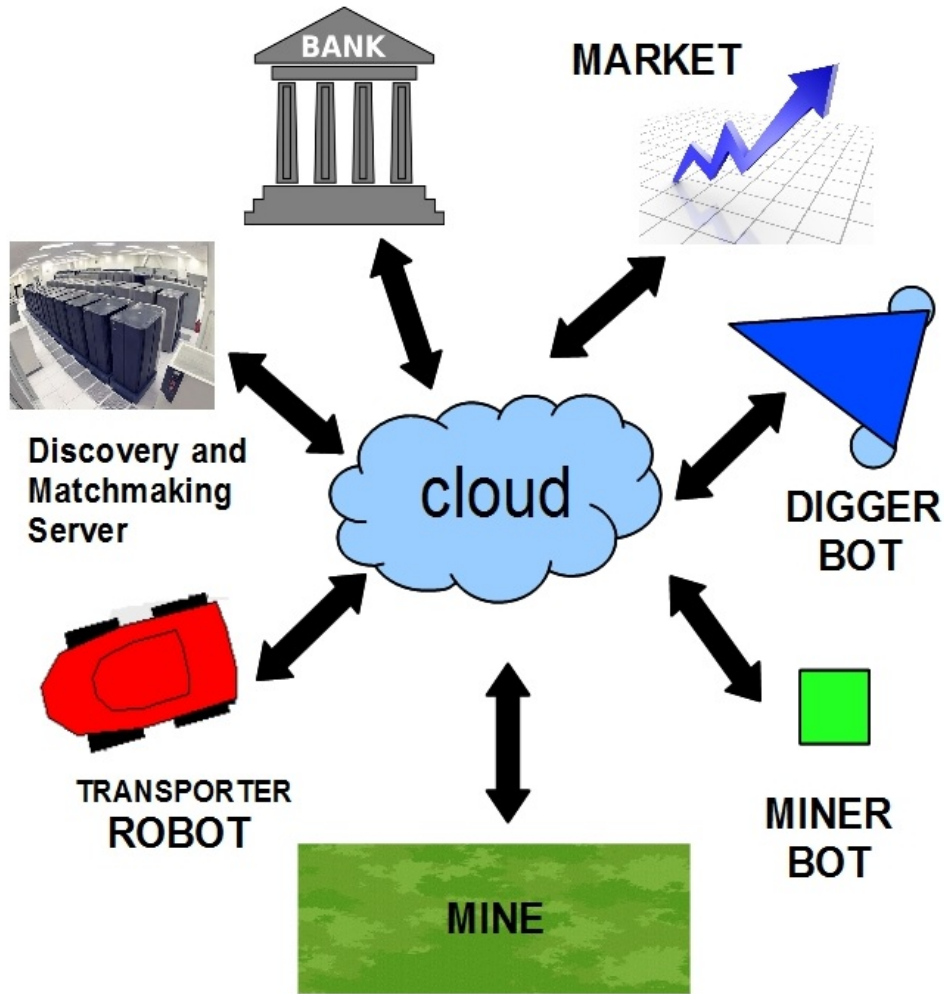


Figure 4.14: The above figure shows cloud entities in the "Mine Cloud" cloud robotic system. "Mine Cloud" is a simulated agent colony representing robotic and non-robotic entities in a digital business ecosystem implemented on a cloud robotic system. The physical robots form short term collaborations to extract minerals from the mine. Miner BOT are robots which select a target mineral based on the current market prices and searches for that mineral ore in the mine field. Once the *Miner BOTs* detects the target mineral, it then hires a *Digger BOT* from a pool of available digging robots based on their ground speed, service delay (digging time) and cost of service parameters. Once the mineral is dug up, another service discovery and matchmaking mechanisms associates *Miner BOT* with a suitable market (based on offered price) and a *Transporter ROBOT* (based on ground speed, loading and dumping time and cost of service). The transfer of money is managed by a banking agent which transfers service fee from one agent account to another. All service providers advertise their quality and cost parameters in respective registries hosted on relation agents. The point of comparison in this experiment is profits made (and system's mining productivity) using a peer-to-peer trade mechanism against fixed teams of collaborating agents. The case study however primarily tests HTM5 as a feasible meta-model for implementing peer-to-peer and fixed-team based trade methodologies.

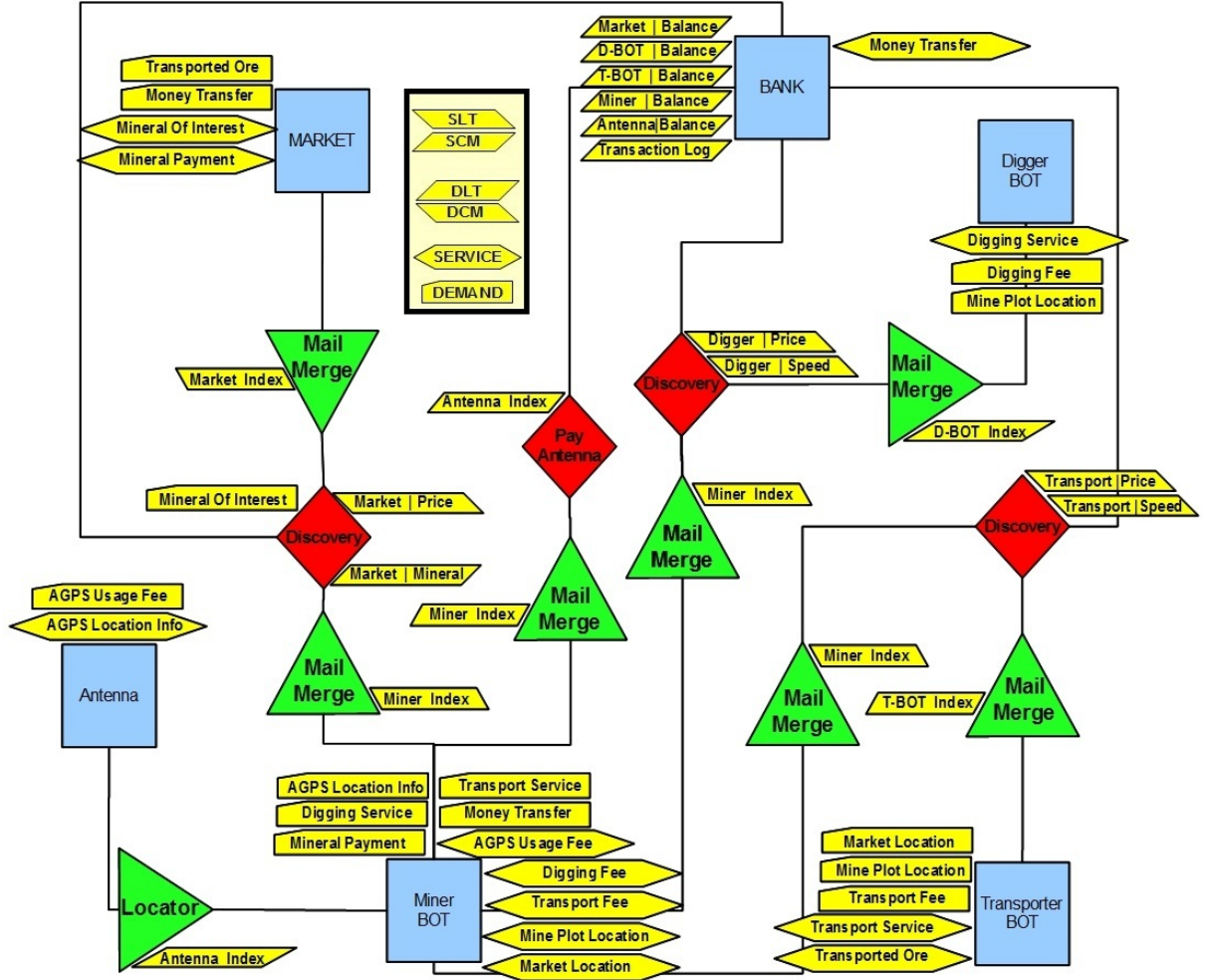


Figure 4.16: Above is the Trade-Agent Relation Chart (T-ARC) for the "Mine Cloud" cloud robotic system. The structural and relational elements of this system are specified in the ARC diagram shown in Fig. 4.15. The Trade-ARC specifies the trade dependencies in the peer-to-peer service economy. Services, demands, lookup tables (Service LT, Demand LT), cost metrics (Service CM, Demand CM), agent indexes and trade variables are specified with respect to each trade item in this peer-to-peer trade ecology. Special *Relations* (e.g. Discovery, Pay Antenna) manage service discovery and matchmaking mechanisms and are assisted by several open ended *Merges* (Mail Merge, Locator) to allow dynamic teams.

4.3.1 Peer-to-Peer Trade modelling in HTM5

In Chapter 1 of this thesis we discussed the advantages of peer-to-peer cloud robotics over a traditional client-server like model. We believe that a typical cloud robotic system will have elements of both peer-to-peer and client-server mechanisms. A meta-model designed for modelling such systems should enable specification of both kinds of design elements. HTM5 meta-model by itself does not inhibit or endorse a particular kind of trade model. HTM5 has 5 different views (models that separate a particular kind of design concern) of which the trade view is designed to support both peer-to-peer and a traditional client-server based trade model. In this work we present the anatomical elements of HTM5 that enable peer-to-peer trade modelling based on relationships and contracts between groups of agents.

HTM5 classifies its agents in three main categories. Agents which represent a cloud entity in the cloud ecosystem is a regular *Agent* (represented by rectangles in Agent Relation Charts) while agents which exist in the system to serve a managerial purpose are given special names and graphical representations. Agents which may/may not represent a cloud entity, but are managing relationships between other agents are called Relational agents or *Relations*. *Relations* are represented by rhombuses in Agent Relation Charts and host social and business logics of a relationship. Two or more agents may be attached to relation agents directly or through a *Merge* agent. *Merges* or merge agents are special agents which are ports to which new agents can join a cloud robotic system. The primary functionality of *Merges* is to manage the flow of messages between agents and manage the open [7] system characteristics. All three kinds of HTM5 agents (*Agents*, *Relations* and *Merges*) are further classified as *Active*, *Passive* or *Hyperactive*. This classification is based on the degree to which an agent's autonomy is released to other agents and the extent of object like character they exhibit. For the scope of current thesis, further discussion on *Hyperactivity* characteristics is extraneous.

Implementing cloud computing business logic in cloud robotic systems will require a mechanism for relationship based trade contracts. Out of all available services, an agent may have a business compulsion to prefer or reject certain service providers. Business logics of business owners that deploy cloud entities will require a place to exist in the system. These dynamically evolving transaction controls can be hosted in an agent's trade view class or at the *Relation* agent that is managing an agent's trade with other agents. Advance trade concepts like Digital Institutions [37, 38, 39, 35] and Digital Business Ecosystem [40, 41] could also be implemented through *Relation* agents. An open system [7] is a system where third party entities can plug in and join the system. In an open system the entities are free to join or leave the system at will. In HTM5, open systems can be implemented using *Merges*. An open cloud robotic system could define ports at which agents can dynamically join or leave the system. In HTM5 such ports could be implemented using *Merges*. *Merges* could be used to specify Ad-Hoc open systems with an unknown number of third party entities. *Merges* could also be used when within a known system, agents switch positions in relationships. The concepts and implementation of *Relations* and *Merges* is very flexible and

can be used to specify any kind of trade or social dynamics. HTM5 by itself does not inhibit or endorses any particular trade methodology and provides generic tools and structure that can be used to specify any logic.

A vital functionality that can be implemented at HTM5's *Relation* agents is the service discovery and matchmaking mechanism. Peer to peer trade in cloud robotic systems will require distributed locations where service providers could advertise their services along with their associated costs and quality parameters. As relationships between trading parties are managed by *Relation* agents, it would be preferable to implement service registries and demand-supply matchmaking mechanisms on the *Relation* agents. *Relation* and *Merge* agents have more visibility in the cloud ecosystem since they are connected to a number of other agents and act as open ports. For this reason *Merges* and *Relations* are also ideal for hosting agent indexes and other trade related data items. Lookup tables for services and demands; distance, speed and quality related cost metrics and other trade variable are suitably hosted at associated *Relation* and *Merge* agents.

4.3.2 Case Study Experiments

Following are the key motivations, methods and precautions associated to these case study experiments:

1. The primary objective of this case study was to test the feasibility of HTM5 meta-model as a design methodology for implementing complex trade methodologies on agent oriented cloud robotic systems.
2. The method chosen to achieve the primary objective was to implement a cloud robotic system that implements Peer-to-Peer trade methodology with multiple trade items, service and demand advertising, service discovery, matchmaking and banking mechanisms.
3. Once the system was designed and implemented using HTM5 methodology, the secondary objective was to compare Peer-to-Peer trade methodology with some other popular methodology.
4. In this work, a Peer-to-Peer methodology for trade amongst cloud robotic entities is described as an open [7] system with multiple service providers publishing services which are matched to demands by several other cloud entities. We assumed that examining such a system against a system with fixed trade relationships (fixed teams) would be interesting.
5. Comparison of peer-to-peer trade methodology to any other methodology is a relatively subjective study as every methodology is suited for a particular scenario, ground rules and implementational realities. The observations presented in this section are based on the data collected in 32 test cases and for a particular trade environment. The authors do not claim that these results will stand valid for all trade environments in real world. The baseline idea is to implement complex trade logics

on cloud robotic systems using HTM5 methodology, empowering its claim as a usable methodology for cloud robotic systems.

For the case study experiments, we envisioned a cloud robotic system named "Mine Cloud" with several robotic and non-robotic entities (see Fig. 4.14). Figures 4.15 and 4.16 are HTM5's computation independent meta-models for system level design specification. Computer simulations for the first part of the case study were implemented using VisuaBOT [64] and VBA [65] toolboxes. Fig. 4.17 shows some elements of the simulating environment created for the case study. For the second part, a scaled down version of the simulated environments were implemented on a colony of five TurtleBOTs [66] (see Fig. 4.18). Some run time videos of the simulation experiments and experiments on the physical TurtleBOT robots are available at [67].

Following are the list of elements that constitute the "Mine Cloud" cloud robotic system: (Refer to Fig. 4.14, 4.15, 4.16 and 4.17) **Physical Entities:**

1. Miner Robots **x Nb**
2. Digging Robots **x Nd**
3. Transporter Robots **x Nt**
4. Base Transceiver Station (BTS) Computer **x Na**
5. Bank Server
6. Market Server **x Nm**
7. Discovery and Matchmaking Server

HTM5 Components:

1. **1** *Antenna Agent* Hosted at BTS Station Computer
2. **Nb** *Miner BOT Agents* Hosted on Miner Robot
3. **Nb** *Locator Merges* Hosted on Miner Robot
4. **Nt** *Transporter BOT Agents* Hosted on Transporter Robot
5. **Nd** *Digging BOT Agents* Hosted on Digging Robot
6. **1** *Bank Agent* Hosted at Bank Server
7. **Nm** *Market Agents* Hosted at Market Server
8. **3** *Discovery Relations* Hosted at Discovery and Matchmaking Server
9. **1** *Pay Antenna Relation* Hosted at Discovery and Matchmaking Server
10. **7** *Mail-Merge Merges* Hosted at Discovery and Matchmaking Server

Cloud Networks:

1. *WWW* Between [Discovery and Matchmaking Server] and [Market Server];
[Discovery and Matchmaking Server] and [Bank Server]
2. *GSM* Between [Discovery and Matchmaking Server] And [Miner Robot];
[Discovery and Matchmaking Server] And [Transporter Robot]; [Discovery and Matchmaking Server] And [Digging Robot]; [Miner Robot] And [BTS Station Computer];

Trade Items: Item:[Served By]–[Demand At]

1. Assisted Global Positioning System (AGPS) Location Info :[Antenna]–[Miner BOT]
2. Assisted Global Positioning System (AGPS) Usage Fee :[Miner BOT]–[Antenna]
3. Digging Fee :[Miner BOT]–[Digger BOT]
4. Transport Fee :[Miner BOT]–[Transporter BOT]
5. Mine Plot Location :[Miner BOT]–[Transporter BOT]
6. Market Location :[Miner BOT]–[Transporter BOT]
7. Transport Service :[Transporter BOT]–[Miner BOT]
8. Transported Ore :[Transporter BOT]–[Market]
9. Digging Service :[Digger BOT]–[Miner BOT]
10. Money Transfer :[Bank]–[Miner BOT, Market]
11. Mineral Of Interest :[Market]–[Discovery]
12. Mineral Payment :[Market]–[Miner BOT]

Trade and Matchmaking Data:

1. Antenna Index
2. Miner Index
3. Market Index
4. Transport-BOT (T-BOT) Index
5. Digger-BOT (D-Bot) Index
6. Market X Price (Service Lookup table)
7. Market X Mineral (Demand Cost Metrice)
8. Digger X Price (Service Lookup table)
9. Digger X Speed (Service Lookup table)
10. Transport X Price (Service Lookup table)
11. Transport X Speed (Service Lookup table)



Figure 4.17: Above are a set of screen shots from the simulated experiments conducted on the "Mine Cloud" cloud robotic system. The Cloud in these experiments is simulated by the inter-agent message passing mechanism of the simulator. Part (a) above shows the virgin mine with 10 different kinds of mineral ores randomly embedded at 320 mine plots. The right-most column of the scene simulates the physical locations of the markets and corresponding columns in white displays their preferred mineral and the cost that they offer for that mineral at a given time. At the start of the simulation, all *Digger* and *Transporter Robots* are at their parking positions. Part (b) of the figure shows the mining action. *Miner Robots* search for their respective target mineral ores while *Digger* and *Transporter Robots* are hired by the *Miner Robots* at different stages of the mining process. The *Transporter Robots* loads the mineral ore from a mined plot and delivers the load to the market selected by the *Miner Robot* (Through the matchmaking mechanism). Part (c) shows a mine field near the end of the mining process. Most of the *Miner*, *Digger* and *Transporter Robots* are now free as very few mine plots remains to be mined. Part (d) shows the mining process with matchmaking associations made visible. At the peak of mining process, the number of matchmaking associations between agents could become very frequent. Part (e) is a table of various parameters that could be controlled for the simulated agent colony. It is necessary to have this variability in the simulation environment so that test cases with large variability could be formed.

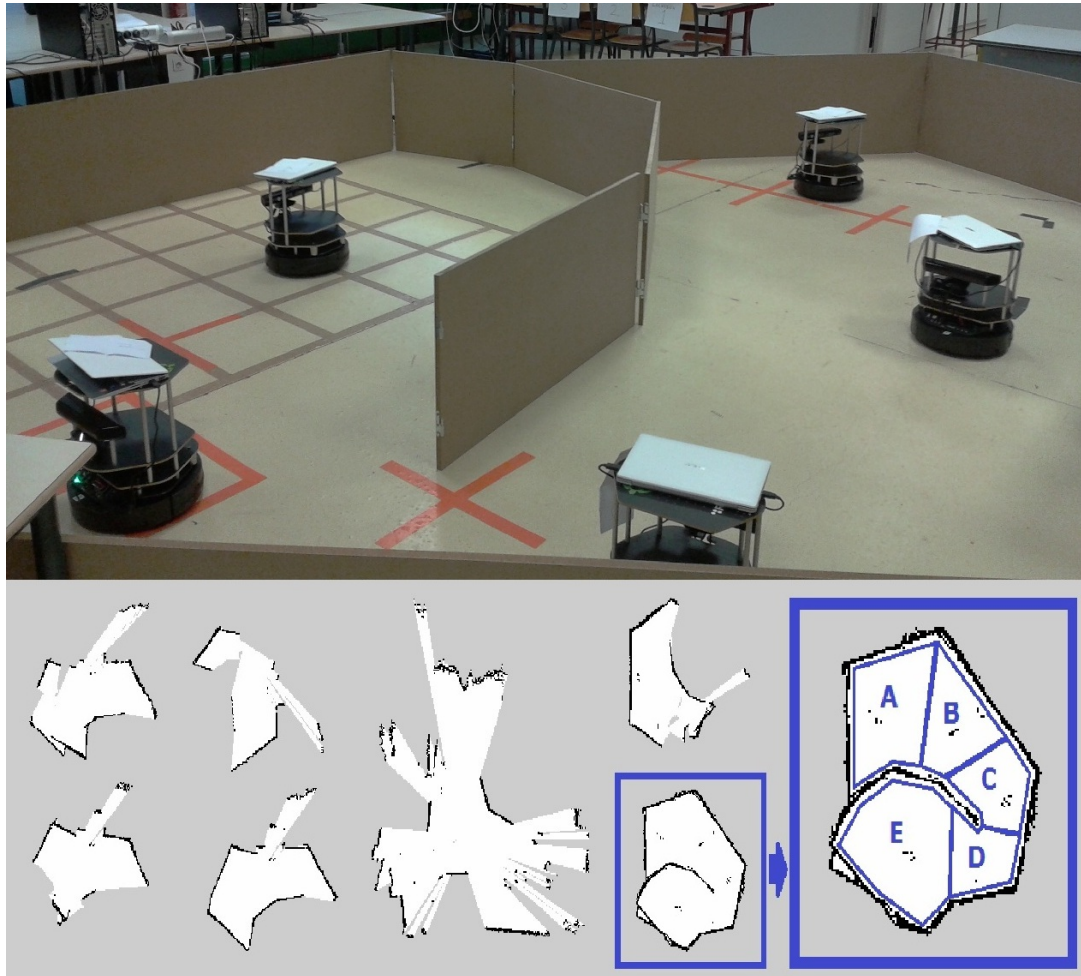


Figure 4.18: The above figure shows the physical robot colony of 5 TurtleBOT robots that was used to implement a scaled down version of the "Mine Cloud" case study. Out of the five robots, three were implemented as *Miner Robots* while the other two were implemented as *Digger Robots*. No *Transporter robots* or *Physical Market locations* were implemented. There were in all five mine plots (named A, B C, D and E) which were both mining locations and parking locations for *Digger Robots*. This limited experiment was conducted to have a real life representation of the ideas tested in the simulated world. The economy (profits and productive) in physical experiments were of limited use due to very few measured parameters and limited variability. For analysis of peer-to-peer and fixed-teams trade methodologies, only data from simulated experiments was utilized. The implementation of peer-to-peer trade ideas on real life robots using HTM5 methodology was encouraging since all three layers of HTM5 design methodology was followed to write software for each of these robots in different roles.

Testcase				Dedicated Teams								Peer-2-Peer Trade								Difference						
Exp#	Rand	#Bot	#DBOT	#TBOT	#Mines	Time	Av	D	T	A	B	#Mines	Time	Av	D	T	A	B	Av	A	Av	D	T	A	B	Av
1	1234	10	2	2	45	5000	1638	3516	3298	1088	2029	37	5000	1659	2986	2748	1147	2199	-8	21	59	-530	550	170		
2	4321	10	2	2	28	5000	1459	2681	2056	1064	1884	27	5000	1390	2207	2135	1099	1660	-1	-69	35	-474	79	-224		
3	1234	60	2	2	47	5000	1275	5315	3710	1140	1104	45	5000	1288	4925	3508	1363	1068	-2	13	223	-390	-202	-36		
4	4321	60	2	2	16	5000	1112	3336	1610	1140	1011	18	5000	1123	3484	1965	1380	931	2	11	240	148	355	-80		
5	1234	10	10	2	80	5000	1852	5235	1790	1136	2668	63	5000	1860	4374	1676	1187	2886	-17	8	51	-861	-114	218		
6	4321	10	10	2	68	5000	1875	4512	1644	1113	3101	76	5000	1915	4294	1675	1164	3182	8	40	51	-218	31	81		
7	1234	60	10	2	76	5000	1400	4986	1842	1167	1285	70	5000	1402	4470	1761	1417	1235	-6	2	250	-516	-81	-50		
8	4321	60	10	2	51	5000	1276	3580	1437	1202	1197	49	5000	1287	3458	1519	1437	1127	-2	11	235	-122	82	-70		
9	1234	10	2	10	47	5000	1503	1510	3329	1088	1962	38	5000	1496	1425	2786	1153	1993	-9	-7	65	-85	-543	31		
10	4321	10	2	10	22	5000	1230	1273	1796	1054	1425	31	5000	1401	1312	2388	1107	1880	9	171	53	39	592	455		
11	1234	60	2	10	50	5000	1276	2145	3958	1148	1085	44	5000	1257	1799	3648	1397	1040	-6	-19	249	-346	-310	-45		
12	4321	60	2	10	17	5000	1100	1681	1584	1144	973	17	5000	1105	1631	1704	1356	913	0	5	212	-50	120	-60		
13	1234	10	10	10	78	5000	1803	1858	1740	1141	3135	87	5000	1930	1986	1981	1229	3222	9	127	88	128	241	87		
14	4321	10	10	10	71	5000	1799	1760	1662	1132	3308	109	5000	2186	2068	1986	1210	4456	38	387	78	308	324	1148		
15	1234	60	10	10	206	5000	2065	3576	3279	1322	1859	206	5000	2034	3279	3309	1558	1773	0	-31	236	-297	30	-86		
16	4321	60	10	10	188	5000	1936	2908	2844	1394	1803	192	5000	2039	2954	2898	1619	1883	4	103	225	46	54	80		
17	1234	20	4	4	90	5000	1943	3946	3292	1162	2054	92	5000	2051	3563	3011	1244	2364	2	108	82	-383	-281	310		
18	4321	20	4	4	69	5000	1736	3365	2686	1131	1824	72	5000	1784	3220	2932	1240	1812	3	48	109	-145	246	-12		
19	1234	40	4	4	85	5000	1626	4213	3012	1155	1464	86	5000	1660	3646	2972	1355	1482	1	34	200	-567	-40	18		
20	4321	40	4	4	103	5000	1721	4564	3362	1247	1509	103	5000	1713	4013	3008	1552	1434	0	-8	305	-551	-354	-75		
21	1234	20	8	4	151	5000	2434	5118	2812	1239	2941	158	5000	2667	5019	2725	1345	3494	7	233	106	-99	-87	553		
22	4321	20	8	4	105	5000	1999	3989	2301	1199	2280	121	5000	2260	4445	2532	1311	2663	16	261	112	456	231	383		
23	1234	40	8	4	134	5000	1949	5224	2594	1224	1856	149	5000	2086	4903	2830	1421	1988	15	137	197	-321	236	132		
24	4321	40	8	4	165	5000	2097	5482	2983	1338	1962	170	5000	2181	5382	2747	1651	2012	5	84	313	-100	-236	50		
25	1234	20	4	8	91	5000	1889	2392	3315	1156	2136	93	5000	1988	2343	3064	1248	2372	2	99	92	-49	-251	236		
26	4321	20	4	8	78	5000	1736	2243	2902	1148	1887	77	5000	1770	2182	2972	1265	1870	-1	34	117	-61	70	-17		
27	1234	40	4	8	88	5000	1644	2627	3157	1160	1538	90	5000	1632	2433	2937	1319	1498	2	-12	159	-194	-220	-40		
28	4321	40	4	8	103	5000	1727	2874	3406	1259	1564	103	5000	1740	2491	2963	1580	1548	0	13	321	-383	-443	-16		
29	1234	20	8	8	149	5000	2394	3075	2788	1235	3124	165	5000	2584	3149	2793	1331	3527	16	190	96	74	5	403		
30	4321	20	8	8	132	5000	2209	2921	2590	1226	2754	136	5000	2328	2967	2700	1363	2887	4	119	137	46	110	133		
31	1234	40	8	8	162	5000	2106	3480	3021	1250	2076	174	5000	2199	3433	3168	1440	2138	12	93	190	-47	147	62		
32	4321	40	8	8	173	5000	2203	3441	3069	1362	2203	177	5000	2199	3311	2798	1660	2126	4	-4	298	-130	-271	-77		

Exp# => Experiment Number
Rand => Randomiser seed for Mineral price variability
#Bot => Number of Miner Robots
#DBOT => Number of Digger Robots
#TBOT => Number of Transporter Robots
Time => Length of Experiment

#Mines => Total number of plots mined by the colony (Out of 320)
<All Robots and Antennas started with an initial bank balance of 1000>
Av => Average bank balance for all robots and antennas at end of experiment
D Av => Average bank balance for all Digger robots at end of experiment
T Av => Average bank balance for all Transporter robots at end of experiment
A Av => Average bank balance for all Antennas at end of experiment
B Av => Average bank balance for all Miner robots at end of experiment

Figure 4.19: Test cases and corresponding productivity and profit data for "peer-to-peer" and "fixed-teams" trade models. Test cases 1 till 12 (marked red) are scenarios with bottleneck where one part of the mining process lacks resources (e.g. Availability of *Digger* or *Transporter Robots*).

In "Mine Cloud" cloud robotic system, there are more than one service providers for any particular service and a service could be demanded at several agents. All agents advertise their services and demands to the *Relation agent* that is hosting the demand or service registry (implemented as lookup tables) for their respective services and demands. A service is discovered by agents by reading the lookup tables associated with that service. Matchmaking is done based on preferences that an agent has for a particular service provider and for a particular quality and cost parameters. Preference vectors (A sequence of variables profiling an agent's service and demand preferences) of agents are matched with the quality, source and cost parameters of all available services (from different sources). This matchmaking chooses the closest matching service to an agent's preference. The matchmaking mechanism in "Mine Cloud" was hosted at the "Relation Agents" (Discovery). HTM5 however allows a designer to implement the matchmaking mechanism on any of the agents. In the absence of a matchmaking mechanism, agents could just utilize the service and demand discovery services and then negotiate bilaterally for a service contract.

Fig. 4.17 Part (e) shows the variability that exists in the simulation environment. Fig. 4.19 shows the results and comparisons for the 32 test cases simulated on the "Mine Cloud" system. Implementation of complicated trade logics on cloud robotic colonies (Simulations and real world robots) justifies the claim of HTM5 meta-model as a usable model for development of agent oriented cloud robotic systems. As mentioned earlier in this section of the thesis, comparison of peer-to-peer trade methodology to any other methodology is a relatively subjective study as every methodology is suited for a particular scenario, ground rules and implementational realities. The observations presented in this section are based on the data collected in 32 test cases and for a particular trade environment. The authors do not claim that these results will stand valid for all trade environments in real world. The baseline idea is to implement complex trade logics on cloud robotic systems using HTM5 methodology, empowering its claim as a usable methodology for cloud robotic systems.

Key Observations :

1. Average profits made by *Antenna Agents* are always higher in the case of Peer-to-Peer trade (see Fig. 4.19). Unlike in fixed-teams scenario, the dynamic allocation of robots to a mine plot requires multiple location requests by the *Miner Robots* to the *Antenna Agents*. Service discovery and matchmaking mechanisms are hosted on a server which is connected to robots through a GSM cloud (simulated). The services of *Antenna Robots* thus are more in demand in peer-to-peer trade scenario and thus the profits made by them are always higher in the case of peer-to-peer trade.
2. Average profits made by *Agents* (Miner Bot, Digging Bot, Transporter Bot and Antenna Agents) are mostly higher in the case of Peer-to-Peer trade methodology (see Fig. 4.19). In 25 out of 32 cases (78.125 per cent) the peer-to-peer methodology resulted in a greater average profit as against fixed-teams trade methodology.

3. The productivity of the colony in mining the available plots is generally higher when peer-to-peer trade methodology is used (see Fig. 4.19). Only in 9 out of 32 cases (28.125 per cent), the productivity (Number of Mine plots processed) is lower in peer-to-peer scenario.
4. Out of the 9 cases when the productivity is lower in peer-to-peer scenario, 8 cases are from the bottleneck set (cases 1 till 12).
5. In non-bottleneck scenarios, only 1 out of 20 cases (5.0 per cent) scenarios result in a lower productivity in peer-to-peer scenario.

4.3.3 Conclusion

In this work we presented HTM5 as a feasible meta-model for designing and implementing complex trade methodologies on an agent oriented cloud robotic system. The anatomical elements of HTM5 with respect to the peer-to-peer relational trade were presented. A case study with a comparative economic analysis of peer-to-peer approach against a fixed-teams approach was presented. The primary objective of the case study was to test the feasibility of HTM5 meta-model as a design methodology for implementing complex trade methodologies on agent oriented cloud robotic systems. Design and Implementation of complex trade methodologies using HTM5 validates its claim as a feasible meta-model for agent oriented cloud robotic system development. The complete HTM5 meta-model, a domain specific language supporting HTM5 and a case study specific for peer to peer trade variability in HTM5 is currently submitted to well-known journals. Next steps in development of HTM5 are to involve industry professionals to use and improve the meta-model. A detailed user guide and a graphical design interface for HTM5 based designing is also currently under development.

4.4 Industrial and Research Feasibility

Architecture has a strong influence on lifecycle of a system. Architecture is "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." [1]. "An architectural description (AD) is a collection of products to document an architecture" [1]. The influence of software architecture in design, construction, deployment and evolutions of a system is even more in software intensive systems. Fig. 4.20 shows the conceptual model of architecture and IEEE recommended practices for development of software intensive systems. The architecture of a system is described by a set of architecture documents. Models and views organize and aggregate the architecture documents. Stakeholders in a system include clients, managers, architects, developers, evaluators and users of the system. Stakeholders provide inputs in the creation of architecture documents, and later use these documents for their role in the system development life cycle such as analysis, development, planning, maintenance, operational and infrastructure support, budget, human resource and analysis of alternative architectures.

Object management group (OMG) is a consortium of standards in computer industry. OMG released a guide to model driven architecture (MDA) [3] (Fig. 4.21) in the year 2003. OMG-MDA has gained popularity in the software industry in the past decade. Many domain specific meta-models based on MDA are now available for development of software for specific domains. There is a strong emphasis on the development of system models [2] in MDA. Software product line engineering (SPLE) [12, 13, 14] encourages creation of reusable entities. These entities could be individual code units, components, classes, libraries, models or components within a model. It is thus necessary that a meta-model should support a modular design, with clear boundaries between components, and between different functionalities within a component. MDA is a three layer design where every layer has a different degree of abstraction and is meant to be used at different stages of the product development and by different people (see Fig. 1.2). Platform specific model (PSM), platform independent model (PIM) and computation independent model (CIM) are designed for software developers, software designers and domain experts respectively.

Current cloud robotic systems are structured around the client server methodology. One or more robotic clients access cloud resources through one or many web servers. This is a useful approach since services such as algorithms, maps, text and image based search engines are easily available as a free service on the internet. A client server system provides a simple migration of cloud computing applications to robotic platforms. Although client server based cloud robotic systems are useful, they are centralized systems with a limited scope of services. The authors of this thesis are supporters of a peer-to-peer cloud robotic methodology that will enable all members of the cloud robotic ecosystem to act like a server and a client as and when required. A peer-to-peer cloud robotic system will enable individual robots to offer and avail services from other robots and auxiliary systems.

The traditional client server based system can be part of the proposed peer-to-peer systems and the end result will be a flexible, dynamic exchange of services over the cloud.

Cloud computing is a business methodology and is a remoulded use of existing computing and internet technologies. There are two levels at which cloud robotics could evolve from cloud computing. Cloud services like cloud storage, software and platform as a service (IaaS, PaaS, SaaS) could be readily adopted to work for robots by treating robots as any other clients to these cloud services. In this form, cloud robotics is cloud computing with a robot's computer as client and the underlying tools and mechanisms for the two remain the same. In other adaptations, cloud robotics could mean that robots should be made capable to provide their physical and functional capabilities as a service to other robots and computers across the cloud. In this second form, the ideas of cloud computing are adopted as such to cloud robotics, but special tools and mechanisms will be required to implement these ideas on traditional robots. Tele-operated robotics is an example for the second form, where a robot acts as a server offering its functionalities to a remote user. In both form, there is a robotic/non-robotic server that provides its functionalities as a service to other non-robotic/robotic clients. This matches with the traditional client-server mechanism in cloud computing and could be identified as client-server like cloud robotics.

Development of cloud robotics as a business model will require new tools and methodologies. It is essential to develop methodologies that are industry and business oriented. The cloud robotic methodologies should go one step further to include models that incorporate concepts like Distributed Artificial Intelligence (DAI) [9], registry based service discovery and automated match-making mechanism. Many of the services offered by a robot to other robots will have a physical world component. A robot's physical reach will determine the scope of the physical services offered by it and any business model for a robotic ecosystem should include provisions to model factors that codify physical world interactions. A cloud robotic ecosystem will also include many non-robotic entities. These entities could range from ambient intelligence to server banks. In theory any device that can communicate through a network could be included as a working component of a cloud robotic system. The communication networks that collectively build the cloud could be of different kinds and visible in selective physical regions. A methodology that allows modelling of these non-robotic devices, networks and interfaces will give a complete design toolset to designers of cloud robotic systems. Fig. 1.1 shows a typical cloud robotic ecosystem with robotic and non-robotic entities. A design methodology for cloud robotic ecosystem should provide tools to model all physical and theoretical aspects of these systems. Key theoretical elements of a cloud robotic ecosystem would be its network structure, event driven behaviour, social interactions, norm driven peer-to-peer trade, micro level competitions and dynamically regulating collaboration [11].

When writing software for the industry, managerial concerns such as reusability have to be taken into the design process. Products based on multi agent systems are generally very dynamic with newer versions being produced very often and parts of software being used in other products. In

most cases, people working on robots and other hardware components of the multi agent system are not trained in software engineering. Hence the Ad-hoc software development at the time of research is not very useful for the product development. To solve the above mentioned problems, the complete system is split into components which are loosely connected and functionally independent. Component-based software engineering (CBSE) [34] is a reuse-based approach to software development. Once implemented and tested, the components can be reused with ease in other products saving time and the cost of software development. It is common practice in the software industry to implement CBSE using objects.

Hyperactive transaction model (Fig. 4.22, 4.23 and 4.24) is an attempt to provide a model for designing multi agent systems with a different "thought process". A similar example can be seen in object oriented software modeling. Object oriented programming is not only a different way to write code, but a new "thought process" to design software where we think in terms of objects in real life before making their equivalent class in the code. Likewise in Hyperactive Transaction Model, the attempt is to think of agents in terms of humans working together in a team.

Discovery and matchmaking mechanism

We assume most cloud robotics systems will be dynamic with agents having more than one options to get the required services from and where members become online and offline randomly. In such systems there will be a need to have a registry system (like in many service oriented open systems) to publish services and demands of agents. In such system more than one service provider publishes the availability and cost of its services through a lookup table hosted on a relation agent. One possibility for service initiation could be the matchmaking mechanism implemented on the relation. In such scenario it is necessary for the agents to grant authority to the relation agent to decide on their behalf which service provider is allotted to them and on what basis (The relation agent will have to be hyperactive to surpass autonomy of other agents). The other possibility could be to allow agents to negotiate the costs of the services. The relation agent in such a case may store and maintain the registry, the trust and quality of service variables of individual relatives. Agents communicate the updated values (or read requests) for the lookup table entries to relation agents, as and when required. The relation agent adds/deletes records from the lookup tables when agents become online/offline. Any updates made in relationship variables are reflected in the matchmaking mechanism (on the relation agent) or the negotiation mechanisms (between pair of agents).

Relationships and contracts:

Relationships are relative positioning of agents into a predefined social construct. In scenarios where we have relation agents dynamically adding or deleting agents to a relationship (via a merge agent, discussed in part D of the current section), the joining agents instantly know their role in a relationship by the kind of relation agent they are attaching to. As each agent has its personal goals and business logic, a pre-defined trade relationships help reduce otherwise chaotic system of agents. Once agents decide to exchange the services (by matchmaking mechanism or by agent to agent negotiations), the

relation agent or the agents themselves may bind themselves to a contract. This is similar to current pay per use cloud computing services available online, the service provider and the user agree upon a cost and quality of service before initiating the service. The quality of service between a pair of agents is monitored by the relation agent. The denial of service by a service provider, or a problem with the quality of service after contract initiation is reported by the client agent and is recorded by the relation agent. Counter Actions could be initiated by the relation agent like termination of the current contract, penalty and/or lowering of the trust parameter for the service provider. Trust parameter of an agent may affect the matchmaking/negotiations for its future deals. Trust is a complex human factor to model, and its implementation may vary from project to project. In peer to peer (or client server based) cloud robotics, the parties involved in the use of paid services will have to bind themselves in a legal contract. Administrative and banking agents as a part of the cloud computing system could be one of the possible solutions. These measures will be essential in popularizing cloud robotics as a business possibility as relationships and contracts bring reliability and a level of trust in the cloud robotic system.

Dynamic Electronic Institutions:

The most advantageous step towards bringing peer-to-peer cloud robotics closer to a viable business model, would be to establish mechanism for automated dynamic electronic institution formation. An institution is a representation of a norm based society. Social, business and administrative institutions shape the way humans interact. Electronic institutions are a representation of a norm based multi agent system leading to a sociologically-inspired computing. The norms introduced by electronic institutions or dynamic electronic institutions may be seen as a limiting factor in a system's functionality, but by constraining agent's behaviour they decrease the system complexity and make agent behaviour more predictable. For dynamic electronic institutions, one proposed lifecycle is a three phase life cycle (3F life cycle). Formation, Foundation and Fulfillment are the three phases in a digital institution's lifecycle. Agents from an agent community are selected to form a coalition (Formation Phase) followed by establishment of norms of an institution. Once the norms are finalized and a particular kind of institution is chosen, the agents form and participate in the institution (Foundation phase). Re-formation and re-foundation may occur as and when required to keep an institution effective. On completion of an institution's mandate, the institution is dissolved (Fulfillment phase). The 3F approach has one application in agent based business ecosystem development. Cloud robotics is one such ecosystem, and an agent based peer-to-peer approach towards cloud robotics requires an institutional framework to find acceptance as a business model. The digital electronic institution concept can be implemented in HTM5 trade model by using merges for managing formation, foundation and fulfillment phases, and relational agents for re-formation, re-foundation and managerial logic constructs.

The above ideas find relevance as a development methodology for cloud robotic systems. Autonomy, heterogeneity, dynamism and interactions that preserve autonomy are essential for a business oriented, open development

of cloud robotic systems. Robots and auxiliary systems in a cloud robotic ecosystem may be developed by different manufacturers and deployed by various businesses or individuals. The systems could be open or closed, the interactions may be standard or custom, and there could be high variability in terms of software engineering practices followed by organizations.

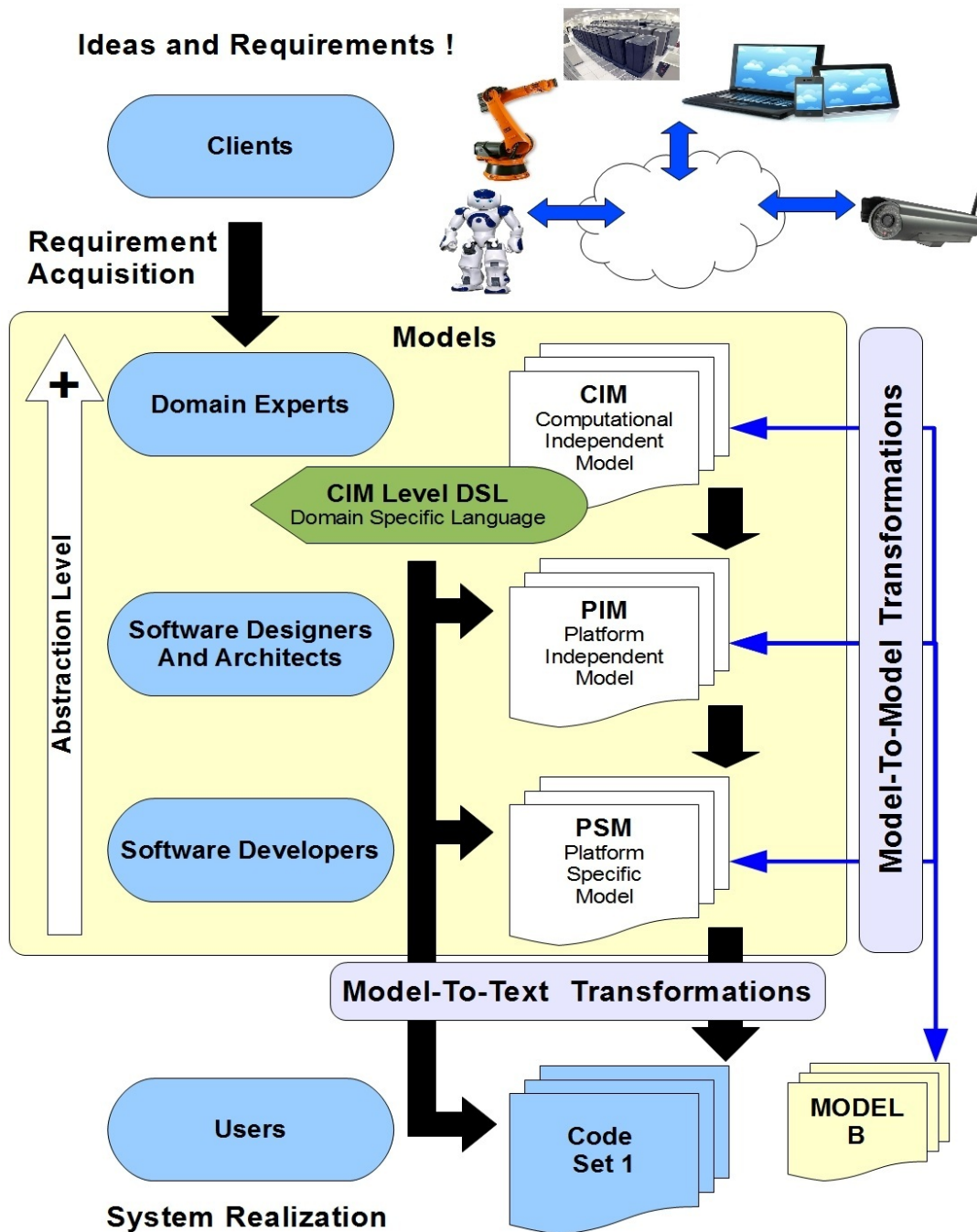


Figure 4.21: Model Driven Architecture proposed by Object Management Group (OMG-MDA). The three layers of MDA cater to different actors in the software development life cycle. Model to model transformations are between the three layers of MDA, or to an external model as the target. Model to Text transformation have a general purpose programming language as target. Models are executed independently or via a domain specific language for model transformations. (Repeated image)

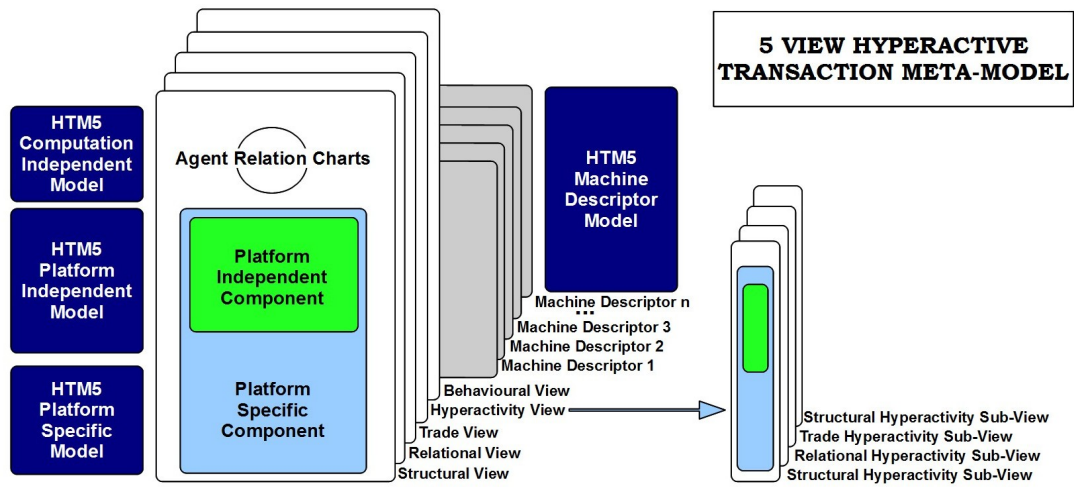


Figure 4.22: An anatomical overview of the 5 views Hyperactive Transaction Meta Model (HTM5). (Repeated image)

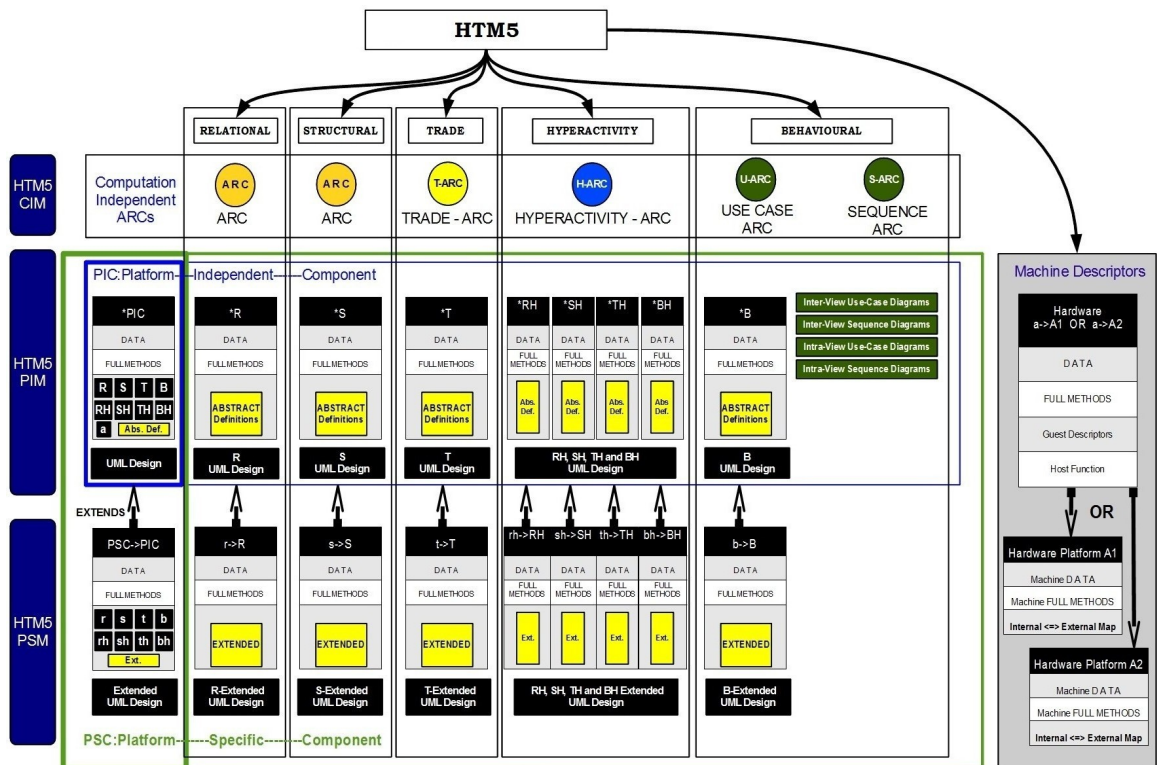


Figure 4.23: An anatomical description of the 5 views Hyperactive Transaction Meta Model (HTM5). (Repeated image)

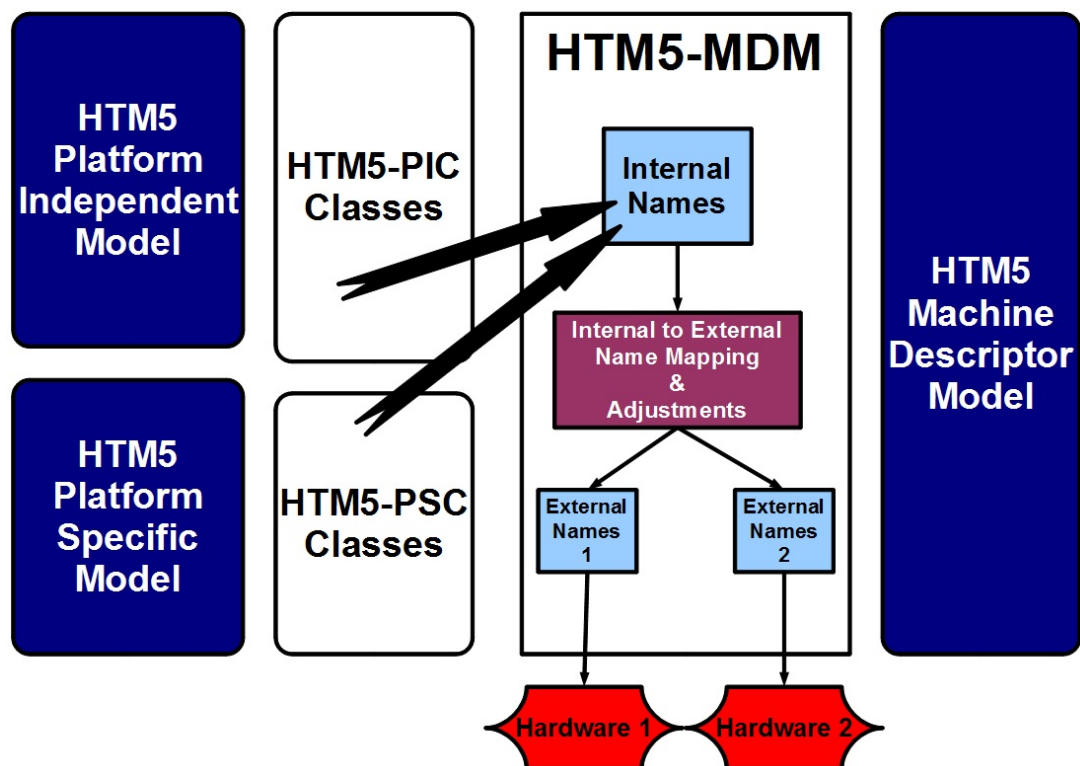


Figure 4.24: Separation of PIM and PSM parts of HTM5-MDM. (Repeated image)

Attributes	TeleNAO	DEI (Simulated)	DEI (Physical)	P2P (Simulated)	P2P (Physical)	Workshop
Type of Study	Physical	Simulated	Physical	Simulated	Physical	CIM Layer
# Entities in cloud robotic ecosystem	10+	38 to 104	7 to 9	57 to 123	13	Innumerable
# Robotic entities	1+	33 to 99	3 to 5	14 to 80	5	Innumerable
Robots	NAO Humanoid robot	Simulated	TurtleBOT robot	Digger Robot; Miner Robot; Transporter Robot	TurtleBOT robot	Innumerable
# Entities that represent cloud infrastructure and devices	6+	5	4	3	3	Innumerable
Cloud infrastructure and devices	Personal computers; Web server; Cloud storage and application hosting; mobile phones; mobile internet enabled devices	Bank; Market; Matchmaking agents; Dynamic electronic institution server (implemented as a Relation)	Personal computers	Bank; Market; Service discovery and matchmaking server;	Personal computers	Innumerable
# Entities that represent auxiliary devices	3+	0	0	40	5	Innumerable
Auxiliary devices	IP Cameras	N/A	N/A	BTS Antenna	Location Marker	Innumerable
# Clouds (networks)	5	3	3	3	3	Innumerable
# Public clouds (networks)	2	1	1	1	1	Innumerable
# Private clouds (networks)	3	2	2	2	2	Innumerable
# Simulated clouds (networks)	0	3	0	3	0	Innumerable
Cloud/networks	LAN; Wireless LAN; Mobile phone data network; Internet; IP Link	Wireless LAN; Internet	Wireless LAN; Internet; Inter Process Communication (IPC)	Wireless LAN; Internet	Wireless LAN; Internet; Inter Process Communication (IPC)	Innumerable
Peer-to-Peer trade (DBE)	Partial	Yes	Yes	Yes	Yes	Yes
Relational trade	No	Yes	Yes	Yes	Yes	Yes
Scalable	Yes	Yes	Yes	Yes	Yes	Yes
Open system	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic service discovery	No	Yes	Yes	Yes	Yes	Yes
Matchmaking	No	Yes	Yes	Yes	Yes	Yes
Service demand cardinality	1 to many	many to many	many to many	many to many	many to many	Innumerable
Business logic implementation	Yes	Yes	Yes	Yes	Yes	Yes
Intermediate models used for component implementation	UML	None	UML	None	UML	N/A
Tools used for final component implementation	NaoQj; Python GPL; Open CV; XML; Google App Engine; AJAX; HTML;	VBA; VisualBOTS; MS Excel	linux shell scripting; ROS routines	VBA; VisualBOTS; MS Excel	linux shell scripting; ROS routines	N/A
Device Operating systems	OpenNao; MS Windows; Linux; Google customized Linux; Android; Symbian, IOS; MacOS	MS Windows; Linux Virtual Machine;	MS Windows; Linux Virtual Machine; Robot Operating System (ROS)	MS Windows; Linux Virtual Machine;	MS Windows; Linux Virtual Machine; Robot Operating System (ROS)	N/A
# Repetitions / # Instances	20+	120+	20+	120+	15+	6 Project Ideas
# Stakeholders (Human)	4	3	3	3	3	24
Domains represented	Electronics; Computer science; Software engineering; Robotics; Computer vision; Image processing; Web application development; Rapid prototyping; Management	Computer science; Software engineering; Robotics; Rapid prototyping; Management	Electronics; Computer science; Software engineering; Robotics; Rapid prototyping; Management	Computer science; Software engineering; Robotics; Rapid prototyping; Management	Electronics; Computer science; Software engineering; Robotics; Rapid prototyping; Management	Robotics; Robot fabrication; Power systems; Digital signal processing; physica; Computer networks; Computer programming; Software engineering; Embedded systems; Computer vision
Level of familiarity with domains	Masters; Doctoral candidate; professor	Doctoral candidate; professor	Doctoral candidate; professor	Doctoral candidate; professor	Doctoral candidate; professor	PHD; Doctoral candidates; Masters; Bachelor; Foundation
Verbal feedback	Yes	Yes	Yes	Yes	Yes	Yes
Presentations/Brainstorming	Yes	Yes	Yes	Yes	Yes	Yes
Feedback questioner	Yes	None	None	None	None	Yes

Figure 4.25: A summary of the case studies and a workshop conducted during the incremental development of HTM5 meta-model and HTM5-DSL

4.5 A summary of all Feasibility and Viability studies

In this section we present some of the case study experiments and a workshop conducted over the last 3 years to test the feasibility and viability of the HTM5 meta model and its corresponding domain specific language (HTM5-DSL). The objective of these studies was to improve the proposed model through feedback and usage and to present a defense to the major claimed

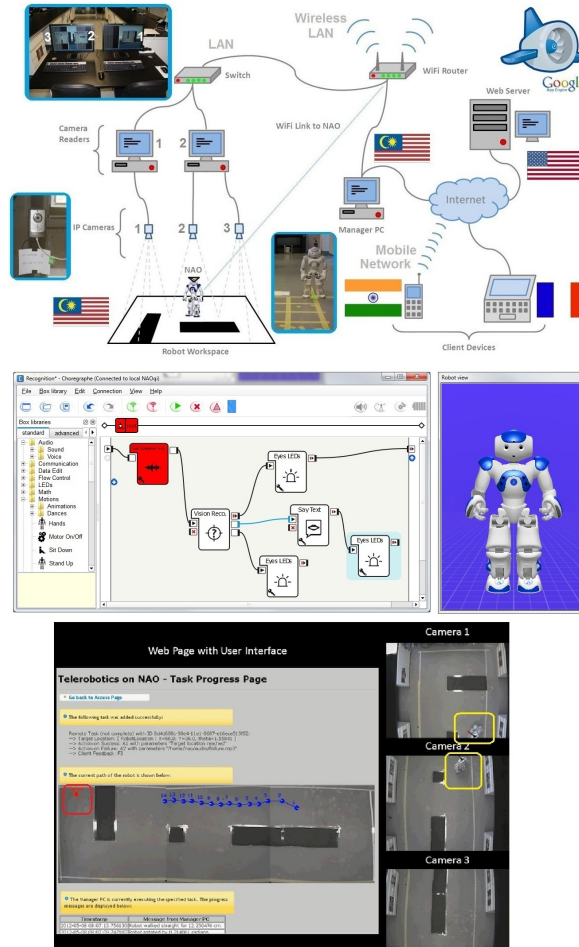


Figure 4.26: Top: Structural Overview of 'Telerobotics On Nao' System. Centre: Development of Nao Agent using choreograph, NaoQi and OpenNao platforms. Bottom: The graphical user interface at the *Client Agent*. Location of the Nao Robot is updated to the human user as it moves towards a target location specified by the user.

benefits of the HTM5-DSL and the HTM5 methodology. Fig. 4.25 is a summary of the case studies and a workshop conducted during the incremental development of HTM5 meta-model and HTM5-DSL. These projects, experiments and workshop are essential for the continuous improvement of the meta-model and the proposed domain specific language. The studies capture a wide scope of cloud robotic ecosystems ranging from simulations, real world projects and ideas at the computation independent model (CIM). Most of the participants and stakeholders in these studies were first time users of HTM5 and HTM5-DSL. The stakeholders were from a variety of domains making these studies closer to the real life development of cloud robotic ecosystems. The development tools, platforms, networks, component development methodologies touched a number of popular practices in the industry. These studies also implemented complex behavioural models for cloud robotic ecosystems and proved the usability of HTM5 as viable platform for research and business needs. Fig. 4.26, 4.27, 4.28 and 4.29 present brief accounts on these studies and workshops. This section will present a

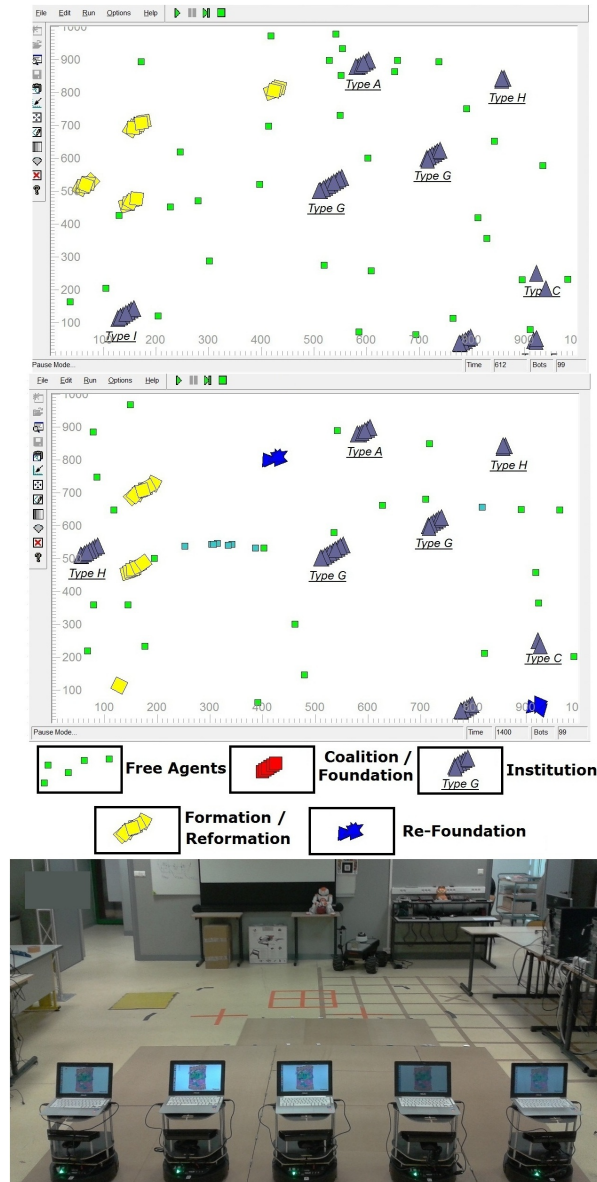


Figure 4.27: Above are some instants from Dynamic Electronic Institutions based Digital Business Ecosystem simulations. Top: Formation of eight Institutions of different cardinality and type. Some of the groups are in Formation or Re-Formation phase. Centre: Two institutions have moved to Re-Foundation phase while one of the institutes has finalized freeing its member BOTs. Bottom: TurtleBOT setup for a scaled down version of Dynamic Electronic Institutions based Digital Business Ecosystem.

short introductory text on these studies followed by an analysis on how these studies accumulate elements to defend the claimed benefits of HTM5 and HTM5-DSL.

Telerobotics on NAO: This is a simple telepresence application for a humanoid robot supported by a number of auxiliary devices. The humanoid robot in our system does not have localization capability. Three roof mounted cameras provide their feed as a service to camera reading

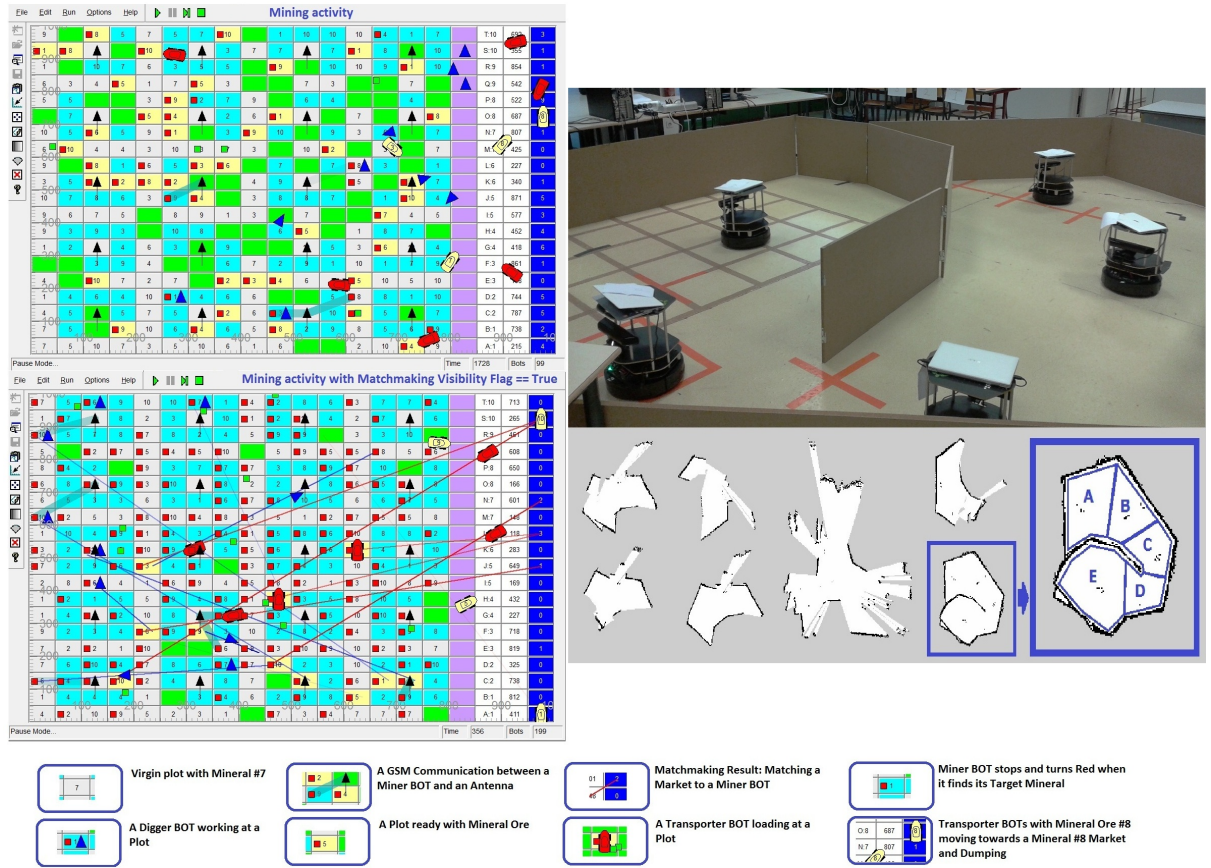


Figure 4.28: Above are a set of screen shots from the simulated experiments conducted on the "Mine Cloud" P2P cloud robotic system. The Cloud in these experiments is simulated by the inter-agent message passing mechanism of the simulator. Left: The mining action. *Miner Robots* search for their respective target mineral ores while *Digger* and *Transporter Robots* are hired by the *Miner Robots* at different stages of the mining process. The *Transporter Robots* loads the mineral ore from a mined plot and delivers the load to the market selected by the *Miner Robot* (Through the matchmaking mechanism). Right: The physical robot colony of 5 TurtleBOT robots that was used to implement a scaled down version of the P2P "Mine Cloud" case study. Out of the five robots, three were implemented as *Miner Robots* while the other two were implemented as *Digger Robots*. No *Transporter robots* or *Physical Market locations* were implemented. There were in all five mine plots (named A, B C, D and E) which were both mining locations and parking locations for *Digger Robots*.

computers. The manager computer combines the information coming from these sources and provides direction and location as a service to the humanoid robot. The manager computer is connected to a web server across the internet cloud which runs a secure access telepresence service for the humanoid robot. Computers and Mobile devices around the world can register to the service and access the robot.

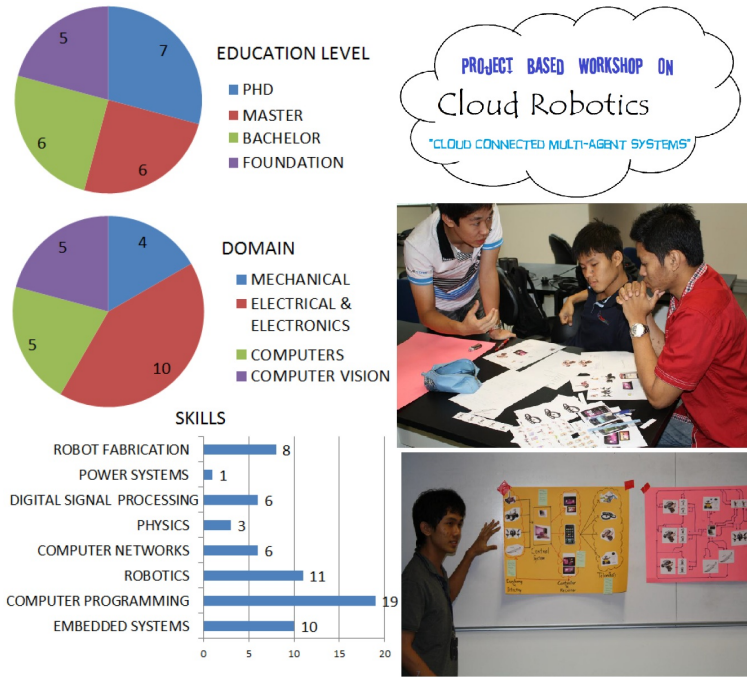


Figure 4.29: A project based workshop on cloud connected multi-agent systems.

#	Checks	Description
1	A.1.a	HTM5-DSL supports heterogeneity in robotic platforms.
2	A.1.b	HTM5-DSL supports heterogeneity in the kinds and number of communication networks (Clouds).
3	A.1.c	HTM5-DSL supports heterogeneity in kinds of cloud infrastructures.
4	A.1.d	HTM5-DSL supports heterogeneity in number and kind of auxiliary devices in the cloud ecosystem.
5	A.1.e	HTM5-DSL supports design heterogeneity at component level.
6	A.2	HTM5-DSL follows object management group's (OMG) model driven architecture guidelines (OMG-MDA) for model transformations.
7	A.3.a	HTM5-DSL is designed to work at computation independent layer (CIM) of HTM5 which is the most suited placement for a DSL used for requirement elicitation and concept development.
8	A.3.b	A CIM level DSL promotes contribution of clients and non-domain personals in system design process.
9	A.4	The code written in HTM5-DSL is abstract and has an intuitive grammar which serves as a design document even before execution.
10	A.5	HTM5-DSL supports automated model to text (M2T) and model to model (M2M) transformations. The targets for these execution results are Java (general purpose language, GPL) and UML (unified modelling language) which are widespread in software industry.
11	B.1	HTM5 is agent oriented.
12	B.2	HTM5 enables implementation of complex behavioural models on cloud robotic systems.
13	B.3	HTM5 supports peer-to-peer service-demand interactions.
14	B.4	HTM5 is a multi-view meta-model which allows clear separation of view specific concerns.
15	B.5.a	HTM5 views supports engineering needs at initial stages of the design process and in substituent increments by allowing quick fix solutions (flexible agency).
16	B.5.b	HTM5 views supports inclusion of business logic at initial stages of the design process and dynamically during a system run.
17	B.5.c	HTM5 views supports research/rapid-prototyping needs of researchers.
18	B.6	HTM5 supports incremental development of cloud robotic ecosystem.
19	B.7	HTM5 enables smooth design up gradation when a base software/machine is changed/updated. (Inclusion of a machine model)
20	B.8	HTM5 allows flexibility in the concept of agency as and when required.
21	B.9	HTM5 allows hyperactive controls from one to another agent in the system. This is useful since at implementation layer unnecessary design clutter and bottlenecks can be avoided by an elegant engineering hack.
22	C.1.a	The case study is scalable in number of participating entities.
23	C.1.b	The case study is scalable in number of transactions amongst participating entities.
24	C.1.c	The case study is scalable in number of rules governing the transactions amongst participating entities.
25	C.2.a	The proposed methodology is usable as a tool to capture cloud robotic design ideas.
26	C.2.b	The proposed methodology is usable as a tool to communicate captured cloud robotic design ideas to first-time readers.
27	C.2.c	The proposed methodology is usable as a tool for personals migrating from a purely UML based designing paradigm.
28	C.2.d	The proposed methodology can be easily adapted for technical and non-technical personals from various domains.
Category A: Claimed Benefits of HTM5-Domain Specific Language (HTM5-DSL)		
Category B: Design characteristics of HTM5 (extends to HTM5-DSL)		
Category C: Model feasibility checks		

Figure 4.30: A list of inspection cases (checks) for the case studies.

The robot can be directed to a particular location in its workspace by the remote users. Once reached, the robot performs a set of actions

		TeleNAO	DEI	DEI	P2P	P2P	Workshop	Net	
#	Check		Simulator	Physical	Simulator	Physical			%
1	A.1.a	✓ 3	! 1	✓ 2	! 1	✓ 2	✓ 3	✓ 12	66.7
2	A.1.b	✓ 3	! 1	! 1	! 1	! 1	✓ 3	✓ 10	55.6
3	A.1.c	✓ 3	! 1	! 1	! 1	! 1	✓ 3	✓ 10	55.6
4	A.1.d	✓ 3	! 1	! 1	! 1	! 1	✓ 3	✓ 10	55.6
5	A.1.e	✓ 3	! 1	! 1	! 1	! 1	✓ 3	✓ 10	55.6
6	A.2	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	! 1	✓ 16	88.9
7	A.3.a	✓ 2	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 17	94.4
8	A.3.b	✓ 2	✓ 2	✓ 2	✓ 2	✓ 2	✓ 3	✓ 13	72.2
9	A.4	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 18	100.0
10	A.5	✓ 2	✓ 2	✓ 3	✓ 2	✓ 3	✗ 0	✓ 12	66.7
11	B.1	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 2	✓ 17	94.4
12	B.2	! 1	✓ 3	✓ 3	✓ 3	✓ 3	✓ 2	✓ 15	83.3
13	B.3	✓ 2	✓ 3	✓ 3	✓ 3	✓ 3	✓ 2	✓ 16	88.9
14	B.4	! 1	✓ 3	✓ 3	✓ 3	✓ 3	! 1	✓ 14	77.8
15	B.5.a	✓ 3	! 1	✓ 2	! 1	✓ 2	! 1	✓ 10	55.6
16	B.5.b	! 1	✓ 3	✓ 3	✓ 2	✓ 2	! 1	✓ 12	66.7
17	B.5.c	✓ 2	✓ 3	✓ 3	✓ 3	✓ 3	! 1	✓ 15	83.3
18	B.6	! 1	✓ 3	! 1	✓ 3	! 1	✗ 0	! 9	50.0
19	B.7	✓ 2	! 1	! 1	! 1	! 1	✗ 0	! 6	33.3
20	B.8	✓ 3	! 1	! 1	! 1	! 1	! 1	! 8	44.4
21	B.9	✓ 3	! 1	! 1	! 1	! 1	! 1	! 8	44.4
22	C.1.a	✓ 2	✓ 3	! 1	✓ 3	! 1	! 1	✓ 11	61.1
23	C.1.b	! 1	✓ 3	! 1	✓ 3	! 1	! 1	✓ 10	55.6
24	C.1.c	! 1	✓ 3	✓ 3	✓ 3	✓ 3	! 1	✓ 14	77.8
25	C.2.a	✓ 2	✓ 3	✓ 3	✓ 3	✓ 3	✓ 3	✓ 17	94.4
26	C.2.b	✓ 2	✓ 2	✓ 2	✓ 2	✓ 2	✓ 3	✓ 13	72.2
27	C.2.c	! 1	! 1	! 1	! 1	! 1	✓ 3	! 8	44.4
28	C.2.d	✓ 2	✓ 2	! 1	! 1	! 1	✓ 3	✓ 10	55.6
Net		60	60	56	58	55	52	341	67.7
%		✓ 71.4	✓ 71.4	✓ 66.7	✓ 69.0	✓ 65.5	✓ 61.9	67.7	
0: Not Tested 1: Proof of Concept 2: Tested with satisfaction 3: Exclusively Tested									

Figure 4.31: Pairwise analysis of inspection cases (checks) presented in Fig. 4.30 with the case studies and the workshop presented in Fig. 4.25. The legends rate the studies based on the extent to which an inspection case was tested. The combined result of each of the studies for an inspection case gives a measure of the extent to which an inspection case was tested in these studies. The combined result of each of the checks for a study gives a measure of the extent to which a study contributed in testing the HTM5 methodology. The combined sum of all contributions tests the overall satisfaction measure 341 (67.7 per cent of 504=3x6x28). A perfect 504 would mean each of the implementation cases was rigorously proven while any score above 336 (2x6x28) would mean that at an average all inspection cases were tested with satisfaction. Further studies are planned to test the inspection cases with scores lower or equal to 50 per cent.

as directed by the remote user and sends images or information as a service to the remote clients. The whole system is location independent and is a simple but adequate example of a cloud robotic system. This is a scalable system with provisions to include any number of NAO robots, cameras and clients in the system. Automatic initialization

of new members however is implemented only for clients and not for robots or cameras.

The peer-to-peer trade in the 'Telerobotics on Nao' system is not based on predefined contracts or institutionalization. There is no actual transfer of money in the system and the services are not governed by measurement of Trust or Quality of Service. This is a scalable system with provisions to include any number of NAO robots, cameras and clients in the system. Automatic initialization of new members however is implemented only for clients and not for robots or cameras.

Dynamic Electronic Institutions: Dynamic

Electronic Institutions (DEI) are modelled on the ideas of Institutions in Human societies. Norms based on trade contracts, social relationships and institutions bring a sense of order in multi agent systems. The aim of this study is to test feasibility of HTM5 methodology in implementing Dynamic Electronic Institutions on a digital business ecosystem (DBE). Dynamic Electronic Institutions are Electronic Institutions where formation, reformation and dissolution of institutions are automated processes. The norms and objectives of institutionalization are dynamically adapting to the needs of its member agents. Following are the stages in a DBE:

1. Search for a business opportunity
2. Analysis of opportunity by business owner
3. Coalition establishment (Formation and re-formation phase)
4. Selection of an appropriate DBE
5. Acceptance by business owner
6. Establishment of a DBE (Foundation and re-foundation phase)
7. Finalization of the DBE and release of acquired resources and agents

The most advantageous step towards bringing peer-to-peer cloud robotics closer to a viable business model, would be to establish mechanism for automated dynamic electronic institution formation.

Peer-to-Peer Trade: Peer-to-Peer methodology for

trade amongst cloud robotic entities is described as an open system with multiple service providers publishing services which are matched to demands by several other cloud entities. Comparison of peer-to-peer trade methodology to any other methodology is a relatively subjective study as every methodology is suited for a particular scenario, ground rules and implementational realities. The authors do not claim that these results will stand valid for all trade environments in real world. The baseline idea is to implement complex trade logics on cloud robotic systems using HTM5 methodology, empowering its claim as a usable methodology for cloud robotic systems.

A vital functionality that is implemented for HTML5 relation agents is the service discovery and matchmaking mechanism. Peer to peer trade in cloud robotic systems will require distributed locations where service providers could advertise their services along with their associated costs and quality parameters. As relationships between trading parties are managed by Relation agents, it could be preferable to implement service registries and demand-supply matchmaking mechanisms on the Relation agents. Relation and Merge agents have more visibility in the cloud ecosystem since they are connected to a number of other agents and act as open ports.

Workshop: In order to test the feasibility of ARC based designing, we conducted a project based feasibility study at a workshop organized at University Technology Petronas, Malaysia in April 2012. A group of 24 students participated in the study. In total of 6 project ideas were developed using the ARC model, followed by detailed analysis of the usability of the model. A comparison was made to the traditional methods to represent ideas at the most abstract level and participant feedback was taken on various usability and functional aspects of the ARC model.

In order to simulate different levels of technical knowledge of the users of the ARC model, participants were chosen from various levels of education. The participants had a wide breadth of technical skills and were working in different domains. This was a crucial component of the participating group as it enabled to test the feasibility of the ARC model in representing ideas coming from people with different kind of expertise. It was important that the model was proved useful to all the participants in representing their ideas, but another important aspect of the study was to see how useful the model is to establish a common standard to represent ideas within the group, and its communication to the other groups.

A number of robots and devices were used by the participants to form the project ideas. The inclusion of a wide variety of agent-hosts in idea formation was mainly because of the freedom the participants had while working at an abstract level, without being influenced by the implementational details. The top layer of the proposed model (ARCs) is designed for this abstract level idea formation where all project stakeholders can contribute into the design.

The 6 teams were given time to brainstorm on their own group project idea. In the first section, they were told to use any kind of textual or diagrammatical tools to make abstract design documents for their designs. Later in second section they were trained to use ARCs and then told to use ARCs to make their abstract design documents. The teams gave presentations of their design ideas twice, once using their

own textual/ diagrammatical aids and once again using the ARCs. In the end the participants gave their feedback on 22 subjective data points. The feedback target was ranging from their understanding of the tasks given to them, to comparing the tools they used to make abstract design documents for their ideas. They were also asked about how easy it was for them to use the different tools and to understand the presentations of other teams when they used those tools. In all, 6 project ideas were developed and discusses. ARCs were developed for all 6 projects and the workshop provides a number of pointers to improve and validate the usability of the ARC based modeling.

Chapter 5

Conclusions

This thesis proposed a 5-view meta-model for agent oriented development of cloud robotic systems. Cloud robotic systems are systems with multiple robots and auxiliary devices such as computers, server banks, cameras, sensor networks and ambient intelligence devices. In an industrial setup, these devices and robots are developed by different vendors and have a multitude of hardware and software constructs. In order to unify the design process in these systems, it is necessary that the meta-model should be multi layered and should permit flexibility in the development process. It is also important that the systems should be capable to adopt intelligent and evolving technologies in the field of cloud computing and robotics. The cloud computing business model and elements of distributed artificial intelligence should find their way into the cloud robotic domain to make it a business possibility. In this work we proposed an agent oriented approach towards cloud robotic systems. We also discussed the advantages of an agent oriented approach to the development of cloud robotics. Peer-to-Peer exchange of services over cloud robotic ecosystem will empower the idea of intelligent environments. We placed a great importance in developing a methodology that is feasible for research, industry as well as business. The 5-View Hyperactive Transaction Meta-model (HTM5) for agent oriented development of cloud robotic systems is based on Model Driven Architecture (MDA) guidelines of the object management group (OMG). The methodology is agent oriented, component based, reusability oriented and compatible with current industrial practices for product development. HTM5 supports heterogeneity in the cloud robotic ecosystem. Agent orientation in HTM5 is concept based and flexible. System designers and developers are empowered by mechanisms to relax the Agency concepts as and when required. The model itself does not put any constraints in a flexible, multi-platform, multi style development of Agents for various machines. This thesis discusses the status of cloud robotics and how the proposed model is designed to support upcoming ideas in the domain. In this thesis we have explained the design drivers behind the proposed HTM5 methodology. The anatomical constructs and concepts associated to design decisions were expressed. The significance of the 5 views and the concerns represented by them were clearly stated. The domain specificity and usability of HTM5 was justified with extensive examples and case studies. The case studies presented in this thesis does not reflect on all the benefits of

the methodology like large complex systems developed in a commercial scenario. However, even with the small set of machines and platforms touched in the case studies, HTM5 was shown to be a feasible methodology. This thesis was also a generous discussion on the thought process that went behind development of HTM5 methodology.

Following are the research objectives presented in this work:

- To develop an OMG-MDA based Meta-Model for agent oriented development of cloud robotic systems.
- To specify the 5 views and their scope in the development of cloud robotic system.
- To provide provisions in HTM5 to have a graphical representation (Agent Relation Charts) for inter-agent interactions specific to each of the 5 views.
- To provide a mechanism (Hyperactivity) by which an agent's autonomy can be released for specific agents. This is to give flexibility to system designers by inducing an object-like character to some agents, without fully dissolving their agency characteristics.
- To enable Components built in PIM layer of HTM5 be extendable to PSM layer.
- To provide provisions in HTM5 to support a relationship based, peer-to-peer exchange of services. HTM5 should propose mechanisms to implement cloud computing business logic.
- To develop a domain specific language HTM5-DSL with CIM layer abstraction.
- To automatically translate HTM5-DSL code to Java class hierarchy and equivalent UML class diagrams.
- To conduct diverse case studies on real and simulated robot colonies justifying the usability of HTM5 in cloud robotic systems. The case studies should incorporate the following test cases for HTM5 usability:
 - Incorporating multiple robotic platforms in one study
 - Incorporating internet based agents in at least one case study
 - Incorporating internet based commercial servers in at least one case study
 - Incorporating non robotic elements of a cloud robotic system
 - Real time execution response for all case studies
 - Simulations for cloud robotic agent colonies in at least two case studies
 - At least 5 cloud robotic entities in all physical case studies

- Upto 1000 cloud robotic entities in simulated case studies
- Incorporation of Dynamic Electronic Institutions in at least one of the case study
- Incorporation of Digital Business Ecosystem in at least two of the case study
- Incorporation of Peer-to-Peer trade dynamics in at least one of the case study
- A scaled down version of simulated case studies to be implemented on physical robots
- Incorporation of cloud robotic business model in at least two case studies

The key contributions of this doctoral research are as follows:

- *Development of a 5 View Hyperactive Transaction Meta-Model (HTM5) for development of agent oriented cloud robotic systems.*
- *Incorporation of OMG-MDA guidelines for development of HTM5, an industrial standard for development of MDA meta-models.*
- *Development of a computation independent Domain Specific Language (HTM5-DSL) for HTM5.*
- *Model and Model and Model to Text automated transformations from HTM5-DSL to UML and Java class hierarchy*
- *Numerous case study experiments to test and demonstrate the usability of HTM5 in real projects and with complicated cloud robotic implementation scenarios*

Key advantages that HTM5 brings to the Industry, Scientific and Business community working towards cloud robotic ecosystems:

- The proposed meta-model supports heterogeneity in robotic platforms.
- The proposed meta-model supports heterogeneity in the kinds and number of communication networks (Clouds).
- The proposed meta-model supports heterogeneity in kinds of cloud infrastructures.
- The proposed meta-model supports heterogeneity in number and kind of auxiliary devices in the cloud ecosystem.
- The proposed meta-model supports design heterogeneity at component level.

- The proposed meta-model follows object management groups (OMG) model driven architecture guidelines (OMG-MDA) for model transformations.
- The proposed meta-model is designed to work at computation independent layer (CIM) of HTM5 which is the most suited placement for a DSL used for requirement elicitation and concept development.
- A CIM level DSL promotes contribution of clients and non-domain personals in system design process.
- The code written in HTM5-DSL is abstract and has an intuitive grammar which serves as a design document even before execution.
- HTM5-DSL supports automated model to text (M2T) and model to model (M2M) transformations. The targets for these execution results are Java (general purpose language, GPL) and UML (unified modelling language) which are widespread in software industry.
- The proposed meta-model is agent oriented.
- The proposed meta-model enables implementation of complex behavioural models on cloud robotic systems.
- The proposed meta-model supports peer-to-peer service-demand interactions.
- The proposed meta-model is a multi-view meta-model which allows clear separation of view specific concerns.
- The proposed meta-model views support engineering needs at initial stages of the design process and in subsequent increments by allowing quick fix solutions (flexible agency).
- The proposed meta-model views supports inclusion of business logic at initial stages of the design process and dynamically during a system run.
- The proposed meta-model views supports research/rapid-prototyping needs of researchers.
- The proposed meta-model supports incremental development of cloud robotic ecosystem.
- The proposed meta-model enables smooth design upgradation when a base software/machine is changed/updated. (Inclusion of a machine model)
- The proposed meta-model allows flexibility in the concept of agency as and when required.

Future Work

- GUI tools to support ARC designs.
- Automated ARC to HTM5-DSL transformation.
- Extended HTM5-DSL to UML transformation to include behaviour specific methods/functions.
- Extended HTM5-DSL to Java transformation to include behaviour specific methods/functions.
- Integration of Simulation tools with HTM5-DSL.
- Industrial projects.
- Extended documentation.
- Product website and integration with open source community.

In HTM5 the behaviour of the agent oriented cloud robotic system is recorded in the behaviour view. In dynamic agent systems, the number of possible scenarios could be very large and not all foreseeable. In such systems, the behaviour view might not be able to capture all behaviour scenarios of the system. Furthermore, in learning systems and systems which have a global time/agent-age dependence on the behaviour model, the current behaviour model may seem limited. Although a good designer may still use the HTM5 behaviour model to capture all possible behaviour scenarios, the current model is not designed to automatically identify (based on other view designs) all possible use case scenarios that may occur. HTM5 was developed incrementally based on the case studies and feedbacks from the users. All captured features or suggestions from the users were incorporated in the model. A more rigorous analysis of the weaknesses or shortcomings of HTM5 will be possible in future industrial projects when HTM5 will be deployed in real life projects industrial constraints and feature variability

Model transformations and Domain Specific Language (DSL) are essential supplements to any model driven methodology. Development of HTM5-DSL, and making it an executable language required knowledge of various aspects and cultures of software industry. Automated Model-to-Text (to Java component classes) and Model-to-Model (to UML) transformations required a lot of engineering. The transformations are still a work in progress and more can be done to include all elements of HTM5 in automated code generation. Results of these transformations are design as well as implementation templates for individual development of Agents (representing robots and auxiliary systems) by various vendors. Future line of research would be to conduct case studies with industrial partners to popularize and improve HTM5 methodology. Automated transformations need more refinement to make them suitable for commercial use. Industry Participation; contributions from open source community for development of graphical tools to construct and transform the model; inclusion of dynamic digital institutions in HTM5 methodology are key requirements for establishing HTM5 as a usable methodology for future **Cloud Robotics**.

A list of publications:

1. [**IEEE CSNT 2012**] Nagrath, V.; Meriaudeau, F.; Malik, A.S.; Morel, O. Agent Relation Charts (ARCs) for Modeling Cloud based transactions, Communication Systems and Network Technologies (CSNT), 2012 International Conference on, On page(s): 704 - 709
2. [**IJCSN Korea 2012**] Nagrath, V.; Meriaudeau, F.; Malik, A.S.; Morel, O. A multi-view approach for modeling agent transactions over cloud, International Journal of Communication Systems and Networks published by SERSC Korea (IJCSN) **EXTENDED VERSION OF CONFERENCE PAPER [1] AS REQUESTED BY THE JOURNAL**
3. [**IEEE CSNT 2013**] Nagrath, V.; Meriaudeau, F.; Malik, A.S.; Morel, O. "Introducing the Concept of Hyperactivity in Multi Agent Systems", Communication Systems and Network Technologies (CSNT), 2013 International Conference on, On page(s): 542 - 546
4. [**IEEE CSNT 2013**] Furler, L.; Nagrath, V.; Malik, A.S.; Meriaudeau, F. "An Auto-Operated Telepresence System for the Nao Humanoid Robot", Communication Systems and Network Technologies (CSNT), 2013 International Conference on, On page(s): 262 - 267
5. [**UTP APC 2013**] Nagrath, V.; Malik, A.S. ; Saad, N.M.; Meriaudeau, F. "A Multi-View Approach On Platform Independent Component Design for Multi Agent Systems", Annual Postgraduate Conference 2013, University Technology Petronas (APC2013), 27-29 June 2013
6. [**IJCSN 2012 Vol1 Iss 3**] Nagrath, V.; Meriaudeau, F.; Malik, A.S.; Morel, O. "Inter Agent flow of control parameters by Hyperactivity Mechanism", International Journal of Communication Systems and Networks (IJCSN), 2012, VOLUME 1, ISSUE-3 **EXTENDED VERSION OF CONFERENCE PAPER [3] AS REQUESTED BY THE JOURNAL**
7. [**IJCSN 2012 Vol2 Iss 1**] Nagrath, V.; Furler, L.; Malik, A.S.; Meriaudeau, F. "A Telepresence and Teleoperation Implementation on the Nao Humanoid Robotic Platform", International Journal of Communication Systems and Networks (IJCSN), 2013, VOLUME 2, ISSUE-1 **EXTENDED VERSION OF CONFERENCE PAPER [4] AS REQUESTED BY THE JOURNAL**
8. [**SAI 2013 London**] Nagrath, V.; Morel, O.; Malik, A.S.; Saad, N.M.; Meriaudeau, F. "HTM5-Trade Model For Relationship Based Trade Modelling In Multi Agent Systems", Science and Information Conference (SAI), London, United Kingdom, 7-9 October 2013, On page(s): 188-196

9. **[IROS 2013 Tokyo]** Nagrath, V.; Morel, O.; Malik, A.S.; Saad, N.M.; Meriaudeau, F. "Agent driven Peer-to-Peer Cloud Robotics", Cloud Robotics Workshop, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2013), Tokyo Big Sight, Japan, 3-9 November 2013
10. **[Springer Software and Systems Modeling (SOSYM)]** Nagrath, V.; Morel, O.; Malik, A.S.; Saad, N.M.; Meriaudeau, F. "HTM5 Domain Specific Language for Agent Oriented Cloud Robotic Systems" **Accepted, awaiting publication**
11. **[SpringerPlus]** Nagrath, V.; Morel, O.; Malik, A.S.; Saad, N.M.; Meriaudeau, F. "Dynamic Electronic Institutions in HTM5 Meta Model for Agent Oriented Cloud Robotic Systems" **Accepted, awaiting publication**
12. **[Springer Peer-to-Peer Networking and Applications]** Nagrath, V.; Morel, O.; Malik, A.S.; Saad, N.M.; Meriaudeau, F. "Peer to Peer Trade in HTM5 Meta Model for Agent Oriented Cloud Robotic Systems" **Accepted, awaiting publication**

Bibliography

- [1] “IEEE recommended practice for architectural description of software-intensive systems,” *IEEE Std 1471-2000*, pp. i–23, 2000.
- [2] E. Seidewitz, “What models mean,” *IEEE Softw.*, vol. 20, no. 5, pp. 26–32, Sep. 2003. [Online]. Available: <http://dx.doi.org/10.1109/MS.2003.1231147>
- [3] OMG, *Model Driven Architecture (MDA) Guide*, 2003, oMG doc. ab/2003-06-01.
- [4] M. Luck, P. McBurne, O. Shehory, and S. Willmott, *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.
- [5] M. Luck, P. McBurney, and C. Preist, *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, 2003.
- [6] M. Wooldridge, *An Introduction to Multi-agent Systems*. John Wiley & Sons, February 2002.
- [7] J. I. Hungate and M. M. Gray, “Conference report: Application portability profile and open system environment users forum gaithersburg, may 910, 1995,” *Journal of Research of the National Institute of Standards and Technology*, vol. Volume 100, Number 6, pp. 699–709, 1995.
- [8] N. R. Jennings and S. Bussmann, “Agent-based control systems. Why are they suited to engineering complex systems?” *IEEE Control Systems Magazine*, vol. 23, no.3, pp. 61–73, Jun. 2003.
- [9] P. Stone and M. Veloso, “Multiagent systems: A survey from a machine learning perspective,” *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000. [Online]. Available: <http://dx.doi.org/10.1023/A3A1008942012299>
- [10] L. Panait and S. Luke, “Cooperative multi-agent learning: The state of the art,” *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005. [Online]. Available: <http://dx.doi.org/10.1007/s10458-005-2631-2>

- [11] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. Cambridge, MA, USA: MIT Press, 1999.
- [12] D. M. Weiss and C. T. R. Lai, *Software Product-line Engineering: A Family-based Software Development Process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
- [13] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.
- [14] K. Pohl, G. Böckle, and F. J. V. D. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.
- [15] *Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*, ISO/IEC Std. 9075-1, 2008.
- [16] *IEEE Standard VHDL Language Reference Manual 2009*, IEEE Std. ISBN 978-0-7381-6854-8, 2009.
- [17] “HTML by the internet engineering task force (IETF).” [Online]. Available: <http://tools.ietf.org/html/>
- [18] “Matrix laboratory by mathworks. [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.mathworks.com/products/matlab/>
- [19] D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. New York, NY, USA: John Wiley & Sons, Inc., 2002.
- [20] T. Stahl, M. Voelter, and K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.
- [21] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, “Viewpoints: A framework for integrating multiple perspectives in system development.” *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, no. 1, pp. 31–57, 1992. [Online]. Available: <http://dblp.uni-trier.de/db/journals/ijseke/ijseke2.html#FinkelsteinKNFG92>
- [22] D. Alonso, C. Vincent-Chicote, F. Ortiz, J. Pastor, and B. Ivarez, “V3CMM: a 3-view component meta-model for model-driven robotic software development,” *Journal of Software Engineering for Robotics – JOSE*, vol. 1, pp. 3–17, January 2010.
- [23] WWW, “Internet of Things. [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.rfidjournal.com/articles/view?4986>

- [24] —, “Web of Things. [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.webofthings.org/>
- [25] “Rosbridge applications layer network protocol specification [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.rosbridge.org/>
- [26] “ROS the robot operating system [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.ros.org/>
- [27] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. Kumar, K. D. Meng, and G. W. Kit, “DAvinCi: A cloud computing framework for service robots,” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 3084–3089.
- [28] “Rosjava [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://wiki.ros.org/rosjava>
- [29] M. Waibel, M. Beetz, J. Civera, R. D’Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, “RoboEarth,” *Robotics Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, 2011.
- [30] “GostaiNet [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.gostai.com/>
- [31] S. Candido and J. Kuffner, “Cloud-based robot grasping with the google object recognition engine,” in *IEEE Intl Conf. on Robotics and Automation*, 2013, p. 8.
- [32] “Google goggles [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.google.com/mobile/goggles/>
- [33] “The point cloud library (PCL) [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.pointclouds.org/>
- [34] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [35] P. Noriega, *Agent mediated auctions: the fishmarket metaphor*. Institut d’Investigació en Intel·ligència Artificial, 1999.
- [36] F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Eds., *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Springer US, 2004, no. ISBN 978-1-4020-8057-9.
- [37] H. Aldewereld, J. Vazquez-Salceda, F. Dignum, and J.-J. C. Meyer, “Norm compliance of protocols in electronic institutions,” in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS ’05. New

- York, NY, USA: ACM, 2005, pp. 1291–1292. [Online]. Available: <http://doi.acm.org/10.1145/1082473.1082738>
- [38] F. L. Y. Lopez, “Social power and norms: Impact on agent behaviour,” Tech. Rep., 2003.
- [39] V. Dignum, “A model for organizational interaction: based on agents, founded in logic,” Ph.D. dissertation, Universiteit Utrecht, 2004.
- [40] E. Muntaner-Perich and J. Rosa Esteva, “Using Dynamic Electronic Institutions to Enable Digital Business Ecosystems,” in *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, ser. Lecture Notes in Computer Science, P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, Eds. Springer Berlin Heidelberg, 2007, vol. 4386, pp. 259–273.
- [41] ———, “Towards a formalisation of dynamic electronic institutions,” in *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, ser. Lecture Notes in Computer Science, J. Sichman, J. Padget, S. Ossowski, and P. Noriega, Eds. Springer Berlin Heidelberg, 2008, vol. 4870, pp. 97–109.
- [42] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.
- [43] A. S. Rao and M. P. Georgeff, “BDI Agents: From Theory to Practice,” in *In Proceedings of the first international conference on multi-agent systems (ICMAS-95, 1995*, pp. 312–319.
- [44] S. D. Ramchurn, D. Huynh, N. R. Jennings *et al.*, “Trust in multi-agent systems,” *The Knowledge Engineering Review*, vol. 19, no. 1, pp. 1–25, 2004.
- [45] M. P. Singh, “An ontology for commitments in multiagent systems,” *Artificial Intelligence and Law*, pp. 97–113, 1999.
- [46] “Abstract Types [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html>
- [47] O. M. Group, “OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2,” Tech. Rep., Nov. 2007. [Online]. Available: <http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF>
- [48] G. Booch, *Software Components With Ada: Structures, Tools, and Subsystems*, ser. The Benjamin/Cummings Series in Ada and Software Engineering. Benjamin-Cummings Publishing Company, 1987.
- [49] C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object-oriented Programming*, ser. ACM Press Series. ACM Press, 2002. [Online]. Available: <http://books.google.co.in/books?id=U896iwmTiagC>

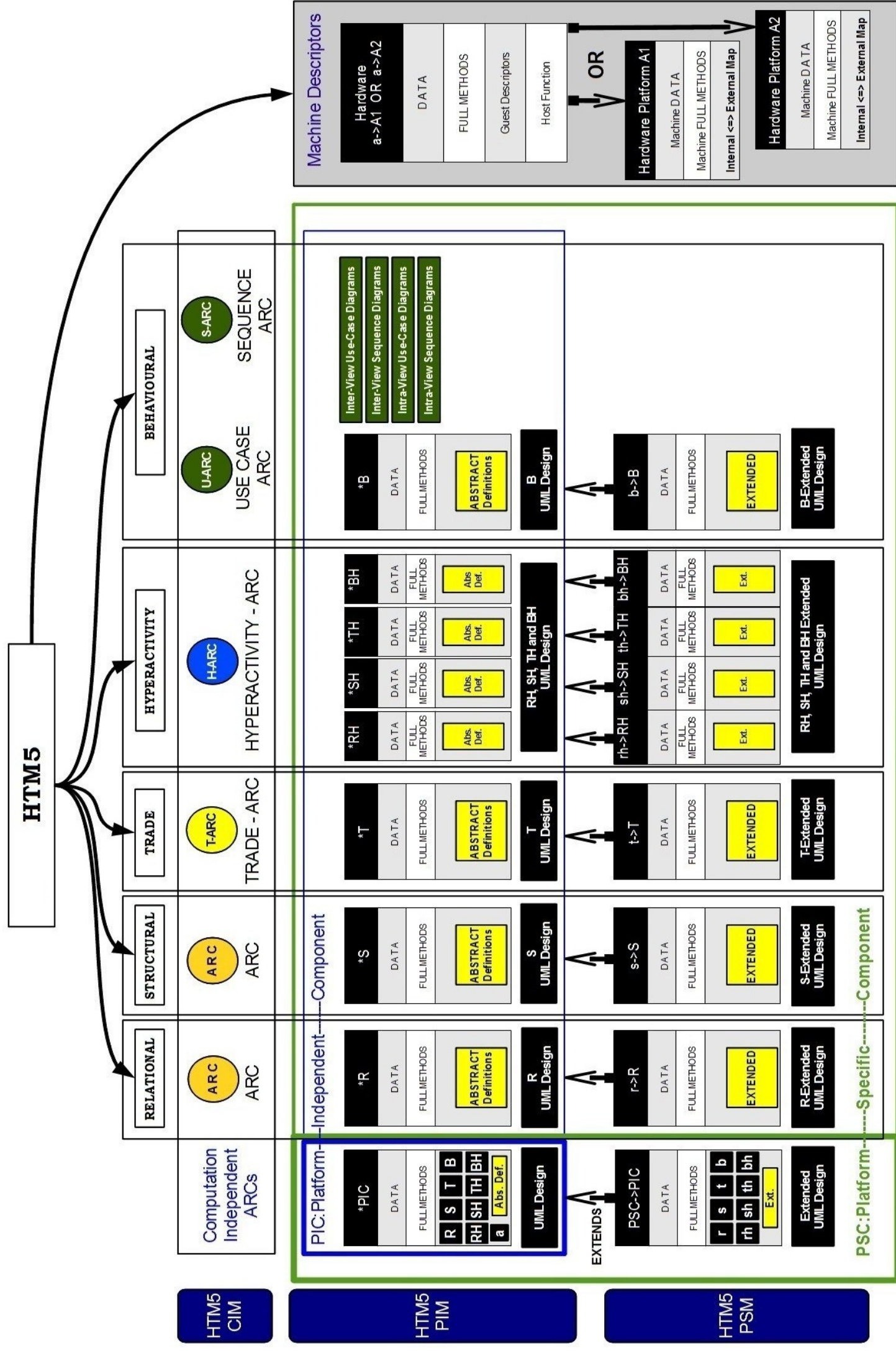
- [50] “Xtext, an open-source framework for development of programming languages and domain-specific languages. [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.eclipse.org/xtext>
- [51] “Xtend, a general-purpose high-level programming language for the JAVA virtual machine. [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.eclipse.org/xtend>
- [52] “PlantUML, an open-source tool that uses simple textual descriptions to draw UML diagrams.” [Online]. Available: <http://plantuml.sourceforge.net/>
- [53] “GraphViz, Graph Visualization Software: A package of open-source tools initiated by AT&T labs research for drawing graphs.” [Online]. Available: <http://www.graphviz.org/>
- [54] A. van Deursen, P. Klint, and J. Visser, “Domain-specific languages: An annotated bibliography,” *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, Jun. 2000. [Online]. Available: <http://doi.acm.org/10.1145/352029.352035>
- [55] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005. [Online]. Available: <http://doi.acm.org/10.1145/1118890.1118892>
- [56] “Open Office DRAW, Graphics editing software included as part of the Open Office Suite. [Official Website].” [Online]. Available: <http://www.openoffice.org/product/draw>
- [57] V. Nagrath, F. Meriaudeau, A. S. Malik, and O. Morel, “Agent relation charts (ARCs) for modeling cloud based transactions,” in *Proceedings of the 2012 International Conference on Communication Systems and Network Technologies*, ser. CSNT ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 704–709. [Online]. Available: <http://dx.doi.org/10.1109/CSNT.2012.156>
- [58] V. Nagrath, F. Meriaudeau, A. Malik, and O. Morel, “Introducing the concept of hyperactivity in multi agent systems,” in *Communication Systems and Network Technologies (CSNT), 2013 International Conference on*, 2013, pp. 542–546.
- [59] V. Nagrath, O. Morel, A. Malik, N. Saad, and F. Meriaudeau, “HTM5-Trade model for relationship based trade modelling in multi agent systems,” in *Science and Information Conference (SAI), 2013*, 2013, pp. 188–196.
- [60] ———, “5-View Hyperactive Transaction Meta-Model for Agent-Oriented Cloud Robotic Systems ”Manuscript submitted for publication”.
- [61] D. C. North, “Economics and Cognitive Science,” Econ-WPA, Economic History 9612002, Dec. 1996. [Online]. Available: <http://ideas.repec.org/p/wpa/wuwpeh/9612002.html>

- [62] H. E. Pattison, D. D. Corkill, and V. R. Lesser, “Instantiating descriptions of organizational structures,” in *Distributed Artificial Intelligence*, ser. Research Notes in Artificial Intelligence, M. N. Huhns, Ed. Pitman, 1987, ch. 3, pp. 59–96.
- [63] E. Werner, “Cooperating agents: A unified theory of communication and social structure,” in *Distributed Artificial Intelligence (Vol. II)*, L. Gasser and M. N. Huhns, Eds. San Mateo, CA: Kaufmann, 1989, pp. 3–36.
- [64] “VisualBots Simulator [latest access on 06/11/2014 0344 hrs].” [Online]. Available: <http://www.visualbots.com/>
- [65] “Visual Basic for Applications. [latest access on 06/11/2014 0344 hrs] [Official Website].” [Online]. Available: <http://excelvbatutor.com>
- [66] “TurtleBOTs. [latest access on 06/11/2014 0344 hrs] [Official Website].” [Online]. Available: <http://www.turtlebot.com/>
- [67] “Dynamic Electronic Institutions, Digital Business Ecosystem and Peer to Peer Cloud Robotics Simulation Videos.” [Online]. Available: <https://sites.google.com/site/deidbep2psimulations/>
- [68] F. Panzieri, z. Babaoglu, S. Ferretti, V. Ghini, and M. Marzolla, “Distributed computing in the 21st century: Some aspects of cloud computing.” in *Dependable and Historic Computing*, ser. Lecture Notes in Computer Science, C. B. Jones and J. L. Lloyd, Eds., vol. 6875. Springer, 2011, pp. 393–412.
- [69] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, “Peer-to-peer cloud provisioning: Service discovery and load-balancing,” in *Cloud Computing*, ser. Computer Communications and Networks, N. Antonopoulos and L. Gillam, Eds. Springer London, 2010, pp. 195–217.

Appendices

Appendix A

Full page version of intricate images.




```

1 package HTM5.DSL.ObjMiner.NetworkClouds
2 {
3     /*Specifying Different Clouds */
4     Cloud-Network Local_WiFi
5     {
6         Is-WiFi,
7         Is-Multiplex
8     }
9
10    Cloud-Network WWW
11    Cloud-Network Trader_Machine_Inter_Com
12    Cloud-Network C_Server_Inter_Com
13    Cloud-Network A_Server_Inter_Com
14 }

```

A



```

1 package HTM5.DSL.ObjMiner.TradeItems
2 {
3     /*Specifying Trade items */
4     Trade Mine_Coordinates
5     Trade Cargo
6     Trade Search_Space
7     Trade Cargo_Payment
8     Trade Sub_Search_Space
9     Trade Trader_ID
10    Trade Miner_ID
11    Trade Initial_Location
12    Trade Initial_Coordinates
13    Trade Miner_Salary
14    Trade Target_Minerals
15    Trade Sub_Space_Hit
16
17    TradeDatum Mineral_X_Price
18    { Is-LookUpTable ,
19      For-Demand: Cargo_Payment }
20    TradeDatum MineralID_X_Hz
21    { Is-CostMetric ,
22      For-Trade: Target_Minerals }
23    TradeDatum MineralID
24    { Is-Variable,
25      For-Demand: Target_Minerals }
26    TradeDatum TraderID_X_Cargo
27    TradeDatum MinerID_X_Salary
28    TradeDatum Location_X_MineralID
29    TradeDatum MinedAreaPc
30    TradeDatum TraderID_X_NumMiner
31    TradeDatum TraderID_X_Coordinates
32    TradeDatum TraderID_X_Salary
33    TradeDatum MinerID
34    TradeDatum MinerID_X_JobStatus
35    TradeDatum MinerID_X_MinSalary
36    TradeDatum Pi_TraderID_X_Salary
37 }

```

B



```

package HTM5.DSL.ObjMiner.Connections
{
import HTM5.DSL.ObjMiner.NetworkClouds.*
import HTM5.DSL.ObjMiner.HTM5Components.*
/*
 * Specifying Connections between HTM5 components
 * via Various Cloud Networks */
Connection L01 Miner [Nb] <-- Local_WiFi --> M4
Connection L02 Miner [Nc] <-- Local_WiFi --> M3
Connection L03 M4 <-- Trader_Machine_Inter_Com --> #r1# R4
Connection L04 R4 #r2# <-- Trader_Machine_Inter_Com --> Trader
Connection L05 Trader [Na] <-- WWW --> M1
Connection L06 M1 <-- C_Server_Inter_Com --> #r2# R1
Connection L07 R1 #r1# <-- C_Server_Inter_Com --> Collector
Connection L08 Trader [Na] <-- WWW --> M2
Connection L09 M2 <-- A_Server_Inter_Com --> #r1# R2
Connection L10 R2 #r2# <-- A_Server_Inter_Com --> Agency
Connection L11 Agency <-- A_Server_Inter_Com --> #r1# R3
Connection L12 R3 #r2# <-- A_Server_Inter_Com --> M3
}

```

C



```

1 package HTM5.DSL.ObjMiner.Machines
2 {
3     import HTM5.DSL.ObjMiner.TradeItems.*
4     import HTM5.DSL.ObjMiner.HTM5Components.*
5     import HTM5.DSL.ObjMiner.Connections.*
6
7     Cardinality Na Cardinality Nb Cardinality Nc
8     Cardinality N Cardinality r1 Cardinality r2
9
10    /*Specifying Machines*/
11    many [N] Component-Host MinerBot
12    <i>Mobile_Robot </i>
13    {
14        ROS
15        many Camera
16        many Movement
17        IRSensor
18        many ProximitySensor
19        WiFiCommunicator
20    }
21    many[Na] Component-Host TraderMachine
22    Component-Host CollectorServer
23    <i>Web_Server </i>
24    {
25        LinuxOperatingSystem
26        WiredInternet
27        SocketCommunicator
28    }
29    Component-Host AgencyServer
30    Component-Host External
31 }

```

D



```

1 package HTM5.DSL.ObjMiner.UseCases
2 { import HTM5.DSL.ObjMiner.HTM5Components.*
3   import HTM5.DSL.ObjMiner.NetworkClouds.*
4
15    package UARC004Trader_is_Allocated_Miners{
16        Act a001 Between M4, R4;
17        <i>Reads_MinerID_of_MinerID_x_Salary</i>
18        Act a002 Between R4, Trader;
19        <i>Updates_MinerID_x_Salary</i>
20        Act a003 Between Trader, Agency;
21        <i>Allocates_List_of_MinerIDs</i>
22        Act a004 Between Agency, R2;
23        <i>Agency_Reads_TraderID_x_Miner_and_
24        TargetID_x_Coordinates_and_TraderID_x_Salary</i>
25        Act a005 Between Agency, R3;
26        <i>Agency_Reads_MinerID_and_MinerID_x_
27        MinSalary_and_MinerID_x_JobStatus</i>
28        Act a006 Between Agency, Miner;
29        <i>Allocates_TraderID_and_Initial_Location_
30        and_Communication_Coordinates</i>
31    }
32 }

```

F



```

1 package HTM5.DSL.ObjMiner.HTM5Components
2 {
3     import HTM5.DSL.ObjMiner.Machines.*
4     import HTM5.DSL.ObjMiner.TradeItems.*
5     /*External Actor */
6     HTM5component MineField hosted-at External
7     /*Specifying HTM5 Components */
8     /*At MinerBot */
9     many [N] HTM5component Miner hosted-at MinerBot
10    {
11        Type: Is-agent,
12        Is-HyperActive
13
14        Associates:
15        Is-H-Slave of Trader
16
17        Service: Sub_Space_Hit
18        Demand: Trader_ID
19        Initial_Location
20        Target_Minerals
21        Sub_Search_Space
22        Miner_Salary
23    }
24    /*At Trader Machine */
25    many [Na] HTM5component R4 hosted-at TraderMachine
26    {
27        Type: Is-relation,
28        Is-Passive
29        TradeData: MinerID_X_Salary
30        MineralID
31        Location_X_MineralID
32        MinedAreaPc
33        MineralID_X_Hz
34    }
35    many [Na] HTM5component M4 hosted-at TraderMachine
36    many [Na] HTM5component Trader hosted-at TraderMachine
37    /*At Collector Server */
38    HTM5component Collector hosted-at CollectorServer
39    HTM5component M1 hosted-at CollectorServer
40    HTM5component R1 hosted-at CollectorServer
41 }

```

E



```

1 package HTM5.DSL.ObjMiner.Sequence
2 {
3     import HTM5.DSL.ObjMiner.HTM5Components.*
4     import HTM5.DSL.ObjMiner.NetworkClouds.*
5
6    package SARC001Trader_Gets_Search_Space {}
7    package SARC002Trader_Requests_Miners {}
8    package SARC003Miners_Register_With_Agency {}
9    package SARC004Trader_is_Allocated_Miners {}
10   package SARC005Collector_Publishes_Mineral_Prices {}
11   package SARC006Miners_Registers_With_Trader {}
12   package SARC007Active_Miner_Looking_For_Switching_Jobs
13 }

```

G



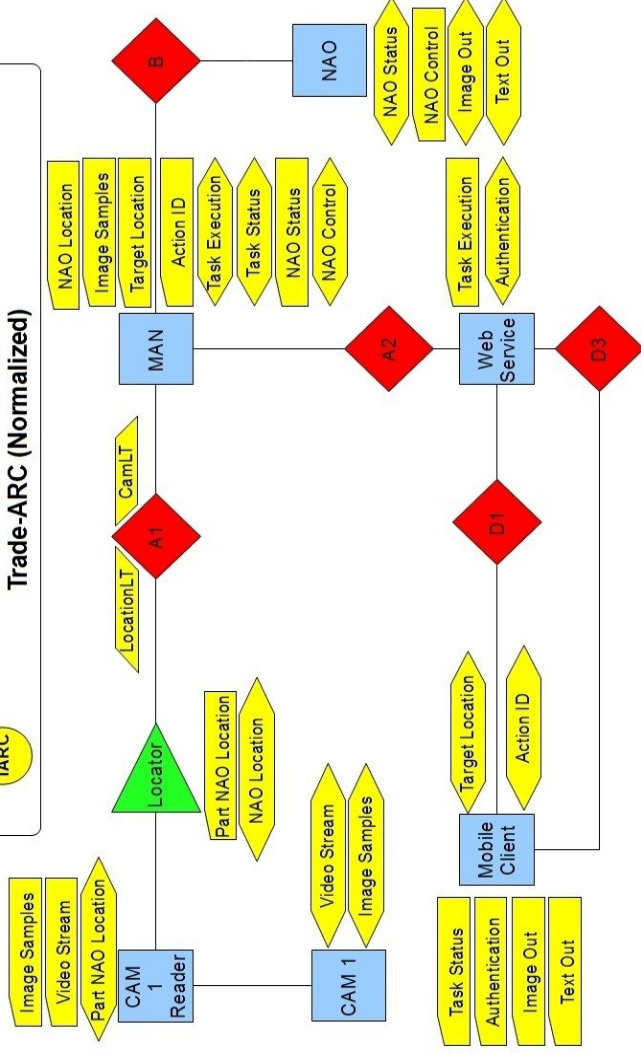
```

21    package SARC008Mining_Process {
22        Action ID001 [Miner] [A]<i>Reaches_A_Location_In_The_Miners_Sub_Search_Space</i>
23        Action ID002 [Miner] [A]<i>Records_Sensor_Data</i>
24        Action ID003 [Miner] [A]<i>Makes_A_List_Of_Positive_Matches_HITS_with_the_Target_IDs</i>
25        Action ID004 [Miner] [0]<i>Sends_the_List_of_Positive_HITS_to_M4</i>
26        Action ID005 [M4] [I]<i>Recieves_the_list_of_positive_hits</i>
27        Action ID006 [M4] [A]<i>Combines_positive_hits_from_all_Miners</i>
28        Action ID007 [M4] [0]<i>Sends_update_request_for_Location_x_MinerID_to_R4</i>
29        Action ID008 [R4] [I]<i>Recieves_update_request_for_Location_x_MinerID</i>
30        Action ID009 [R4] [A]<i>Updates_Location_x_MinerID</i>
31        Action ID010 [M4] [A]<i>Calculates_Positive_Hit_Frequencies_for_all_MineralIDs</i>
32        Action ID011 [M4] [0]<i>Sends_update_request_for_MinerID_x_HZ_to_R4</i>
33        Action ID012 [R4] [I]<i>Recieves_update_request_for_MinerID_x_HZ</i>
34        Action ID013 [R4] [A]<i>Updates_MinerID_x_HZ</i>
35        Action ID014 [M4] [A]<i>Calculates_the_percentage_of_area_mined</i>
36        Action ID015 [M4] [A]<i>Sends_update_request_for_MinedAreaPc_to_R4</i>
37        Action ID016 [R4] [0]<i>Recieves_update_request_for_MinedAreaPc</i>
38        Action ID017 [R4] [I]<i>Updates_MinerAreaPc</i>
39    }
40    package SARC009Trader_Initiates_Trade_With_Collector {}
41    package SARC010Collector_Accepts_Cargo {}
42 }

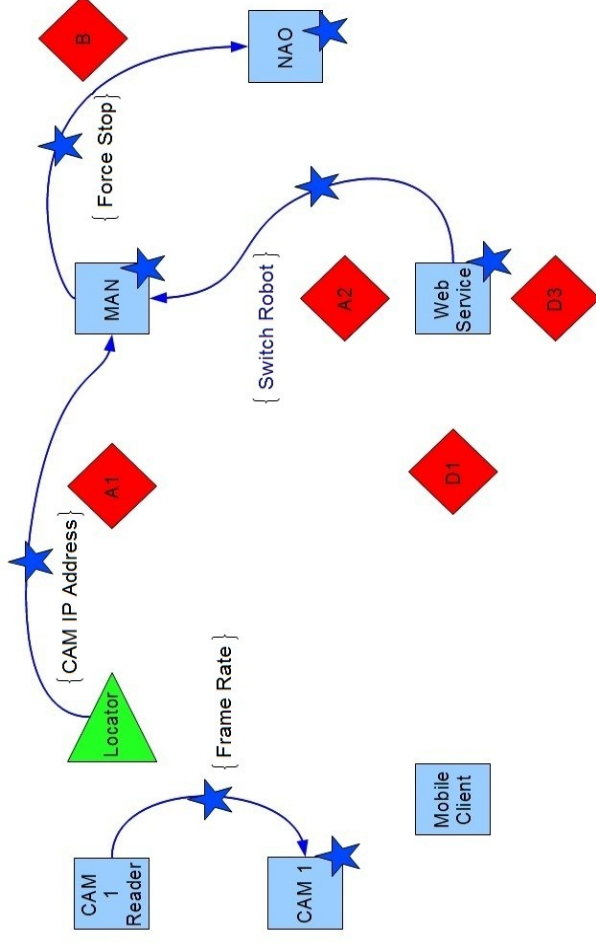
```



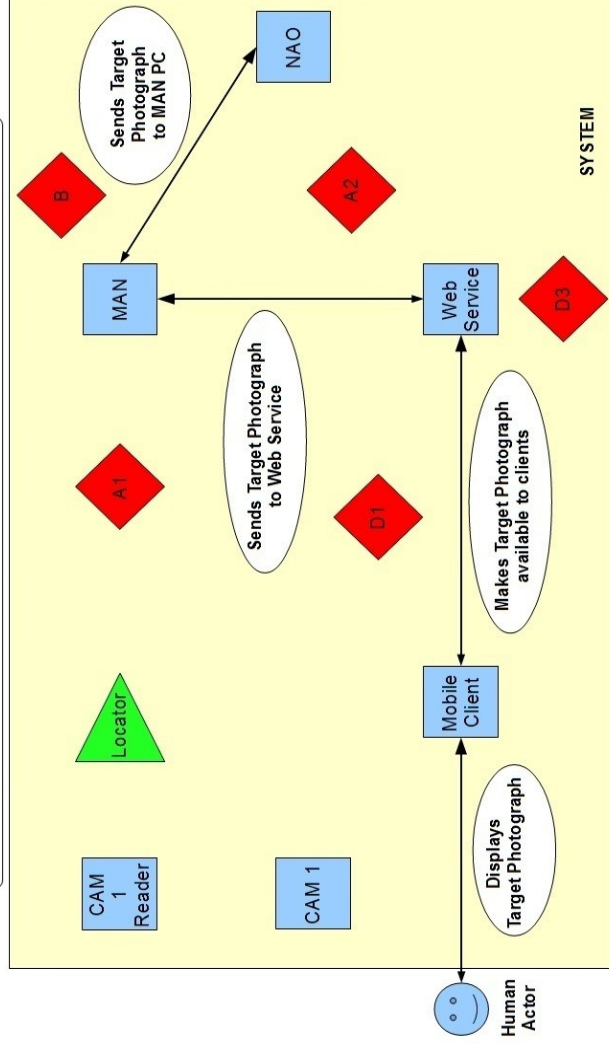

TELEROBOTICS ON NAO Trade-ARC (Normalized)



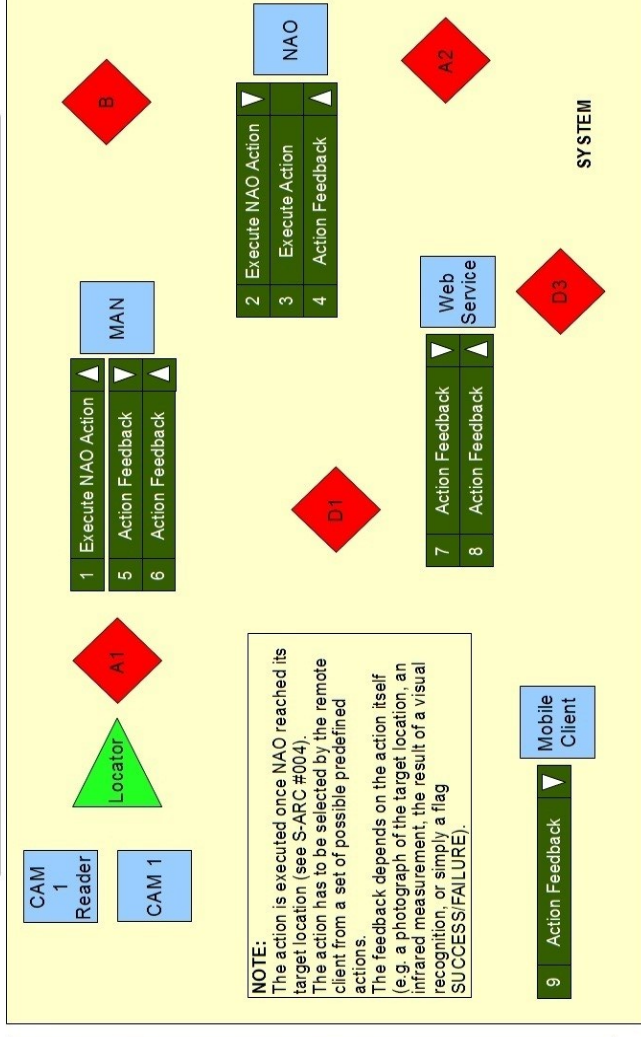
TELEROBOTICS ON NAO Hyperactivity-ARC (Normalized)



TELEROBOTICS ON NAO UARC #002: DISPLAY TARGET PHOTOGRAPH



TELEROBOTICS ON NAO SEQUENCE ARC #005: EXECUTE ACTION




```

1 package HTM5.DSL.TeleNao.NetworkClouds
2 {
3     /*Specifying Different Clouds */
4     Cloud-Network LAN /*Cloud #1*/
5     { Is-Ethernet, Is-Multiplex }
6     Cloud-Network Wireless_LAN /*Cloud #2*/
7     { Is-WiFi, Is-Multiplex }
8     Cloud-Network Mobile_Network /*Cloud #3*/
9     { Is-GSM, Is-Multiplex }
10    Cloud-Network WWN /*Cloud #4*/
11    { Is-Ethernet, Is-Multiplex }
12    Cloud-Network IP_Link /*Cloud #5*/
13    { Is-Ethernet, Is-Duplex }
14    Cloud-Network Manager_Inter_Com
17    Cloud-Network Web_Server_Inter_Com
20    Cloud-Network Nao_Inter_Com
23 }

```

```

1 package HTM5.DSL.TeleNao.TradeItems
2 {
3     /*Specifying Trade items */
4     Trade Image_Sample
5     Trade Video_Stream
6     Trade Part_NAO_Location
7     Trade NAO_Location
8     Trade Target_Location
9     Trade Action_ID
10    Trade NAO_Status
11    Trade NAO_Control
12    Trade Task_Execution
13    Trade Task_Status
14    Trade Authentication
15    Trade Image_Out
16    Trade Text_Out
17
18    TradeDatum LocationLT
19    { Is-LookUpTable ,
20      For-Demand: Target_Location }
21    TradeDatum CamLT
22    { Is-CostMetric ,
23      For-Service: Image_Sample}
24 }

```

```

1 package HTM5.DSL.TeleNao.Machines
2 {
3     import HTM5.DSL.TeleNao.TradeItems.*
4     import HTM5.DSL.TeleNao.HTM5Components.*
5     import HTM5.DSL.TeleNao.Connections.*
6
7     Cardinality x3 Cardinality xR Cardinality xN
8     Cardinality r1 Cardinality r2
9
10    /*Specifying Machines*/
11    many [x3] Component-Host Cam_Reader_Machine
17    many[xR] Component-Host NAO_Robot
18    <i> Humanoid_Robot </i>
19    {
20        many Camera      many Movement
21        Face_Recognition many LED
22        many Bump_Sensor many Sound
23        NaoQi             OpenNao
24        WiFiCommunicator WiredInternet
25    }

```

```

26    Component-Host Manager_PC
33    Component-Host Web_Server
41    Component-Host IP_Camera
43    many[xN] Component-Host Mobile_Client_Machine

```

```

1 package HTM5.DSL.TeleNao.HTM5Components
2 {
3     import HTM5.DSL.TeleNao.Machines.*
4     import HTM5.DSL.TeleNao.TradeItems.*
5     /*External Actor */
6     HTM5component HumanActor hosted-at External
7     /*Specifying HTM5 Components */
8     HTM5component MAN hosted-at Manager_PC
9
10    {
11        Type: Is-agent,
12              Is-HyperActive
13
14        Associates:
15        Is-H-Master of NAO
16        Is-H-Slave of Web_Service Locator
17
18        Service: Task_Execution Task_Status
19                 NAO_Control
20        Demand:  NAO_Location Image_Sample
21                 Target_Location
22                 Action_ID NAO_Status
23    }
24    many [N] HTM5component Mobile_Client
25    hosted-at Mobile_Client_Machine
26    {
27        Type: Is-agent,
28              Is-Passive
29
30        Associates:
31
32        Service: Target_Location Action_ID
33        Demand: Task_Status Authentication
34                 Image_Out Text_Out
35    }
36    many[xR] HTM5component NAO hosted-at NAO_Robot
38    HTM5component Web_Service hosted-at Web_Server
40    HTM5component Locator hosted-at Manager_PC

```

```

92 package SARC005Execution {
93     Action ID001 [MAN] [0]<i>Execute_NAO_Action</i>
94     Action ID002 [NAO] [I]<i>Execute_NAO_Action</i>
95     Action ID003 [NAO] [A]<i>Eecuting_Action</i>
96     Action ID004 [NAO] [0]<i>Action_Feedback</i>
97     Action ID005 [MAN] [I]<i>Action_Feedback</i>
98     Action ID006 [MAN] [0]<i>Action_Feedback</i>
99     Action ID007 [Web_Service] [I]<i>Action_Feedback</i>
100    Action ID008 [Web_Service] [0]<i>Action_Feedback</i>
101    Action ID009 [Mobile_Client] [I]<i>Action_Feedback</i>
102 }

```

```

10 package UARC002Display_Target_Phograph{
11    Act a001 Between NAO, MAN;
12    <i>Sends_Target_Phograph_To_MAN</i>
13    Act a002 Between MAN, Web_Service;
14    <i>Sends_Target_Phograph_To_Web_Service</i>
15    Act a003 Between Web_Service, Mobile_Client;
16    <i>Makes_Phograph_Available_to_Clients</i>
17    Act a004 Between HumanActor, Mobile_Client;
18    <i>Displays Target Photograph</i>
19 }

```



A
CAM
Agent

```

27 class CamReader(object):
28     """
29     Main class of the **CamReader** application.
30     Launch this module from a command line to start a new CamReader instance.
31     The script accepts the path to the camreader configuration file as optional parameter.
32     If not given, the default file name 'camreader_config.xml' is used.
33     """
34
35     def __init__(self, configFile, showOutput, logLevel):
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187

```

```

65 class ManagerPC(object):
66     """
67     Main class of the **ManagerPC** application.
68     Should be launched on a computer connected to the required number of CamReader agents and the WebServer
69     agent.
70     """
71
72     def __init__(self, configFile, taskFile, withWebServer, showOutput, logLevel):
73         """
74         Constructor: Initializes the ManagerPC application.
75
76         - configFile: Path to the ManagerPC XML configuration file.
77         - taskFile: Path to the XML task file that should be executed. If not given, no task is
78         executed locally.
79         - withWebServer: 1 [DEFAULT] -> The WebServer is active, waiting for remote tasks.
80         - showOutput: 1 [DEFAULT] -> Map and Robot Path are shown.
81         - logLevel: 0 -> No graphical output.
82
83         NOTE: It does not start the ManagerPC main loop. Use 'start()' to start it.
84         """
85
86         # Store the boolean parameters
87         self.withWebServer = withWebServer
88         self.showOutput = showOutput
89
90         # Initialize the logger
91         log.basicConfig(level=logLevel, format=LOG_FORMAT)
92
93         # Initialize the objects used during the task execution
94         self.currRobotLocation = None
95         self.currCamLocation = None
96         self.currCameraImage = None
97         self.currCamReaderProxy = None
98         self.currRobotProxy = None
99         self.currRobotPath = None
100         self.currTask = None
101
102         # Initially the ManagerPC is not running
103         self.running = False
104
105         # Create a Locator (for robot localization and path planning)
106         self.locator = Locator()
107         # Create an Image Processor (for XML transformation and image display)
108         self.imageProcessor = ImageProcessor()

```

C
MAN
Agent

```

stopStreaming()
    Disconnects the CameraProxy from the camera.
    ! Internal Method: Should not be called externally!

```

ImageProcessor Module

```

class camreader.image.ImageProcessor.ImageProcessor

```

```

Bases: object

```

```

Class that utilizes OpenCV to provide several image processing methods for the CamReader application.

```

```

detectMarker(img)

```

```

    Method to detect the location of a RGB circle marker within a color image. Determines the marker location
    by detecting the centers of a pure red, green and blue region in the input image.

```

```

    Returns the marker location (X, Y, orientation) if the marker could be fully detected, None otherwise.

```

```

extractColorBlob(imgHSV, hueMin, hueMax, satMin, satMax, valMin, valMax)

```

```

    Method used to extract a blob (subregion) of the given image 'imgHSV' based on color information.
    The input image must be a three channel HSV image. The six additional parameters define the mini-
    mum/maximum Hue/Saturation/Value values respectively.

```

```

    Returns a tuple (mask, center):

```

- mask is a filtered binary image in which all pixels belonging to the blob are set to 1
- center is a tuple (x, y) containing the center coordinates of the blob with subpixel precision

```

    If at least MIN_REGION_SIZE pixels that match the criteria are found in the filtered mask (i.e. the region
    is large enough), the tuple described above is returned. Otherwise (None, None) is returned indicating that
    a region with the given color values could not be found in the image.

```

```

findIdx(img, val)

```

```

    Finds the indices of all the cells in the input image 'img' that are equal to the given value 'val'. Returns
    a tuple (x, y) containing the two lists of corresponding row and column indices. If no matching cells are
    found, the returned lists are empty.

```

```

getImageFromXML(xmlStr)

```

```

    Creates a new OpenCV IplImage instance from the data present in the input XML string 'xmlStr'. See
    getXMLFromImage(img) for the inverse operation.

```

```

getMarkerPos(redCenter, greenCenter, blueCenter)

```

```

    Method used to compute the position of the RGB marker. The three parameters must be (x, y) tuples
    containing the center coordinates of the red, green and blue blobs (marker circles). The red center is
    assumed to be the front, i.e. the heading direction of the marker.

```

```

    Returns a RobotLocation object containing the (X, Y) position and orientation (in PI radians) of the marker

```

```

getXMLFromImage(img)

```

```

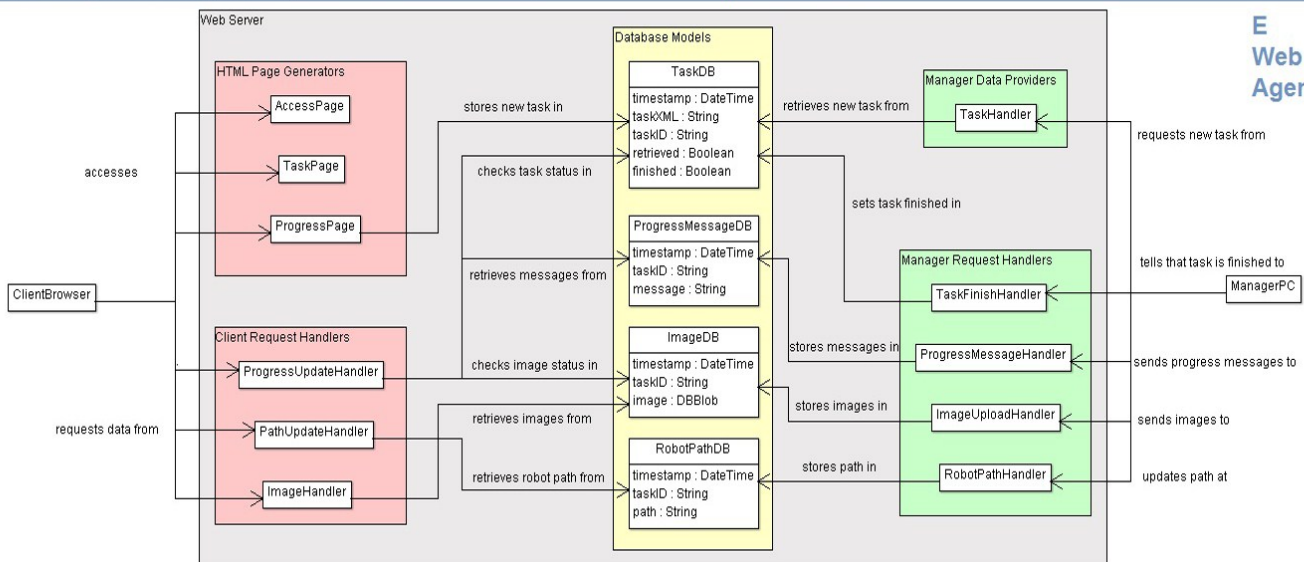
    Transforms the input OpenCV IplImage 'img' to an XML string. See getImageFromXML(xmlStr) for the
    inverse operation.

```

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <manager_pc_config>
3   <system>
4     <config_name>NAO Telerobotics</config_name>
5     <config_description>The setup consists of three CamReader
6     agents observing the workspace from a top view. The workspace
7     contains several obstacles. A single NAO humanoid robot is
8     executing the tasks.</config_description>
9   </system>
10
11   <workspace>
12     <map_file>C:\Documents and
13     Settings\Administrator\Desktop\Thesis\Project\Code\ManagerPC\map
14     \map.bmp</map_file>
15     <mask_file>C:\Documents and
16     Settings\Administrator\Desktop\Thesis\Project\Code\ManagerPC\map
17     \mask.bmp</mask_file>
18     <metric_to_pixel_scale>2.3</metric_to_pixel_scale>
19     <cell_size>15</cell_size>
20   </workspace>
21
22   <webserver>
23     <server_ip>naotelerobotics.appspot.com</server_ip>
24     <server_port>80</server_port>
25     <access_key>nao_manager_otp_2012</access_key>
26     <proxy_ip>nbproxy</proxy_ip>
27     <proxy_port>8080</proxy_port>
28   </webserver>

```

B
CAM
AgentD
MAN
AgentE
Web Service
Agent

```
##### IMPORTS #####
from google.appengine.ext import db
from google.appengine.api import users
from google.appengine.ext import webapp
from google.appengine.ext.webapp import template
from google.appengine.util import run_wsgi_app
```

A Client Agent

```
import handlers.Task as Task
import handlers.RobotLocation as RobotLocation
import handlers.datamodels as dm
```

```
##### HANDLER CLASSES #####
```

```
class AccessPage(webapp.RequestHandler):
    """
    RequestHandler class to generate the Access page at the root URL.
    """

    def get(self):
        """
        Handles incoming HTTP GET requests.
        """

        # Add the message data to the template value dictionary
        message = self.request.get('msg')
        templateValues = {'message': message}

        # Render the template and send it to the client
        renderTemplate(self, '../templates/access.html', templateValues)
```

```
class TaskPage(webapp.RequestHandler):
    """
    RequestHandler class to generate the page to define a new task.
    """
```

```
    def post(self):
        """
        Handles incoming HTTP POST requests.
        """

        # Extract the client access key from the request
        accessKey = self.request.get('access_key')

        # If it is not valid, redirect the client to the access page
        if accessKey is None or accessKey != CLIENT_ACCESS_KEY:
            redirectToAccessPage(self)
            return

        # Create the template value dictionary
        templateValues = {'accessKey': accessKey,
                          'actionList': Task.ACTIONS,
                          'feedbackList': Task.FEEDBACKS}

        # Render the template and send it to the client
        renderTemplate(self, '../templates/task.html', templateValues)
```

```
class ProgressPage(webapp.RequestHandler):
    """
    RequestHandler class to generate the page to display the task progress.
    """
```

B Client Agent

Web Page with User Interface

Telerobotics on NAO - Task Progress Page

Go back to Access Page

The following task was added successfully:

Remote Task (not complete) with ID 844d58c-9944-11e1-9687-e16ae519f52:
 --> Target Location: [RobotLocation] (x=66.0, y=36.0, theta=1.55841)
 --> Action on Success: A1 with parameters "Target location reached"
 --> Action on Failure: A2 with parameters "Home/hay/audio/failure.mp3"
 --> Client Feedback: F3

The current path of the robot is shown below:

The Manager PC is currently executing the specified task. The progress messages are displayed below:

Timestamp	Message from Manager PC
2012-05-08 08:07:13.756130	Robot walked straight for 12.250476 cm.
2012-05-08 08:07:13.756130	Task execution started.

Camera 1



Camera 2



Camera 3



C Client Agent

Web Page with User Interface

The Manager PC produced the following image as feedback:

2012-05-08 08:07:24.464859 Robot walked straight for 16.666390 cm.
 2012-05-08 08:07:19.380197 Robot rotated by 0.785398 radians.
 2012-05-08 08:07:13.756130 Robot walked straight for 12.250476 cm.
 2012-05-08 08:07:09.747187 Robot rotated by 0.214061 radians.
 2012-05-08 08:07:05.962940 Robot walked straight for 12.955685 cm.
 2012-05-08 08:07:01.021026 Robot walked straight for 11.320295 cm.
 2012-05-08 08:06:56.973003 Robot rotated by 0.189898 radians.
 2012-05-08 08:06:44.064643 Robot walked straight for 12.985771 cm.
 2012-05-08 08:06:27.576382 Robot walked straight for 13.628528 cm.
 2012-05-08 08:06:22.342581 Robot walked straight for 13.007227 cm.
 2012-05-08 08:06:27.748244 Robot walked straight for 12.850659 cm.
 2012-05-08 08:06:23.537696 Robot rotated by 0.189898 radians.
 2012-05-08 08:06:19.319367 Robot walked straight for 12.661418 cm.
 2012-05-08 08:06:14.877882 Robot rotated by -0.106726 radians.
 2012-05-08 08:06:09.856227 Robot walked straight for 14.121488 cm.
 2012-05-08 08:06:04.460766 Robot walked straight for 13.831867 cm.
 2012-05-08 08:05:58.480444 The robot is now in the view of camera 1.
 2012-05-08 08:05:58.081931 Robot walked straight for 12.495646 cm.
 2012-05-08 08:05:54.068419 Robot rotated by 0.235545 radians.
 2012-05-08 08:05:49.902228 Robot walked straight for 19.592214 cm.
 2012-05-08 08:05:44.440932 Robot rotated by 0.694738 radians.
 2012-05-08 08:05:35.352162 Robot walked straight for 13.043101 cm.
 2012-05-08 08:05:31.116326 Robot rotated by -2.774419 radians.
 2012-05-08 08:05:20.298489 The robot is now in the view of camera 3.
 2012-05-08 08:19:665657 Task execution started.

Camera 1



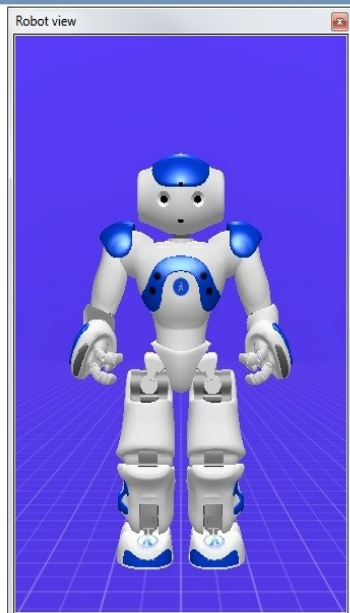
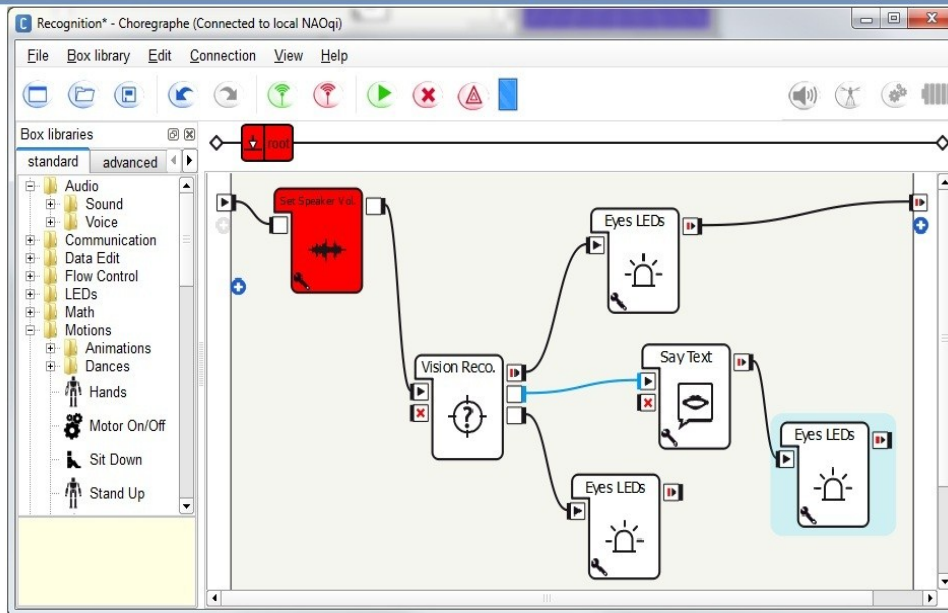
Camera 2

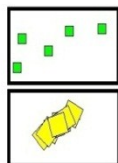
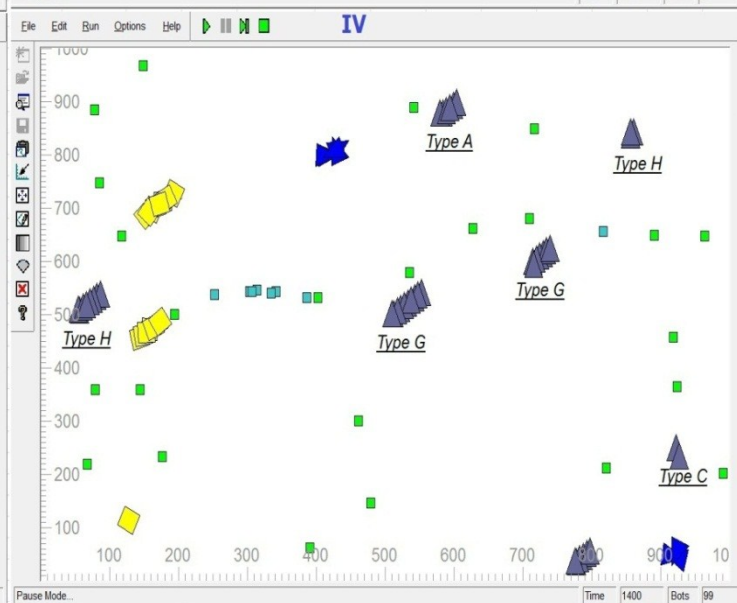
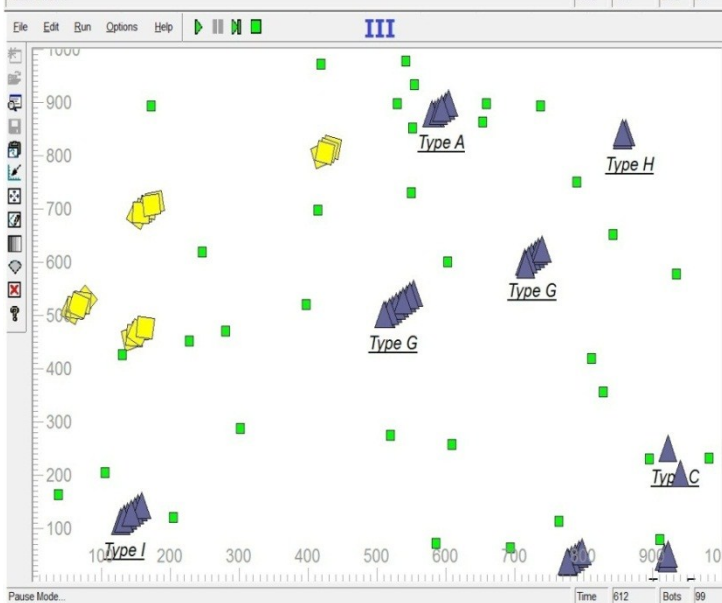
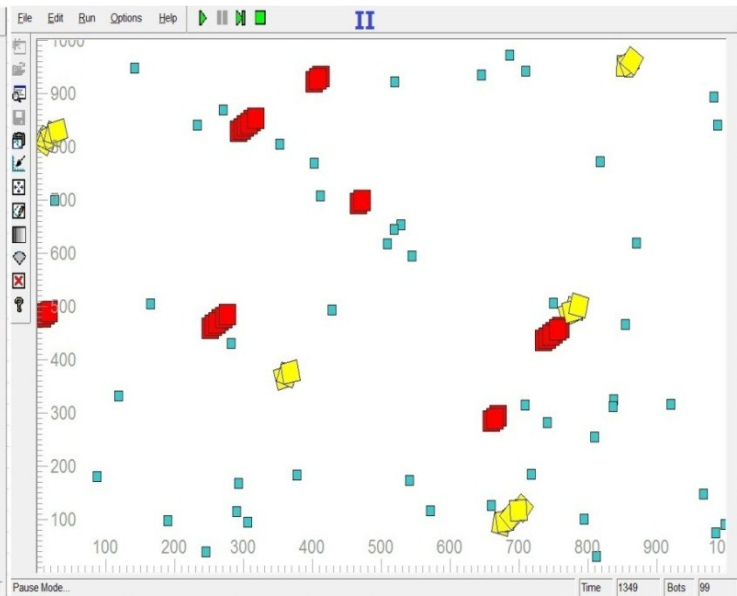
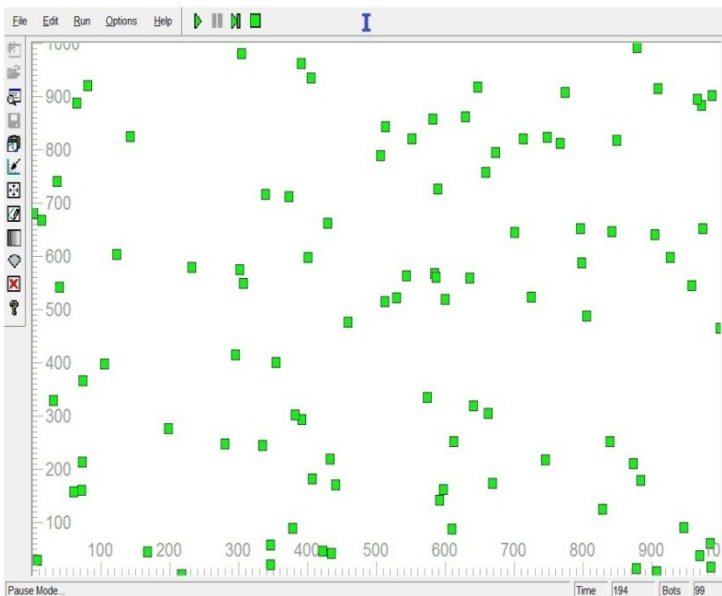


Camera 3

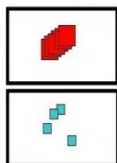


D NAO Agent

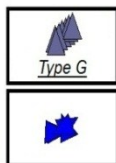




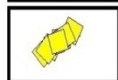
Free Agents



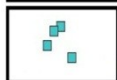
Coalition /
Foundation



Institution



Formation /
Reformation



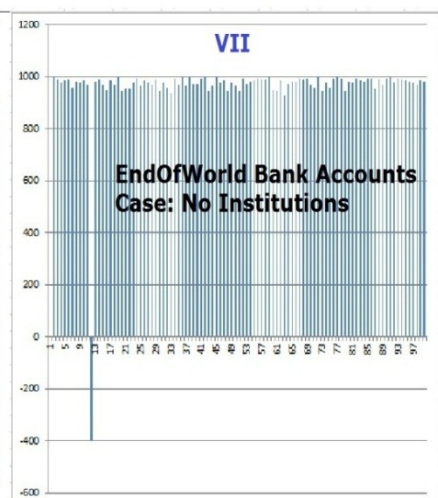
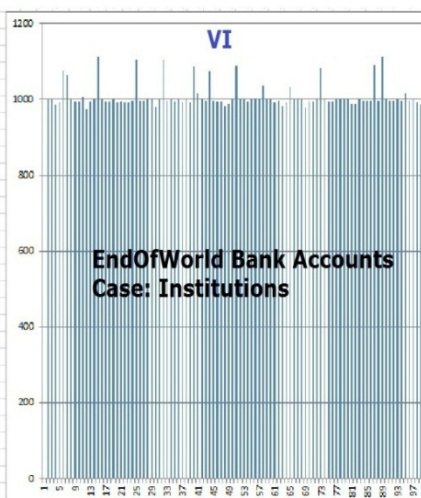
Free Agents
(Post Coalition)

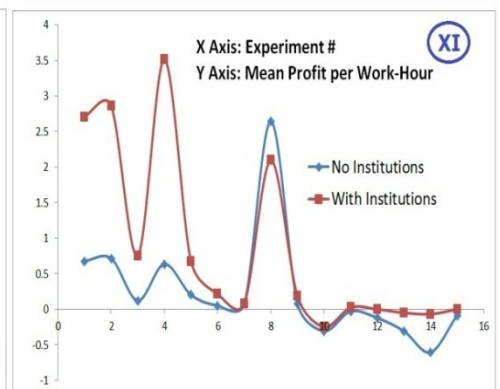
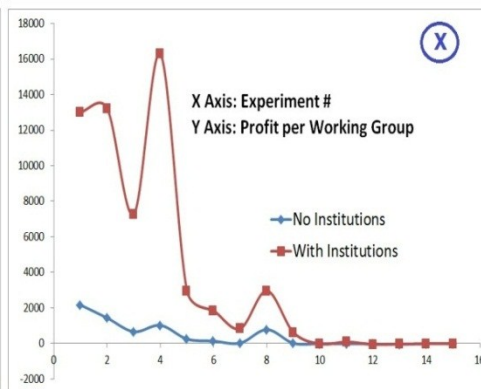
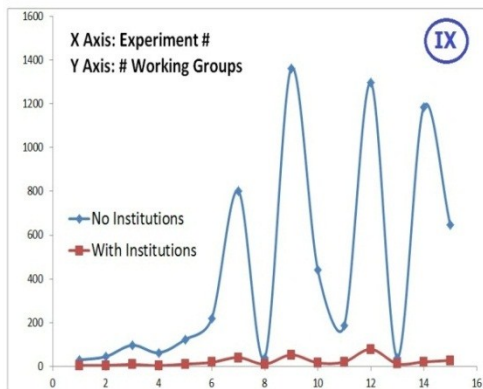
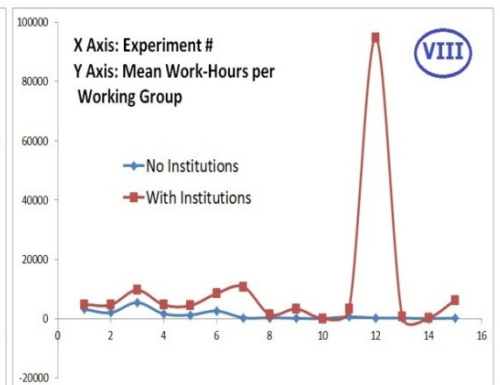
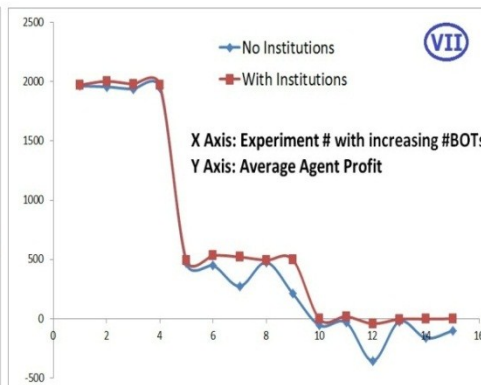
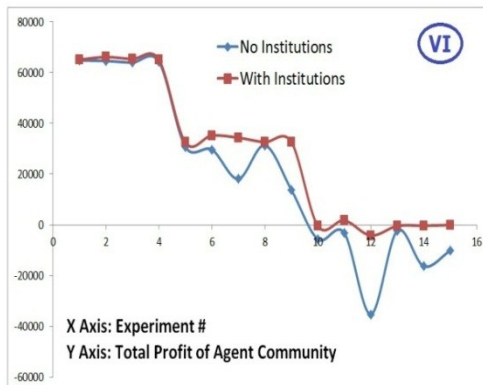
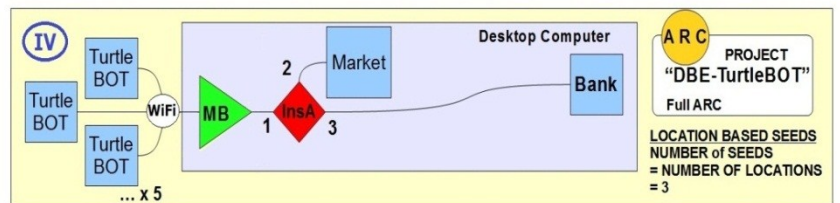
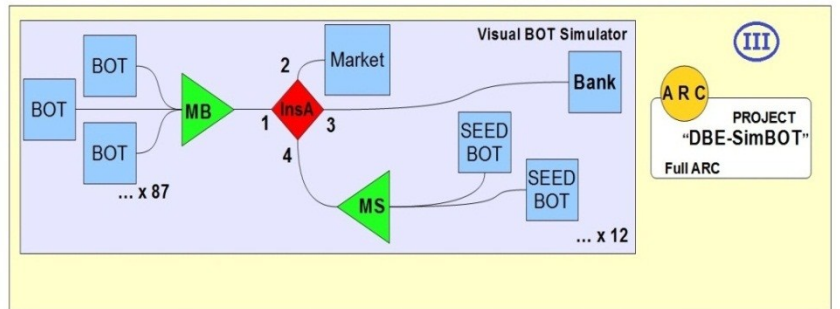
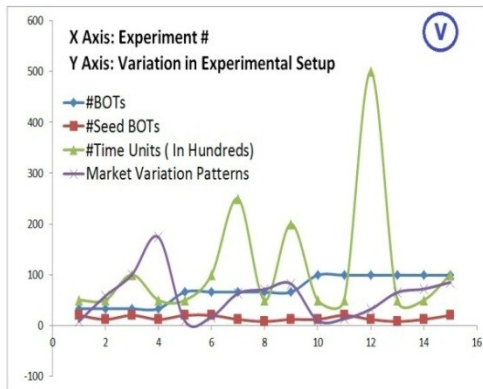


Re-Foundation

Number of Bots (Agents): 99
Proximity Rule: Physical Distance
Number of Seeds: 12

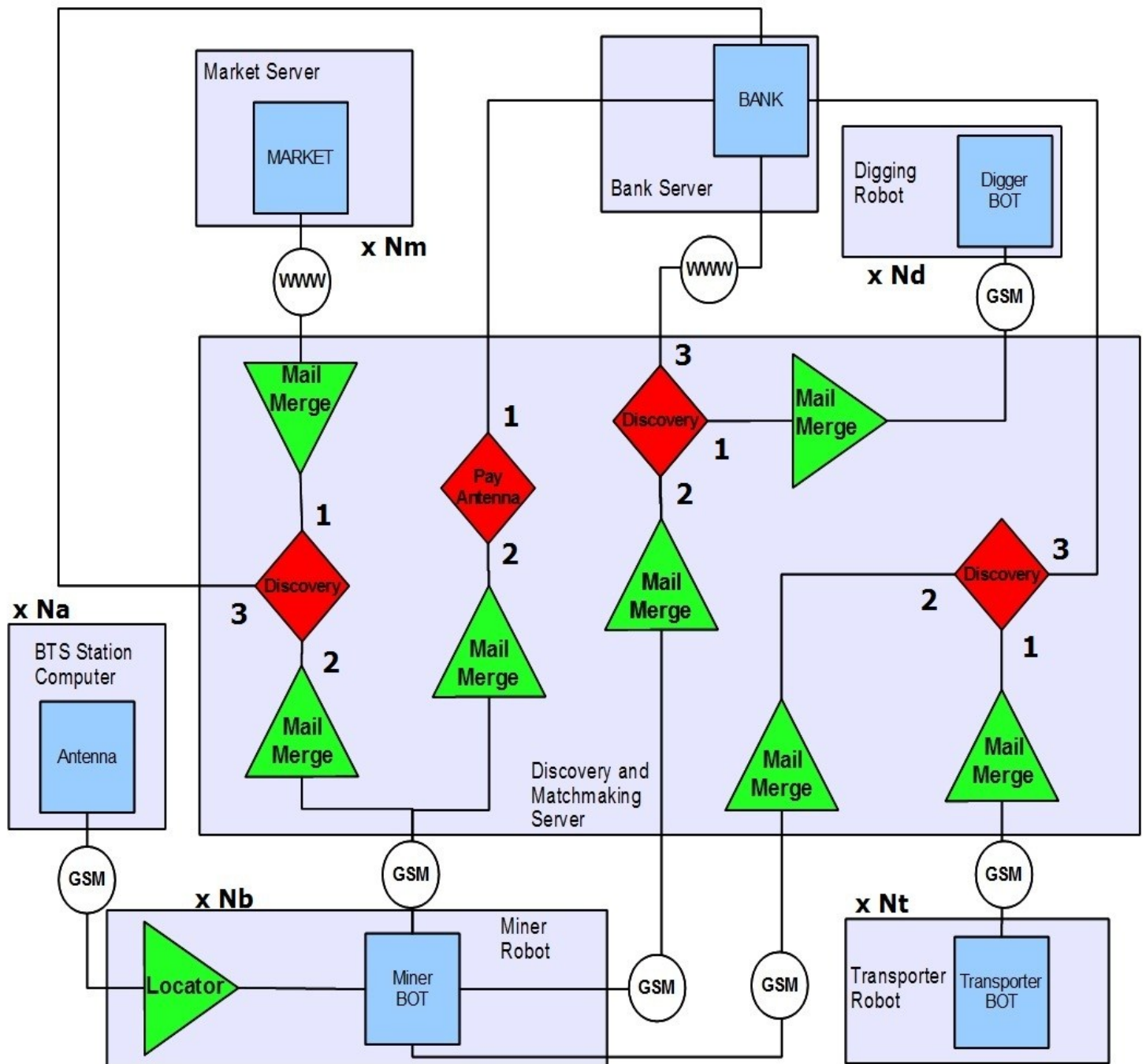
Development Tools: VBA, VisualBOTS

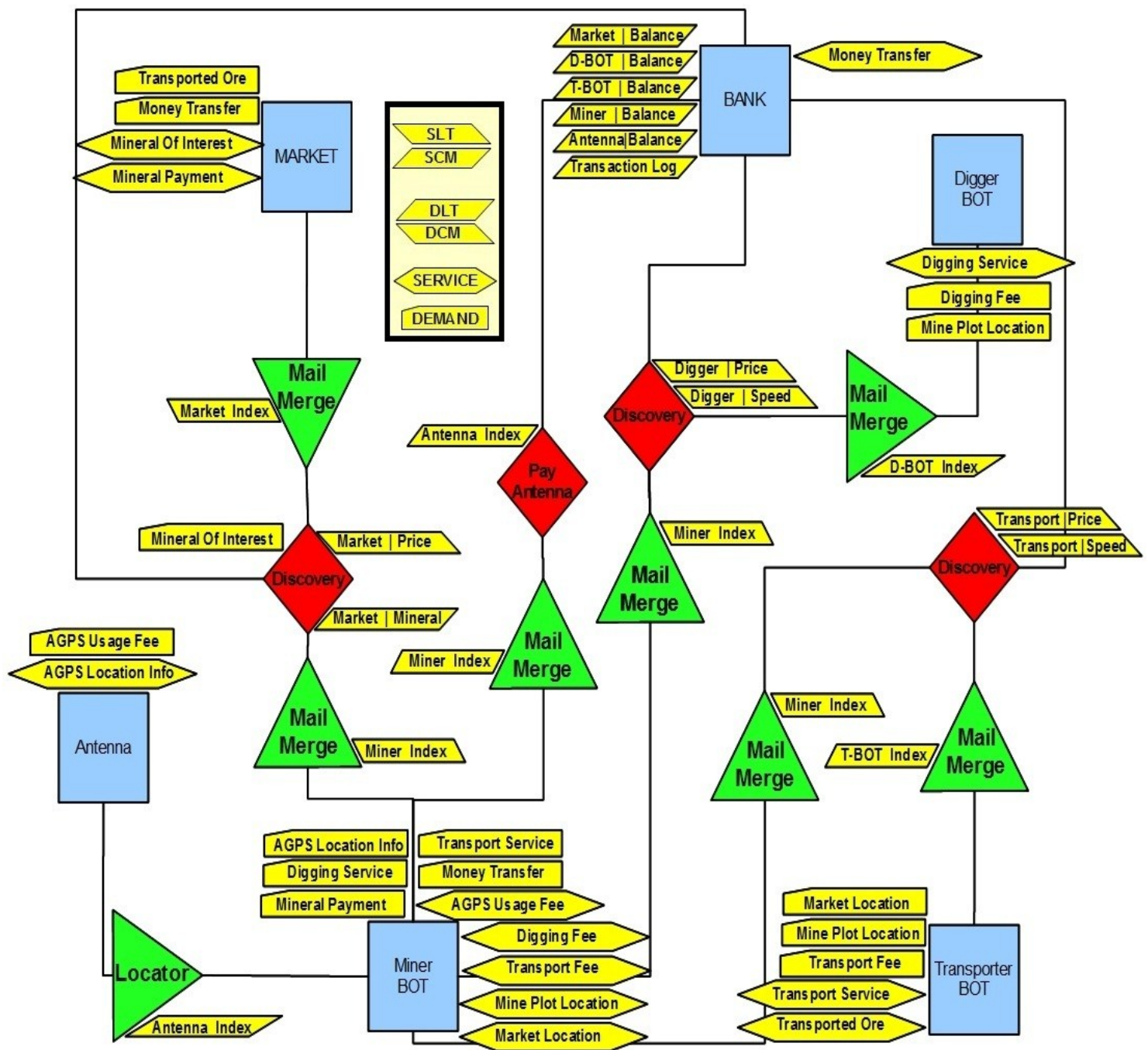


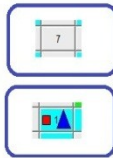
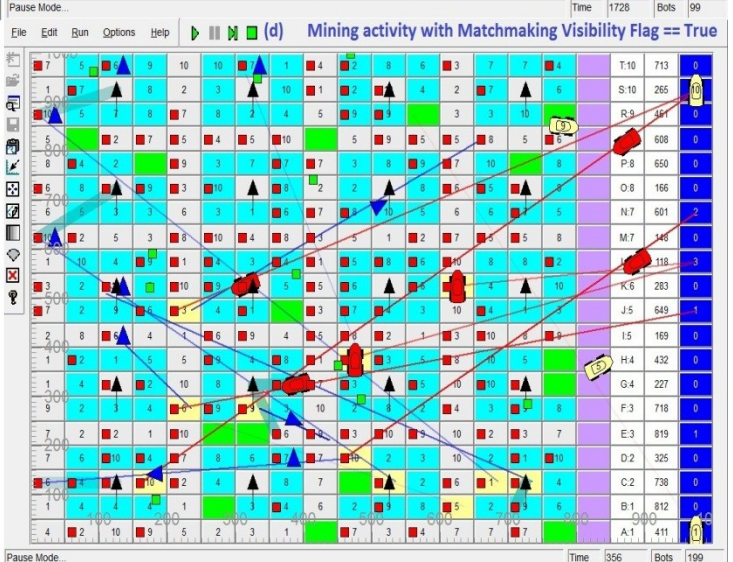
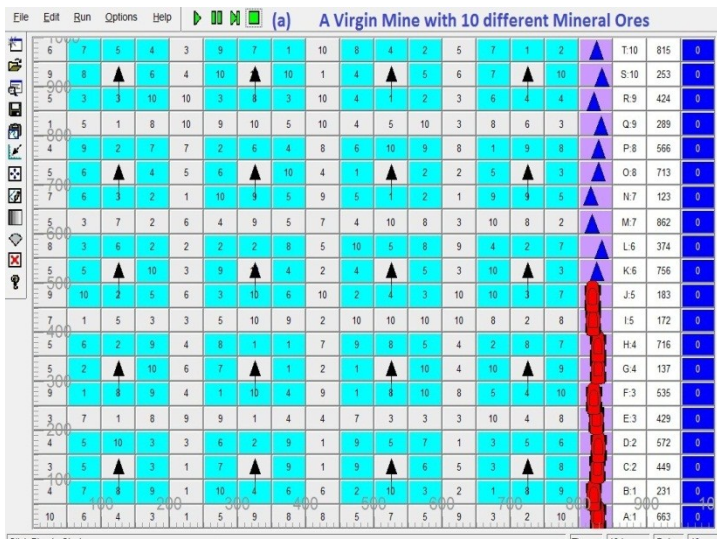


						NO Institute			With Institute			
#	#BOT	#Seed	EoW	Rand#	Bnk t0	Bnk EoW	Profit	Profit/BOT	Bnk EoW	Profit	Profit/BOT	Advantage
1	33	20	5000	8	33000	97798.1	64798.1	1963.578	98086.1	65086.09	1972.3058	288
2	33	12	5000	58	33000	97579.1	64579.1	1956.942	99102.1	66102.09	2003.0936	1523
3	33	20	10000	100	33000	96960.1	63960.1	1938.185	98335.1	65335.09	1979.8512	1375
4	33	12	5000	174	33000	97319.1	64319.1	1949.063	98164.1	65164.09	1974.6694	845
5	66	20	5000	9	66000	96736	30736	465.697	98515	32515	492.65152	1779
6	66	20	10000	17	66000	95675	29675	449.6212	101192	35192	533.21212	5517
7	66	12	25000	63	66000	84109	18109	274.3788	100344	34343.73	520.35955	16234.73
8	66	8	5000	71	66000	97347.1	31347.1	474.9559	98593.1	32593.09	493.8347	1246
9	66	12	20000	82	66000	79902	13902	210.6364	98739.9	32739.89	496.05894	18837.89
10	99	12	5000	10	99000	93377	-5623	-56.79798	98651	-349	-3.525253	5274
11	99	20	5000	13	99000	95820	-3180	-32.12121	100781	1781	17.989899	4961
12	99	12	50000	32	99000	63743	-35257	-356.1313	94826	-4174	-42.16162	31083
13	99	8	5000	65	99000	96732	-2268	-22.90909	98570	-430	-4.343434	1838
14	99	12	5000	72	99000	82738	-16262	-164.2626	98680	-320	-3.232323	15942
15	99	20	10000	85	99000	88810	-10190	-102.9293	98960.1	-39.875	-0.402778	10150.13
	1056	220	2E+05				308645			425539.2		116893.7

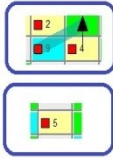
NO Institute					With Institute				
#Set	Time/Set	Time	Profit/Set	Profit/Time	#Set	Time/Set	Time	Profit/Set	Profit/Time
30	3184.88	95546.4	2159.94	0.6781845	5	4806.85	24034.3	13017.2	2.7080558
45	2007.1	90319.5	1435.09	0.7150072	5	4616.99	23085	13220.4	2.8634279
97	5432.29	526932	659.382	0.121382	9	9627.67	86649	7259.45	0.75401987
63	1607.5	101273	1020.94	0.6351091	4	4637.79	18551.2	16291	3.51266929
124	1218.76	151126	247.871	0.2033796	11	4383.83	48222.1	2955.91	0.67427548
220	2580.58	567728	134.886	0.0522698	19	8482.91	161175	1852.21	0.21834612
803	288.59	231738	22.5517	0.0781444	41	10760.9	441197	837.652	0.07784218
40	296.117	11844.7	783.677	2.6465122	11	1413.73	15551	2963.01	2.09587982
1362	128.332	174788	10.207	0.0795363	53	3285.38	174125	617.734	0.18802506
442	41.4437	18318.1	-12.7217	-0.306964	17	82.6364	1404.82	-20.5294	-0.24843062
187	590.862	110491	-17.0053	-0.028781	21	3340.16	70143.3	84.8095	0.02539089
1297	223.237	289538	-27.1835	-0.12177	79	94826	7491254	-52.8354	-0.00055718
41	180.29	7391.89	-55.3171	-0.306823	13	664.417	8637.42	-33.0769	-0.04978338
1185	22.671	26865.1	-13.7232	-0.60532	21	224.765	4720.07	-15.2381	-0.06779568
647	185.93	120297	-15.7496	-0.084707	28	6237.08	174638	-1.42411	-0.00022833
6583	17988.6	2524196	6332.84	3.7551611	337	157391	8743388	58976.3	12.7511372



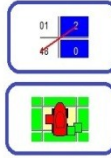




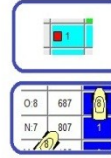
Virgin plot with Mineral #7



A GSM Communication between a Miner BOT and an Antenna



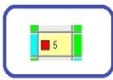
Matchmaking Result: Matching a Market to a Miner BOT



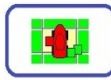
Miner BOT stops and turns Red when it finds its Target Mineral



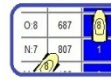
A Digger BOT working at a Plot



A Plot ready with Mineral Ore



A Transporter BOT loading at a Plot



Transporter BOTs with Mineral Ore #8 moving towards a Mineral #8 Market and Dumping

(e)

Item	Count	Variability									
		Movement Speed		Service Delay		Service Fee		Mineral Ore Count		Mineral Ore Price	
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
plot		320	320	X	X	X	X	1	10	X	X
Market		1	20	X	X	0	0	1	10	0	1000
Antenna		20	20	X	X	X	X	X	X	X	X
Digger BOT		1	10	2	20	200	500	50	150	X	X
Transporter BOT		1	10	2	20	(100, 100)	(300, 500)	50	150	X	X
Miner BOT		1	200	2	20	X	X	X	X	X	X

Testcase						Dedicated Teams						Peer-2-Peer Trade						Difference						
Exp#	Rand	#Bot	#DBOT	#TBOT	#Mines	Time	Av	D Av	T Av	A Av	B Av	#Mines	Time	Av	D Av	T Av	A Av	B Av	#Mines	Av	A Avg	D Av	T Av	B Av
1	1234	10	2	2	45	5000	1638	3516	3298	1088	2029	37	5000	1659	2986	2748	1147	2199	-8	21	59	-530	-550	170
2	4321	10	2	2	28	5000	1459	2681	2056	1064	1884	27	5000	1390	2207	2135	1099	1660	-1	-69	35	-474	79	-224
3	1234	60	2	2	47	5000	1275	5315	3710	1140	1104	45	5000	1288	4925	3508	1363	1068	-2	13	223	-390	-202	-36
4	4321	60	2	2	16	5000	1112	3336	1610	1140	1011	18	5000	1123	3484	1965	1380	931	2	11	240	148	355	-80
5	1234	10	10	2	80	5000	1852	5235	1790	1136	2668	63	5000	1860	4374	1676	1187	2886	-17	8	51	-861	-114	218
6	4321	10	10	2	68	5000	1875	4512	1644	1113	3101	76	5000	1915	4294	1675	1164	3182	8	40	51	-218	31	81
7	1234	60	10	2	76	5000	1400	4986	1842	1167	1285	70	5000	1402	4470	1761	1417	1235	-6	2	250	-516	-81	-50
8	4321	60	10	2	51	5000	1276	3580	1437	1202	1197	49	5000	1287	3458	1519	1437	1127	-2	11	235	-122	82	-70
9	1234	10	2	10	47	5000	1503	1510	3329	1088	1962	38	5000	1496	1425	2786	1153	1993	-9	-7	65	-85	-543	31
10	4321	10	2	10	22	5000	1230	1273	1796	1054	1425	31	5000	1401	1312	2388	1107	1880	9	171	53	39	592	455
11	1234	60	2	10	50	5000	1276	2145	3958	1148	1085	44	5000	1257	1799	3648	1397	1040	-6	-19	249	-346	-310	-45
12	4321	60	2	10	17	5000	1100	1681	1584	1144	973	17	5000	1105	1631	1704	1356	913	0	5	212	-50	120	-60
13	1234	10	10	10	78	5000	1803	1858	1740	1141	3135	87	5000	1930	1986	1981	1229	3222	9	127	88	128	241	87
14	4321	10	10	10	71	5000	1799	1760	1662	1132	3308	109	5000	2186	2068	1986	1210	4456	38	387	78	308	324	1148
15	1234	60	10	10	206	5000	2065	3576	3279	1322	1859	206	5000	2034	3279	3309	1558	1773	0	-31	236	-297	30	-86
16	4321	60	10	10	188	5000	1936	2908	2844	1394	1803	192	5000	2039	2954	2898	1619	1883	4	103	225	46	54	80
17	1234	20	4	4	90	5000	1943	3946	3292	1162	2054	92	5000	2051	3563	3011	1244	2364	2	108	82	-383	-281	310
18	4321	20	4	4	69	5000	1736	3365	2686	1131	1824	72	5000	1784	3220	2932	1240	1812	3	48	109	-145	246	-12
19	1234	40	4	4	85	5000	1626	4213	3012	1155	1464	86	5000	1660	3646	2972	1355	1482	1	34	200	-567	-40	18
20	4321	40	4	4	103	5000	1721	4564	3362	1247	1509	103	5000	1713	4013	3008	1552	1434	0	-8	305	-551	-354	-75
21	1234	20	8	4	151	5000	2434	5118	2812	1239	2941	158	5000	2667	5019	2725	1345	3494	7	233	106	-99	-87	553
22	4321	20	8	4	105	5000	1999	3989	2301	1199	2280	121	5000	2260	4445	2532	1311	2663	16	261	112	456	231	383
23	1234	40	8	4	134	5000	1949	5224	2594	1224	1856	149	5000	2086	4903	2830	1421	1988	15	137	197	-321	236	132
24	4321	40	8	4	165	5000	2097	5482	2983	1338	1962	170	5000	2181	5382	2747	1651	2012	5	84	313	-100	-236	50
25	1234	20	4	8	91	5000	1889	2392	3315	1156	2136	93	5000	1988	2343	3064	1248	2372	2	99	92	-49	-251	236
26	4321	20	4	8	78	5000	1736	2243	2902	1148	1887	77	5000	1770	2182	2972	1265	1870	-1	34	117	-61	70	-17
27	1234	40	4	8	88	5000	1644	2627	3157	1160	1538	90	5000	1632	2433	2937	1319	1498	2	-12	159	-194	-220	-40
28	4321	40	4	8	103	5000	1727	2874	3406	1259	1564	103	5000	1740	2491	2963	1580	1548	0	13	321	-383	-443	-16
29	1234	20	8	8	149	5000	2394	3075	2788	1235	3124	165	5000	2584	3149	2793	1331	3527	16	190	96	74	5	403
30	4321	20	8	8	132	5000	2209	2921	2590	1226	2754	136	5000	2328	2967	2700	1363	2887	4	119	137	46	110	133
31	1234	40	8	8	162	5000	2106	3480	3021	1250	2076	174	5000	2199	3433	3168	1440	2138	12	93	190	-47	147	62
32	4321	40	8	8	173	5000	2203	3441	3069	1362	2203	177	5000	2199	3311	2798	1660	2126	4	-4	298	-130	-271	-77

Exp# => Experiment Number

Rand => Randomiser seed for Mineral price variability

#Bot => Number of Miner Robots

#DBOT => Number of Digger Robots

#TBOT => Number of Transporter Robots

Time => Length of Experiment

#Mines => Total number of plots mined by the colony (Out of 320)

<All Robots and Antennas started with an initial bank balance of 1000>

Av => Average bank balance for all robots and antennas at end of experiment

D Av => Average bank balance for all Digger robots at end of experiment

T Av => Average bank balance for all Transporter robots at end of experiment

A Av => Average bank balance for all Antennas at end of experiment

B Av => Average bank balance for all Miner robots at end of experiment

[illegible]

Attributes	TeleNAO	DEI (Simulated)	DEI (Physical)	P2P (Simulated)	P2P (Physical)	Workshop
Type of Study	Physical	Simulated	Physical	Simulated	Physical	CIM Layer
# Entities in cloud robotic ecosystem	10+	38 to 104	7 to 9	57 to 123	13	Innumerable
# Robotic entities	1+	33 to 99	3 to 5	14 to 80	5	Innumerable
Robots	NAO Humanoid robot	Simulated	TurtleBOT robot	Digger Robot; Miner Robot; Transporter Robot	TurtleBOT robot	Innumerable
# Entities that represent cloud infrastructure and devices	6+	5	4	3	3	Innumerable
Cloud infrastructure and devices	Personal computers; Web server; Cloud storage and application hosting; mobile phones; mobile internet enabled devices	Bank; Market; Matchmaking agents; Dynamic electronic institution server (implemented as a Relation)	Personal computers	Bank; Market; Service discovery and matchmaking server;	Personal computers	Innumerable
# Entities that represent auxiliary devices	3+	0	0	40	5	Innumerable
Auxiliary devices	IP Cameras	N/A	N/A	BTS Antenna	Location Marker	Innumerable
# Clouds (networks)	5	3	3	3	3	Innumerable
# Public clouds (networks)	2	1	1	1	1	Innumerable
# Private clouds (networks)	3	2	2	2	2	Innumerable
# Simulated clouds (networks)	0	3	0	3	0	Innumerable
Cloud/networks	LAN; Wireless LAN; Mobile phone data network; Internet; IP Link	Wireless LAN; Internet	Wireless LAN; Internet; Inter Process Communication (IPC)	Wireless LAN; Internet	Wireless LAN; Internet; Inter Process Communication (IPC)	Innumerable
Peer-to-Peer trade (DBE)	Partial	Yes	Yes	Yes	Yes	Yes
Relational trade	No	Yes	Yes	Yes	Yes	Yes
Scalable	Yes	Yes	Yes	Yes	Yes	Yes
Open system	Yes	Yes	Yes	Yes	Yes	Yes
Dynamic service discovery	No	Yes	Yes	Yes	Yes	Yes
Matchmaking	No	Yes	Yes	Yes	Yes	Yes
Service demand cardinality	1 to many	many to many	many to many	many to many	many to many	Innumerable
Business logic implementation	Yes	Yes	Yes	Yes	Yes	Yes
Intermediate models used for component implementation	UML	None	UML	None	UML	N/A
Tools used for final component implementation	NaoQj; Python GPL; Open CV; XML; Google App Engine; AJAX; HTML;	VBA; VisualBOTS; MS Excel	linux shell scripting; ROS routines	VBA; VisualBOTS; MS Excel	linux shell scripting; ROS routines	N/A
Device Operating systems	OpenNao; MS Windows; Linux; Google customized Linux; Android; Symbian, IOS; MacOS	MS Windows; Linux Virtual Machine;	MS Windows; Linux Virtual Machine; Robot Operating System (ROS)	MS Windows; Linux Virtual Machine;	MS Windows; Linux Virtual Machine; Robot Operating System (ROS)	N/A
# Repetitions / # Instances	20+	120+	20+	120+	15+	6 Project Ideas
# Stakeholders (Human)	4	3	3	3	3	24
Domains represented	Electronics; Computer science; Software engineering; Robotics; Computer vision; Image processing; Web application development; Rapid prototyping; Management	Computer science; Software engineering; Robotics; Rapid prototyping; Management	Electronics; Computer science; Software engineering; Robotics; Rapid prototyping; Management	Computer science; Software engineering; Robotics; Rapid prototyping; Management	Electronics; Computer science; Software engineering; Robotics; Rapid prototyping; Management	Robotics; Robot fabrication; Power systems; Digital signal processing; physica; Computer networks; Computer programming; Software engineering; Embedded systems; Computer vision
Level of familiarity with domains	Masters; Doctoral candidate; professor	Doctoral candidate; professor	Doctoral candidate; professor	Doctoral candidate; professor	Doctoral candidate; professor	PHD; Doctoral candidates; Masters; Bachelor; Foundation
Verbal feedback	Yes	Yes	Yes	Yes	Yes	Yes
Presentations/Brainstorming	Yes	Yes	Yes	Yes	Yes	Yes
Feedback questioner	Yes	None	None	None	None	Yes

Appendix B

A Detailed explanation for Object Miner project

The object miner project is designed as a model project to demonstrate the computation independent design process in HTM5. The project is used in the thesis to draw sample ARC diagrams and associated DSL code in HTM5-DSL. Section 3.3 gives details of the Machines, Network clouds, HTM5-Components, Items in Trade, External actors and Data items associated with the trade items. In Figures 3.4 till 3.11 (inclusive) the ARC, T-ARC, H-ARC and two examples each of U-ARC and S-ARCs are presented. In this appendix we will provide some more details about the project for the readers benefit. After the textual explanation, the appendix contains the whole set of ARC diagrams with all behavioural models (Use case and Sequence ARCs) for the project. **The text followed by the complete set of ARC diagrams should enable the reader have a better understanding of the project and the ARC based design methodology.**

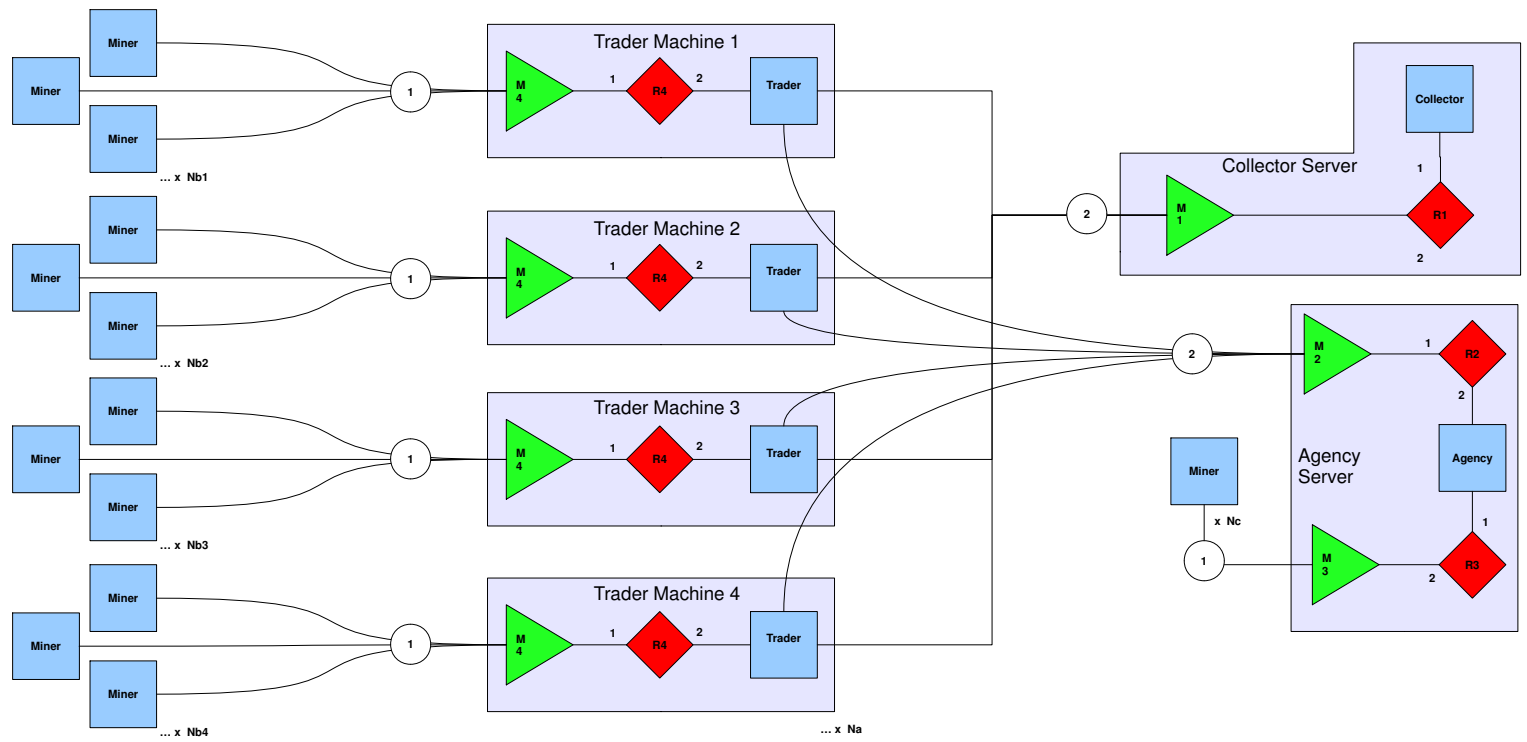
The Object miner project simulates a real life mining operation. There are a number of miners which are equipped to search the mine for specific minerals. Each miner has an ID and an initial location. In the beginning of the ecosystem, all miners are affiliated to the agency which simulates a job agency in real life. The miners cannot be hired directly. The miners are assigned to specific traders (employers for miners) based on the criteria specified by the miner and the trader. The criteria that the agency utilises to match miners to traders is based on a miners expected salary, miners availability and the salaries offered by a particular trader.

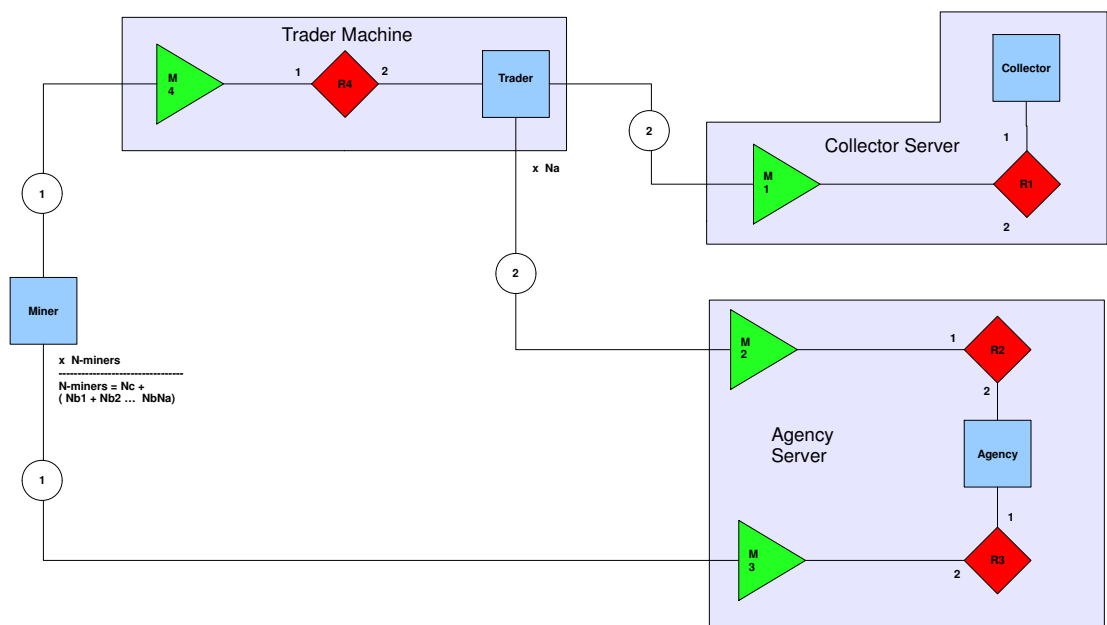
Traders hire miners for their mining operations. Traders have access to a search space within the mine (the complete mine area is subdivided between several traders) and the trader chooses the number and kind of miners based on its requirements. The trader directs the miners that are hired by the trader to search for specific minerals. The choice of target minerals keeps on changing with the changing prices of the minerals in the market (here the market is the collector). The trader regularly checks the mineral prices and adjusts its mining and staffing parameters accordingly. When a miner finds a mineral, it is directed to deposit it to the trader. The trader holds the mineral till a favourable price is available in the market (collector). The load of minerals sold to the collector by the trader is termed as the cargo and

the trader is paid for the cargo as per the prices advertised by the collector. The trader has to pay the miners their predefined salaries irrespective of the market situations and the amount of profits made. The trader may leave a miner when its profits are low, or hire more miners when it has enough funds to expand its mining operations.

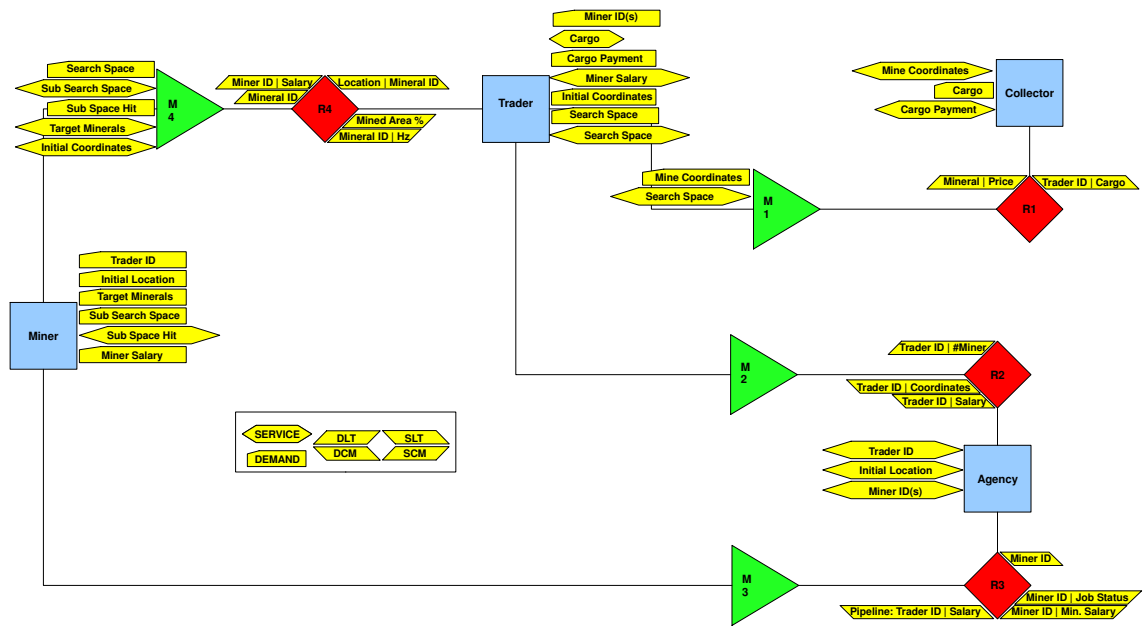
The collector is the market where the traders sell their cargo. The collector is also the owner of the mine, and gives several traders different parts of the mine for their mining operations. Based on their performance, the collector may reduce or expand the region of the mine where the trader is allowed to carry out its mining operations.

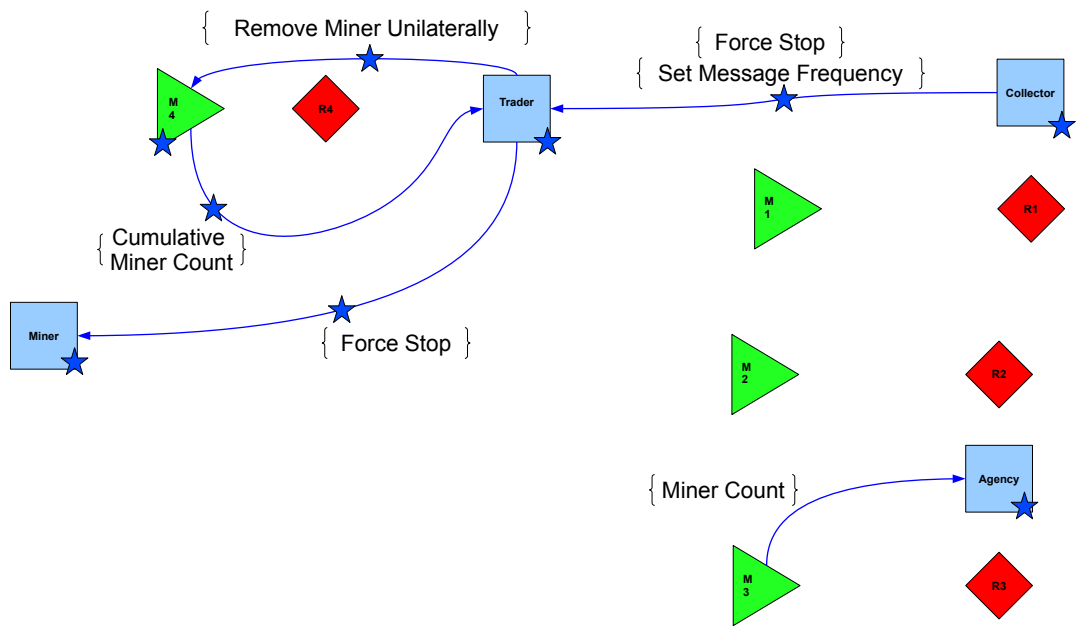
For easy implementation and operation of such a system, certain hyperactivity controls are added. The trader has the hyperactive control over removing a miner or stop a miners operation without the miners will. This may be required when a miner is required to be stopped or when it deviates from the predefined area of the mine where the trader is allowed to carry out its mining operations. The M4 merge can access the actual number of miners (cumulative miner count) from the miner as this will allow M4 to check the number of miners requesting to join the mining operation. This simple hyperactive data access to traders internal knowledge saves the M4 merge performing a number of additional checks whenever a new miner requests to join the trader as an employee. A similar hyperactive data access is also given to M3 merge to keep tract of the total number of miners in the ecosystem. The collector is given a hyperactive access over trader to set the frequency by which the trader is allowed to check the mineral prices in the market. This is to avoid excessive pinging by a trader to access the market prices (and probably blocking the access to the market data to other traders). The collector may force stop a trader is the trader expands its mining operations beyond the permissible area.





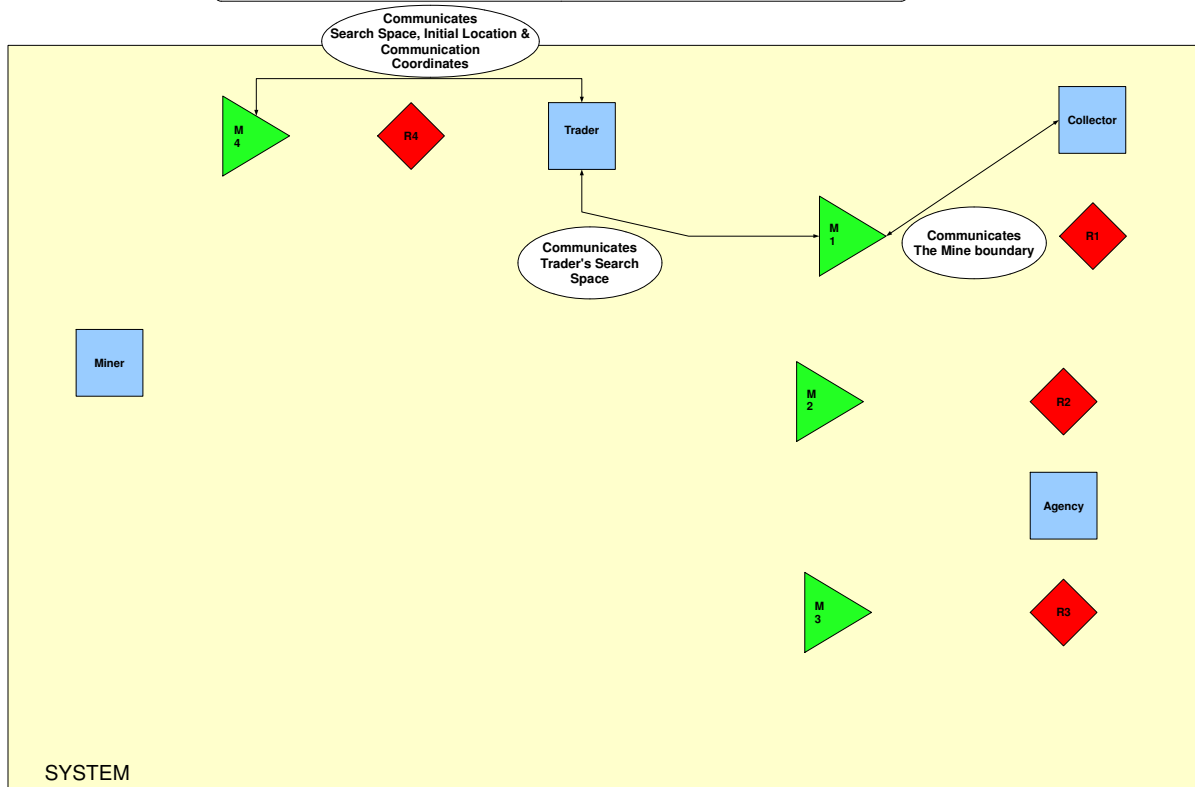
OBJECT MINERS PROJECT
TARC (Normalized)





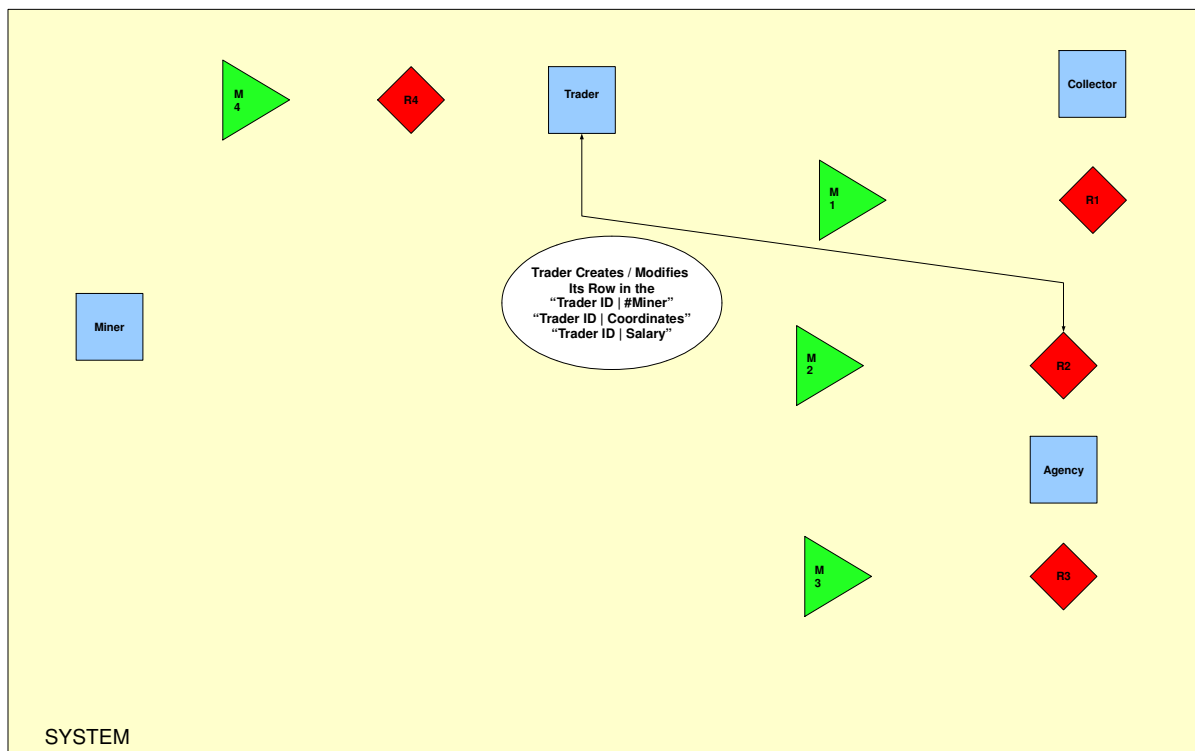
U-ARC	OBJECT MINERS PROJECT UARC 001 : Trader Gets Search Space
U-ARC	OBJECT MINERS PROJECT UARC 003 : Miners Register With Agency
U-ARC	OBJECT MINERS PROJECT UARC 005 : Collector Publishes Mineral Prices
U-ARC	OBJECT MINERS PROJECT UARC 007 : Active Miner Looking For Switching Jobs
U-ARC	OBJECT MINERS PROJECT UARC 009 : Trader Initiates Trade With Collector
U-ARC	OBJECT MINERS PROJECT UARC 011 : Trader Re-accesses Production
U-ARC	OBJECT MINERS PROJECT UARC 013 : Trader Requests Additional Miners
U-ARC	OBJECT MINERS PROJECT UARC 015 : Agency Pipelines Trader Request For Miners
U-ARC	OBJECT MINERS PROJECT UARC 017 : Agency Manages Pipeline
U-ARC	OBJECT MINERS PROJECT UARC 019 : Trader Dispatches Miner Salaries

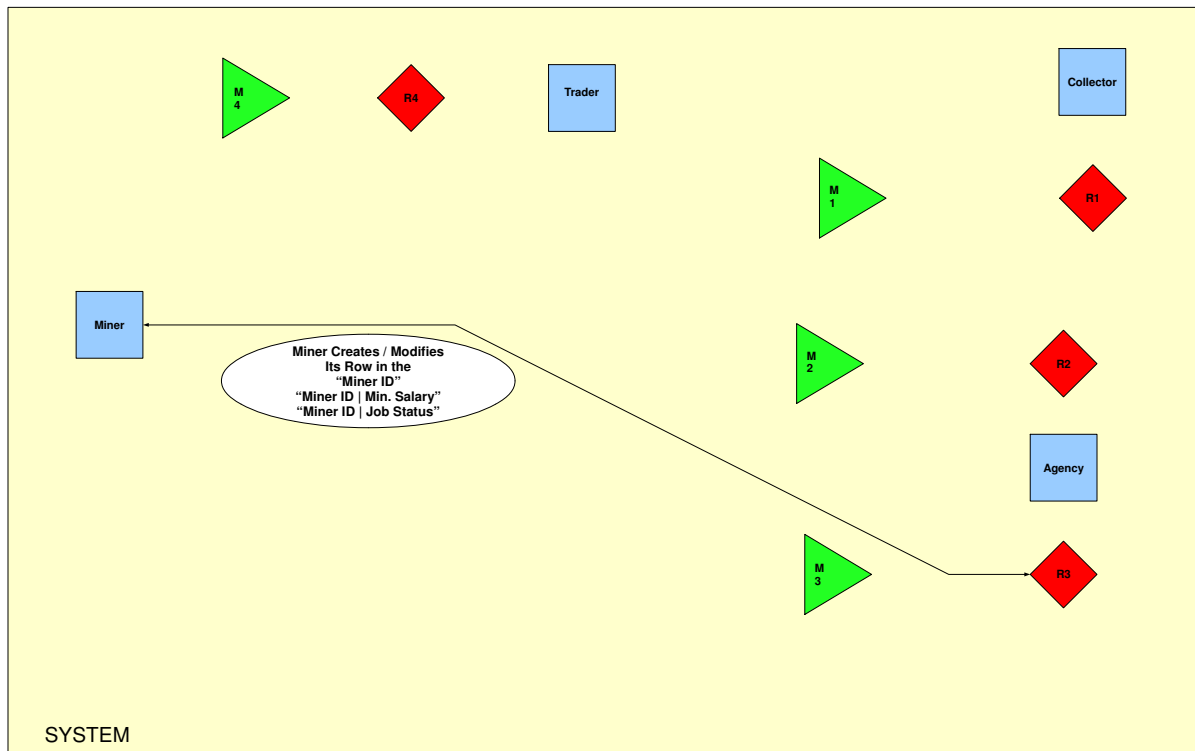
U-ARC	OBJECT MINERS PROJECT UARC 002 : Trader Requests Miners
U-ARC	OBJECT MINERS PROJECT UARC 004 : Trader is Allocated Miners
U-ARC	OBJECT MINERS PROJECT UARC 006 : Miners Registers With Trader
U-ARC	OBJECT MINERS PROJECT UARC 008 : Mining Process
U-ARC	OBJECT MINERS PROJECT UARC 010 : Collector Accepts Cargo
U-ARC	OBJECT MINERS PROJECT UARC 012 : Trader Releases Miners
U-ARC	OBJECT MINERS PROJECT UARC 014 : Agency Releases Additional Miners
U-ARC	OBJECT MINERS PROJECT UARC 016 : Miner Leaves Trader
U-ARC	OBJECT MINERS PROJECT UARC 018 : Collector Redefines Minerals And Prices





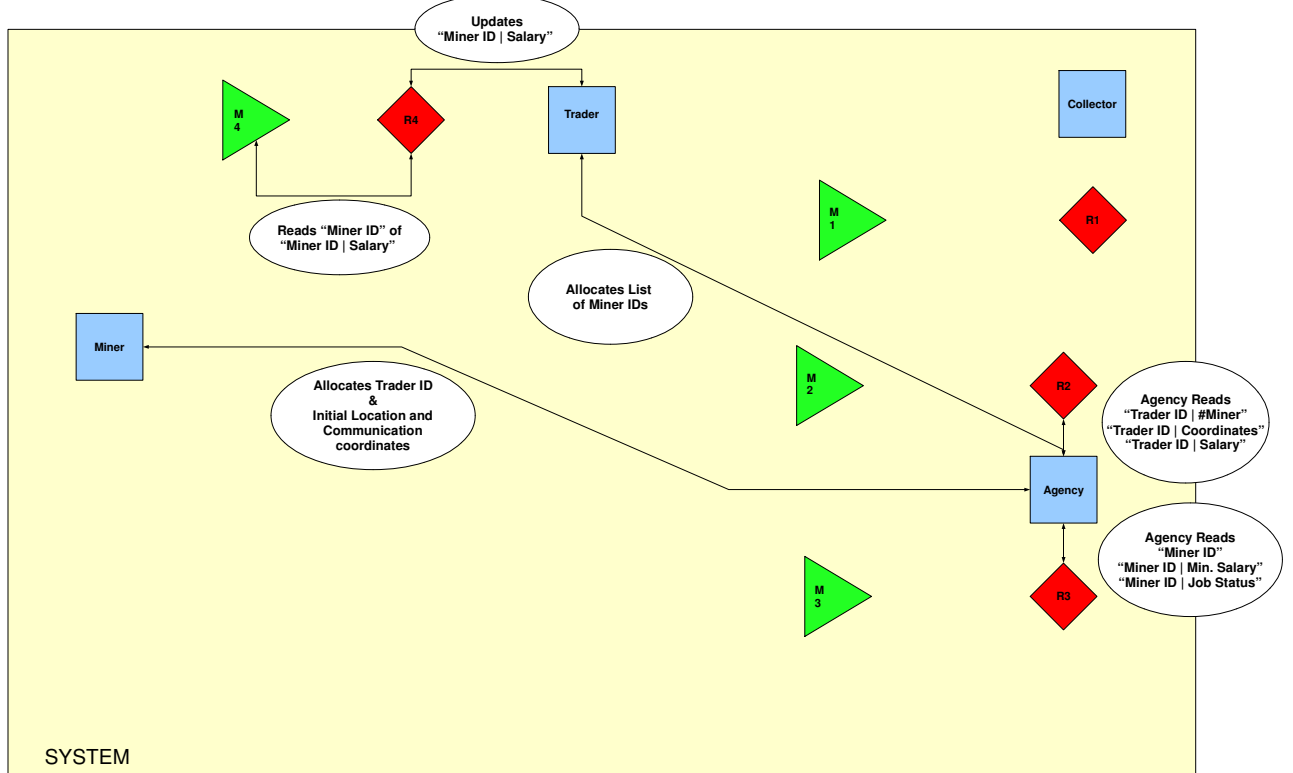
OBJECT MINERS PROJECT
UARC 002 : Trader Requests Miners





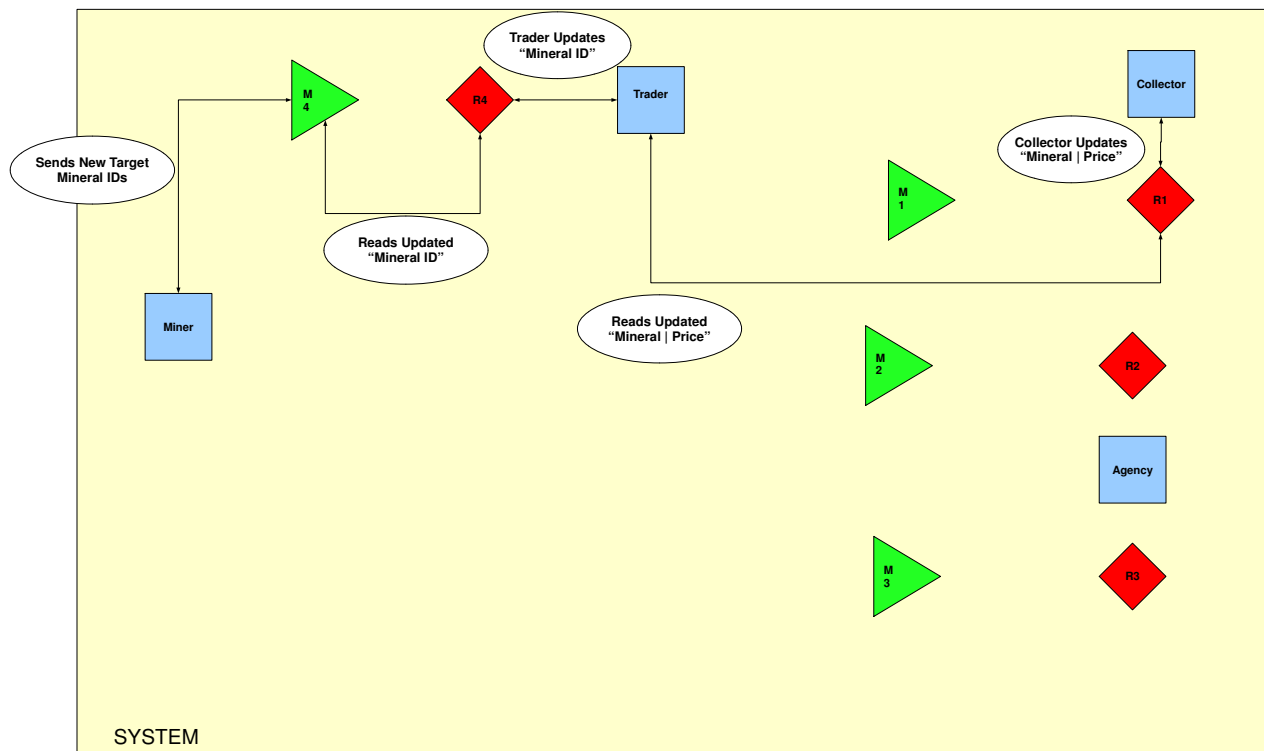


OBJECT MINERS PROJECT
UARC 004 : Trader is Allocated Miners



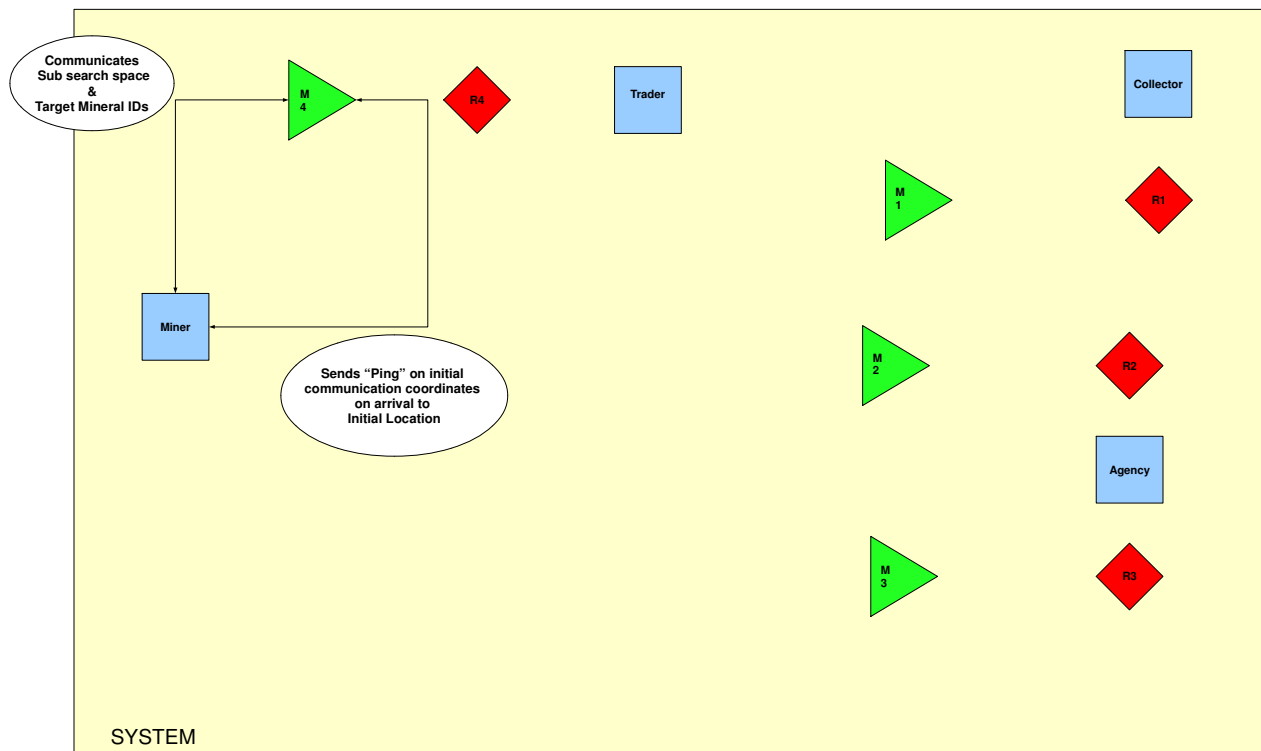


OBJECT MINERS PROJECT
UARC 005 : Collector Publishes Mineral Prices



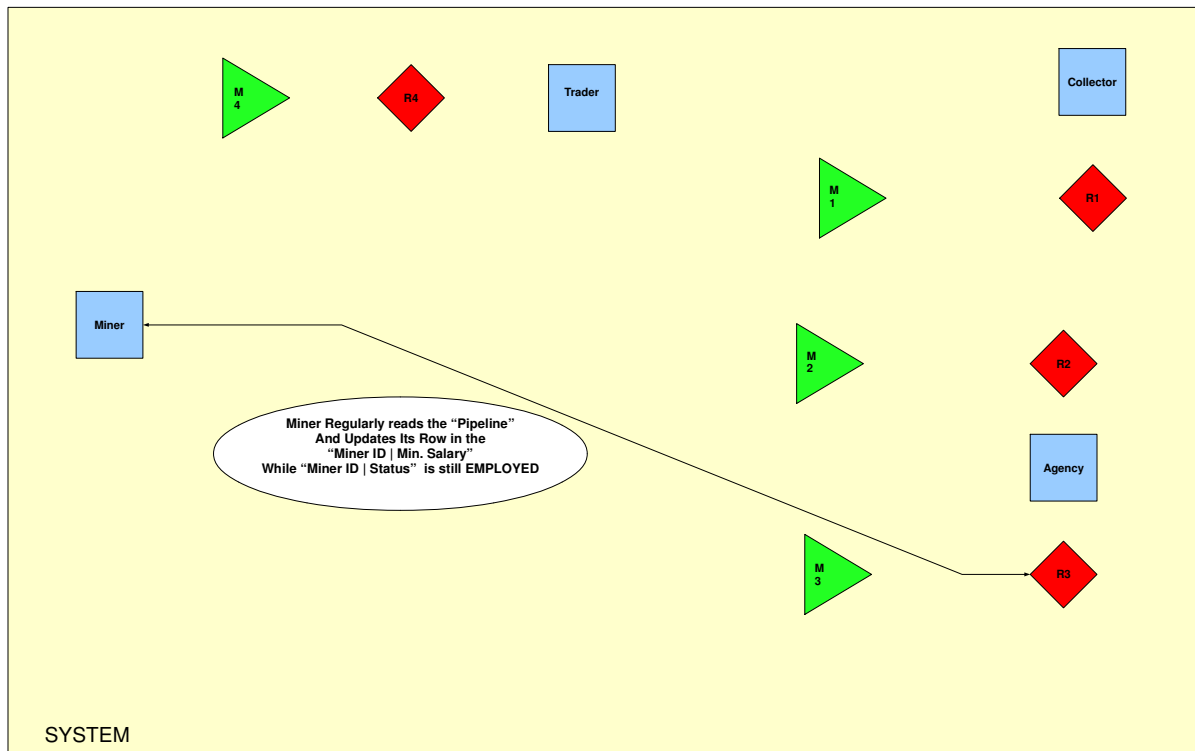


OBJECT MINERS PROJECT
UARC 006 : Miners Registers With Trader

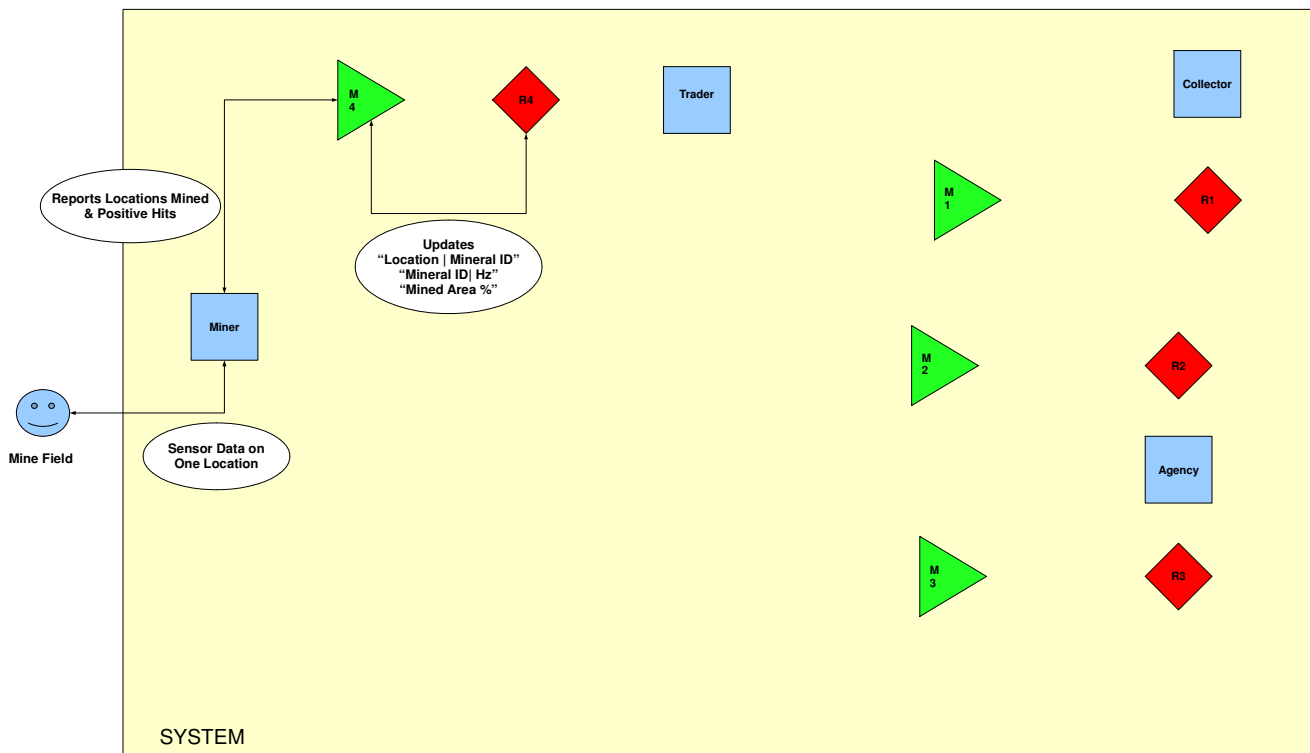




OBJECT MINERS PROJECT
UARC 007 : Active Miner Looking For Switching Jobs

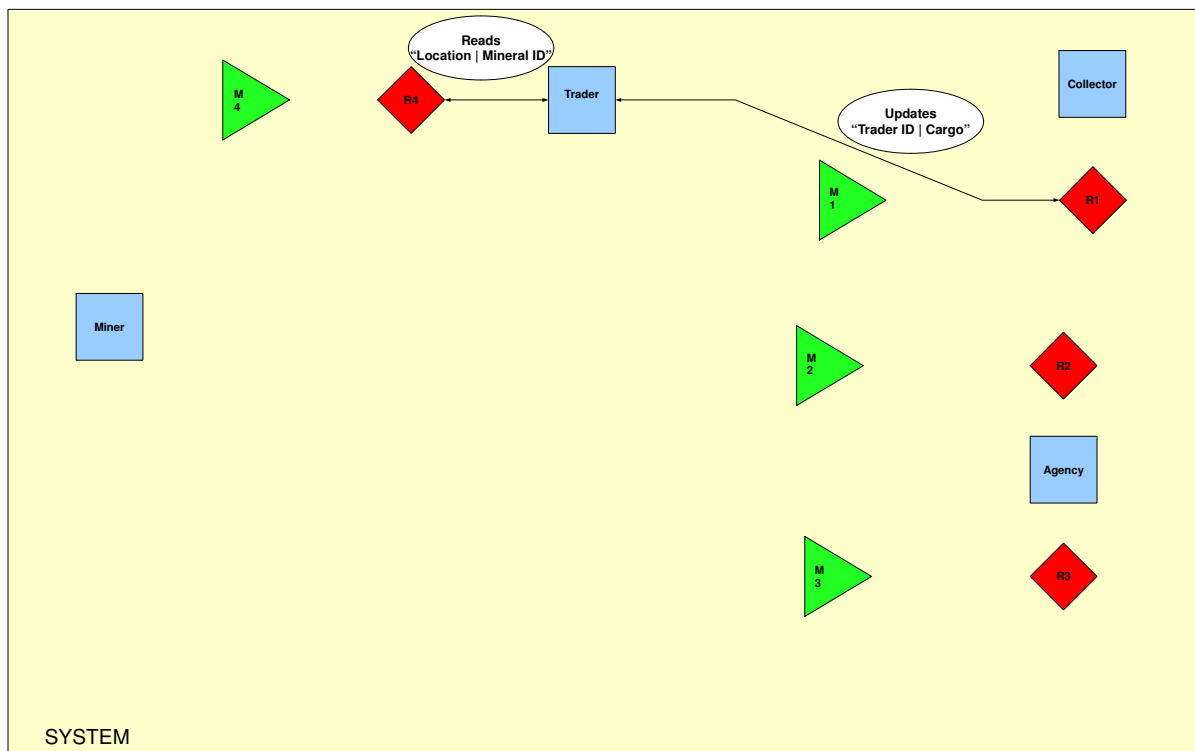


Assumption: The miner may / may not have loyalty to any one Trader.
Depending on its Personality Program. It may stay with the trader or leave as soon as a better job is offered by another trader.



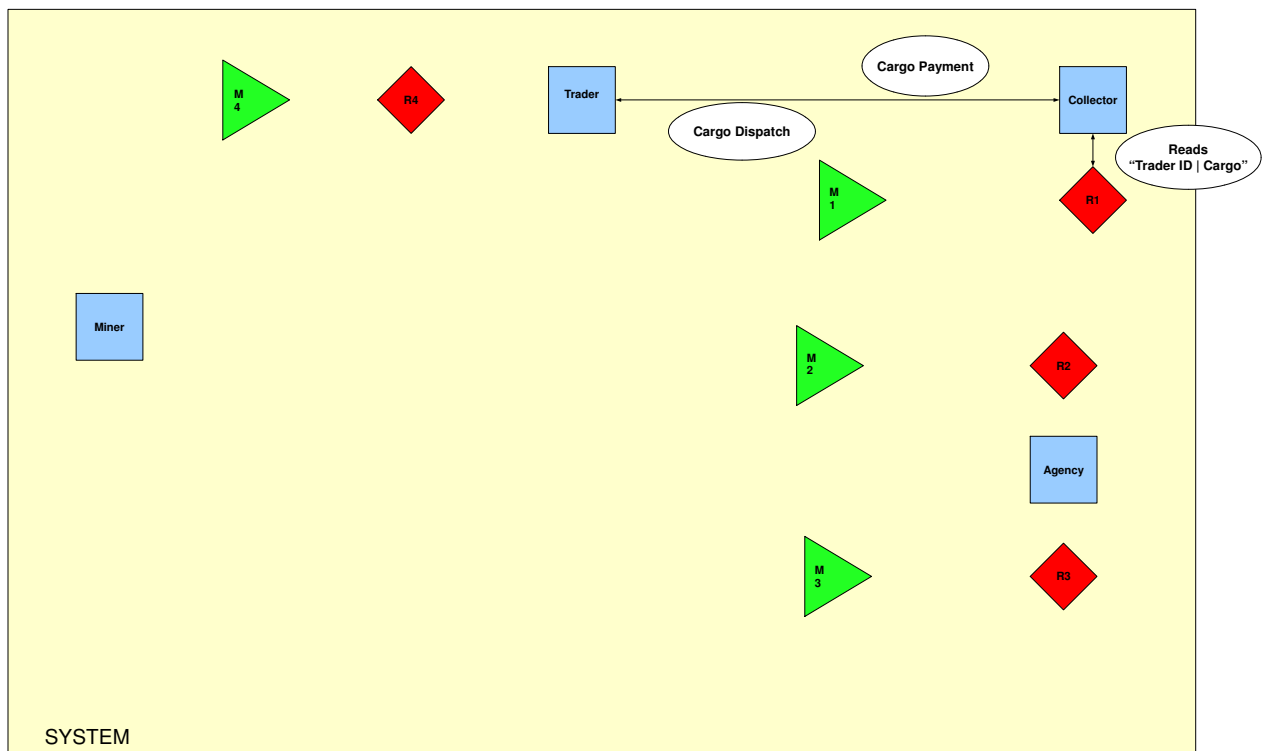


OBJECT MINERS PROJECT
UARC 009 : Trader Initiates Trade With Collector



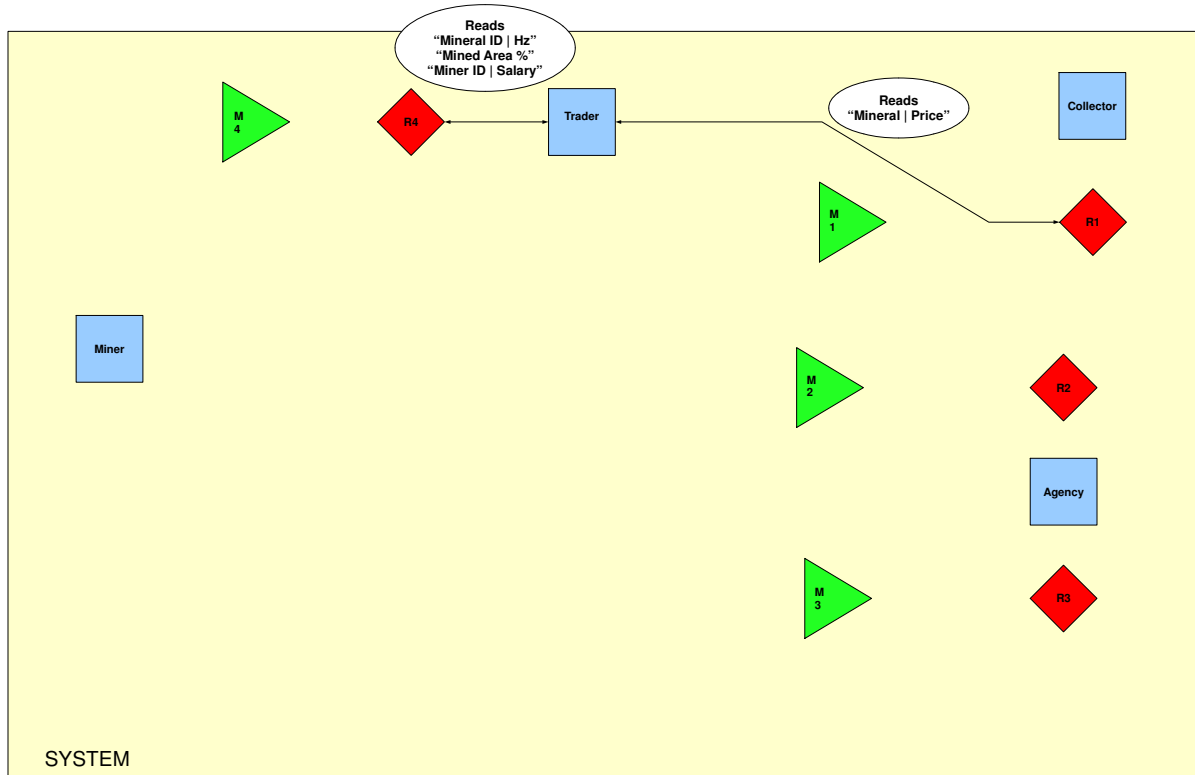


OBJECT MINERS PROJECT
UARC 010 : Collector Accepts Cargo



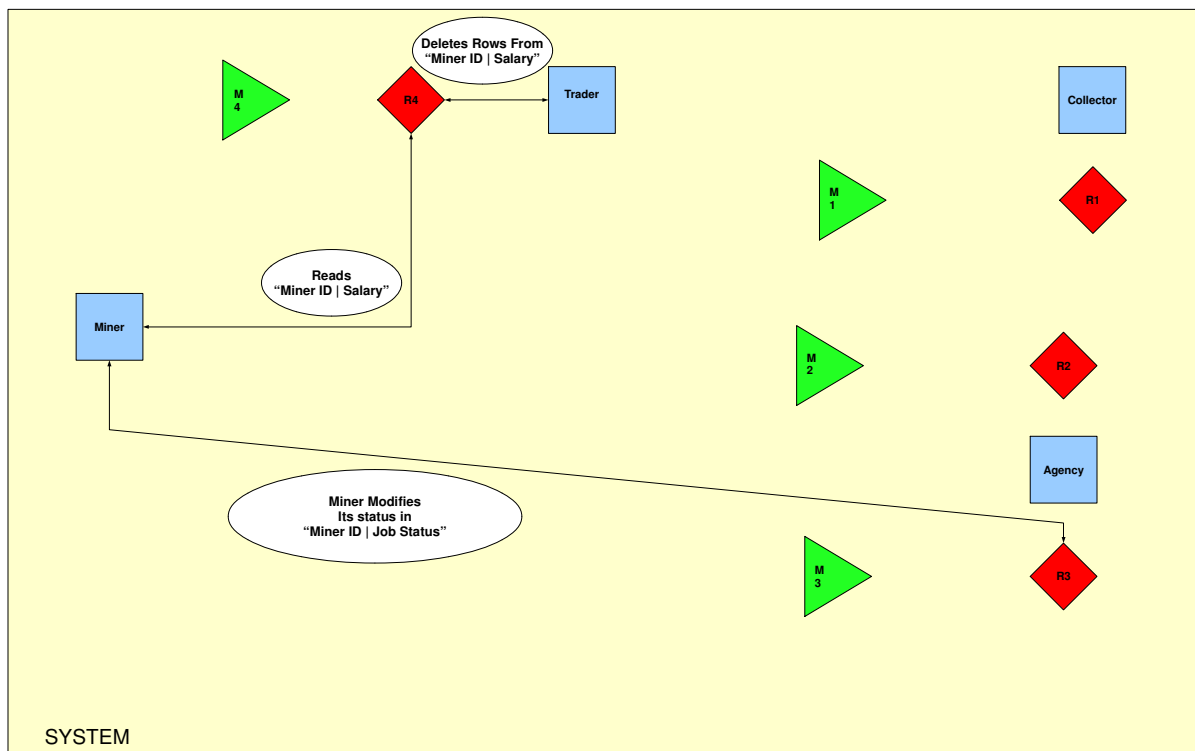


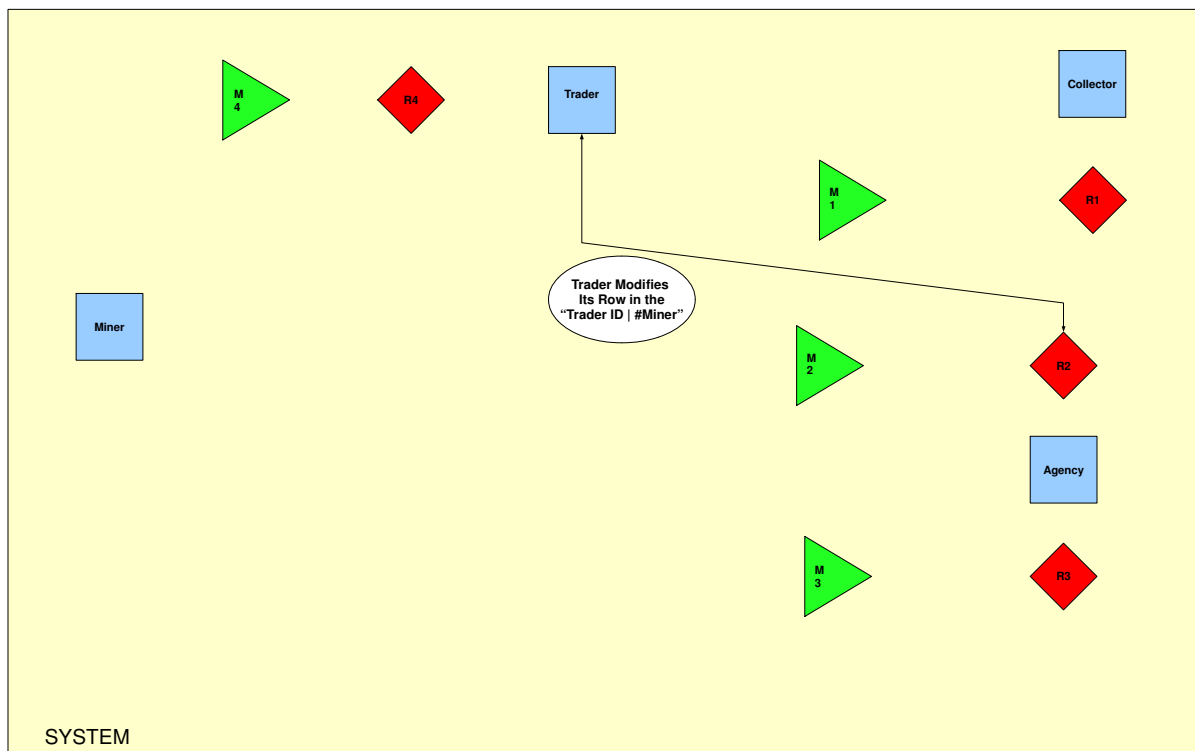
OBJECT MINERS PROJECT
UARC 011 : Trader Re-accesses Production





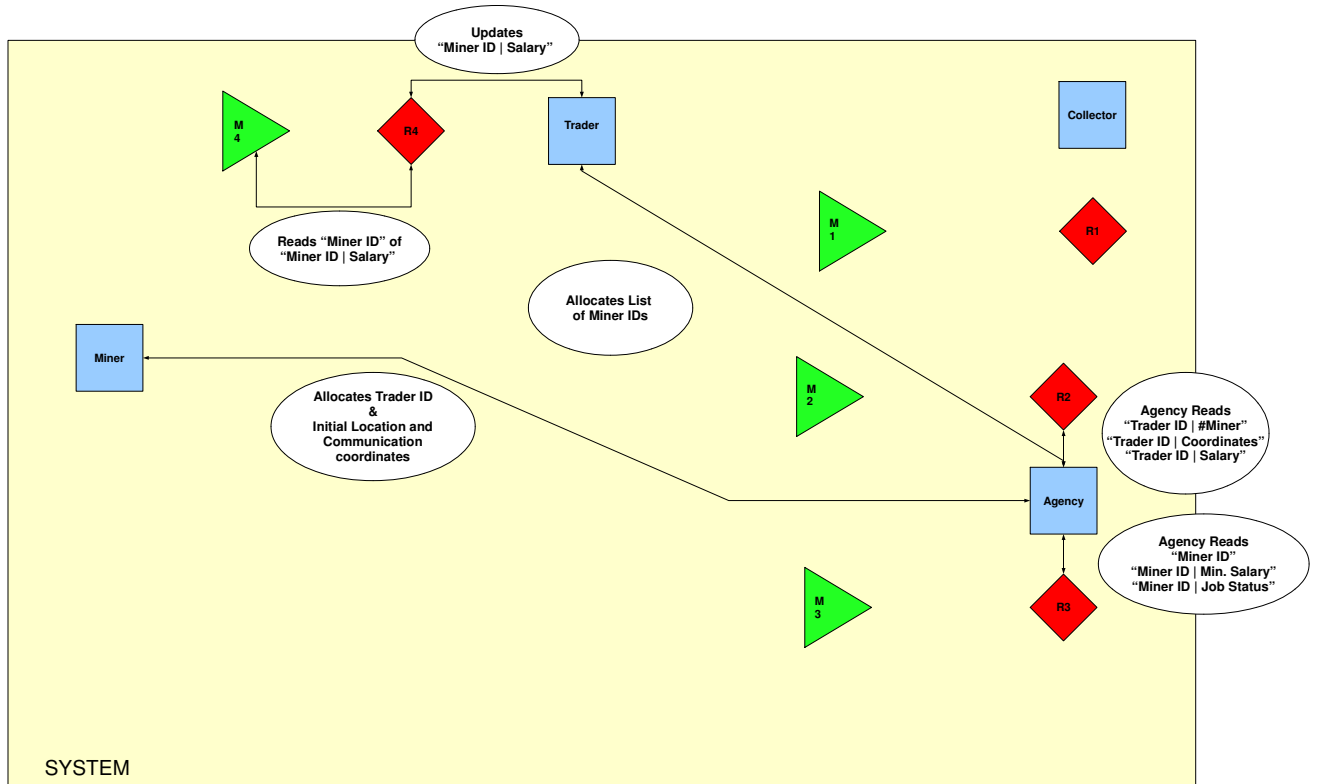
OBJECT MINERS PROJECT
UARC 012 : Trader Releases Miners

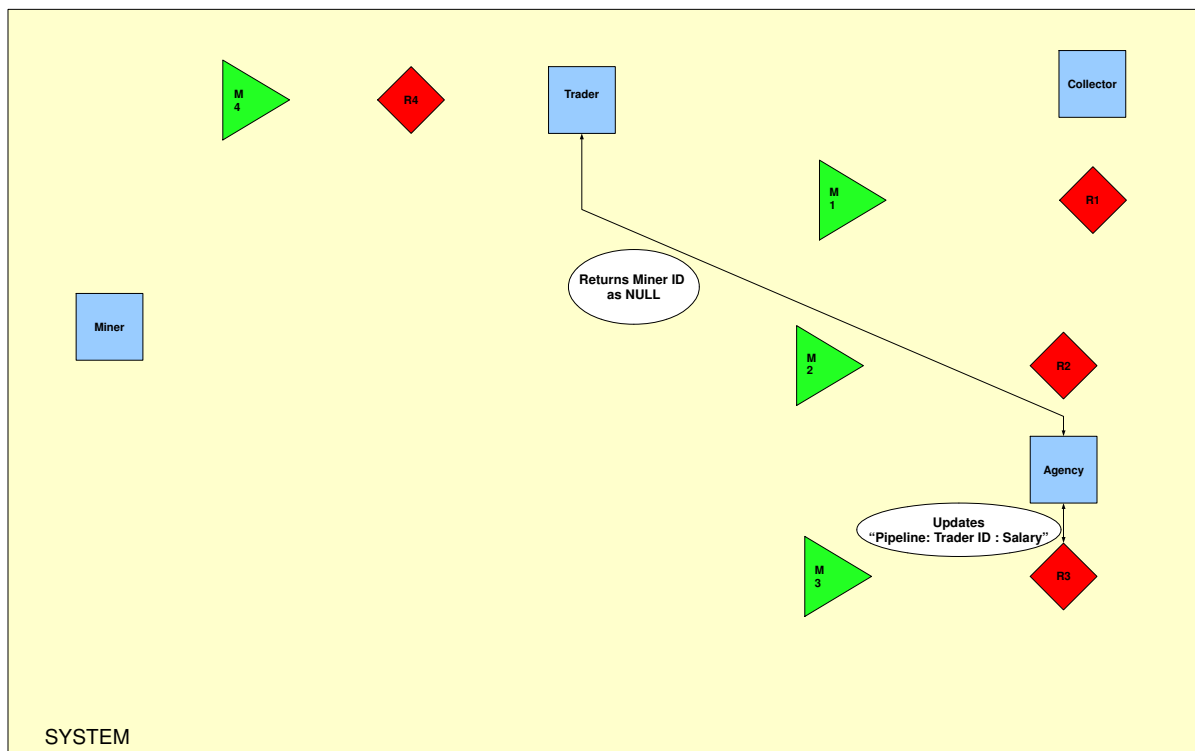




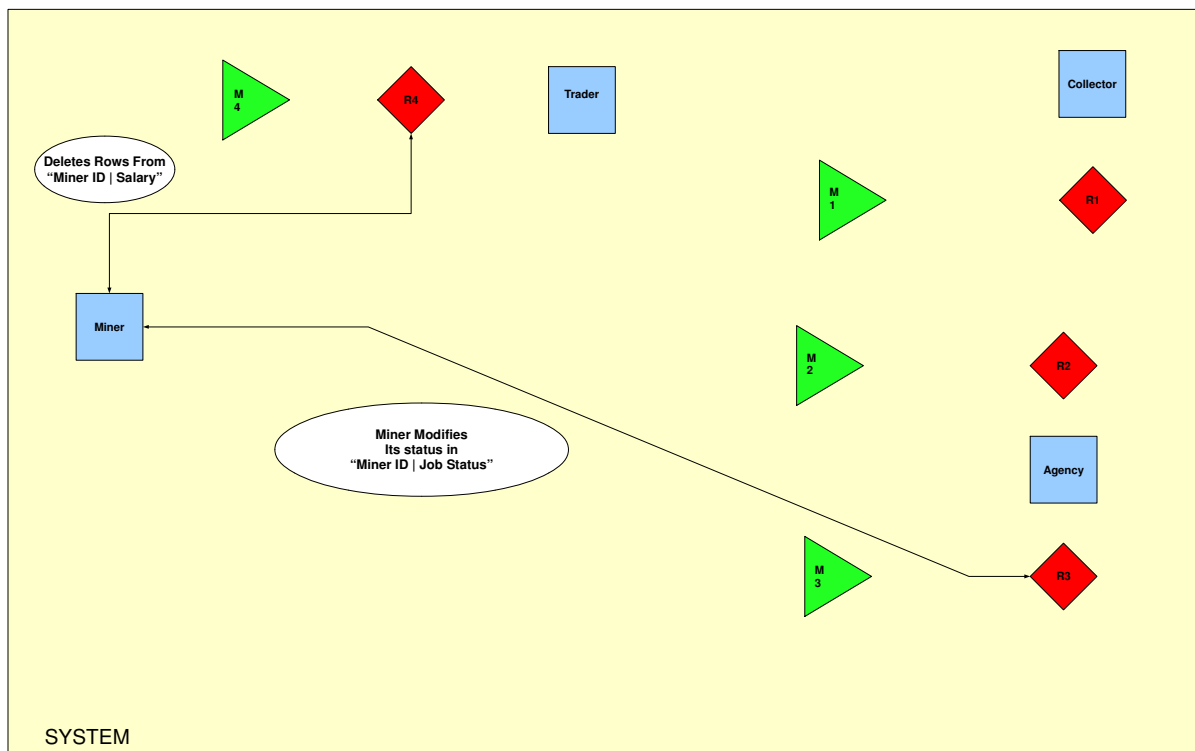


OBJECT MINERS PROJECT
UARC 014 : Agency Releases Additional Miners



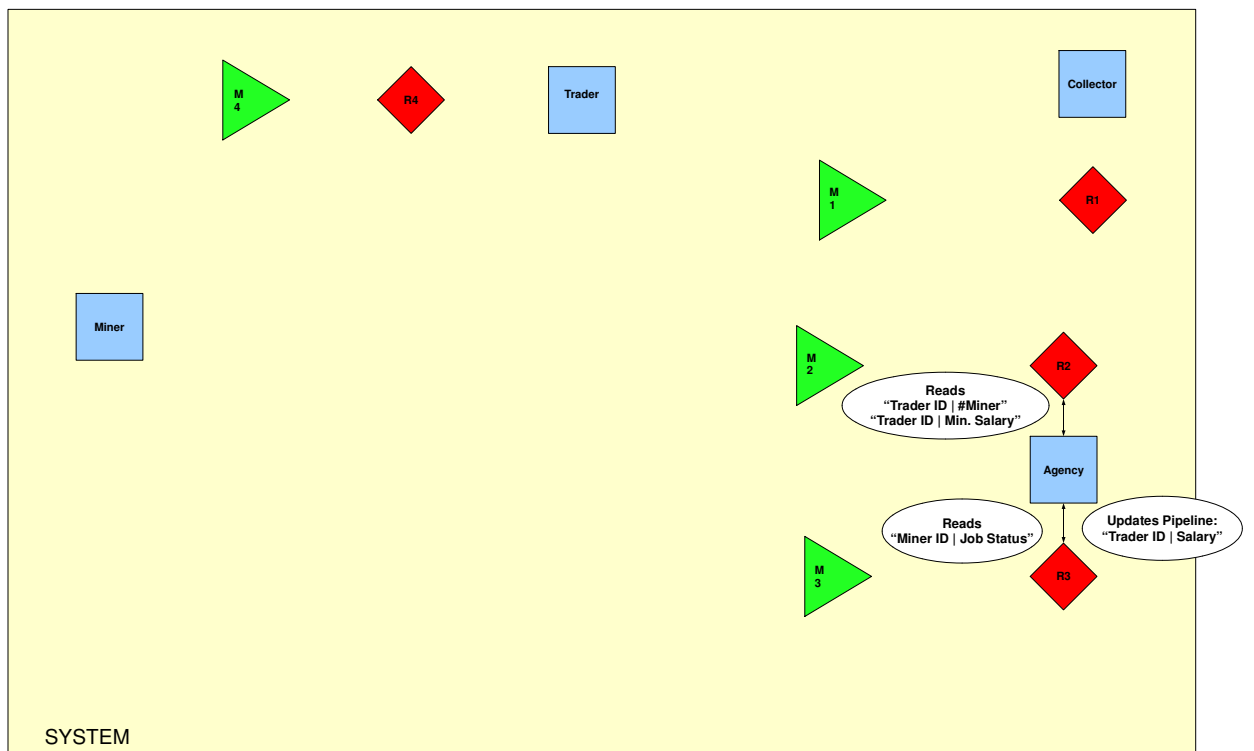


On receiving a NULL Miner ID, an active Trader may Increase the offered Salary based on its Business logic.





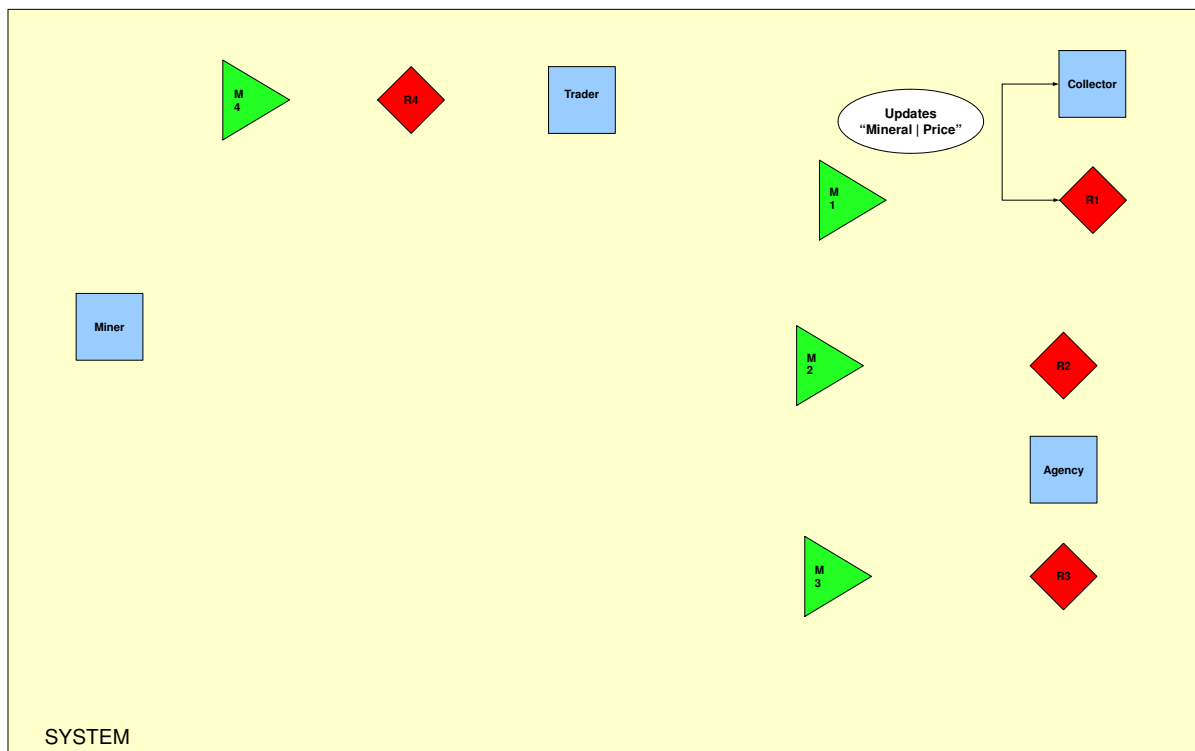
OBJECT MINERS PROJECT
UARC 017 : Agency Manages Pipeline



The Pipeline Updates are the trigger for Miners to switch their current job with a new one.
Assumption: A Miner cannot switch to a new job offered by its current Trader. Hence it will not react to jobs
In the pipeline from its current Trader. [Though this scenario can be explored]

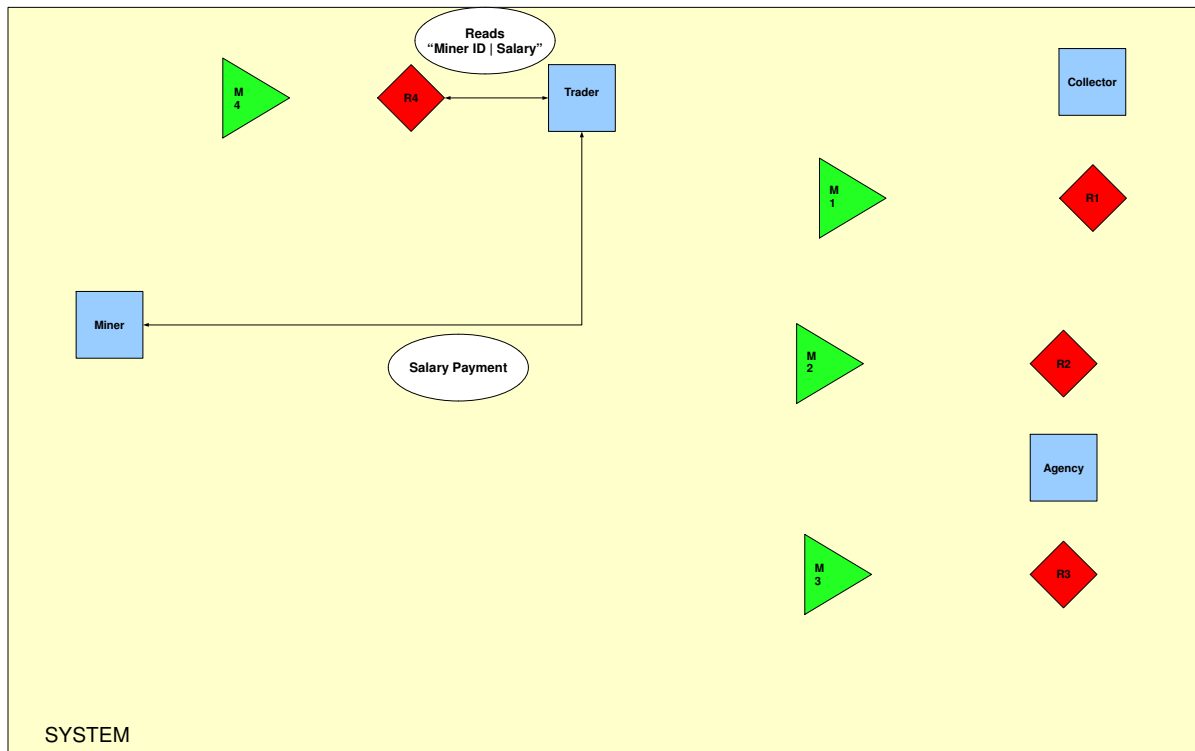


Stories of human like Agent behaviour can be published. What if Miners stop working for a Trader is offering too much to new Miners, will there be a Strike? (Means employed but not working for a salary hike)









OBJECT MINERS PROJECT
UARC 019 : Trader Dispatches Miner Salaries



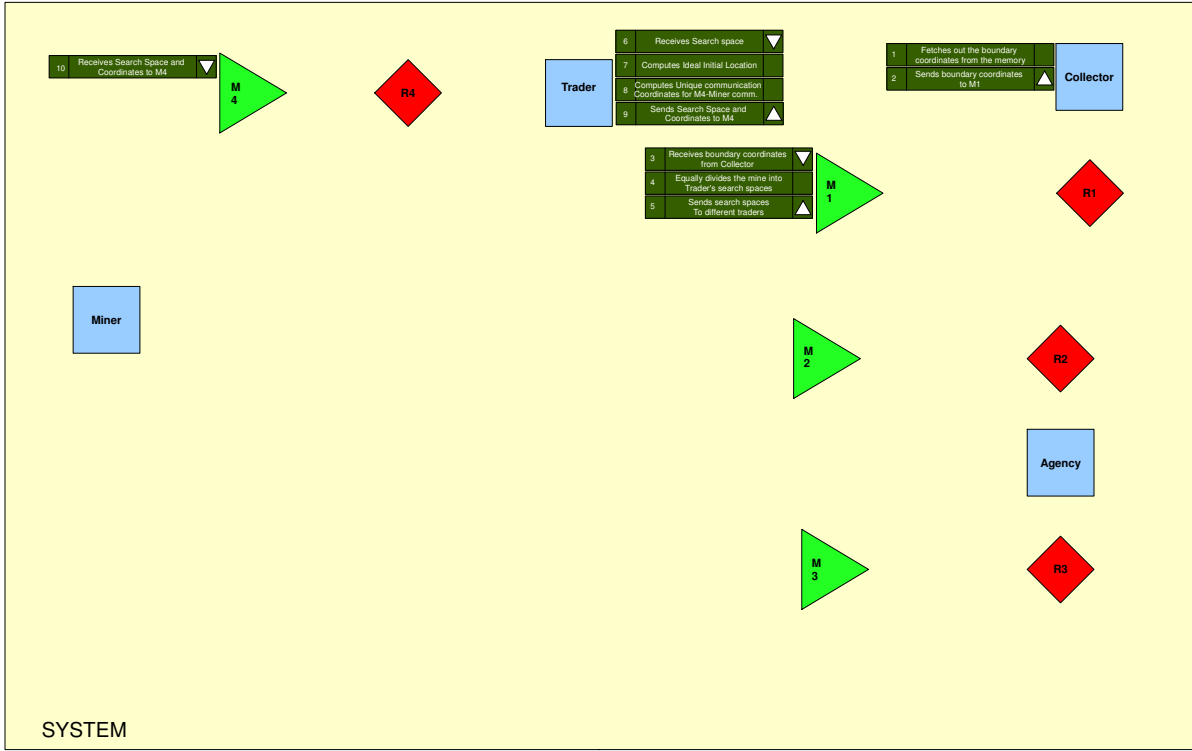
A retirement Scenario can be explored where after saving enough money a Miner decides to Retire.

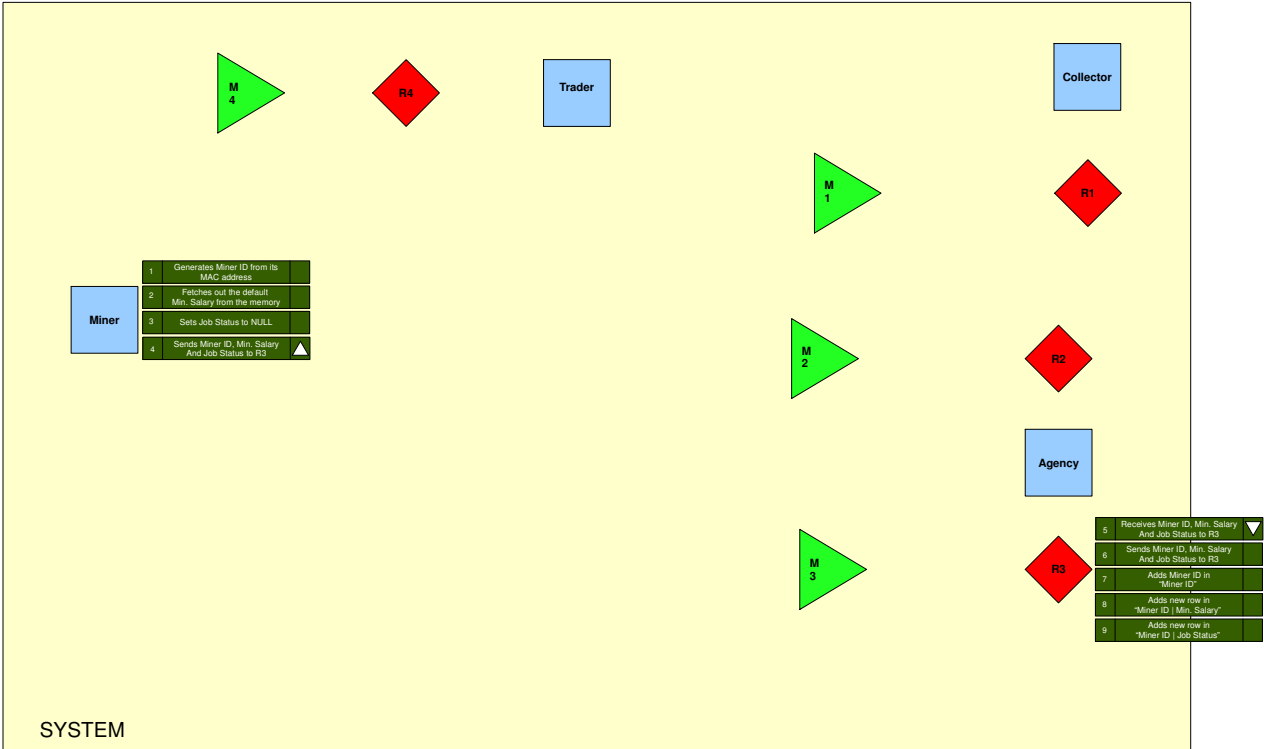
	OBJECT MINERS PROJECT SARC 001 : Trader Gets Search Space
	OBJECT MINERS PROJECT SARC 003 : Miners Register With Agency
	OBJECT MINERS PROJECT SARC 005 : Collector Publishes Mineral Prices
	OBJECT MINERS PROJECT SARC 007 : Active Miner Looking For Switching Jobs
	OBJECT MINERS PROJECT SARC 009 : Trader Initiates Trade With Collector
	OBJECT MINERS PROJECT SARC 011 : Trader Re-accesses Production
	OBJECT MINERS PROJECT SARC 013 : Trader Requests Additional Miners
	OBJECT MINERS PROJECT SARC 015 : Agency Pipelines Trader Request For Miners
	OBJECT MINERS PROJECT SARC 017 : Agency Manages Pipeline
	OBJECT MINERS PROJECT SARC 019 : Trader Dispatches Miner Salaries

	OBJECT MINERS PROJECT SARC 002 : Trader Requests Miners
	OBJECT MINERS PROJECT SARC 004 : Trader is Allocated Miners
	OBJECT MINERS PROJECT SARC 006 : Miners Registers With Trader
	OBJECT MINERS PROJECT SARC 008 : Mining Process
	OBJECT MINERS PROJECT SARC 010 : Collector Accepts Cargo
	OBJECT MINERS PROJECT SARC 012 : Trader Releases Miners
	OBJECT MINERS PROJECT SARC 014 : Agency Releases Additional Miners to Trader
	OBJECT MINERS PROJECT SARC 016 : Miner Leaves Trader
	OBJECT MINERS PROJECT SARC 018 : Collector Redefines Minerals And Prices



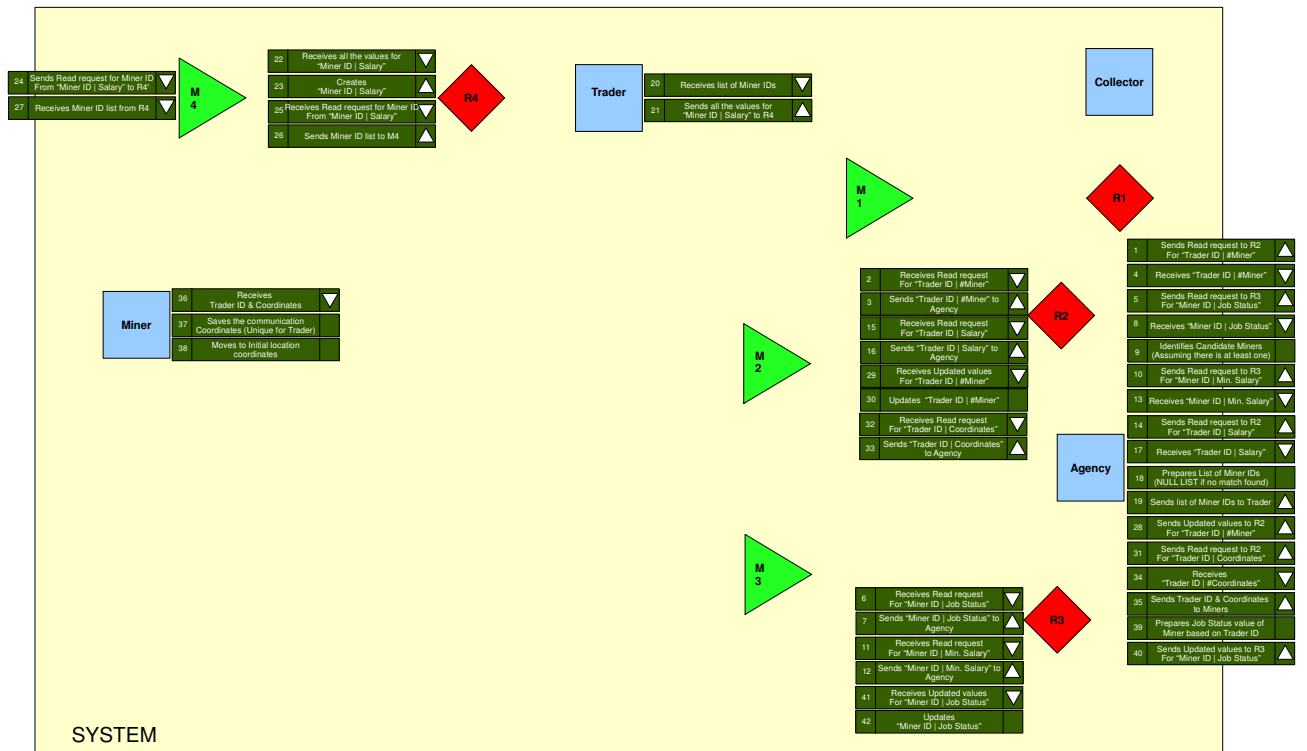
OBJECT MINERS PROJECT
SARC 001 : Trader Gets Search Space





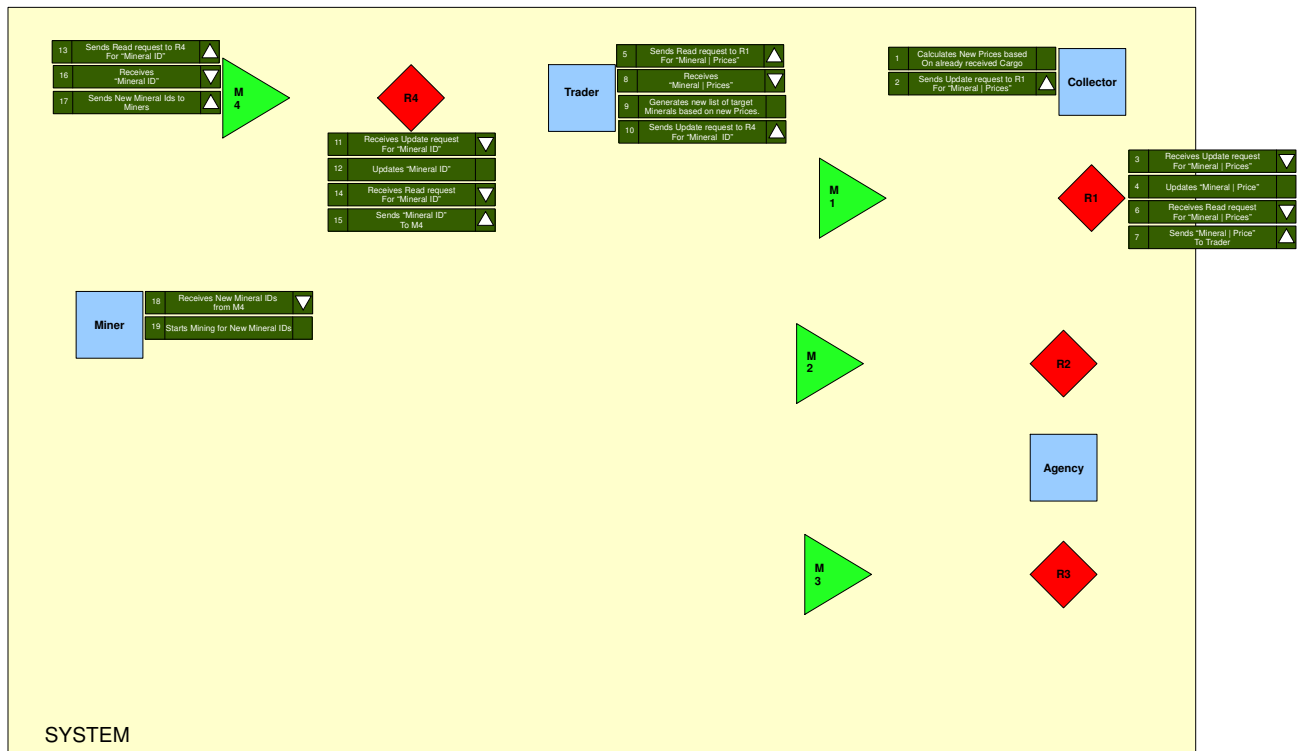


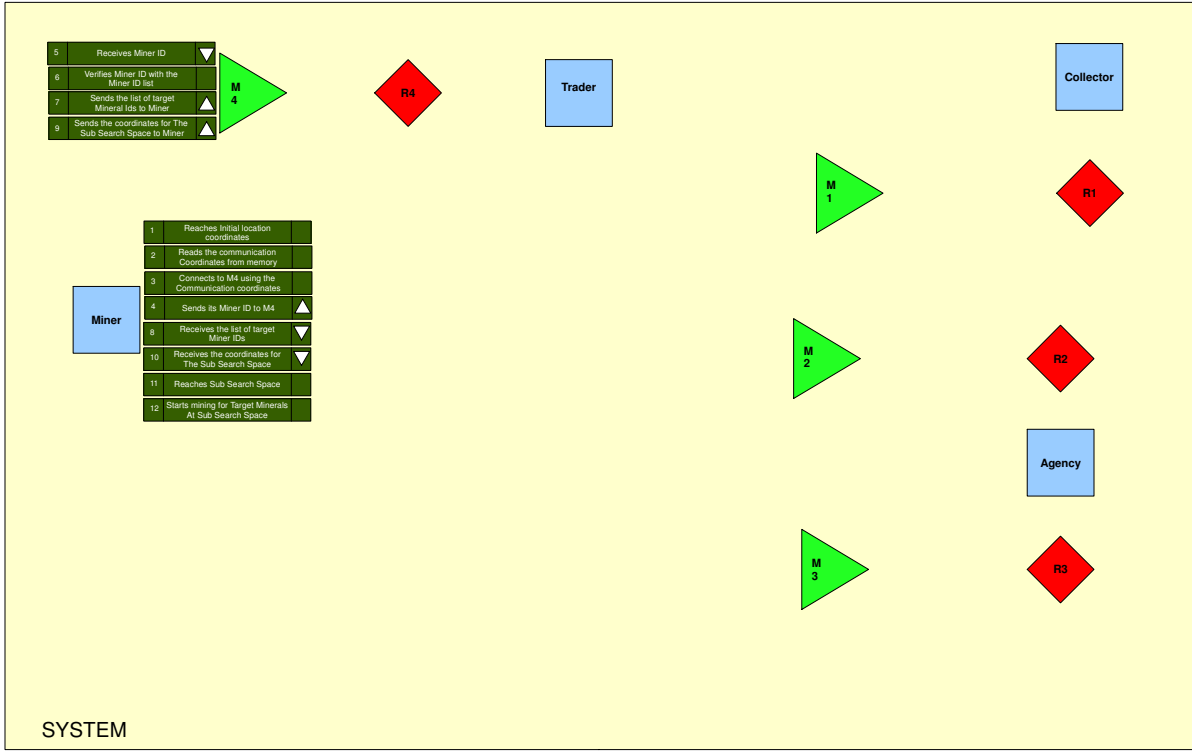
OBJECT MINERS PROJECT
SARC 004 : Trader is Allocated Miners





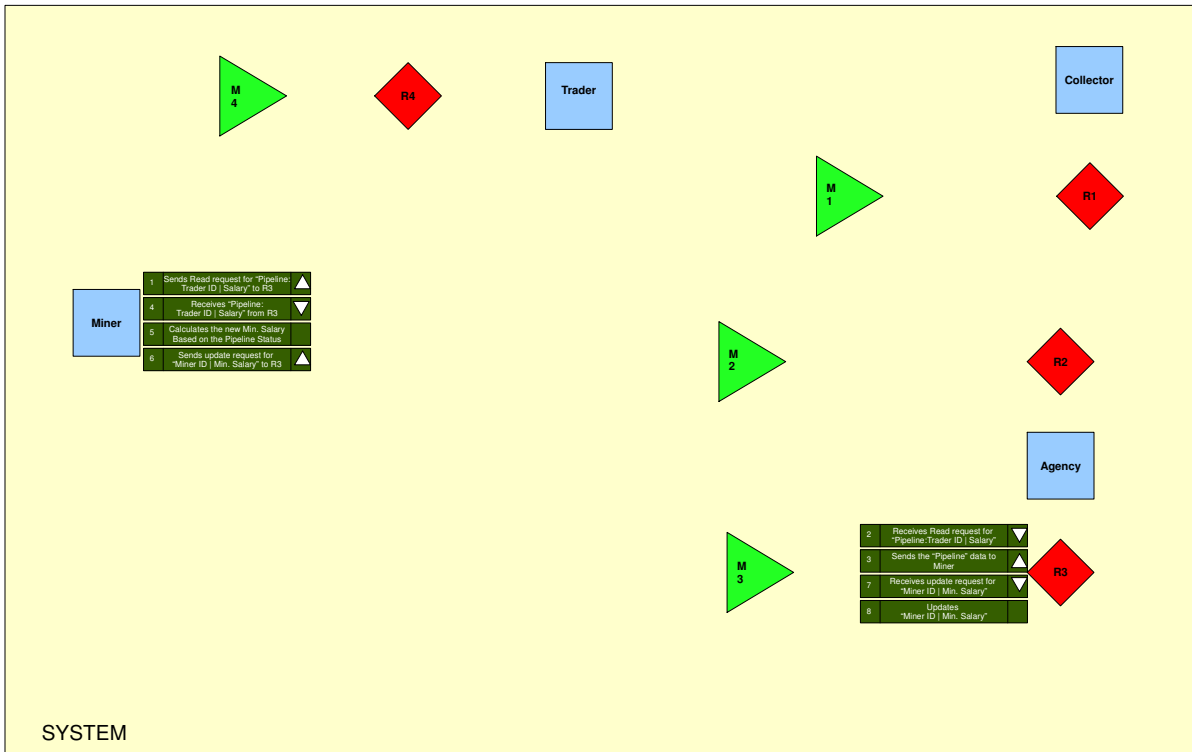
OBJECT MINERS PROJECT
SARC 005 : Collector Publishes Mineral Prices







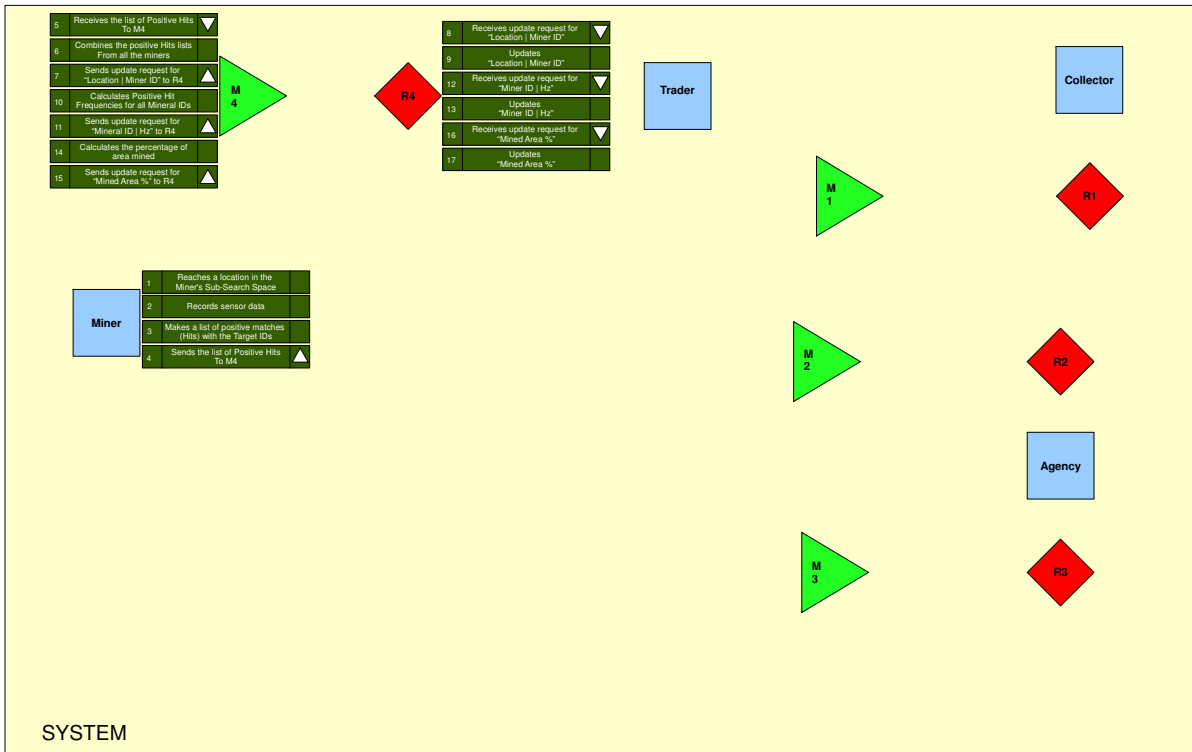
OBJECT MINERS PROJECT
SARC 007 : Active Miner Looking For Switching Jobs



Assumption: The miner may / may not have loyalty to any one Trader.
Depending on its Personality Program. It may stay with the trader or leave as soon as a better job is offered by another trader.

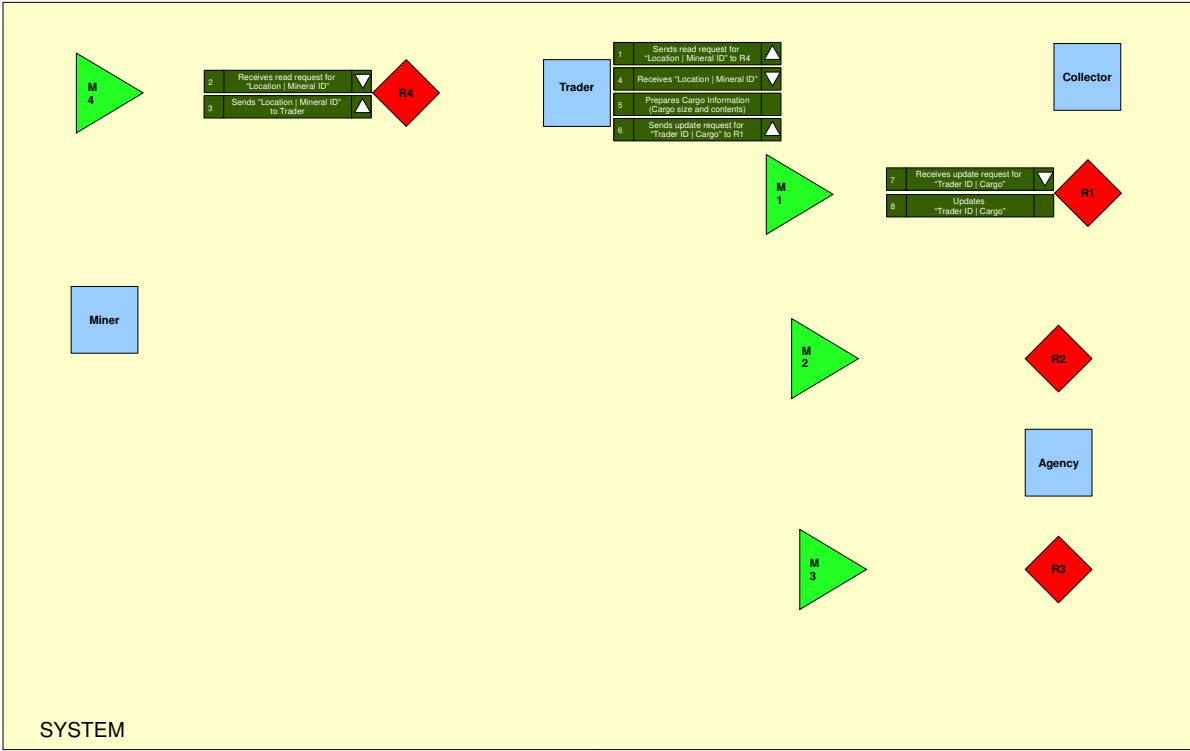


OBJECT MINERS PROJECT
SARC 008 : Mining Process



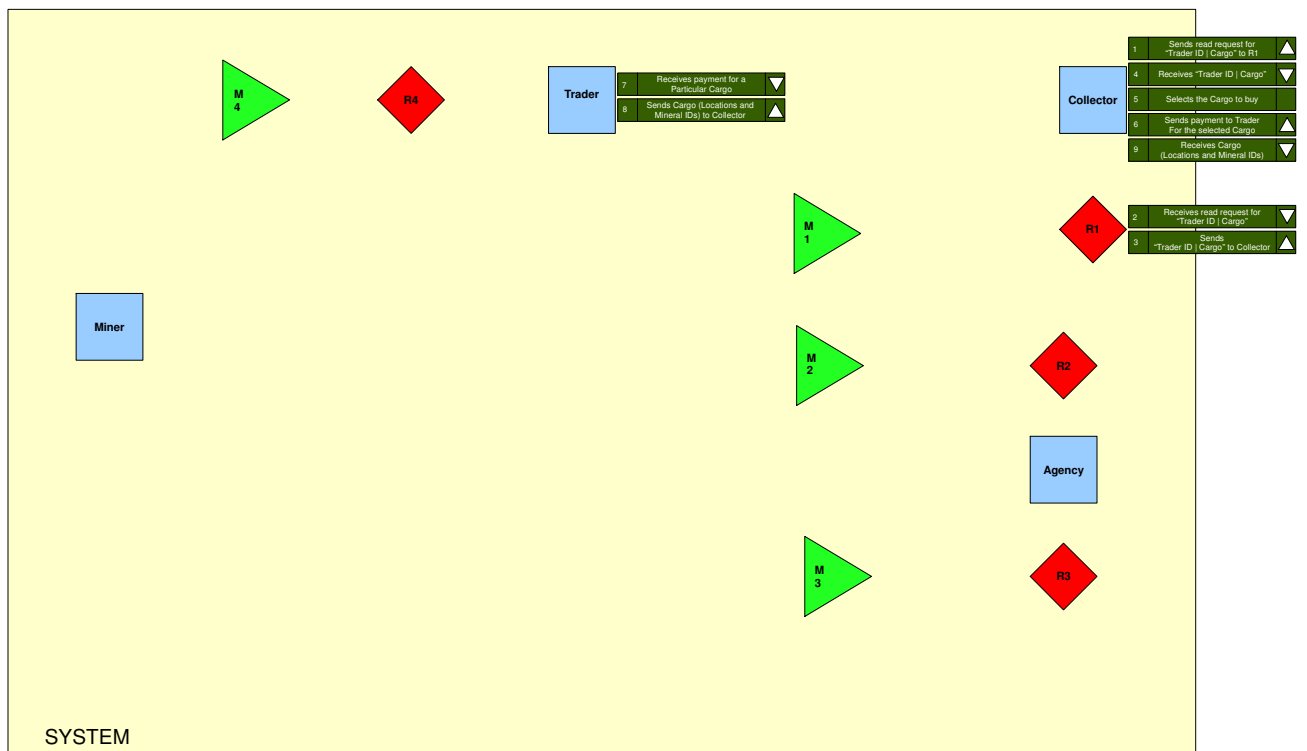


OBJECT MINERS PROJECT
SARC 009 : Trader Initiates Trade With Collector



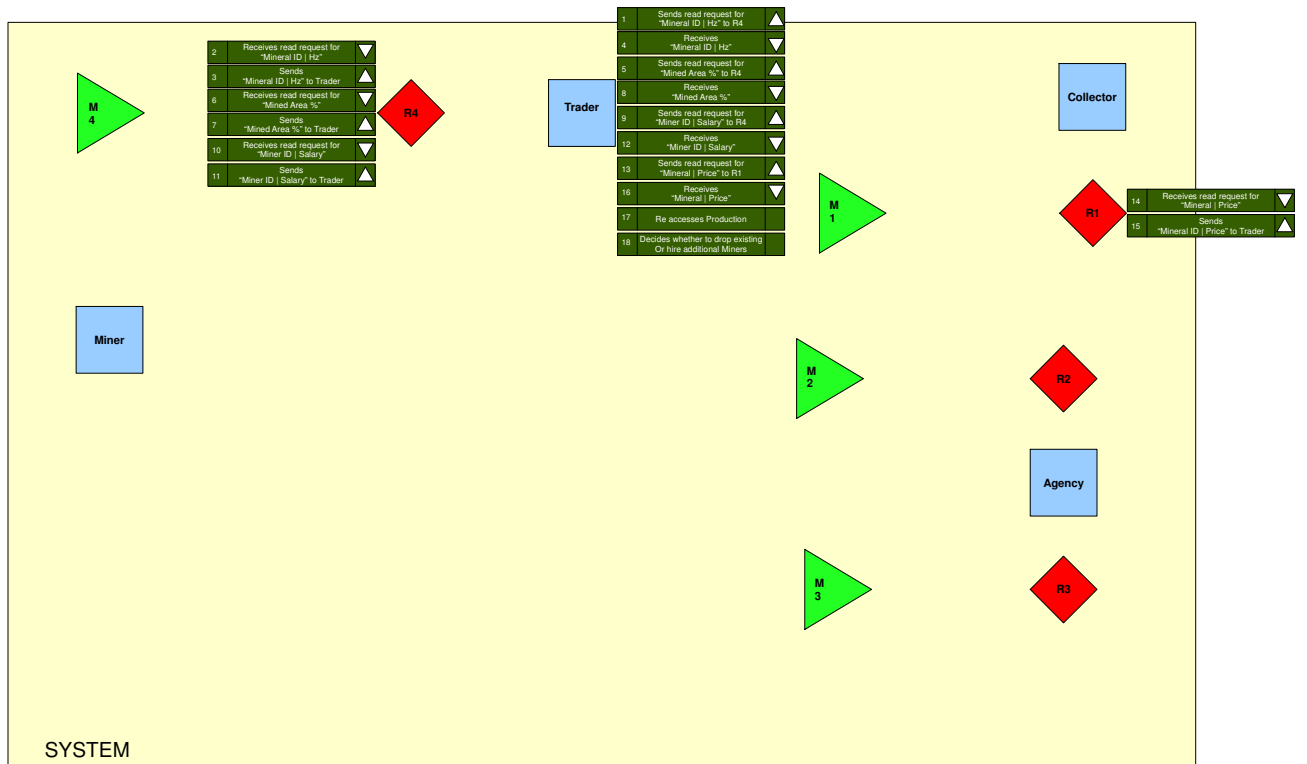


OBJECT MINERS PROJECT
SARC 010 : Collector Accepts Cargo



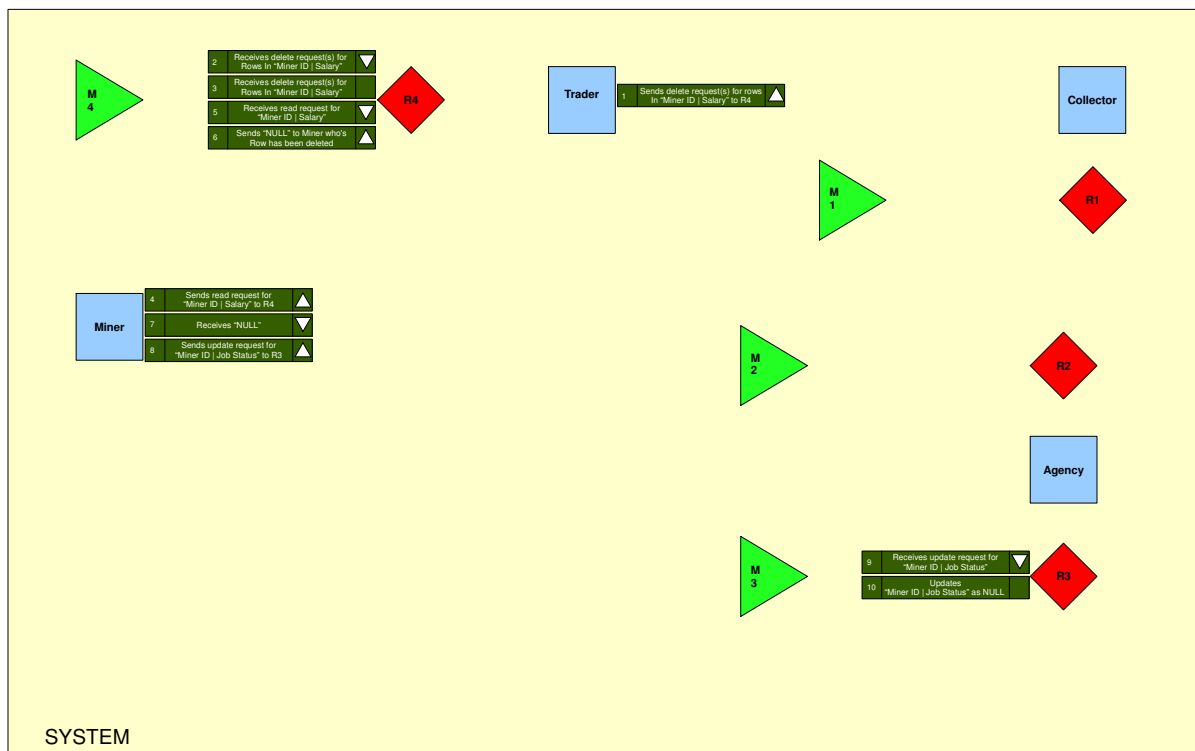


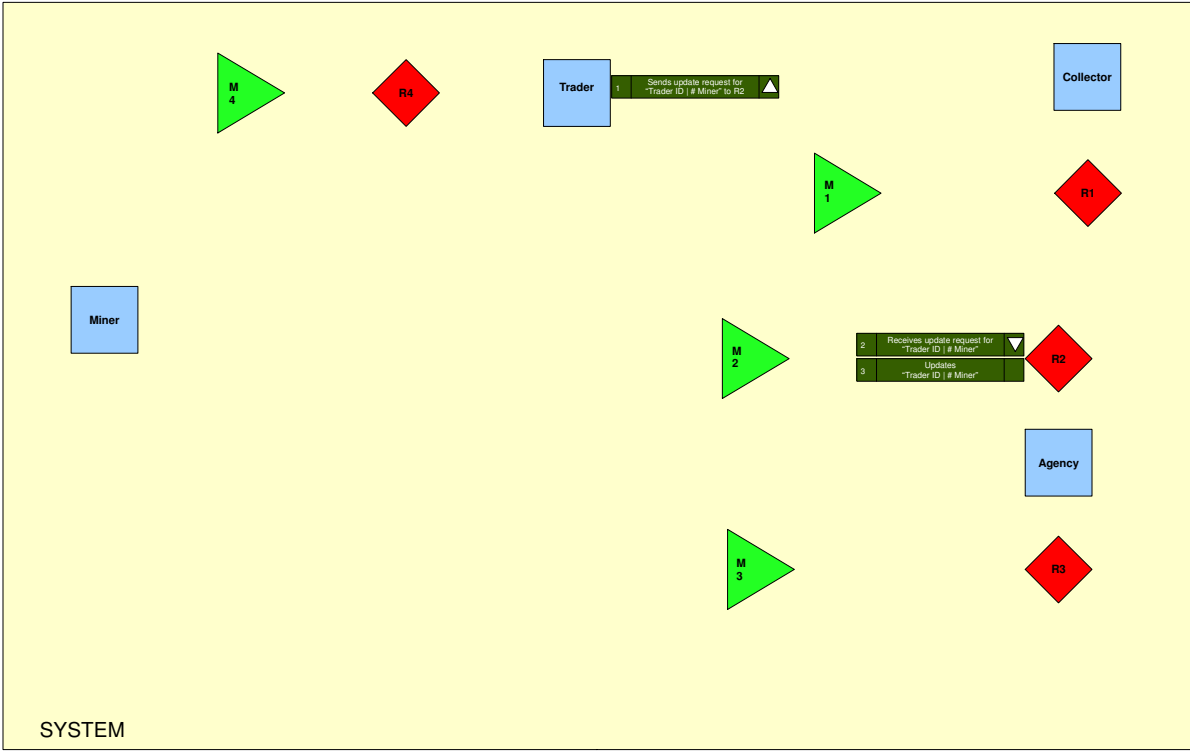
OBJECT MINERS PROJECT
SARC 011 : Trader Re-accesses Production





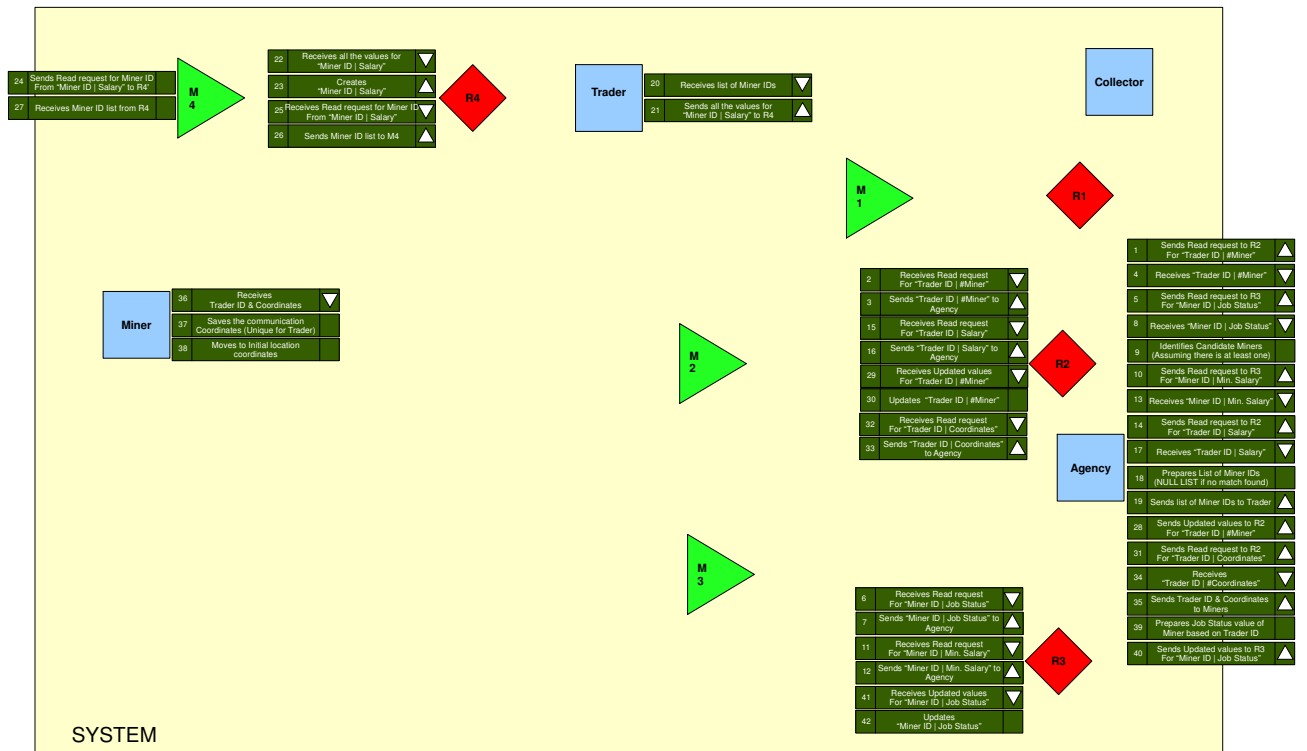
OBJECT MINERS PROJECT
SARC 012 : Trader Releases Miners





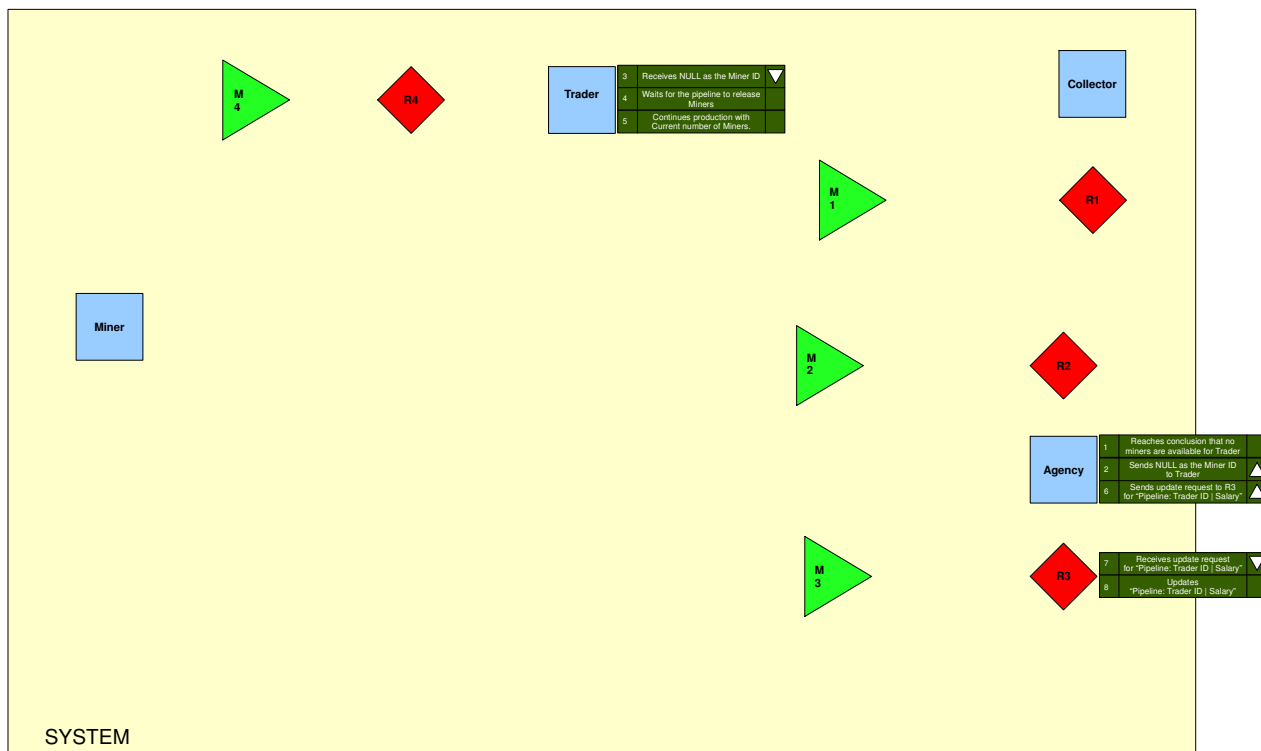


OBJECT MINERS PROJECT
SARC 014 : Agency Releases Additional Miners to Trader

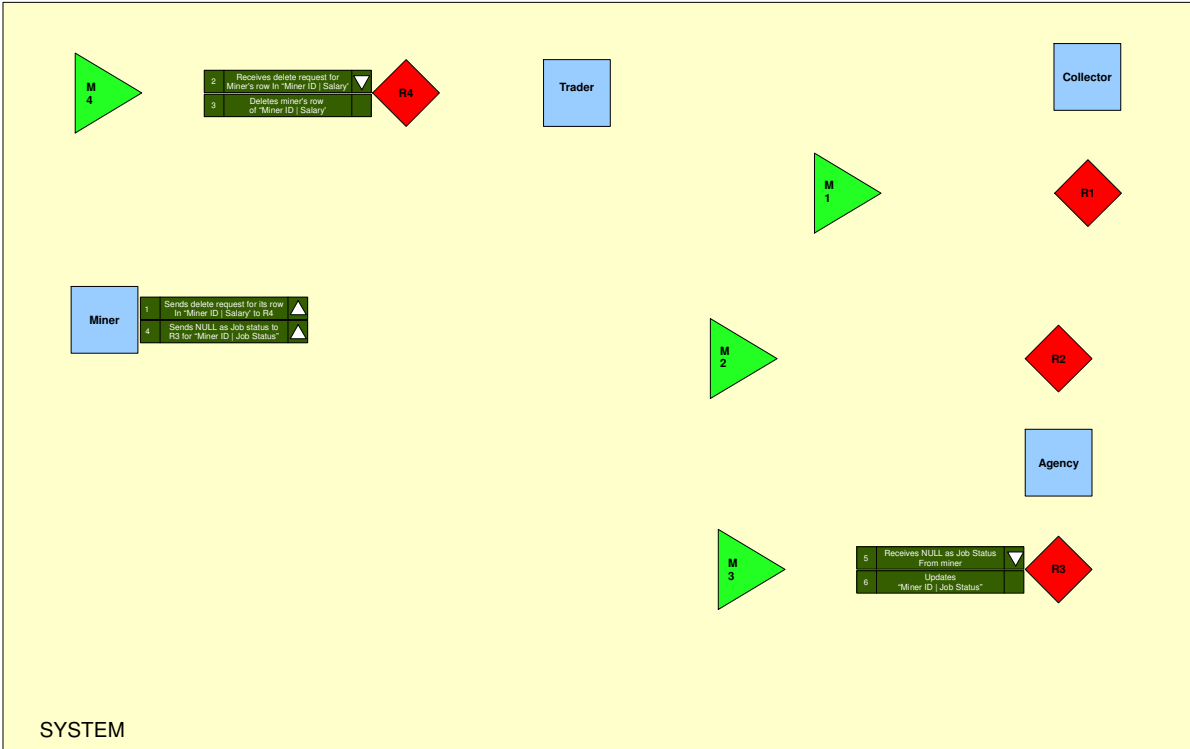




OBJECT MINERS PROJECT
SARC 015 : Agency Pipelines Trader Request For Miners



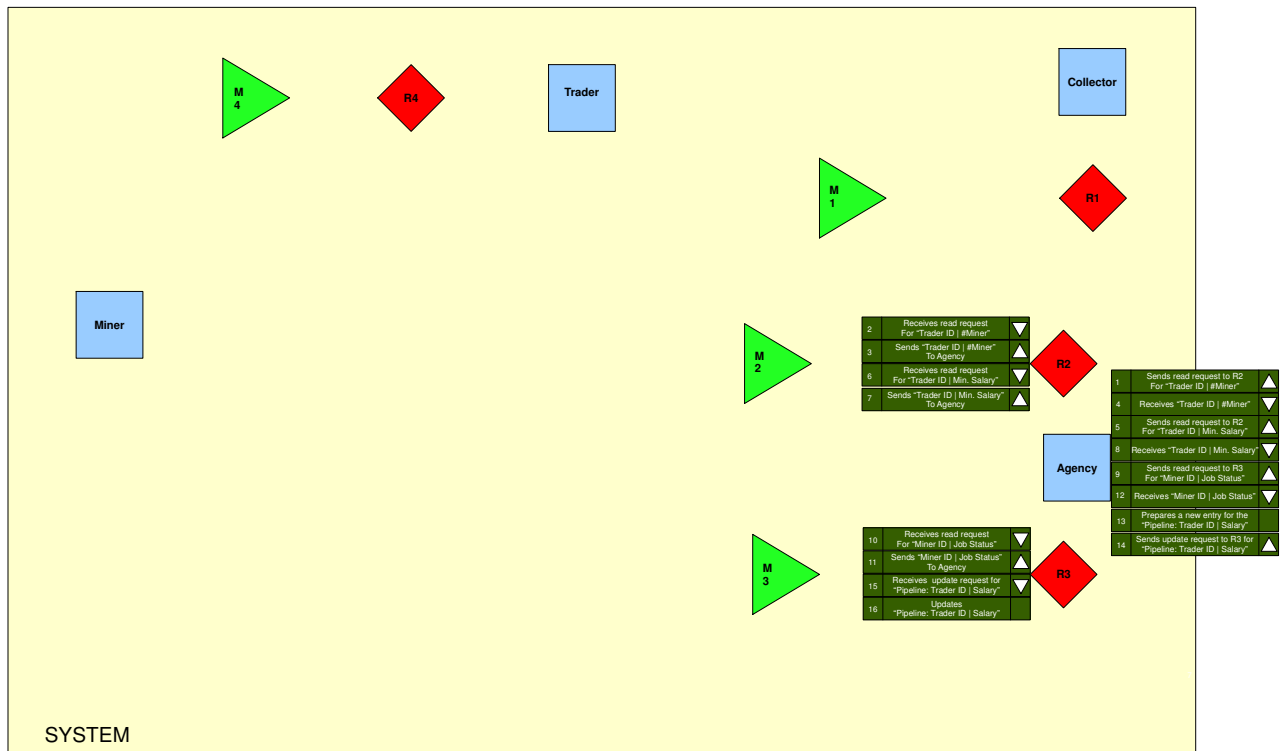
On receiving a NULL Miner ID, an active Trader may Increase the offered Salary based on its Business logic.



SYSTEM



OBJECT MINERS PROJECT
SARC 017 : Agency Manages Pipeline



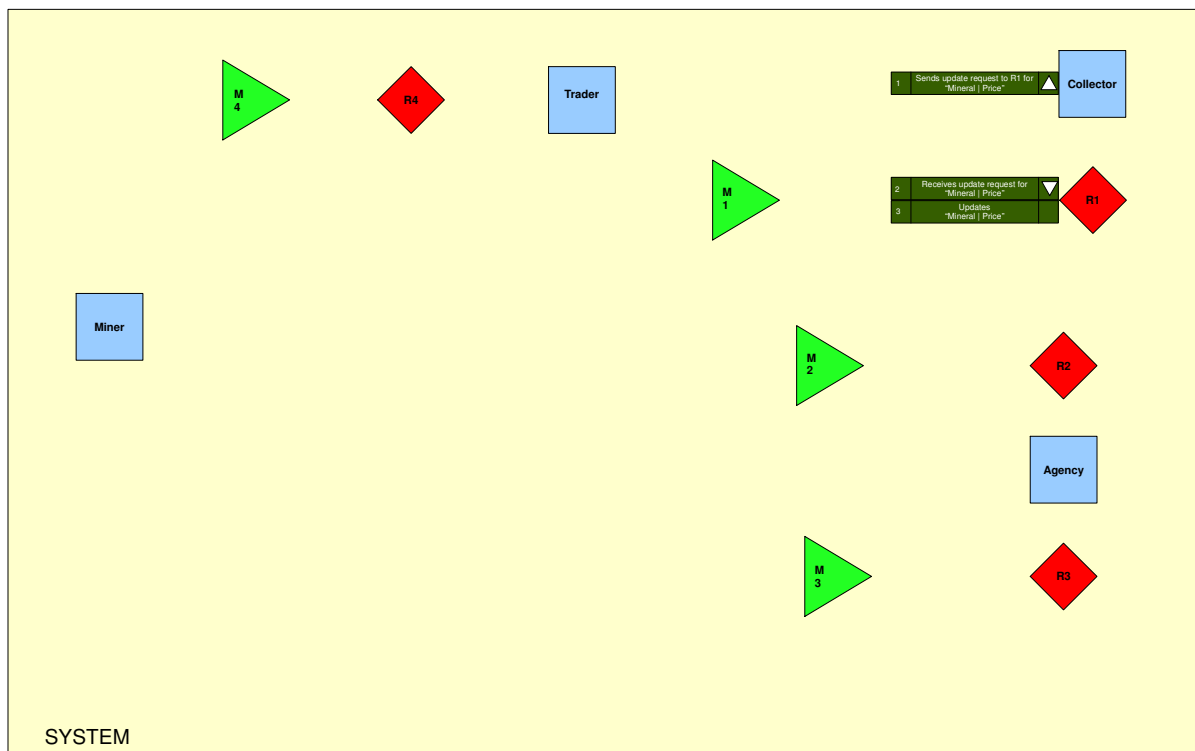
The Pipeline Updates are the trigger for Miners to switch their current job with a new one.
Assumption: A Miner cannot switch to a new job offered by its current Trader. Hence it will not react to jobs
In the pipeline from its current Trader. [Though this scenario can be explored]



Stories of human like Agent behaviour can be published. What if Miners stop working for a Trader is offering too much to new Miners, will there be a Strike? (Means employed but not working for a salary hike)

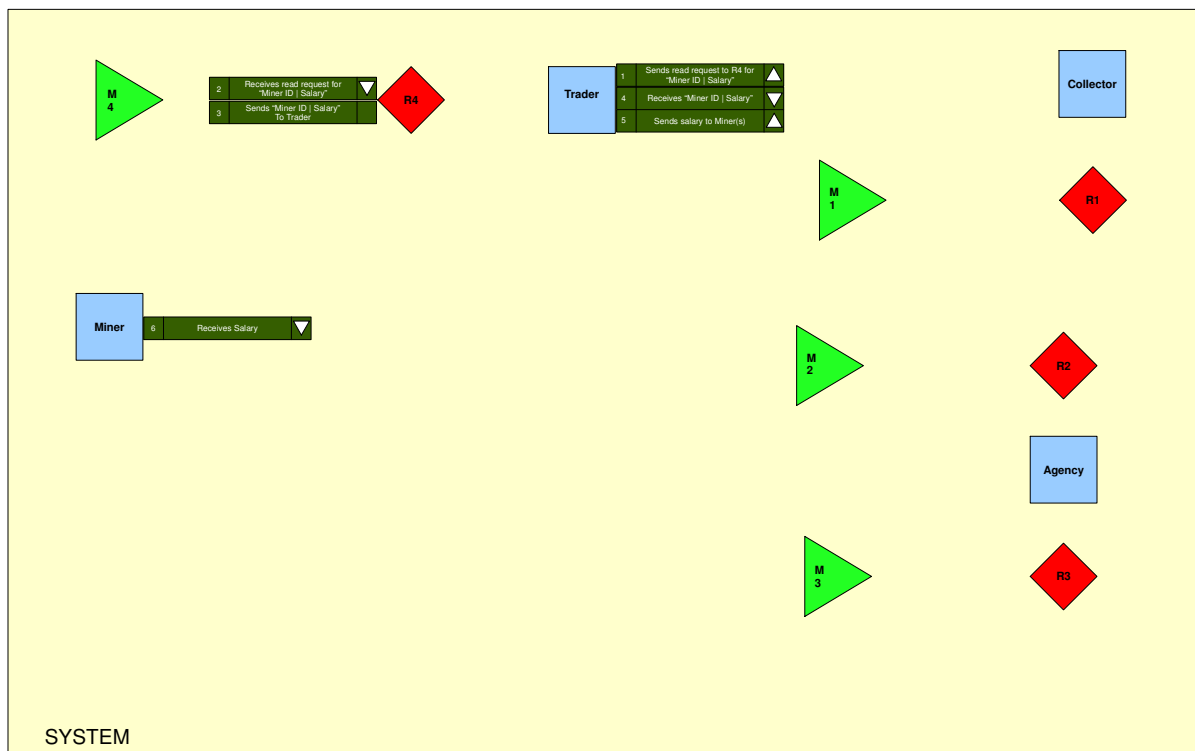


OBJECT MINERS PROJECT
SARC 018 : Collector Redefines Minerals And Prices





OBJECT MINERS PROJECT
SARC 019 : Trader Dispatches Miner Salaries



A retirement Scenario can be explored where after saving enough money a Miner decides to Retire.

Appendix C

Cloud Robotic Workshop at UTP

In order to test the feasibility of ARC based designing, we conducted a project based feasibility study at a workshop organized at University Technology Petronas, Malaysia in April 2012. A group of 24 students participated in the study. In total of 6 project ideas were developed using the ARC model, followed by detailed analysis of the usability of the model. A comparison was made to the traditional methods to represent ideas at the most abstract level and participant feedback was taken on various usability and functional aspects of the ARC model.

In order to simulate different levels of technical knowledge of the users of the ARC model, participants were chosen from various levels of education. The participants had a wide breadth of technical skills and were working in different domains. This was a crucial component of the participating group as it enabled to test the feasibility of the ARC model in representing ideas coming from people with different kind of expertise. It was important that the model was proved useful to all the participants in representing their ideas, but another important aspect of the study was to see how useful the model is to establish a common standard to represent ideas within the group, and its communication to the other groups.

A number of robots and devices were used by the participants to form the project ideas (Table. 1). The inclusion of a wide variety of agent-hosts in idea formation was mainly because of the freedom the participants had while working at an abstract level, without being influenced by the implementational details. The top layer of the proposed model (ARCs) is designed for this abstract level idea formation where all project stake holders (which in this study are simulated by a diversified participant composition Fig. 8) can contribute into the design.

ROBOTS AND OTHER AGENT HOSTS
RESCUE ROBOTS
IP THERMAL SENSORS
WIFI BOTS
PCs
LAPTOPS
SMART PHONES
TABLET PC
IP SMOKE SENSORS
LEGO MINDSTROM
WEB SERVERS
NAO HUMANOID ROBOT
PROGRAMMABLE IP CAMERAS
AMBIENT INTELLIGENCE RF TAGS

A. Proceedings

The 6 teams were given time to brainstorm on their own group project idea. In the first section, they were told to use any kind of textual or diagrammatical tools to make abstract design documents for their designs. Later in second section they were trained to use ARCs and then told to use ARCs to make their abstract design documents. The teams gave presentations of their design ideas twice, once using their own textual/ diagrammatical aids and once again using the ARCs. In the end the participants gave their feedback on 22 subjective data points. The feedback target was ranging from their understanding of the tasks given to them, to comparing the tools they used to make abstract design documents for their ideas. They were also asked about how easy it was for them to use the different tools and to understand the presentations of other teams when they used those tools. In all, 6 project ideas were developed and discusses. ARCs were developed for all 6 projects and the workshop provides a number of pointers to improve and validate the usability of the ARC based modeling.

B. Rating Scale

We used a subjective 5 level scale for all the data points in the feedback form.

Not at All	Little	Good	Very Much	Exceptional
------------	--------	------	-----------	-------------

C. Findings

Following are other findings which validate the “Need” of an ARC like system to model multi-agent systems as well as the feasibility of the top layer of the model (ARCs) as a potential solution.

When asked the usefulness of verbal discussions as a tool to develop the project idea. Average Rating: between “ good ” and “ very much ”.
When asked the usefulness of textual representations as a tool to develop the project idea. Average Rating: between “ little ” and “ good ”.
When asked the usefulness of Diagrammatical representations (Other than ARCs) to develop the project idea. Average Rating: between “ good ” and “ very much ”.
When asked the usefulness of ARC representations as a tool to develop the project idea. Average Rating: between “ very much ” and “ exceptional ”.
When asked the readability of the ARC representations developed by <u>other teams</u> compared to their first presentation. Average Rating between “ good ” and “ very much ”
When asked about the completeness of the set of ARCs to represent all the ideas into documents. Average Rating between “ good ” and “ very much ”
Those participants who have already worked with UML , the rating for the UARC and SARC was between “ very good ” and “ exceptional ”
There was a general comment from most of the participants that once the initial level of understanding is reached in making ARCs, it becomes easier and easier to use ARCs with every usage.
Most of the participants liked the flexibility of ARCs to represent Agents running on different Agent Hosts.
Another common comment was that having a standard template greatly helped in the brainstorming session.

All the participants were using the **ARCs** for the first time. ARCs require some level of training and experience to be really useful. Despite of limited training and experience the average score for its **usability** was between “**Very much**” and “**Exceptional**”. This is very encouraging for the further promotion of the model and provides very necessary pointers for the development of lower layers of the model.

Subjective Data Form

Use Blue/Black Ink.

Use BLOCK LETTERS

Workshop ID:

--

FULL NAME	
Team Name and ID	
Email (Use BLOCK)	

Please first read the complete form before you start answering the questions.

[01] How good you found the concept of the workshop.
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[02] Did you understand the task that was given to you?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[03] Did the team understand the task that was given to you?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[04] Were the background concepts well explained before the task begins?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[05] How useful was the verbal discussion amongst the team members, for Idea development?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[06] How useful were Textual representations (Plain text), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[07] How useful were Diagrammatical representations (Diagrams), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[08] Did you understand the concept behind ARC representations (Agent Relation Chart)?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
PLEASE EXPLAIN ARC:
[09] How useful were ARC representations (Agent Relation Chart), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:

[10] Did you understand the concept behind TARC representations (Trade Agent Relation Chart)?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
PLEASE EXPLAIN TARC:
[11] How useful were TARC representations (Trade Agent Relation Chart), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[12] Did you understand the concept behind HARC representations (Hyperactivity Agent Relation Chart)?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
PLEASE EXPLAIN HARC and Hyperactivity:
[13] How useful were HARC representations (Hyperactivity Agent Relation Chart), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[14] Have you already worked with UML (Unified Modeling Language)
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
What do you know about UML and its role in Software Development :
[15] Did you understand the concept behind UARC representations (Use Case Agent Relation Chart)?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
PLEASE EXPLAIN UARC:
[16] How useful were UARC representations (Use Case Agent Relation Chart), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:
[17] Did you understand the concept behind SARC representations (Sequence Agent Relation Chart)?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
PLEASE EXPLAIN UARC:
[18] How useful were SARC representations (Sequence Agent Relation Chart), for expressing the Idea?
Not at all <input type="checkbox"/> Little <input type="checkbox"/> Good <input type="checkbox"/> Very much <input type="checkbox"/> Exceptional <input type="checkbox"/>
Comments:

[19] Were ARCs (All of them) a complete set of representations to convert your ideas into design documents?

| Not at all ☐ | Little ☐ | Good ☐ | Very much ☐ | Exceptional ☐ |

PLEASE EXPLAIN:

[20] What additional components you would suggest to make ARCs (All of them) a complete set of representations to express your ideas?

PLEASE EXPLAIN:

[21] How Well were you able to read and understand ARCs made by other teams?

| Not at all ☐ | Little ☐ | Good ☐ | Very much ☐ | Exceptional ☐ |

PLEASE EXPLAIN:

[22] How useful was the whole workshop in terms of its "Quality" and "Content".

| Not at all ☐ | Little ☐ | Good ☐ | Very much ☐ | Exceptional ☐ |

Feel free to add any suggestions, comments or improvements:
(Add additional Pages if required)

Signature:

Date:

Place:

Résumé :

Le développement de logiciels pour les robots connectés est une difficulté majeure dans le domaine du génie logiciel. Les systèmes proposés sont souvent issus de la fusion de une ou plusieurs plateformes provenant des robots, des ordinateurs autonomes, des appareils mobiles, des machines virtuelles, des caméras et des réseaux. Nous proposons ici une approche orientée agent permettant de représenter les robots et tous les systèmes auxiliaires comme des agents d'un système. Ce concept de l'agence préserve l'autonomie sur chacun des agents, ce qui est essentiel dans la mise en oeuvre logique d'un nuage d'éléments connectés. Afin de procurer une flexibilité de mise en oeuvre des échanges entre les différentes entités, nous avons mis en place un mécanisme d'hyperactivité ce qui permet de libérer sélectivement une certaine autonomie d'un agent par rapport à ces associés. Actuellement, il n'existe pas de solution orientée méta-modèle pour décrire les ensembles de robots interconnectés. Dans cette thèse, nous présentons un méta-modèle appelé HTM5 pour spécifier la structure, les relations, les échanges, le comportement du système et l'hyperactivité dans un système de nuages de robots. La thèse décrit l'anatomie du méta-modèle (HTM5) en spécifiant les différentes couches indépendantes et en intégrant une plate-forme indépendante de toute plate-forme spécifique. Par ailleurs, la thèse décrit également un langage de domaine spécifique pour la modélisation indépendante dans HTM5. Des études de cas concernant la conception et la mise en oeuvre d'un système multi-robots basés sur le modèle développé sont également présentés dans la thèse. Ces études présentent des applications où les décisions commerciales dynamiques sont modélisées à l'aide du modèle HTM5 confirmant ainsi la faisabilité du méta-modèle proposé.

Mots-clés : Nuage de Robotique, le développement de logiciels utilisant des modèles

Abstract:

Software development for cloud connected robotic systems is a complex software engineering endeavour. These systems are often an amalgamation of one or more robotic platforms, standalone computers, mobile devices, server banks, virtual machines, cameras, network elements and ambient intelligence. An agent oriented approach represents robots and other auxiliary systems as agents in the system.

Software development for distributed and diverse systems like cloud robotic systems require special software modelling processes and tools. Model driven software development for such complex systems will increase flexibility, reusability, cost effectiveness and overall quality of the end product. The proposed 5-view meta-model has separate meta-models for specifying structure, relationships, trade, system behaviour and hyperactivity in a cloud robotic system. The thesis describes the anatomy of the 5-view Hyperactive Transaction Meta-Model (HTM5) in computation independent, platform independent and platform specific layers. The thesis also describes a domain specific language for computation independent modelling in HTM5.

The thesis has presented a complete meta-model for agent oriented cloud robotic systems and has several simulated and real experiment-projects justifying HTM5 as a feasible meta-model.

Keywords: Cloud Robotics, Model driven software development