Towards Specifying Swarm-Based Systems using Categorical Modeling Language: A Case Study

Noorulain Khurshid

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Software Engineering) at

Concordia University

Montreal, Quebec, Canada

August 2011

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Noorulain Khurshid

Entitled: Towards Specifying Swarm-Based Systems using Categorical Modeling Language: A Case Study

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________________________________ Chair
Dr. Nematollaah Shiri

________________________________ Examiner
Dr. Terry Fancott

________________________________ Examiner
Dr. Joey Paquet

________________________________ Supervisor
Dr. Olga Ormandjieva

________________________________ Supervisor
Dr. Stan Klasa

Approved By ________________________________
Chair of Department or Graduate Program Director

_____________ 20_____ ________________________________
Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

ABSTRACT

Towards Specifying Swarm-Based Systems using
Categorical Modeling Language: A Case Study

Noorulain Khurshid

One of the solutions to the software complexity crisis of this era is the proposition of self-managing systems like autonomous and autonomic systems. The idea has gained wide acceptance in the IT industry but it has also introduced the challenge of specification and development of such systems. Swarm intelligence is finding its applications in research and design of self-managing systems because of the coincidental resemblance between the two domains. However, specification of a swarm-based self-managing system is faced with the difficulty of specifying the complex evolving behavior.

This thesis presents an adaptation of a mathematical technique known as Category Theory to serve as a ‘reasoning and modeling’ paradigm for specifying high-level behavioral patterns of a swarm-based self-managing systems. The crux of this paradigm is the formal categorical modeling language (CML). CML syntax and semantics have been defined using an EBNF-based context-free grammar. The language helps to generate a formal specification of different scenarios/behavioral patterns of a swarm-based system. Moreover, a prototype tool has been implemented as part of this research work to serve as a modeling tool based on CML. In this thesis, NASA’s ANTS-based Prospecting Asteroid Mission (PAM) serves as a case study to analyze the applicability and usability of CML as a formal method of choice.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	**Definition**
CAT	Category Theory
SI	Swarm Intelligence
NASA	National Aeronautics and Space Administration
ANTS	Autonomous Nano Technology Swarms
PAM	Prospecting Asteroid Mission
SARA	Saturn Autonomous Ring Array
LARA	ANTS Application Lunar Base Activities
CSP	Communicating Sequential Processes
WSCCS	Weighted Synchronous Calculus of Sequential Systems
DESML	Dynamic Emergent System Modeling Language
PTN	Process Transition Networks
CML	Categorical Modeling Language
ASSL	Autonomic System Specification Language
EBNF	Extended Backus-Naur Form

1 Introduction

This chapter presents a synopsis of the context of this research work, the motivations that drove it, a mention of the objectives and contributions followed by the organization of this thesis.

1.1 Research Context

The computing systems of this era have introduced amongst many other challenges, the challenge of managing systems with ever growing complexity. Systems such as enterprise systems, sensor networks, grid systems and storage systems comprise of a large number of heterogeneously interacting components. This renders the system fault-prone and very dynamic. Consequently, it becomes very difficult to manage and configure these systems. Researchers have proposed a way to deal with this complexity by enabling the systems to manage themselves or with very little human intervention.

The idea of self-managing systems finds its shape in the form of reactive, autonomous and autonomic systems. All these systems incorporate different levels of self-management properties. Reactive systems are the most widely known systems that have the ability to respond to the dynamic changes in their environment demonstrating intelligent emergent behavior. Autonomous systems can handle major uncertainties in the system and the environment and have the capability to successfully recover from system failures without external intervention. Autonomic systems are based on the concept of how the human nervous system works. Self-management in an autonomic system is defined in terms of the self-* properties which include self-configuration, self-optimization, self-protection and self-healing. Self-managing systems exhibit patterns of intelligent behavior which is analogous to the concept of swarms and swarm intelligence.

Swarm intelligence is motivated by the concept of the collective behavior of a social insect colony towards achieving a goal or a set of goals [10]. Swarm intelligence is extensively being used in problem solving particularly solving optimization problems. In addition to that, swarm intelligence is finding its applications in self-managing systems because of the similarity of the two domains. A swarm comprises of a large number of small entities with local interactions amongst themselves and the environment. Inside a swarm, the agents work in a set of teams exhibiting independent as well as group intelligence. Collectively these local interactions exhibit a very complex and intelligent behavior. Specification of this behavioral complexity is challenging and existing formal methods are being evaluated by researchers to find a favourable approach for specifying the behavioral/structural complexity of swarm-based systems with self-* properties. The NASA has initiated a mission architecture known as Autonomous Nano Technology Swarm that is based on swarm intelligence and possesses the self-* properties.

In this thesis, NASA’s ANTS-based Prospecting Asteroid Mission (PAM) has been selected as the case study. PAM being a concept mission consists of different classes of spacecraft with different set of responsibilities and goals, both individual and collective. Some spacecraft carry science instruments and are also called sciencecraft; others only serve the purpose of communication or control.

1.2 Motivations

This thesis presents the work done in presenting Category Theory as a formal method and a reasoning framework for identifying and specifying the patterns of behavior of a swarm-based PAM with self-* properties. Category Theory or CAT in mathematics is used to reason about structures at a very abstract level of detail. As discussed in detail in section 2.6, in order to specify the behavior of a system, representation of structure/behavioral details of the system are of vital importance and this is best dealt with functions abstraction from mathematics. This ideal matured to be the first and the most driving motivation for the work done in this research.

A Category in CAT consists of objects and relations between them known as morphisms representing structure of the category. The relations of objects with other objects in a category define their social life. This social life concept is similar to the social behavior of entities inside a swarm and laid another foundation, another motivation, for this research.

Lastly and most importantly, another motive that guided this research was to follow-up on the remark in [21] on the possibility of using CAT as a formal method for specifying ANTS-based missions.

1.3 Objectives & Contributions

This research started off as a study on application of CAT to the software engineering domain. Soon after the list of objectives started taking shape when Emil Vassev [38] presented his work on ASSL specification with ANTS as a case-study:

1. **Study CAT as a Formal Method, as a Reasoning Framework**

   The first and most important objective was to study CAT from the point of view of a 'Reasoning Framework' based on the ideal of reasoning about structures from CAT. Next, the goal was to study CAT as a formal method and application of CAT as a formal method for the behavioral specification of a swarm-based system. After reasoning about the ANTS-based mission scenarios in CAT, only a subset of CAT constructs were selected to study the possibility of using CAT for the behavioral specification of the mission scenarios.

2. **Proposition of a Modeling Language based on CAT**

   After doing a thorough study on CAT as a formal method and a reasoning framework, we felt the need to come up with a formal specification language based on CAT mathematical definitions. As part of the work done in constructing the modeling language named Categorical Modeling Language (CML), the following contributions have been made:

   a. Construction of a grammar for the specification language in order to construct "well-formed" specification in the language.

b. Defining the visual/graphical model notation.

3. **Application of the proposed modeling language to ANTS case study**

    a. Choice of a subset of ANTS-based mission scenarios.

    b. Choice of a subset of CML constructs for the chosen mission scenarios.

    c. Formal specification of the selected mission scenarios using CML.

4. **Tool Support for CML**

    Another important landmark of this research work was the implementation of a prototype modeling tool, named CATCanvas, to support the categorical modeling language. The contribution/highlights of the tool are:

    a. Separate workspace for Category and Functor Constructs.

    b. Ability to draw the visual models on a canvas.

    c. An intelligent mapping tool for the functor construct.

    d. Generate XML specification for a constructed model.

    e. Import XML specification to render the graphical model.

    f. PNG Export for the graphical models.

    g. Web-based Rich UI.

    h. Web-based application available online.

Throughout this research, in addition to the mentioned objectives, an inherent objective was to choose only the very basic CAT constructs for research and application to the case study. In addition to that, the primary goal was to convey CAT as a formal method to researchers in software engineering domain without overwhelming them with lengthy and complex mathematical details/definitions.

1.4 Thesis Organization

This thesis is organized into seven chapters in total. Chapter 1 presents an outline of the problem domain, the proposed solution and list of objectives to be achieved during the course of this research. Chapter 2 has two parts; the first part presents a discussion of self-managing systems, swarm and swarm intelligence and the attributes of a formal method for specifying the PAM. The second part includes a discussion on application of existing formal methods/modeling methodologies for specification of swarm-based systems, in particular, the PAM. The chapter concludes with a comparison of the existing methodologies against a subset of formal method attributes discussed in the first part. Chapter 3 consists of the basic categorical definitions along with examples for the constructs used in the rest of the chapters. Chapter 4 introduces the Categorical Modeling Language along with the grammar for the language. Chapter 5 presents the case study: Specification of PAM using CML, where different mission scenarios have been modeled using different CML constructs. Chapter 6 presents a discussion of the working of the CAT modeling tool, CATCanvas, in the form of a list of features. Finally chapter 7 consists of the conclusion statement followed by a list of possibilities for future work. The appendix consists of some other CAT constructs that were studied as part of this research but not included in this thesis.

2 Background & Related Work

2.1 Introduction

The background section in this chapter presents a brief discussion of self-managing systems, swarms and swarm intelligence, ANTS-based PAM and attributes of a formal method for specifying PAM. The section on related work includes a discussion on existing formal methods that have been used to specify swarm-based systems, in particular, the PAM. The last section presents an evaluation of the existing formal methods against the attributes discussed earlier.

2.2 Self-managing Systems & Behavioral Complexity

In 2001, IBM declared a manifesto according to which the tens of millions of lines of code of this era of computing systems present a threat to halt the progress in computing. They define this halt in progress as the result of the difficulty of managing complex computing systems, for example, a system that requires integrating several heterogeneous environments into corporate-wide computing systems that extend into the Internet [1]. Researchers have started to realize the need for self-controlling systems, or in simpler words, systems that can manage themselves. Solutions are being proposed to develop systems that are independent of human intervention. If not completely independent, the human involvement is at a very high level and the low-level complex tasks are handled by the system itself.

This hypothesis of completely independent systems is still in its infancy but the idea finds its applications in reactive, autonomous and autonomic systems. The hypothesis though presents a worthwhile solution to the crisis of handling large scale computing systems, but at the same time it also introduces the challenge of specifying the now more complex emergent behavior of a self-managing system. This section includes an introduction to reactive, autonomic and autonomous systems highlighting the inherent behavioral complexity.

2.2.1 Reactive Systems

A system that must respond to dynamic changes in its environment is termed to be reactive [39]. The complexity of a reactive system stems from an on-going interactivity with its environment, complex computations, concurrent response to sensory data and management of dataflow are amongst the numerous other contributing factors.

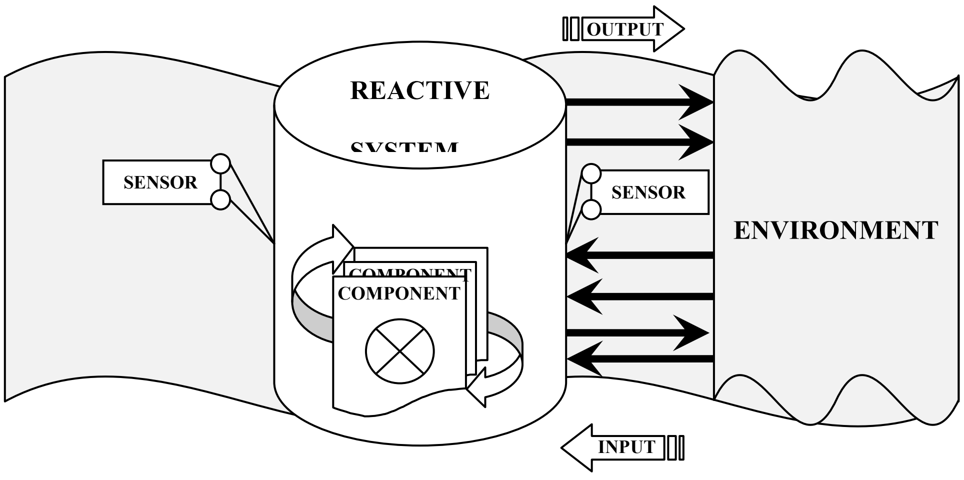


Figure 2.1: Block Diagram of a Typical Reactive System

An on-going interaction is similar to that of an infinite loop with a desire of non-termination. A sensor for instance is constantly transmitting data to the process loop that triggers relevant actuators and the entire process goes on to loop forever [40]. The system is thus constantly exchanging information with its environment and demonstrates a very complex emergent behavior because of the ever-changing sensory data. Figure 2.1 shows a block diagram of a classic reactive system and its interaction with its environment.

2.2.2 Autonomous Systems

Currently, most of the self-managing systems materialize themselves in the form of autonomous systems. These systems are increasingly being used in the industrial and commercial domains [8]. Autonomous systems are designed to perform well under significant uncertainties in the system and the environment for extended periods, and they have the ability to compensate for system failures without external intervention [9]. Every entity in the system has a certain degree of autonomy assigned to it for self-management and self-configuration at the entity level. In most of the systems, this degree of autonomy ranges to interactions of the entity with the environment as well.

Taking example of Microsoft Windows, in a windows environment the system has an ability to recover from failures to a certain extent. In addition, depending on the number of processes in the queue it regulates the CPU usage and performs memory allocation at runtime. Another example of a self-managing or self-configuring system could be that of the intelligent routing of network traffic. The network traffic monitors sense the bottleneck and route the traffic to the relatively less busy routes. Autonomous systems have the ability to carry out self-management tasks; self-management is classified into a set of properties in "Autonomic System".

2.2.3 Autonomic Systems

In the light of what they refer to as the "looming software complexity crisis", IBM proposed a new paradigm of computing known as "Autonomic Computing" drawing an analogy between the software systems and the human autonomic nervous system [2, 3].

The human nervous system is a master-controller that keeps track of the changes inside the human body and its environment. It gets data from the net of different sensors installed all over the body and sends appropriate response to maintain a certain state of balance inside the body [6]. This state of stable equilibrium is important for survival of a human being. Similarly, for a computing system, it is vital that the system maintains a certain state of equilibrium by protecting itself from crashes, has the ability to recover from a failure, and has the capability to reconfigure as required and when required. While in the process of maintaining a certain state of equilibrium, the system has to analyze the situation and choose a behavior from a set of behaviours in order to bring the system to a desired state. "Sensing", "Analyzing", "Planning" and "Execution" are the keywords used in literature to discuss an autonomic system [7, 2]. In an autonomic system the self-management properties are termed as the self-* properties. These include self-awareness, self-configuration, self-optimization, self-protection and self-healing [4, 5].

As we know from the introduction chapter, the discussions presented in this thesis are based on the NASA's ANTS-based PAM mission. ANTS being a swarm-based mission architecture, defines a swarm to be autonomous and autonomic.

2.3 Swarm Intelligence

'Swarm Intelligence' (SI) is a mindset rather than a technology. It provides a foundation to explore collective (or distributed) problem solving without centralized control or the provision of a global model. SI is being used to understand and explore reactive, autonomous and autonomic systems because of the similarity of behavioral complexity. A swarm comprises of a large number of small entities with local interactions amongst themselves and the environment [11]. The authors in [10] call these small entities 'unsophisticated agents' that interact locally amongst themselves and with the environment causing collective behavioural patterns to emerge globally. The local interactions between these agents demonstrate simple behavior, but the combination of these simple behaviours result in emergence of very complex behavior [15]. Inside a swarm, the agents work in a set of teams exhibiting independent as well as group intelligence [12]. The central idea of SI has been inspired by how social insects operate in an insect colony and demonstrate a certain hive culture.

2.3.1 Social Insects, Social Swarm, Social Behavior

An insect colony is analogues to the way reactive, autonomous and autonomic systems are built and work. Inside an insect colony, be that an ants' colony or a beehive, these insects interact to achieve a goal or a set of goals. The colony can respond to external challenges as well as internal perturbations and is robust, in that, tasks are completed upon failure of an individual, indicating the ability of the swarm entities to self-heal. In addition, an insect colony does not have a central controller in the colony that directs the workflow towards achievement of a goal. Lastly, paths to a solution are emergent rather than being predefined which indicates to the ability of the swarm entities to self-organize. The individual ants when seen interacting inside a group or swarm working to achieve a target demonstrate a social life, social behavior or structure of that swarm. The collective behavior or structure looks very complex and presents challenges on understanding the inherent goals of the swarm.

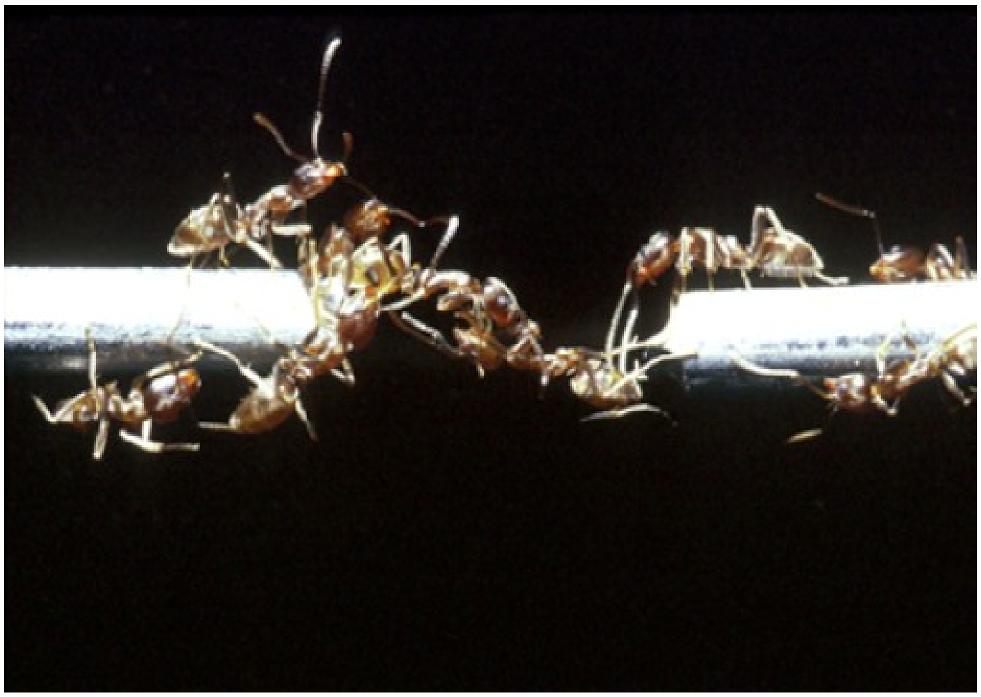


Figure 2.2: Interacting Ants

In view of this, specification of this social behavior or structure of a swarm in a swarm-based system is a challenge. The following points should be kept in mind while specifying the social behavior of a swarm [10]:

1. It is difficult to predict collective behavior from simple/individual rules.
2. Modeling of group-level behavior is possible through bottom-up approach.
3. A participant/agent inside a swarm is unaware of the function of the group.
4. Slight changes in the rules result in a different group level behavior.
5. Efficient control of organization or manipulation of groups inside a swarm is possible through simple rules.

The challenge of modeling and specification of a swarm-based system actually comes from the difficulty in specifying the social behavior of the swarm along with the self-* properties.

2.4 Autonomous Nano Technology Swarms

Autonomous Nano Technology Swarms or ANTS is a swarm-based mission architecture for concept mission by the NASA driven by the need to collect more data than is possible by a single spacecraft. The missions based on ANTS will be unmanned and highly autonomous. In an ANTS mission a hundred or even thousand picospacecraft weighing less than or equal to 1Kg moving in a swarm, will work cooperatively in order to explore the space entities (planets, asteroid belt, moon depending on the mission). The spacecraft must work both individually and collectively and must also possess autonomic self-* properties in order to endure the harsh space environment [15, 17]. ANTS consists of a number of concept missions: The Saturn Autonomous Ring Array (SARA), ANTS Application Lunar Base Activities (LARA) and Prospecting Asteroid Mission (PAM). For the course of this thesis, Prospecting Asteroid Mission will serve as the baseline case study.

2.4.1 Prospecting Asteroid Mission, A Case Study

PAM, an ANTS-based concept mission is a future autonomous robotic mission for exploration of the asteroid belt [17, 18]. The mission will comprise of a swarm of autonomous pico-class spacecraft, weighing approximately 1kg, which will explore the asteroid belt for asteroids with certain characteristics such as mass, density, morphology, and chemical composition. A few of these spacecraft will form teams and for example use their scientific instruments to record properties of the asteroid(s). Other spacecraft will communicate with the data collectors and send updates to Earth station [15]. The teams and groups formed inside a swarm may be ad-hoc and temporary depending on the requirements of the mission, for example, sharing of resources or surveying a new asteroid [16].

The mission is discussed in detail in chapter 5. The discussions in the rest of the sections are related to specification of the PAM, discussion of what a formal method is, what is a formal specification, and what are the attributes of a formal method in relation to specifying PAM. Section 2.5 and 2.6 are mainly based on the discussions in [20].

2.5 What is a Formal Method?

Applying a formal method is using mathematical techniques like abstract algebra, logic and discrete mathematics for representing information required to build software systems. The word "formal" from "formal logic" indicates the ability to reason using "structure" and not the "content". A specification based on a formal method has to be a "well-formed" mathematical set of statements and is verifiable by logical deduction in the formal method.

2.6 Why and What to Specify?

From Engineering, Architecture and Software domain, a specification is a description of the structure and behavior of the product to be developed. The word complexity in this thesis refers to the structural and behavioral complexity of a software system. A proper specification can help represent and control complexity of a software. The most familiar and effective way of dealing with complexity is via 'abstraction' while behavioral/structural complexity is best dealt with function abstractions from Mathematics.

The very first level of specification of a software system is the precise and unambiguous description of the system behavior in terms of externally observable functional characteristics. In this thesis, the word specification refers to the behavioral specification only and should not be confused with the design specification.

2.7 Specifying Swarm Behavior

From the previous text, we witness the challenges of specifying complex emergent behavior of a swarm and a swarm-based system per se. Specification of the swarm behavior is challenging because of the very nature of the interactivity of its entities and the resulting behavior at different levels of hierarchy in the system. This section includes the characteristics of a formal method in general and for specifying PAM in particular.

2.7.1 Attributes of a Formal Method for Specifying PAM

A formal method's characteristic whether its language is graphical or whether its underlying logic is first-order influences the style in which user applies it [24]. Formal methods are proven approaches for ensuring the correct operation of complex interacting systems. Once written, a formal specification can be used to prove properties of a system correct and check for particular types of errors (e.g., race conditions), and can be used as input to a model checker. Verifying emergent behaviour is one area that, unfortunately, most formal methods have not addressed well [23]. In [22], the authors have combined several methodologies for specification of the PAM swarm and conclude that integration of the evaluated methods seems to be the best approach so far.

This section includes a list of attributes coming from both the solution presented in this research and the problem domain. Together these characteristics advocate for a formal method to be termed as a favourable approach for specifying swarm-based PAM mission.

Formal Basis

Like most of the modeling paradigms, formalization of the semantics should have a mathematical basis [20]. This includes logic, algebra or any other mathematical theory. It is thus important to take note of the fact that visual modeling methodologies without a mathematical basis could not qualify as a formal method.

Language Abstraction

Abstraction promotes the declarative specification in a language. This property supports powerful primitives for defining and manipulating information and data at the logical level. Logical data definition should not imply any specific data representation [20].

Modularity

A specification language allows construction of large and complex specification by assembling smaller constructs. This attribute supports modular design and incremental specification that in turn adds to the expression power of the language [20].

Adaptability to Programming

All kind of computer system specifications find home in some sort of program/algorithm. It is thus necessary for the specification to be adaptable to the requirements of a computer program [21].

Reasoning

It is desirable for the methodology used for specifying PAM to enable intelligent reasoning with possibly inconsistent and uncertain information [21].

Visual Formalism

"A picture is worth a thousand words". A modeling paradigm that has visual models aids the process of representing complexity of a system with interacting components and organization of these components to demonstrate local and global behaviours [25].

Hierarchical Abstraction

Inside a PAM swarm, there are different levels of hierarchical distributions of the spacecraft. This granularity demands specification at different levels of hierarchy rather than a few defined architectural levels. Abstraction of details while representing a certain system scenario is desirable. The idea follows from the basic human direction of problem solving: "Divide and Conquer". Breaking a bigger problem into manageable smaller problem, the bigger problem still in mind could aid in representation of complex behavior by representation of simple behavior and deductions at a local level [30].

Ease of Comprehension

Most of the modeling methodologies suffer from complexity of constructs either mathematical or algorithmic. This affects the ability of the formalism to be used as a favourite approach especially in computer science and software engineering domains [25].

Tool Support

Model-checking and verification of the specification is of vital importance, especially in the case of verifying the complex behaviour specification. Tool support with the formal method makes it possible to verify a generated specification [21].

Emergent Behavior Specification & Verification

One of the most important attributes of a formal method for specifying a PAM is its ability to predict and verify emergent behavior [21]. Prediction of emergent behavior has come across as an enormous challenge to researchers to date.

Specification of Probability or Frequency

A means of expressing the probability of certain actions, and the frequency with which they will occur is desired to be specified [21].

2.8 Related Work

This section presents different approaches that have been used to model swarm-based systems; in particular, NASA's ANTS based missions.

2.8.1 Communicating Sequential Processes (CSP)

Communicating Sequential Processes, or CSP [27], is a language used to describe the patterns of interaction between a set of interacting entities. CSP [28] has been used for specifying NASA's PAM in [29]. In [28] each of the spacecraft is assigned goals to fulfill their mission and the emergent behavior of all these goals is considered equal to the goals of the mission. The specification makes use of the CSP command language. CSP has also been suggested as a preferred approach in [22] for specification and verification of the emergent behavior of intelligent swarms.

2.8.2 Weighted Synchronous Calculus of Sequential Systems (WSCCS)

WSCCS is a process algebra proposed by the author in [35] and was used to model social insects. Tofts [35] specified models where ants were the components, and all the component ants together composed the entire colony. WSCCS was also used in combination with a dynamical systems method for analyzing the nonlinear characteristics of social insects [34]. WSCCS has been used as one of the favoured formal methods for specification and verification of the NASA's swarm-based missions [21, 28, 29, 22].

2.8.3 X-Machines

Introduced by Eilenberg [36], X-machines is a specification formalism [59] capable of modelling both the data and the control. X-machines is a diagrammatic approach, which is an extension to the finite state automata. Transitions between states are in X-machines are not performed through simple input symbols but through the application of functions. These functions specify the processing of the data and are written in a formal notation. X-machines have memory attached in order to hold data. Functions receive symbols and memory values as input, and produce output modifying the memory values wherever required. X-Machines is claimed to provide a highly executable environment for specifying the ANTS spacecraft. It enables data storage in the memory and represents the transition between states as functions involving inputs and outputs. This aids keeping track of the actions of the ANTS spacecraft as well as memory keeping of every step of the goals. This ability makes X-Machines highly effective for tracking and affecting changes in the goals and the model [21, 28, 29, 22].

2.8.4 Unity Logic

Unity Logic makes use of the propositional logic syntax for reasoning about the ANTS spacecraft and the states they imply. It can also be used for defining specific mathematical, statistical and other simple calculations to be performed. Though, unity logic is not expressive enough to allow easy specification and validation of more abstract concepts such as mission goals. It can serve as a good candidate for specifying and validating the actual reasoning programming portion of the ANTS Leader spacecraft, if and when required in future [21, 28].

2.8.5 Temporal Logic

Formal specification frequently witnesses the use of temporal logic. It has also been used to specify the swarm behaviours including emergent behaviours. The author in [19] explores linear time temporal logic for the formal specification of the behaviour exhibited by the swarm robots. The swarm robots have been modeled as concurrent processes. Temporal logic is a favoured approach but it tends to ignore the multi-level, compositional nature of a swarm-based system.

2.8.6 Autonomic System Specification Language (ASSL)

The author in [37] has proposed a framework called Autonomic System Specification Language (ASSL) for formal specification and generation of autonomic systems. ASSL makes it possible to specify high-level behavior policies, as part of overall system behavior. ASSL has been accepted as a very suitable candidate for specifying the autonomic behavior of swarm-based missions and has been applied to specify self-configuration, self-healing, and safety properties of NASA's swarm-based missions [38]. ASSL is directed towards the system's specification at the design level.

2.8.7 Dynamic Emergent System Modeling Language (DESML)

The author in [31] has proposed a variant of UML called DESML providing several new graphical constructs to the basic UML. DESML was mainly proposed for specification of emerging distributed systems, and swarm-based systems [21]. The author has indicated it in [15] to be a possible candidate for specifying NASA's swarm-based systems.

2.8.8 Process Transition Networks (PTN)

PTN is a graphical language for specifying behavior of entities in an autonomous system. PTN has formal semantics and enables simultaneous behavioral specification of independent components of a system and the environment. PTN does not have the ability to represent hierarchical abstractions and thus PTN specifications are more flat than that of Statecharts [26].

2.9 Category Theory

Category theory, concisely, is an advanced mathematical formalism, independent of any modeling or programming paradigm, capable of representing "structure" [41, 42, 43]. In mathematics, category theory is an abstract way of agreement between various mathematical structures and relationships between them. Everything is abstracted from its original meaning to a corresponding categorical meaning. For example, sets in set theory abstract to objects in category theory and functions over those sets abstract to morphisms in categorical terminology. Therefore, the category named **SET** will have objects that are sets and morphisms or arrows as all of the functions over those set objects. Although category theory is a relatively new domain of mathematics, introduced and formulated in 1945 [44], categories have been frequently discussed and used to relate sets, vector spaces, groups, and topological spaces all of which naturally correspond to distinct categories [45].

2.9.1 Social Life in CAT

The concept of social life is innate to the structure inside and outside of a category of related entities and is analogous to the social life theory in swarms. Entities in real world may or may not interact with each other. This presence or absence of interaction of an entity with other entities defines the way an entity behaves socially. This social "behavior" or in broad terms the social life of an entity is mainly defined by its role in the group it belongs to. A group in category theory represents a structure and presents ground for reasoning about this structure in relation to other structures.

2.9.2 CAT, A Formal Method, A Reasoning Framework

Based on the basic definition in 2.5, CAT qualifies as a formal method but in order to specify the complex behavior of a software system the CAT specification language does not have any metalanguage or grammar to construct 'well-formed" specification statements. The mathematical formula/notations used to specify models in CAT are not standard and expressive enough. There is a need for a specification language for CAT-based specifications in order to construct well-formed specification.

CAT enables expression of semantics for interconnection, theory, instantiation, and composition. For all these attributes are of significance when reasoning about a swarm-based system.

2.10 Categorical Modeling Language (CML)

In this thesis, a specification language for CAT has been proposed along with the grammar for constructing well-formed sentences in the language. The language is named Categorical Modeling Language, in short, CML. A CML model includes a formal specification as well as a graphical model of the problem domain. A modeling tool for CML is also written as part of this research. The tool also serves as a static verifier of the imported CML specification in XML format.

CML can prove to be a viable candidate for specification of swarm-based systems along with other mentioned methodologies. Table 2.1 evaluates the strengths of the mentioned methodologies along with CML against a subset of attributes discussed earlier in this chapter. From the table, we can compare CML with the rest of the methodologies to conclude that CML can qualify as a favourable approach for specification of swarm-based systems. The one attribute that CML currently lacks is the verification of the emergent behavior in a swarm-based system. This is discussed further in chapter 7, conclusion and future work. The other attributes of CML are discussed in the subsequent chapters.

Table 2.1: Attributes of Specification Methodologies for Swarm-based Systems

	Formal Basis	Visual Formalism	Adaptability to Programming	Tool Support	Modularity	Emergent Behavior Verification
CSP	✓	✘	✘	✓	✓	✘
WSCCS	✓	✘	✘	✘	✓	✓
Temporal Logic	✓	✘	✘	✘	✓	✘
X-Machines	✓	✓	✓	✘	✓	✓
Unity Logic	✓	✘	✘	✘	✓	✘
ASSL	✓	✘	✓	✘	✓	✘
PTN	✓	✓	✘	✘	✓	✘
DESML	✘	✓	✓	✓	✓	✓
CML	✓	✓	✓	✓	✓	✘*

* Verification of emergent behavior is the next step and part of the future work

3 Category Theory

This chapter presents a subset of the different constructs in Category theory. These constructs serve to both abstract and unify many concrete theories in diverse branches of mathematics. The definitions have been supplemented with easy to understand examples avoiding extensive mathematical details.

3.1 Category and Social Life

The central notion in CAT is the interesting concept of a 'Category'. Obvious by its name, a category is a class or group of entities related to each other in some way that defines the category they belong to, and the reason why they belong to that category. In the previous chapter, we briefly discussed how category theory or in short CAT talks about objects living a social life in a category, defined by the relationships between these objects. Keeping in view the coincidental similarity of the insect swarms, the swarm-based systems and the categorical social life, this chapter refers to hive culture of insect swarms in support to explain the different constructs of CAT.

Let us try to look at the biological classification of species in terms of categories in CAT. A category of all species consists of some 7 - 100 million species [32]. Likewise, a category of insects consists of only the species that are characterized as insects. Further, down the classification of insects, a honey bee category is different from category bumble bee, and so on. The noticeable attribute of this example is the abstraction of the category construct. The notation used in this chapter when talking about a category would be bold capital letters, e.g. **HB** for category honeybee. As social insects living in a colony, honey bees must communicate with one another using movement, odour cues, and even food exchanges to share information.

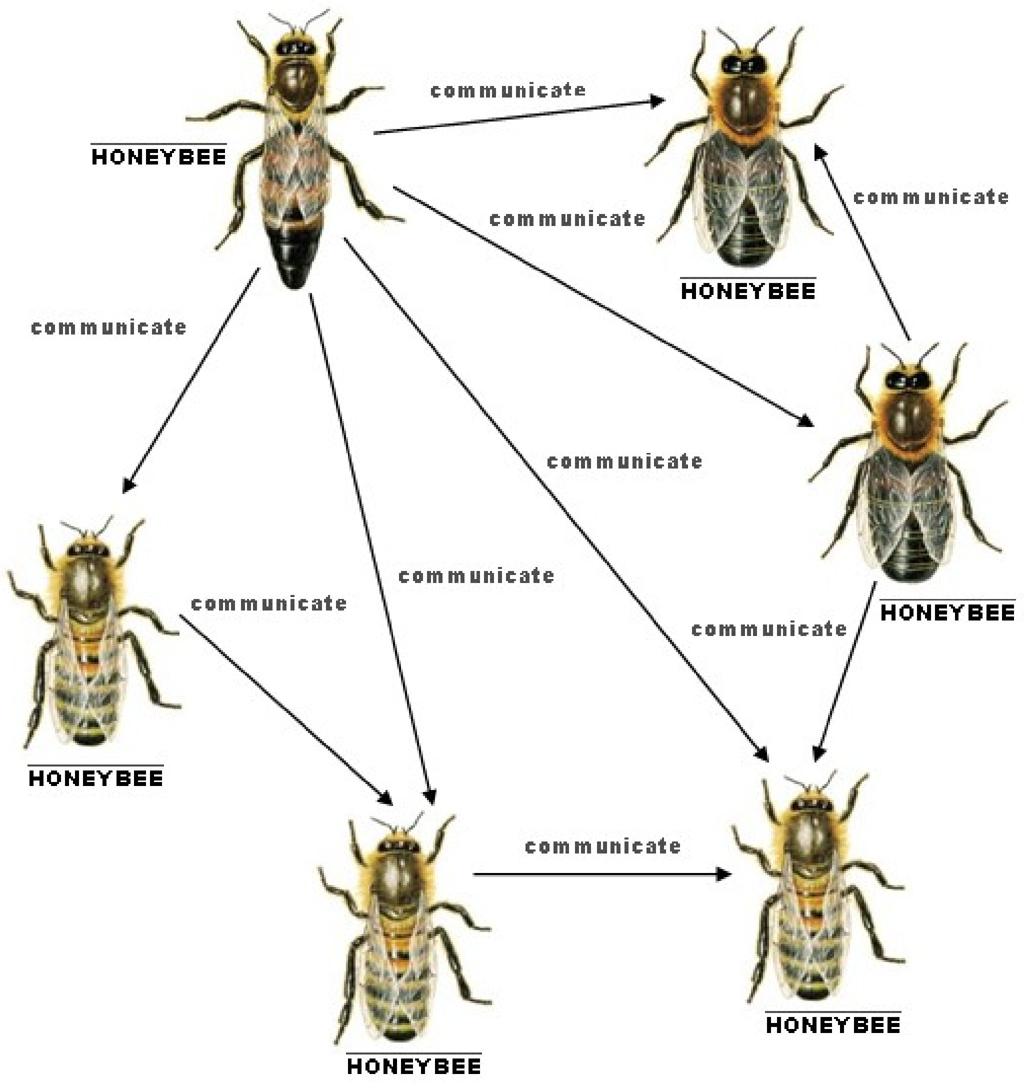


Figure 3.1: Category HB

Figure 3.1 shows a snapshot of the social life in a category honey bee, **HB**. The objects being honeybees and relationships/morphisms between these object representing communication in the form of signals [33]. The readers should keep in mind while going through this thesis that only an instance of a category has been included in the diagrams/examples for the purpose of discussions and not the entire category. For example, Figure 3.1 shows a category of only six bees and some morphisms between them, not the entire category HB.

Formally, a category **C** consists of:

1. A collection Obj (**C**) of objects.
2. $\forall$ x,y $\in$Obj(**C**) a collection **C**(x,y) of morphisms.
3. Identity: $\forall$ x $\in$Obj(**C**), a morphism Id(x): x $\rightarrow$ x, Id(x) $\in$ Identity(**C**)
4. Composition: $\forall$ x,y,z $\in$Obj(**C**), then we have a function (composition),

   C(x,y) $\circ$ **C**(y,z) $\rightarrow$ **C**(x,z)
5. Following axioms hold true:
   a. Identity: $\forall$ x,y $\in$Obj(**C**), f $\in$ **C**(x,y), f: x $\rightarrow$ y,

      Id(x): x $\rightarrow$ x, Id(y): y $\rightarrow$ y and Id(x), Id(y) $\in$ Identity (**C**) then,

      Id(y) $\circ$ f = f = f $\circ$ Id(x)

   b. Associativity: $\forall$ x,y,z $\in$ Obj(**C**), f,g,h $\in$ **C**(x,y), and
      f: x $\rightarrow$ y, g: y $\rightarrow$ z and h: z $\rightarrow$ m, then,
      h $\circ$ (g $\circ$ f) = (h $\circ$ g) $\circ$ f

The definition is explained in detail in sections to follow.

3.2 Typed Category

A category typically declares all objects belonging to that category to be of one type. For instance, the category **SET** has all objects that are sets. However, every set could also be defined to be of a certain type. For example, a set can be of type integer or natural numbers and so on. This classification of objects inside a category into types promotes the need for the Typed Category construct. A typed category is not native to the category theory but is a result of the requirements that arose during this research to better adapt to modeling of the swarm-based systems.

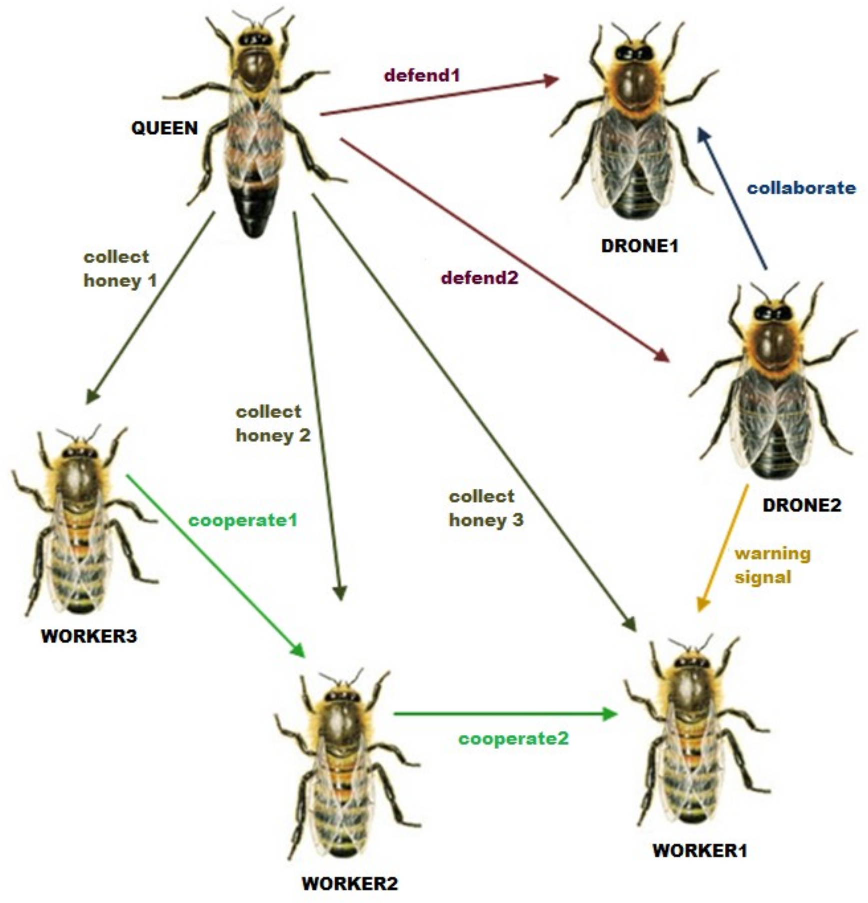


Figure 3.2: Social Life –Typed Category HB

Along with objects, morphisms too can have certain types. Every morphism thus belongs to a certain type and each type of morphism consists of a set of morphism instances under it. For the most part of this thesis, categories would be typed categories. Figure 3.2 re-demonstrates an instance of category **HB** but with typed objects and typed morphisms.

3.2.1 Objects & Morphisms

In categorical terminology, the entities we have been referring to in the social life phenomenon are called objects. The diagrammatical notation for a category object typically is a circle with name of the object inside the circle. For the most part of this thesis, for consistency, we have used the circle notation to represent an object. Capital letters up to three letters followed by a subscript, if any, have been used to name objects in the course of this thesis. The abstract notion of an object enables reuse of the very concept, to give it any form possible. Figure 3.3 (a) shows an object in diagrammatical representation. The mathematical representation of an object follows the diagrammatical naming convention. Figure 3.3 (d) shows the mathematical representation of a category.

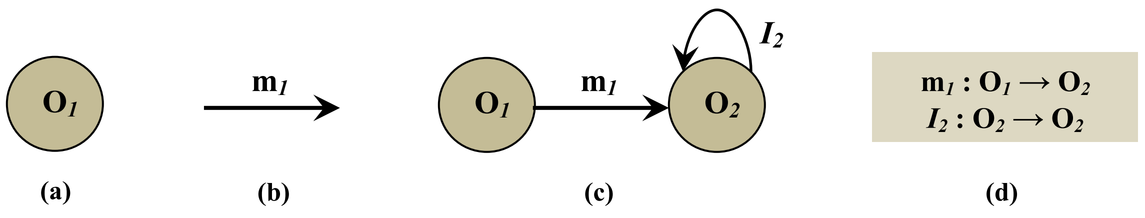


Figure 3.3: (a) Object Notation; (b) Morphism Notation; (c) Morphism w/ Source-Target Objects, Identity Morphism; (d) Mathematical Notation for Morphisms

A relationship between two objects is termed as a 'morphism' in category theory. A set of morphisms between two objects comprises of all morphisms between those two objects. The directions of morphisms are of significance since they determine the overall structure of the category they reside in. The object from which a morphism originates is called the domain or source object. The object to which the morphism is directed to is called the co-domain or the target object. Morphisms define how one category is different from another. The entire structure inside a category takes its shape from these interactions or morphisms. The ideal behind the concept of a morphism being plain interaction in simple terms makes it possible to apply the concept in any scenario where a possible interaction between entities could be represented/modeled. Figure 3.3(b) shows the diagrammatical notation of morphisms. The arrow represents the direction, the name of a morphism in this thesis is always in lower case letters up to three letters followed by a subscript, if any. Morphisms have also been referred to as arrows in literature. Mathematically, name of a morphism follows the same naming convention as its diagrammatical counterpart followed by a colon. The source object name comes next to the colon followed by an arrow to the target object. Figure 3.3 (c), (d) shows the diagrammatical and mathematical morphisms respectively. A special kind of morphism with each object is identity morphism. The identity morphisms are seldom shown on a category diagram. It is assumed to be present, to avoid cluttering the diagram with an identity morphism for each object. The notation for identity morphism is a looped arrow with source and target being the same object. The name for an identity morphism starts with an italicized capital $\boldsymbol{I}$ followed by the object's subscript. Object $\mathbf{O}_{2}$ in Figure 3.3(d) shows identity morphism $\boldsymbol{I}_{2}$ for Object $\mathbf{O}_{2}$.

3.2.2 Composition & Associativity

Inside a category, two composable arrows compose together to form a composition morphism. Composition morphism is an alternative path to the target object and is present for every pair of composable arrows. The absence of composition morphism for any pair of composable arrows compromises the definition of a category. Composition of an identity morphism with any morphism results into the latter morphism.

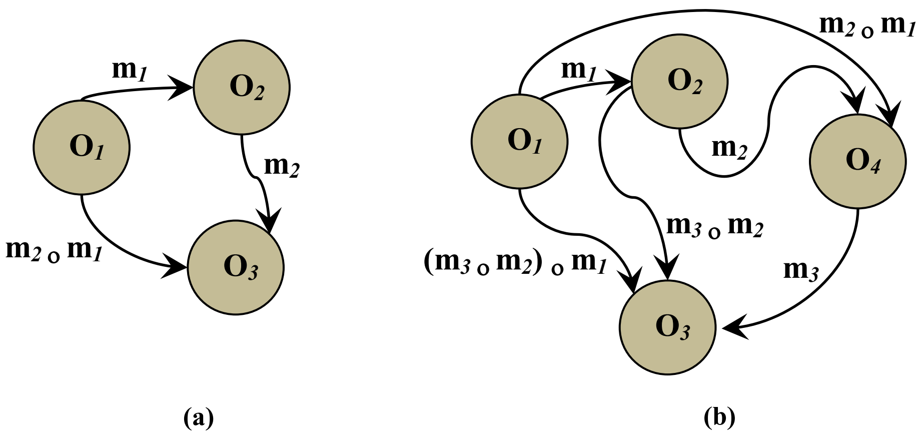


Figure 3.4: (a) Composition of $\mathbf{m}_1$ & $\mathbf{m}_2$; (b) Associativity for $\mathbf{m}_1$, $\mathbf{m}_2$ & $\mathbf{m}_3$

The composition of two arrows is represented with the symbol '$\circ$'. Graphically, the composition arrow is just another arrow with name consisting of the two arrows composed together '$\mathbf{m}_2 \circ \mathbf{m}_1$' (see Figure 3.4 (a)) or equivalent composition morphism name. The composition arrow $\mathbf{m}_2 \circ \mathbf{m}_1$ is read $\mathbf{m}_1$ composed with $\mathbf{m}_2$ keeping in view the direction of composition. Composing three arrows in a direction leads to a complex composition and has to evaluate true for associativity property. Let us assume we have another object $\mathbf{O}_4$ and three morphisms such that, $\mathbf{m}_1: \mathbf{O}_1 \rightarrow \mathbf{O}_2$, $\mathbf{m}_2 : \mathbf{O}_2 \rightarrow \mathbf{O}_4$ and $\mathbf{m}_3 : \mathbf{O}_4 \rightarrow \mathbf{O}_3$. So, $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$ have compositions to give a path such that $\mathbf{m}_3 \circ (\mathbf{m}_2 \circ \mathbf{m}_1)$ =

$(\mathbf{m}_3 \circ \mathbf{m}_2) \circ \mathbf{m}_1$ (see Figure 3.4 (b)). For every morphism between two objects, e.g. $\mathbf{m}_3 : \mathbf{O}_4 \longrightarrow \mathbf{O}_3$, there exists composition of $\mathbf{m}_3$ with the identity morphisms I_3 , I_4 of objects $\mathbf{O}_3$, $\mathbf{O}_4$ respectively, such that:

$$I_3 \circ \mathbf{m}_3 = \mathbf{m}_3 = \mathbf{m}_3 \circ I_4$$

Let us consider a scenario of honeybees in the category of honeybees for composition and associativity. The interaction from a queen to a worker and from a worker to another worker has an equivalent interaction from the queen to the other worker. Similarly, the path of interactions from queen to the last worker has an equivalent interaction from the queen to that last worker. Figure 3.5 demonstrates this concept with categorical explanation in text to follow.

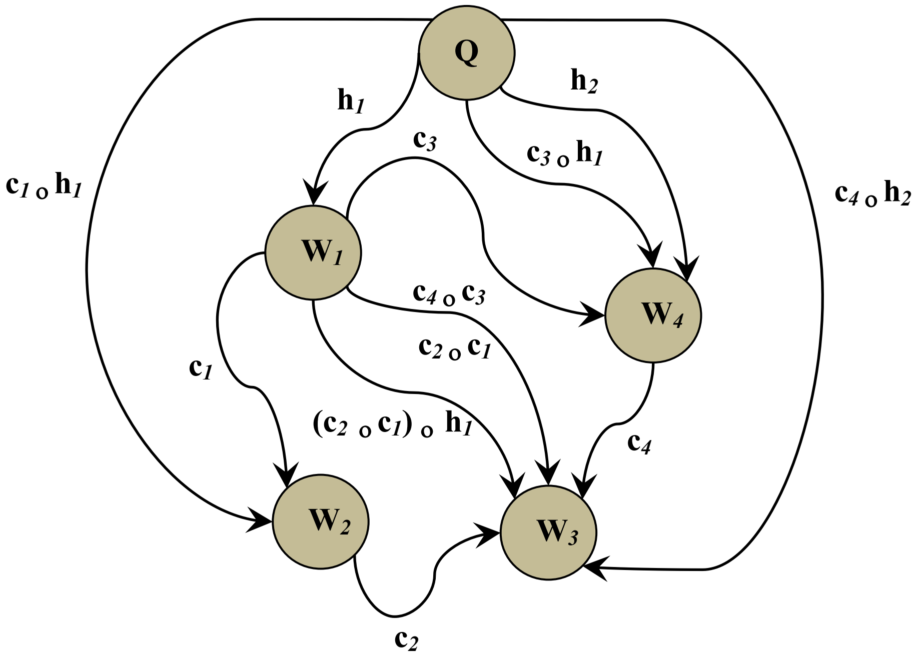


Figure 3.5: Composition of c_1 & c_2, c_3 & c_4, h_1 & c_3, h_1 & c_4 and h_1 & c_1; Associativity for h_1, c_1 & c_2 in Category HB

In category **HB**, the set of objects $\mathbf{O} = \{\mathbf{Q}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4)$ where **Q** is the Queen honey bee object and $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$ worker honeybees. The interaction between the queen bee and the worker bees is of the type honey collection and that between the workers of type cooperation. In the category **HB**, the set of morphisms $\mathbf{m} = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{h}_1, \mathbf{h}_2\}$. The composition morphisms in the category honeybee are $(\mathbf{c}_4 \circ \mathbf{c}_3)$, $(\mathbf{c}_2 \circ \mathbf{c}_1)$, $(\mathbf{c}_4 \circ \mathbf{h}_1)$, $(\mathbf{c}_3 \circ \mathbf{h}_1)$ and $(\mathbf{c}_1 \circ \mathbf{h}_1)$. The associativity property could be easily proved for $(\mathbf{c}_2 \circ \mathbf{c}_1) \circ \mathbf{h}_1 = \mathbf{c}_2 \circ (\mathbf{c}_1 \circ \mathbf{h}_1)$ from the diagram.

3.3 Category Theory Constructs

This section includes definitions of each construct used in the course of this thesis along with adequate examples.

3.3.1 Diagram

Diagram advocates for the concept of a structure within a structure. It is often used for stating and proving properties of other categorical constructs [6]. A diagram consists of a collection of certain objects and morphisms in a category having indices to the parent category. In other words, a diagram consists of objects and morphisms indexed by its parent category. This choice of collection of objects/morphisms is up to the scenario, which considers that structure of a diagram in the parent structure (category). For example, objects $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$, $\mathbf{W}_4$ along with morphisms $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$ form a diagram **D** in category **HB**. A diagram is said to commute when proving properties of a categorical construction. Formally, a diagram **D** in category **HB** is said to commute if all paths between two objects are equal, in the sense that each path in **D** determines a morphism these morphisms are equal in **HB** [6]. Figure 3.6 shows a diagram **D** that commutes in **HB.**

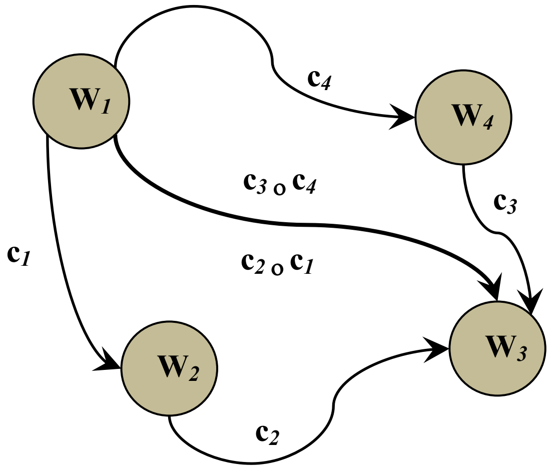


Figure 3.6: Diagram D, Commutes in HB

3.3.2 Functor

A diagram **D** could also be seen as a mapping from of the objects and morphisms inside the diagram to the objects and morphisms inside the parent category **HB** such that the structure in **D** is preserved in **HB**. This kind of mapping of a structure onto another in category theory is performed with the help of the functor construct. Mapping categories of different types with similarity in structures, while preserving the original structures demonstrates the power of category theory to better reason about the behaviours that result from each structure. A functor is shown diagrammatically as an arrow between the source and target category with the name of the functor on top of the arrow.

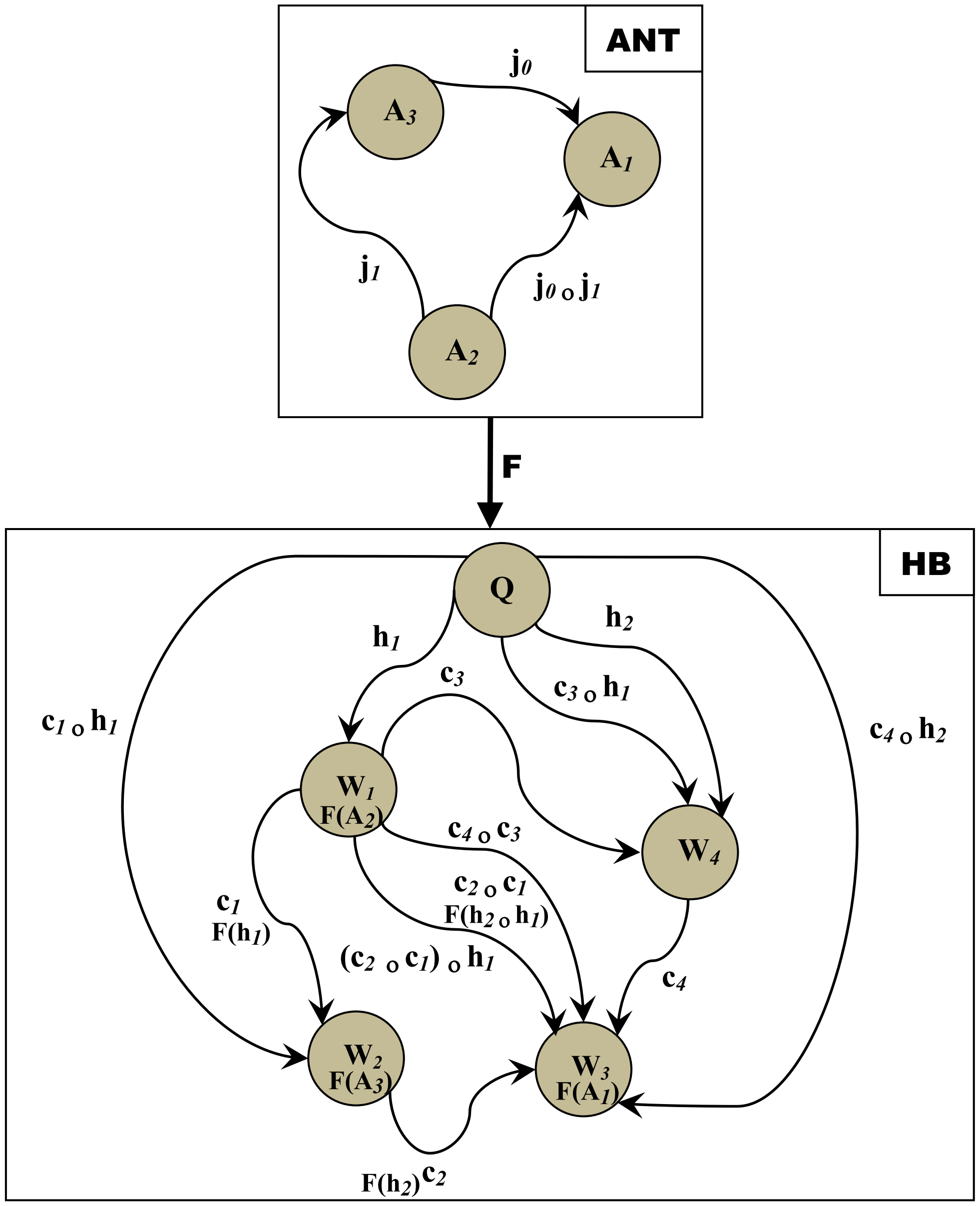


Figure 3.7: Functor F mapping ANT to HB

The name of the functor helps tag or index the mapped objects in target category with the indices of objects and morphisms in the source category. Category ANT in Figure 3.7 shows a scenario of ants working towards achieving a goal. Ants exhibit a social life in the spirit of a hive culture or ants colony and bears similarity with the social life inside a honeybee hive. Figure 3.7 demonstrates the mapping of all objects and morphisms in source category **HB** to a subset of objects and morphisms in target category **ANT** through functor **F**. Preservation of structure of **HB** could be seen in **ANT** through **F**.

3.3.3 Index Category

The diagram and functor constructs give birth to the concept of an Index category. An index category is like a diagram, where its objects and morphisms reflect the structure inside some category but unlike the diagram, the objects and morphisms are labeled/named as indices instead of the labels of its parent category.

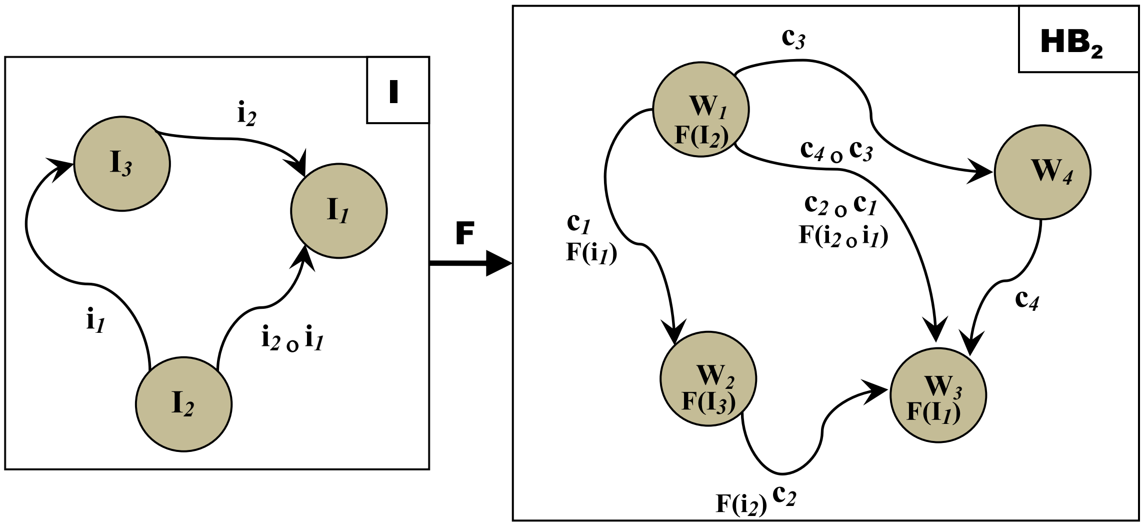


Figure 3.8: Functor F mapping from index category I to target category HB_2

The resulting index category is used in the functor construct as a source category, being mapped to a structure in some target category. An index category could be thought of as a stencil for a structure using which that structure could be traced inside another structure. This makes intuitive relating of two structures with some similarity. This construct comes in handy for reasoning about an occurrence of a behavior in different scenarios. Figure 3.8 shows an index category drawn out using category **HB**, which acts as a source for functor **F**.

3.3.4 Natural Transformation

One of the basic and important notions in category theory is that of a natural transformation. It offers a way to transform a functor into another while preserving the structure of the two categories involved. In other words, a natural transformation could be thought of as morphism of functors [7]. Figure 3.9(a) shows two functors **F**, and **G** from **ANT** category to **HB** category. Figure 3.9(b) shows the natural transformation η from **F** functor to **G** functor in such a way that the natural transformation morphisms are specific to objects in source category mapped to the target category. Therefore, $\eta(A_3)$ is a natural transformation from object A_3 in **HB** mapped by functors **F** and **G**. Similarly, $\eta(A_1)$ is a natural transformation from object A_1 in **HB** mapped by functors **F** and **G** and so on for all objects in source category **ANT** mapped by functors **F** and **G** in the target category **HB**. Evident from it name, a natural transformation occurs naturally and defines the change in an object because of its mapping by two different functors.

The diagram in Figure 3.9(b) commutes such that all directed paths in the diagram with the same endpoints lead to the same result by composition. A natural transformation provides adequate means of reasoning about two structures mapped using different functors.

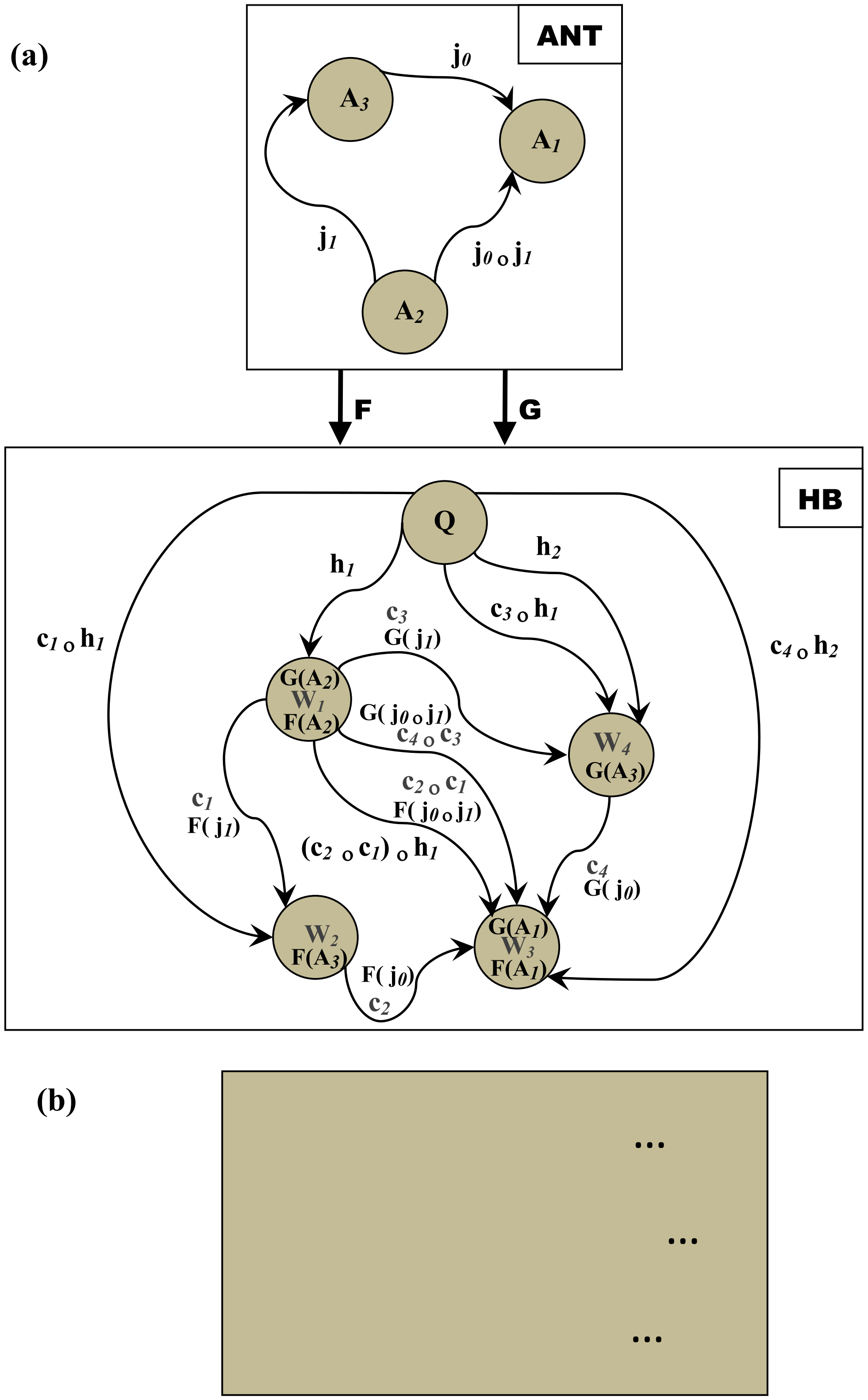


Figure 3.9: (a) Functors F and G from ANT to HB; (b) Natural Transformation η

3.3.5 Cone & Co-cone

The diagram construct is further used in another categorical construct called cone. Co-cone is the inverse or dual of the cone concept. A cone consists of one other object apart from the objects inside a diagram, and additional morphisms going from diagram objects to this other object forming a cone-like shape. This cone object has also been referred to as an apical object in some literature because of its position in the apex of the cone. Figure 3.10(a) shows a cone inside category **HB** where **Q** would be the apical object and $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{h}_3$ morphisms from **Q** to diagram **D**. The morphisms in bold represent apical object or cone morphisms to differentiate from the diagram morphisms.

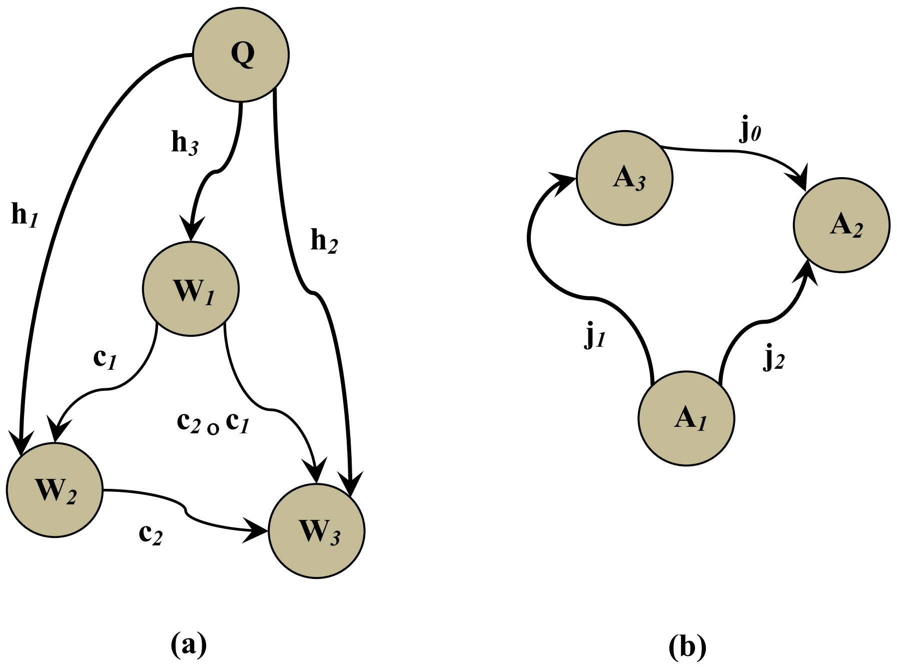


Figure 3.10: (a) Cone; (b) Co-cone

The direction of morphisms from the objects of the diagram to the apical object forms a cone and morphisms coming from the apical object going to the objects inside the diagram form a co-cone. Hence, cone and co-cone are dual of each other. Figure 3.10(b) shows a co-cone in **ANT**. The diagram in this co-cone comprises of objects $\mathbf{A}_2$, $\mathbf{A}_3$ and morphism $\mathbf{j}_0$. The apical object from this co-cone is $\mathbf{A}_1$ with apical morphisms $\mathbf{j}_1$ and $\mathbf{j}_2$ in **ANT**. The cone construct offers a technique to represent the behavior of a group of objects both collectively and individually. The nature of the apical object with its morphisms to and from a group of other objects is similar to that of a host or representation entity for the group with which it forms a structure. This hierarchy of structural representation effectively suits our need for representation of complex structures such as computer networks or autonomic systems.

3.3.6 Limit & Co-limit

A category could possibly house a number of cones and co-cones. Limit is a universal cone such that all other cones factor through it. In this manner, limit is like a specialized cone amongst all cones. Further simplifying the concept, the apical object of the Limit cone has a unique morphism to the apical object of every other cone. Limit construct adds another level to the cone, the limit being on the top. Together this hierarchy results into a very complex structure. The limit construct thus enables representation of an equivalently complex structure therefore modeling the behavior exhibited by that structure. A Limit is diagrammatically represented as shown in Figure 3.11(a). The inverse or dual of a limit is a co-limit. A co-limit makes use of the co-cone construct. A co-limit for that reason is a universal co-cone such that every other co-cone factors through it, or is recognized by a unique morphism from the apical object of the co-limit to every other co-cone. Figure 3.11(b) shows the co-limit in category **ANT**.

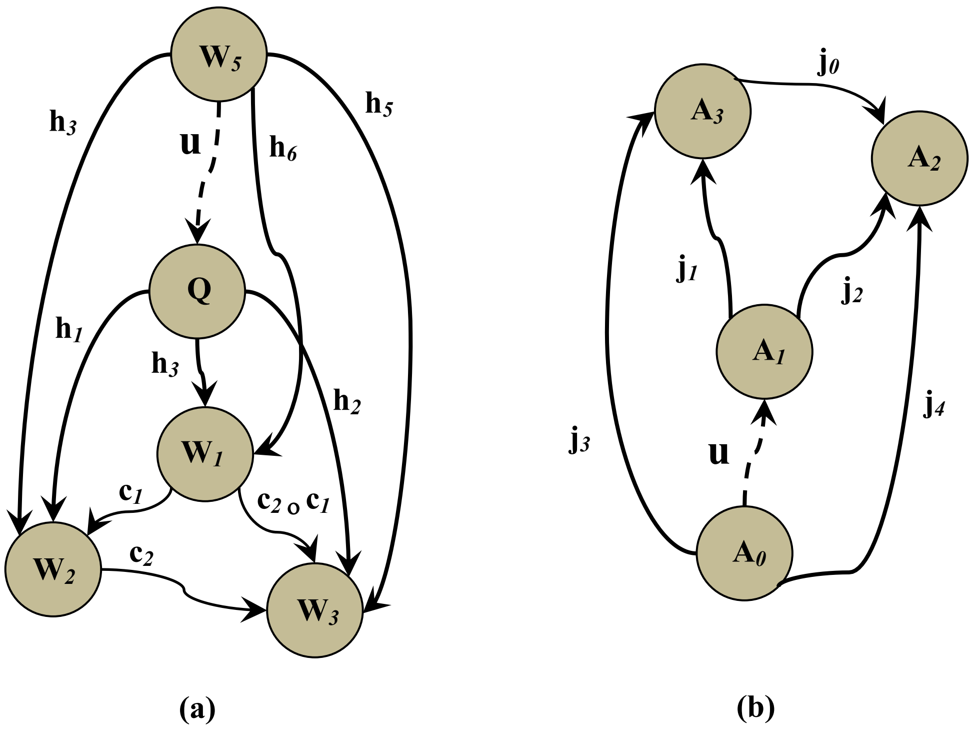


Figure 3.11: (a) Limit; (b) Co-limit

3.3.7 Product & Co-product

Product construct is a special case of limit and a co-product that of a co-limit. Product in category theory follows the idea of the cartesian product of sets. A product construct consists of a limit cone with apical object being the product of objects in diagram or base of the cone such that for any other cone, there exists a unique morphism to the limit cone such that the triangles commute. To better understand the concept, Figure 3.12 shows the diagrammatical product and its dual co-product. Product and co-product in Figure 3.12 represent the cooperation of the entities towards achieving a goal. The product object represents the collective behavior of the involved entities/objects. The product is denoted $\prod_{i \in I} X_i$; if I = {1,…,n}, then, product is $X_1 \Pi \ldots \Pi X_n$. The unique morphism u is the product of morphisms h_1 and h_2.

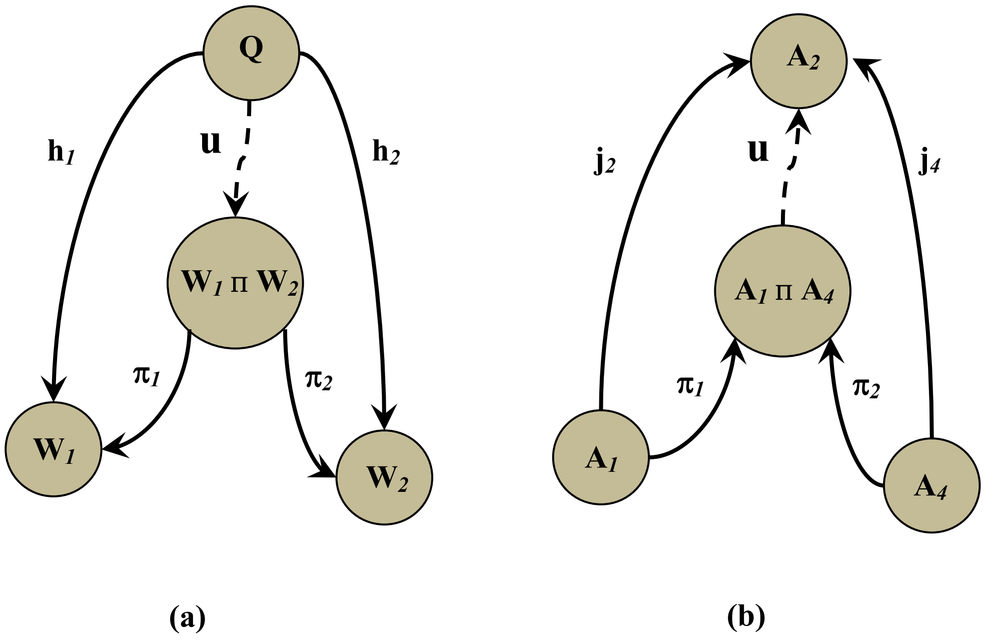


Figure 3.12: (a) Product of objects W_1, W_2; (b) Product of objects A_1, A_4;

3.4 Conclusion

The purpose of this chapter was to present the readers with a precise introduction to the CAT constructs along with examples from biological swarms. This chapter lays the foundation for the grammar discussed in chapter 4 and the case study presented in chapter

5. The readers of these chapters can always refer to chapter 3 in order to refresh the definitions.

4 Categorical Modeling Language

4.1 Introduction

In the previous chapters we presented a discussion of category theory along with a sub set of its constructs that have been studied for this research. We also mentioned in chapter 2 that CAT formalism is not powerful enough to serve as a specification language. A part of this thesis goes into construction of a grammar for what we call the Categorical Modeling language or in short CML. Together with the specification language and the visual/graphical modeling notation, CML will serve as a powerful modeling language, its formal basis coming from the category theory.

4.2 Graphical Models

A very important aspect of the CML is the support of a graphical model with a formal specification. The conventions for the graphical model are an adaptation of the category theory conventions where a circle represents an object in a category and a directed arrow represents a morphism. A category in CML can be represented with a square but specifically in the case where a functor is to be represented between two categories. Otherwise, enclosing objects and morphism of a category in a square limits the available drawing space and is therefore avoided. Functors and natural transformations are represented as arrows too. A functor is a triangle shaped filled arrowhead and for the natural transformation, the arrowhead is an unfilled triangle. A unique morphism is represented with a dashed directed arrow with the name of the morphism represented together with the letter **u** in bold. Table 4.1 lists the graphical/visual notation for CML models. The examples of the visual models could be found in both chapter 5 and chapter 6.

Table 4.1: CML Graphical Notation

Notation	Description
$\bigcirc\!\!\!\!\mathbf{w}_1$	Object
$\xrightarrow{\mathbf{c}_4}$	Morphism
$\boxed{\mathbf{Cat}}$	Category
$\xrightarrow{\mathbf{F}}$	Functor
$\overset{\mathbf{NT}}{\Longrightarrow}$	Natural Transformation
$\xrightarrow{\mathbf{c}_2 \circ \mathbf{c}_1}$	Composition Morphism
$\overset{\mathbf{u}}{\dashrightarrow}$	Universal Morphism

4.3 Formal Specification Language

Along with the graphical modelling tools, CML boasts a formal specification language. A CML specification is constructed using the CML formal grammar. The grammar also serves as a basis for generating the XML for the CML models constructed using the tool discussed in chapter 6. This section introduces the grammar for all of the CML constructs presented in chapter 3. The models constructed using the language are discussed in chapter 5 in detail and would be referenced to wherever required.

4.3.1 Grammar for CML

CML makes use of the Extended Backus-Naur Form (EBNF) for the grammar notation. The grammar can be used to determine the exact syntax for any category construct. An EBNF based grammar consists of "non-terminals" and "terminals." Non-terminals are symbols within a BNF definition, also defined in the grammar. Terminals are endpoints in BNF definition, consisting of CAT keywords. In this section, all non-terminals appear in brackets < > and all terminals appear without brackets.

Table 4.2: CML Grammar Conventions

Attribute	**Description**
<Non-terminal>	Indicates non-terminal symbols
Terminal	Indicates terminal symbols
CONSTRUCT	Terminals in bold face type are reserved words for basic constructs
Construct-Entity	Terminals in bold & italics face are reserve words for parts of a construct
::=	Indicates non-terminal symbol followed by the production rule or expression
|	Vertical bar indicates choice of rules
{ }$^+$	Braces with a plus sign indicates at least one or more
{ }*	Braces with an asterisk indicates zero or more
: ( ) → = **,**	Terminals (For separation)
[ ]	Indicates optional expression
;	Indicates end of line for a production rule
<Type>	<Id>
<*_type_Id>	<Id> (* indicates all)
<*_name>	<Id>
<*_instance_Id>	<Id>
<*_Id>	<Id>
ε	Empty String

Table 4.2 includes a complete list of the grammar conventions for CML grammar along with their description for reference in this chapter and everywhere else in the thesis. The start symbol in the CML grammar corresponds to a list of non-terminals each of which translates to a model in the CML. Sections 4.6 through 4.14 include the grammar for all the symbols in the rule listed in figure 4.1.

```
<Start>::=      <Typed_Category> | <Functor> | <N_Trans>  |
                <Diagram>  | <Cone> | <Co-Cone> | <Limit> |
                <Co-Limit> | <Product> | <Co-Product>;
```

Figure 4.1: CML Grammar Start Symbol

4.4 Grammar Structure & Conventions

This section includes an explanation of the way the grammar has been composed and makes use of the <Typed_Category> grammar for the explanation of the structure and the conventions used in the discussions to follow. The basic structure of all of the non-terminals in the production rule in Figure 4.1 is the same. There are some differences that will be explained along with the description of each grammar. Readers can always refer back to this section to find the underlying structure of all of the grammar to follow.

<Typed_Category> consists of the keyword **TYPED-CATEGORY** followed by the non-terminals for the name of the category and the Id of the category in parenthesis. The keyword ***Types of Objects*** serves as a heading for a list of object types. Enclosing braces with a plus $\{\}^+$ means there should be at least one type defined for objects of this category. The keyword ***Objects*** with the notation for set of objects in the category is

followed by a list of objects inside this category. Similarly, the non-terminals for types of morphism and the list of instance morphisms for each type would be what comes after. Identity morphisms exist for each object in the category and ***Composition*** contains the pairs of composable morphisms. Lastly, the grammar ends with a list of axioms that hold true for the constructed model or in this case the typed category model. Figure 4.2 lists the complete grammar for the non-terminal <Typed_Category> in Figure 4.1.

<Object_Type> consists of the list of the type names and Ids for each type and a list of object type instances with name and Id for each instance. An object type instance adds another level of type definition for expression of a scenario that demands such hierarchy of types. Object_Type_Instances expression is optional in the grammar as it is surrounded by [ ]. <Object> consists of the object type name and Id followed by the instance objects for this type. Similarly, <Morphism_Type> consists of the name of the morphism type followed by a list of morphisms for that type. <Morphism> is <Mor_Instance> that is the list of morphism instances for each morphism type followed by <Mor_Identity>. <Mor_Identity> is the list of Identity morphisms for each object instance in <Object>. <Axiom> consists of all of the properties that must hold true in order to prove the correctness of the models according to CAT. It consists primarily of <Property> that is <Identity> and <Associativity>. <Id> is the symbol for construction of the names and Ids in CML. It consists of one or more characters. The non-terminal <Character> consists of all of the alphabets along with digits from 0 to 9. <Empty> facilitates the termination of a name or Id with an empty space denoted by ε.

```
<Typed_Category>::=    TYPED-CATEGORY <Cat_name>(<Cat_Id>)
                       Types of Objects {<Object_Type>} +
                       Objects: Obj(<Cat_Id>){<Object>} +
                       Types of Morphisms {<Morphism_Type>} +
                       Morphisms: Mor(<Cat_Id>)<Morphism>
                       Composition <Composition>
                       Axioms <Axiom> ;

<Object_Type>::=       Object_Type: <Obj_type_name>(<Obj_type_Id>)
                      [Object_Type_Instances:
                      <Obj_type_name>(<Obj_type_Id>)
                    {,<Obj_type_name>(<Obj_type_Id>)}*] ;

<Object>::=           <Obj_type_Id>:
                      <Obj_type_Id><Obj_instance_Id>
                    {,<Obj_type_Id><Obj_instance_Id>}* ;

<Morphism_Type>::=     Morphism_Type: <Mor_type_name>(<Mor_type_Id>):
                     {<Obj_type_Id> → <Obj_type_Id>}+ ;

<Morphism>::=         <Mor_Instance><Mor_Identity> ;

<Mor_Instance>::=     <Mor_type_Id><Mor_instance_Id>
                      (Obj_type_Id><Obj_instance_Id>) =
                      <Obj_type_Id><Obj_instance_Id>
                      {, <Mor_type_Id><Mor_instance_Id>
                        (<Obj_type_Id><Obj_instance_Id>) =
                         <Obj_type_Id><Obj_instance_Id>}+ ;

<Mor_Identity>::=      Identity: Identity(<Cat_Id>)
                       Id (<Obj_type_Id><Obj_instance_Id>):
                      <Obj_type_Id><Obj_instance_Id> →
                      <Obj_type_Id> <Obj_instance_Id>
                    {, Id (<Obj_type_Id><Obj_instance_Id>):
                           <Obj_type_Id><Obj_instance_Id> →
                           <Obj_type_Id> <Obj_instance_Id>} + ;

<Composition>::=     (<Mor_type_Id> <Mor_instance_Id> o
                      <Mor_type_Id> <Mor_instance_Id>) =
                      <Mor_type_Id> <Mor_instance_Id>
                  {, (<Mor_type_Id> <Mor_instance_Id> o
                      <Mor_type_Id> <Mor_instance_Id>) =
                      <Mor_type_Id> <Mor_instance_Id>} + ;

<Axiom>::=            <Property> ;

<Property>::=          Identity:{<Identity>} +  |
                       Associativity:{<Associativity>}+;

<Identity>::=          ∀ x ∈ Identity(<Cat_Id>), y ∈ Mor(<Cat_Id>),
                       x o y =  y =  y o x ;
```

```
<Associativity>::=    <Mor_type_Id> <Mor_instance_Id> o
                     (<Mor_type_Id> <Mor_instance_Id> o
                      <Mor_type_Id> <Mor_instance_Id>) =
                     (<Mor_type_Id> <Mor_instance_Id> o
                      <Mor_type_Id> <Mor_instance_Id>) o
                      <Mor_type_Id> <Mor_instance_Id> ;

<Id>::=               <Character><Id> | <Empty> ;

<Character>::=        A | B | C |... |Z | a | b | c |... | z |
                      0 | 1 | 2 |... | 9 ;

<Empty>::=            ε ;
```

Figure 4.2: Grammar for Typed-Category Construct

Chapter 5 includes an example of a CML model constructed using <Typed_Category> grammar. Please refer to section 5.2.1 for a detailed discussion with an example from the case study.

4.5 Functor

The grammar for functor construct starts with the symbol <Functor> listed in Figure 4.4. <Functor> comprises of a source and target category constructed using <Typed_Category> grammar followed by the Ids of the source and target categories respectively. This is followed by the keyword **FUNCTOR** and the functor definition in parenthesis. The functor definitions consist of functor type name and Id followed by the Ids of the source and target category. The rest of the grammar structure is same as the <Typed_Category> grammar. The objects and morphisms for the functor, as well as the axioms that hold true for the constructed functor succeed the functor definitions using non-terminals <F_Object>, <F_Morphism> and <F_Axioms> respectively. <F_Object>

is different from <Object> in the sense that it consists of <Obj_mapp> instead of the object type instances in Figure 4.2.

<Obj_mapp> consists of the source category mapping object type and instance Id enclosed in functor Id parenthesis followed by the mapping and mapped objects separated by an arrow. The mapped object is represented with the functor representation of the mapping object. Similarly, <F_Morphism> consists of <Mor_mapp> that is comprised of the mapped and mapping morphisms in the source and target category respectively for all mapped morphisms. The mapping morphism is also represented using the functor.

```
<Functor>::=        Categories: <Typed_Category> <Typed_Category>
                    Category Source: <Cat_Id>
                    Category Target: <Cat_Id>
                    FUNCTOR(<Func_type_name>,<Func_Id>,
                            <Cat_Id>,<Cat_Id>)
                    Functor Objects {<F_Object>}+
                    Functor Morphisms {<F_Morphism>}+
                    Functor Composition {<F_Composition>}+
                    Functor Axioms <F_Axioms> ;

<F_Object>::=      <Obj_type_name>: <Obj_mapp> {, <Obj_mapp>}* ;

<Obj_mapp>::=      <Func_Id>(<Obj_type_Id><Obj_instance_Id>):
                   <Cat_Id>(<Obj_type_Id><Obj_instance_Id>))→ <Cat_Id>
                  (<Func_Id>(<Obj_type_Id><Obj_instance_Id>):
                   <Obj_type_Id><Obj_instance_Id>);

<F_Morphism>::=    <Mor_type_name>: <Mor_mapp> {, <Mor_mapp>}* ;

<Mor_mapp>::=      <Func_Id>(<Mor_type_Id><Mor_instance_Id>):
                   <Cat_Id>(<Obj_type_Id><Obj_instance_Id>,
                   <Obj_type_Id><Obj_instance_Id>,
                   <Mor_type_Id><Mor_instance_Id> ) →
                   <Cat_Id>(<Func_Id>(<Obj_type_Id><Obj_instance_Id>):
                   <Obj_type_Id><Obj_instance_Id>,
                   <Func_Id>(<Obj_type_Id><Obj_instance_Id>):
                   <Obj_type_Id><Obj_instance_Id>,
                   <Func_Id>(<Mor_type_Id><Mor_instance_Id>):
                   <Mor_type_Id><Mor_instance_Id> ) ;
```

```
<F_Composition>::=<Func_Id>(
                             <Mor_type_Id><Mor_instance_Id> o
                             <Mor_type_Id><Mor_instance_Id>) =
                             <Func_Id>(
                             <Mor_type_Id> <Mor_instance_Id>) o
                             <Func_Id>(
                             <Mor_type_Id> <Mor_instance_Id>) =
                             <Func_Id>(
                             <Mor_type_Id> <Mor_instance_Id>) ;

<F_Axioms>::=     <F_Identity> ;

<F_Identity>::=    Identity: {<Func_Id>(
                          Id(<Obj_type_Id><Obj_instance_Id>)) =
                          Id(<Func_Id>(<Obj_type_Id>
                             <Obj_instance_Id>))}+ ;
```

Figure 4.3: Grammar for Functor Construct

<F_Axioms> is similar to the <Axioms> in Figure 4.2 except for the functor representation for the Id and composition morphisms. A model constructed using functor grammar in Figure 4.3 is included in section 5.2.3 as part of the case study.

4.6 Natural Transformation

The start symbol for natural transformation is <N_Trans> as shown in Figure 4.5. <N_Trans> consists of the source and target category constructed using <Typed_Category> followed by the list of functors for the natural transformation. The functors are constructed using <Functor> grammar discussed earlier. After that comes the definition of the natural transformation that consists of the keyword **NAT_TRANSFORMATION** followed by the name and Id of the natural transformation in parenthesis. The keyword ***NTrans Functors*** is followed by the <NT_Functor> which consists of the functor type names followed by the functor type instances. The expression ends with a {}$^+$ indicating the requirement of at least two functors be defined. The keyword ***NTrans Objects*** is followed by the non-terminals <NT_Objects> and <Obj_Mapping>. <NT_Object> consists of the source category followed by a list of mapped objects. <Obj_Mapping> in <N_Trans> consists of the natural transformation Id followed by the object mapped by the two functors in parenthesis.

```
<N_Trans>::=          Categories: <Typed_Category> <Typed_Category>
                      Category Source: <Cat_Id>
                      Category Target: <Cat_Id>
                      Functors: <Functor> <Functor>
                      Functor Ids: <Func_Id>,<Func_Id>
                      NAT_TRANSFORMATION (<NTrans_name>, <NTrans_Id>)
                      NTrans Functors <NT_Functor>
                      NTrans Objects <NT_Object>
                      NTrans Mapping Function {<Obj_Mapping>}+
                      NTrans Morphisms <NT_Morphism>
                      NTrans Axioms <NT_Axioms> ;

<NT_Functor>::=      <Func_type_name>:
                     <Func_name>(<Func_Id>): <Cat_Id>  ⇒ <Cat_Id>
                  {, <Func_type_name>:
                     <Func_name>(<Func_Id>): <Cat_Id>  ⇒ <Cat_Id>}+ ;

<NT_Object>::=       <Cat_Id>:
                     {<Obj_type_name>:<Obj_type_Id><Obj_instance_Id>
                                 { , <Obj_type_Id><Obj_instance_Id}* }+ ;

<Obj_Mapping>::=     <NTrans_Id>(<Obj_type_Id><Obj_instance_Id>):
                     <Func_Id>(<Obj_type_Id><Obj_instance_Id) ⇒
                     <Func_Id>(<Obj_type_Id><Obj_instance_Id) ;

<NT_Morphism>::=    {<Mor_type_Id>:<NT_Arrows>}+ ;

<NT_Arrows>::=       <Func_Id>(<Mor_type_Id><Mor_instance_Id>):
                     <Func_Id>(<Obj_type_Id><Obj_instance_Id) →
                     <Func_Id>(<Obj_type_Id><Obj_instance_Id)
                  {, <AT_Arrows> }* ;

<NT_Axioms>::=        Commutativity:

                      ∀ x,y ∈ Obj(<Cat_Id>), f : x → y ∈ Mor(<Cat_Id>),
                     <Func_Id>(f) o <NTrans_Id>(x)  =
                     <NTrans_Id>)(y) o <Func_Id>(f ) ;
```

Figure 4.4: Grammar for Natural Transformation Construct

This is followed by the same object represented in terms of the functors separated by an arrow ⇒. <NT_Morphism> consists of name of the source category for all the functors followed by <NT_Arrows> that comprises of all of the morphisms between the mapped objects in the source category represented using functors. The | symbol indicate the alternate rule for <NT_Arrows> that is used to construct the identity arrows for all NTrans objects. The grammar ends with <NT_Axioms> that comprises of commutativity for every pair of natural transformation mapping function and morphism.

The model constructed using the grammar in Figure 4.4 is given in section 5.2.4 as part of the PAM case study.

4.7 Diagram

The 'Diagram' construct as discussed in section 3.3.1 is mostly used for stating and proving properties of other categorical constructs. The grammar listed in Figure 4.5 for the 'Diagram' construct would in turn be a part of the grammar for <Cone>, <Co-cone>, <Limit> and <Co-Limit>. The start symbol <Diagram> consists of the grammar for the index category, the grammar for the target category followed by the Ids of the two categories. After that comes the keyword **DIAGRAM** followed by the definition of the diagram in parenthesis. The definition consists of the name, and Id of the diagram and the Ids of the index and target categories respectively. Then comes the typical listing of the objects of the diagram using the keyword ***Diagram Objects*** followed by one or more objects constructed using <D_Object>.

```
<Diagram>::=          Categories: <Typed_Category> <Typed_Category>
                      Index Category: <Cat_Id>
                      Target Category: <Cat_Id>
                      DIAGRAM (<Diag_Id>,<Cat_Id>,<Cat_Id>)
                      Diagram Objects {<D_Object>}+
                      Diagram Morphisms {<D_Morphism>}+ ;

<D_Object>::=        <Obj_type_name>: <Obj_indexing>
                                  {, <Obj_indexing>}* ;

<Obj_indexing>::=    <Diag_Id>(<Vertex_index_Id>):
                     <Cat_Id>(<Vertex_index_Id>) →
                     <Cat_Id>(<Diag_Id> (<Obj_type_Id>
                     <Obj_instance_Id>)) ;

<D_Morphism>::=      <Mor_type_name>: <Mor_indexing>
                                  {, <Mor_indexing>}* ;

<Mor_indexing>::=    <Diag_Id>(<Edge_index_Id>):
                     <Cat_Id>(<Vertex_index_Id>,
                     <Vertex_index_Id>,<Edge_index_Id> →
                     <Cat_Id>(<Diag_Id>(
                     <Obj_type_Id><Obj_instance_Id>),
                     <Diag_Id>(<Obj_type_Id><Obj_instance_Id>)) ;
```

Figure 4.5: Grammar for Diagram Construct

<D_Object> consists of the keyword Object_Type followed by the type name, a colon and one or more <Obj_indexing> separated by a comma ‘,’. <Obj_indexing> comprises of the diagram Id followed by the id of the index vertex in parenthesis. The mapping of the index object follows this from the index category to the target category, the mapping separated by an arrow and the diagram index and the mapping separated by a colon ‘:’. The grammar ends with the keyword Diagram Morphisms followed by one or more morphisms in the diagram constructed using <D_Morphism>. <D_Morphism> consists of the keyword Morphism_Type followed by a colon and the morphisms type Id and another colon followed by one or more indexing morphisms constructed using <Mor_indexing>. <Mor_indexing> consists of the diagram representation of the index

edge Id followed by a three-tuple index category representation and a three-tuple target category representation separated by an arrow. The three-tuple for index category consists of the index category Id followed by the index source vertex, the index target vertex and the edge index Id enclosed in parenthesis. Similarly, the three-tuple for the target category consists of the target category Id followed by the source object, the target object and the name of the morphism enclosed in parenthesis.

An example of a model constructed using the <Diagram> grammar is included in section 5.2.2 as part of the PAM case-study. Readers are suggested to refer back to this section for explanation of the grammar.

4.8 Cone

The grammar for the Cone construct discussed in section 3.3.5 is listed in Figure 4.6. The start symbol <Cone> consists of the grammar for the category using the symbol <Typed_Category> followed by the grammar for the diagram in the category. This is followed by the keyword ***Category Id*** and the category Id separated by a colon ':' and the keyword ***Diagram Id*** followed by the Id of the diagram. The keyword **CONE** indicates the start of the definition for cone followed by the non-terminal <Cone_Obj>. <Cone_Obj> consists of the type and instance Id of the object in the cone external to the diagram. This is followed by a list of diagram objects for the cone constructed using <C_Object>. <C_Object> consists of the diagram Id followed by the index category vertex Id in parenthesis for all diagram objects in the cone. The grammar ends with the keyword ***Cone Morphisms*** followed by the non-terminal <C_Morphism> for all morphism types.

```
<Cone>::=               Diagram: <Diagram>
                        Category Id: <Cat_Id>
                        Diagram Id:  <Diag_Id>
                        CONE (Object:<Cone_Obj>)
                        Cone Objects <C_Object>
                        Cone Morphisms {<C_Morphism>}+ ;

<Cone_Obj>::=           <Obj_type_Id><Obj_instance_Id> ;

<C_Object>::=           <Diag_Id>(<Vertex_index_Id>)
                     {, <Diag_Id>(<Vertex_index_Id>)}* ;

<C_Morphism>::=         <Mor_type_name>: <Cone_Obj>(<Vertex_index_Id>):
                        <Cone_Obj> → <Diag_Id>(<Vertex_index_Id>)
                     {, <Cone_Obj>(<Vertex_index_Id>):
                        <Cone_Obj> → <Diag_Id>(<Vertex_index_Id>)}* |
                        <Mor_type_name>: <Diag_Id>(<Edge_instance_Id>):
                        <Diag_Id>(<Vertex_index_Id>) →
                        <Diag_Id>(<Vertex_index_Id>)
                     {, <Diag_Id>(<Edge_instance_Id>):
                        <Diag_Id>(<Vertex_index_Id>) →
                        <Diag_Id>(<Vertex_index_Id>)}* ;
```

Figure 4.6:Grammar for Cone Construct

<C_Morphism> has two rules as indicated by the '|'. <C_Morphism> for all cone morphisms, comprises of the morphism type name followed by colon preceding the cone object type and instance Id with the vertex Id in parenthesis. This is succeeded by the morphism source object and morphism target object separated by an arrow. <C_Morphism> for all diagram morphisms, comprises of the morphism type name followed by colon preceded diagram Id and edge instance Id in parenthesis. This is followed by the source and target objects in the diagram for this morphism separated by

an arrow. For an example of a model constructed using grammar listed in Figure 4.6, please refer to section 5.2.5, Figure 5.5. The example is part of the PAM case study.

4.9 Co-Cone

The grammar for co-cone is very similar to that of a cone except for some significant differences. Figure 4.7 includes the grammar for constructing a model based on a co-cone construct. The start symbol <Co-Cone> has a similar pattern to <Cone> except for the difference in name and rules of some non-terminals. This section is going to discuss only these differences. <Co-Cone> consists of the grammar for <Typed_Category>, and <Diagram> followed by the Ids of the category and diagram respectively.

```
<Co-Cone>::=           Diagram: <Diagram>
                       Category Id: <Cat_Id>
                       Diagram Id:  <Diag_Id>
                       CO-CONE (Object: <Co-Cone_Obj>)
                       Co-Cone Objects <CC_Object>
                       Co-Cone Morphisms {<CC_Morphism>}+ ;

<Co-Cone_Obj>::=      <Obj_type_Id><Obj_instance_Id> ;

<CC_Object>::=        <Diag_Id>(<Vertex_index_Id>)
                   {, <Diag_Id>(<Vertex_index_Id>)}* ;

<CC_Morphism>::=      <Mor_type_name>: <Co-Cone_Obj><Vertex_index_Id>):
                      <Diag_Id>(<Vertex_index_Id>) → <Co-Cone_Obj>
                   {, <Co-Cone_Obj>(<Vertex_index_Id>):
                      <Diag_Id>(<Vertex_index_Id>) → <Co-Cone_Obj>}* |
                      <Mor_type_name>: <Diag_Id>(<Edge_instance_Id>):
                      <Diag_Id> (<Vertex_index_Id>) →
                      <Diag_Id> (<Vertex_index_Id>)
                   {, <Diag_Id> (<Edge_instance_Id>):
                      <Diag_Id> (<Vertex_index_Id>) →
                      <Diag_Id> (<Vertex_index_Id>)}* ;
```

Figure 4.7: Grammar for Co-Cone Construct

The definition of a co-cone begins with the keyword **CO-CONE** followed by the <Co-Cone_Obj>. <Co-Cone_Obj> is similar to <Cone_Obj> except for the name difference. This is followed by the keyword ***Co-Cone Objects*** followed by the objects in the diagram constructed using <CC_Object>. <CC_Object> is similar to <C_Object> except for the name difference. The grammar ends with the keyword ***Co-Cone Morphisms*** followed by the non-terminal <CC_Morphism> for all morphism types. <CC_Morphism> is different from <C_Morphism> and thus differentiates a cone from a co-cone.

<CC_Morphism> has two rules as indicated by the '|'. <CC_Morphism> for all co-cone morphisms, comprises of the morphism type name followed by colon preceding the co-cone object type and instance Id with the vertex Id in parenthesis. This is succeeded by the morphism source object and morphism target object separated by an arrow. The source object in case of a co-cone is the vertex in the diagram unlike the source object in a cone, which is the cone object. <CC_Morphism> for all diagram morphisms, comprises of the morphism type name followed by colon preceding the diagram Id and edge instance Id in parenthesis. This is followed by the source and target objects in the diagram for this morphism separated by an arrow. An example model constructed using the <Co-Cone> grammar is included in section 5.2.5 Figure 5.6. The example is part of the PAM case study.

4.10 Limit

The grammar for the limit construct is given in Figure 4.8 with the start symbol being <Limit>. <Limit> comprises of the grammar for the typed category and the diagram for the limit along with the keywords ***Category*** and ***Diagram***. This is followed by the keyword ***Cones*** along with the grammar for all cones in the typed category. After that comes the keyword ***Category Id*** with the Id of the category, the keyword ***Diagram Id*** with the diagram Id and the keyword ***Cone Ids*** with the ids of all of the cones of the diagram in the category.

```
<Limit>::=               Diagram: <Diagram>
                         Cones: <Cone> {<Cone>}+
                         Category Id: <Cat_Id>
                         Diagram Id: <Diag_Id>
                         Cone Ids: <Obj_type_Id><Obj_instance_Id>
                                {,<Obj_type_Id><Obj_instance_Id>}+
                         LIMIT (Terminal Object: <Terminal_Obj>,
                               Unique Morphism(u): <Mor_Unique>)
                         Limit Objects <L_Object>
                         Limit Morphisms {<L_Morphism>}+
                         Limit Axioms {<L_Axiom>}+ ;

<Terminal_Obj>::=        <Obj_type_Id><Obj_instance_Id> ;

<Mor_Unique>::=          <Mor_type_Id><Mor_instance_Id>:
                         <Cone_Obj> → <Terminal_Obj>;

<L_Object>::=            <Diag_Id>(Vertex_index_Id>)
                      {, <Diag_Id>(Vertex_index_Id>)}* ;

<L_Morphism>::=          <Mor_type_name>:<Cone_Obj>(<Vertex_index_Id>):
                         <Cone_Obj> → <Diag_Id>(<Vertex_index_Id>)
                      {, <Cone_Obj>(<Vertex_index_Id>):
                         <Cone_Obj>  → <Diag_Id>(<Vertex_index_Id>)}*  |
                         <Mor_type_name>:<Diag_Id>(<Edge_instance_Id>):
                         <Diag_Id>(<Vertex_index_Id>) →
                         <Diag_Id>(<Vertex_index_Id>)
                      {, <Diag_Id>(<Edge_instance_Id>):
                         <Diag_Id>(<Vertex_index_Id>) →
                         <Diag_Id>(<Vertex_index_Id>)}* ;

<L_Axiom>::=             <Mor_Unique> o <Cone_Obj>(<Vertex_index_Id>) =
                         <Cone_Obj>(<Vertex_index_Id>) ;
```

Figure 4.8: Grammar for Limit Construct

The definition of the limit follows this with the keyword **LIMIT** followed by the non-terminal <Terminal_Obj> and <Mor_Unique> separated by a comma ‘,’ and enclosed in parenthesis. <Terminal_Obj> consists of the object type Id and object instance Id and <Mor_Unique> consists of the morphisms type Id followed by a colon separating the arrow from source object that is the cone object to the target object that is terminal object. This is followed by the keyword ***Limit Objects*** along with the non-terminal <L_Object> for all limit objects. <L_Object> consists of the diagram Id followed by the index category vertex Id. The keyword ***Limit Morphisms*** is followed by <L_Morphism> for all morphism types. <L_Morphism> is similar to <C_Morphism> with the cone being the limit cone. The grammar ends with axioms that hold true for the limit indicated with the keyword ***Limit Axioms*** followed by the symbol <L_Axiom>. <L_Axiom> comprises of the composition of the unique morphism with the cone morphism to be equal to the cone morphism for all cones. Section 5.2.6, Figure 5.7 includes an example of a model constructed using the grammar <Limit>. The example is part of the PAM case study. Readers may refer back to this section for clarity of the constructed model.

4.11 Co-Limit

The grammar for the co-limit construct is given in Figure 4.9. The grammar is very similar to the grammar for the limit construct with a few significant differences. Please read section 3.5.6 to find the differences between the two constructs. For the grammar, the start symbol for a co-limit is <Co-Limit>. <Co-Limit> consists of the typed category or index category grammar followed by the grammar for the diagram in the category along with the respective keywords ***Category*** and ***Diagram***.

```
<Co-Limit>::=           Diagram: <Diagram>
                        Co-Cones: <Co-Cone> {<Co-Cone>}+
                        Category Id: <Cat_Id>
                        Diagram Id: <Diag_Id>
                        Co-Cone Ids: <Obj_type_Id><Obj_instance_Id>
                                {,<Obj_type_Id><Obj_instance_Id>}+
                        CO-LIMIT (Initial Object: <Initial_Obj>,
                                  Unique Morphism(u): <CL_Mor_Unique>)
                        Co-Limit Objects <CL_Object>
                        Co-Limit Morphisms {<CL_Morphism>}+
                        Co-Limit Axioms {<CL_Axiom>}+ ;

<Initial_Obj>::=        <Obj_type_Id><Obj_instance_Id> ;

<CL_Mor_Unique>::=      <Mor_type_Id>: <Initial_Obj> →
                        <Cone_Obj>(<Vertex_index_Id>) ;

<CL_Object>::=          <Diag_Id>(Vertex_index_Id>)
                     {, <Diag_Id>(Vertex_index_Id>)}* ;

<CL_Morphism>::=        <Mor_type_name>:
                        <Co-Cone_Obj>(<Vertex_index_Id>):
                        <Diag_Id>(<Vertex_index_Id>) → <Co-Cone_Obj>
                     {, <Co-Cone_Obj>(<Vertex_index_Id>):
                        <Diag_Id>(<Vertex_index_Id>) → <Co-Cone_Obj>}* |
                        <Mor_type_name>:<Diag_Id>(<Edge_instance_Id>):
                        <Diag_Id>(<Vertex_index_Id>) →
                        <Diag_Id>(<Vertex_index_Id>)
                     {, <Diag_Id>(<Edge_instance_Id>):
                        <Diag_Id>(<Vertex_index_Id>) →
                        <Diag_Id>(<Vertex_index_Id>)}* ;

<CL_Axiom>::=          <Co-Cone_Obj>(<Vertex_index_Id>) o <Mor_Unique> =
                       <Co-Cone_Obj>(<Vertex_index_Id>) ;
```

Figure 4.9: Grammar for Co-Limit Construct

This is succeeded by the grammar for all co-cones for the diagram with the keyword ***Co-Cones***. The definition of the co-limit starts with the keyword **CO-LIMIT** followed by <Initial_Obj> and <Mor_Unique> separated by a comma ‘,’ enclosed in parenthesis. This is followed by the keyword Co-Limit objects and the non-terminal <CL_Object>. <CL_Object> is similar to <CC_Object> for all co-limit objects. The keyword ***Co-Limit Morphisms*** consists of <CL_Morphism> for all morphism types. <CL_Morphism> is similar to <CC_Morphism> with the co-cone being the co-limit co-cone. The grammar ends with axioms that hold true for the co-limit indicated with the keyword ***Co-Limit Axioms*** followed by the symbol <CL_Axiom>. <CL_Axiom> comprises of the composition of the unique morphism with the co-cone morphism to be equal to the co-cone morphism for all co-cones. The example for the co-limit construct is included in section 5.2.7, Figure 5.8 as part of the PAM case study. Readers are suggested to refer back to this section for questions on the constructed model based on the grammar.

4.12 Product

The grammar for the product construct starts with the symbol <Product> as shown in Figure 4.10.

```
<Product>::=           Category: <Typed_Category>
                       Category Id: <Cat_Id>
                       PRODUCT(<Prod_name>,<Cat_Id>)
                       Product Objects
                       {<Obj_type_name>: <Prod_Obj>{,<Prod_Obj>}+
                        Product:<P_Obj> }+
                       Product Morphisms {<Prod_Morphism>}+
                       Product Axioms {<Prod_Axiom>}+ ;

<Prod_Obj>::=         <Obj_type_Id><Obj_instance_Id> ;

<P_Obj>::=          < <Obj_type_Id><Obj_instance_Id> Π
                      <Obj_type_Id><Obj_instance_Id> > ;
```

```
<Prod_Morphism>::=    <Mor_type_name>:<Mor_type_Id>:
                                     <Obj_P> → <Prod_Object>  |
                      <Mor_type_name>:<Mor_Unique>:
                                     <Prod_Object> → <Obj_P> ;

<Prod_Axiom>::=       Composition:
                      <Mor_Unique > o <Mor_type_Id> = <Mor_type_Id> ;
```

Figure 4.10: Grammar for Product Construct

<Product> consists of the keyword ***Category*** followed by the grammar for the typed category. This is succeeded by the keyword ***Category Id*** along with the Id of the category constructed using the grammar for typed category. The definition of the product construct begins with the keyword **PRODUCT** followed by the name of the product and Id of the category separated by a comma ‘,’ enclosed in parenthesis. The keyword ***Product Objects*** is followed by the object type name and list of <Prod_Obj> for all object types.

4.13 Conclusion

This chapter presents the grammar for the different constructs of CML along with a detailed explanation of the production rules in the grammar. The chapter does not include any example specification to avoid repetition. The examples are a part of the case study in chapter 6 and have been referred to with all the grammar constructs.

5 Case Study: PAM

In chapter 1, we discussed the different objectives of this research. Application of the category theory and CML to NASA's Prospecting Asteroid Mission was amongst the primary objectives. This chapter includes an introduction of the PAM, the mission's core goals followed by a discussion on modeling of some of the PAM scenarios using CML.

5.1 Prospecting Asteroid Mission

We have already seen an introductory discussion of PAM in chapter 1 and 2. This section is going to elicit the mission's details relevant to the modeling exercises included in section 5.2. The case study is based on the operational characteristics and mission scenarios of the PAM as discussed in [15], [16], [18], [47], [48], [51], [52], [53] and [54].

5.1.1 Asteroid Exploration and PAM Sciencecrafts

The ANTS based PAM mission is an advanced mission concept for the 2020s. Its primary objective is the exploration of the resource potential of the solar system's asteroid belt beyond Mars. The availability of these resources would facilitate uninterrupted presence of humans in space. The asteroid belt consists of thousands of individual asteroids widely separated across the belt. To target these thousands of distinct asteroids, a large group of specialized autonomous workers are required. The concept of PAM is directed towards this requirement of individual workers carrying out a systematic study of the entire population of asteroids. In PAM, these individual workers materialize themselves in the form of intelligent and autonomous bodies termed as Spacecraft or Sciencecraft. Each of these Sciencecraft have specialized instrument capability and heuristics systems that are both evolvable and adaptable.

We saw in chapter 2 and 3 that ANTS is analogous to the social insect swarms. These Sciencecraft operate in the form of subswarms for the purpose of gathering the measurements of their target asteroids. A PAM swarm consists of 1000 such spacecraft based on the carbon-based NEMS [50] technology utilizing Super Miniaturized Addressable Reconfigurable Technology (SMART) [49].

5.2 PAM Swarm

The swarm comprises of science specialist classes with approximately 100 members in each class. The members are identical except for each carrying a specialized 'instrument'.

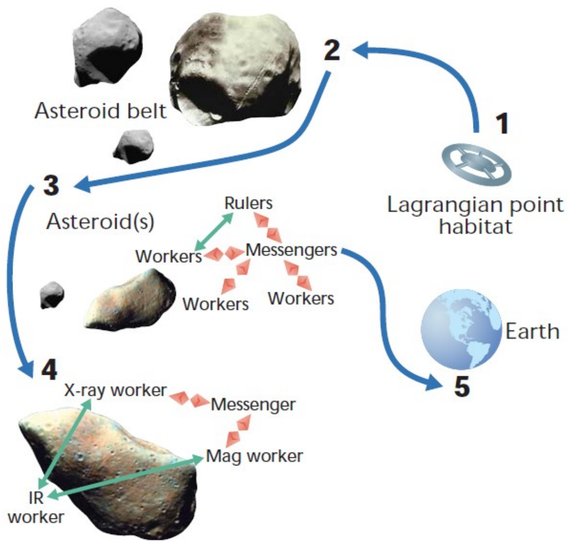


Figure 5.1: ANTS based PAM Concept

The classes of spacecraft include the processors or the CPU known as Rulers, the communication spacecraft known as Messengers, and sciencecraft or Workers including imagers, various spectrometers, altimeters, radio science, and magnetometers. The swarm consists of subswarms with approximately same number of classes in each subswarm. All subswarms inside a swarm operate in parallel. Figure 5.1 provides an overview of the PAM mission concept [15].

The first phase of a PAM mission is to travel from Earth's Lagrange point along with the rest of the swarm. Next, after having raised their orbits to a certain AU level, the Messengers and Rulers position themselves to provide communications and control to the swarm. Workers set about their jobs of detecting and obtaining information about Main Belt asteroids. Some Workers work alone, others are continually forming 'Virtual Teams' [51, 52] to perform science encounters including orbital operations. 'Virtual Instrument Teams' would be formed from those within each class, in order to optimize the accumulation of the data. On occasion, the PAM swarm will send a representative back to Earth or another communication node to report on swarm findings.

5.2.1 CML Model for PAM Sub-swarm Organization

Figure 5.2 (a) and (b) include the CML specification and the visual model respectively for a typical PAM sub-swarm organization. The model is constructed using the typed category construct grammar given in section 4.2 and captures only a part of a typical PAM sub-swarm organization. This model doesn't consider workers' working alone rather depicts organization of workers in the form of a team. The Messengers are represented at two levels of hierarchy, one for the sub-swarms and the other at the team level. The Ruler is present at the swarm level and has interaction with messengers and workers. The typed-category model given in Figure 5.2 obeys all the axioms for it to qualify as a category, i.e. composition, associativity and identity properties.

TYPED-CATEGORY

PAM Sub-swarm (S_1)

Types of Objects

Object_Type:	Ruler (R)
Object_Type_Instances:	Leader (L)
Object_Type:	Messenger (M)
Object_Type_Instances:	Team Messenger (TM), Sub-Swarm Messenger (SM)
Object_Type:	Worker (W)
Object_Type_Instances:	X-Ray (W_{XR}), Gamma Ray(W_{GR}), Infra-Red(W_{IR}), Altimeter(W_{AL})

Objects: Obj(S_1)

R: L_1

TM: TM_1, TM_2

SM: SM_1

W_{XR}: W_{XR1}, W_{XR3}

W_{GR} : W_{GR1}

W_{IR} : W_{IR1}

W_{IR} : W_{AL1}

Type of Morphisms

Morphism_Type: Management (m):

$L \rightarrow TM$, $L \rightarrow SM$, $L \rightarrow W$

Morphism_Type: Cooperation (c):

$W \rightarrow W$, $TM \rightarrow W$, $W \rightarrow TM$

Morphism_Type: Communication (cu):

$TM \rightarrow SM$, $TM \rightarrow TM$, $W \rightarrow SM$, $TM \rightarrow L$

Morphisms: Mor(S_1)

m_1 (L_1) = TM_2, m_2 (L_1) = SM_1,

m_3 (L_1) = W_{XR1}, m_4 (L_1) = W_{GR1},

m_5 (L_1) = W_{IR1}, m_7 (L_1) = TM_1,

c_1 (W_{XR1}) = W_{GR1}, c_2 (TM_2) = W_{XR1},

c_3 (W_{IR1}) = W_{GR1}, c_4 (TM_2) = W_{IR1},

c_5 (TM_2) = W_{GR1}, c_6 (W_{AL1}) = TM_1,

c_7 (W_{XR3}) = TM_1, c_8 (W_{XR3}) = W_{AL1},

cu_1 (TM_2) = TM_1, cu_2 (TM_1) = SM_1,

cu_3 (TM_2) = SM_1, cu_4 (W_{XR3}) = SM_1,

cu_5 (W_{AL1}) = SM_1, cu_8 (TM_2) = L_1

Identity: Identity(S_1)

Id(L_1): $L_1 \rightarrow L_1$, *Id*(SM_1): $SM_1 \rightarrow SM_1$,

Id(TM_1): $TM_1 \rightarrow TM_1$, *Id*(TM_2): $TM_2 \rightarrow TM_2$,

Id(W_{XR1}): $W_{XR1} \rightarrow W_{XR1}$, *Id*($W_{XR3}$): $W_{XR3} \rightarrow W_{XR3}$,

Id(W_{AL1}): $W_{AL1} \rightarrow W_{AL1}$, *Id*($W_{GR1}$): $W_{GR1} \rightarrow W_{GR1}$,

Id(W_{IR1}): $W_{IR1} \rightarrow W_{IR1}$

Composition	***Axioms***
$(c_3\ o\ m_5) = m_4$, $(c_1\ o\ m_3) = m_4$, $(c_4\ o\ m_1) = m_5$, $(c_2\ o$	Identity: $\forall\ x \in \mathit{Identity}(S_1)$, $y \in \mathit{Mor}(S_1)$,
$m_1) = m_3$, $(c_1\ o\ c_2) = c_5$, $(cu_1\ o\ m_1) = m_7$,	$x\ o\ y = y = y\ o\ x$
$(cu_2\ o\ m_7) = m_2$, $(m_2\ o\ cu_8) = cu_3$,	Associativity: $c_1\ o\ (c_2\ o\ m_1) = (c_1\ o\ c_2)\ o\ m_1$
$(m_3\ o\ cu_8) = c_2$, $(m_4\ o\ cu_8) = c_5$, $(m_5\ o\ cu_8) = c_4$,	$c_3\ o\ (c_4\ o\ m_1) = (c_3\ o\ c_4)\ o\ m_1$
$(cu_2\ o\ c_7) = cu_4$, $(cu_5\ o\ c_8) = cu_4$, $(c_6\ o\ c_8) = c_7$, $(cu_2$	$c_1\ o\ (m_3\ o\ cu_8) = (c_1\ o\ m_3)\ o\ cu_8$
$o\ c_6) = cu_5$, $(c_3\ o\ c_4) = c_5$, $(cu_2\ o\ cu_1) = cu_3$, $(cu_3\ o$	$c_3\ o\ (m_5\ o\ cu_8) = (c_3\ o\ m_5)\ o\ cu_8$
$m_1) = m_2$, $(m_7\ o\ cu_8) = cu_1$	$cu_2\ o\ (cu_1\ o\ m_1) = (cu_2\ o\ cu_1)\ o\ m_1$
	$cu_2\ o\ (m_7\ o\ cu_8) = (cu_2\ o\ m_7)\ o\ cu_8$
	$cu_2\ o\ (c_6\ o\ c_8) = (cu_2\ o\ c_6)\ o\ c_8$

Figure 5.2: CML Specification Model of a PAM Swarm Scenario

Figure 5.3 shows the graphical model for the specification in Figure 5.2.

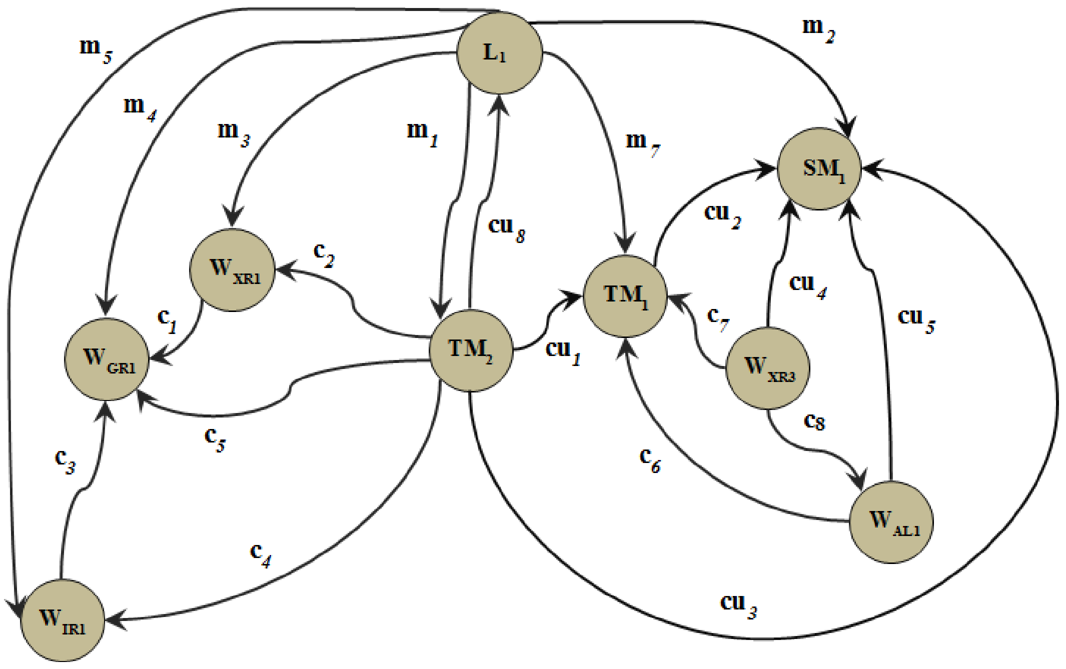


Figure 5.3: CML Graphical Model of a PAM Swarm Scenario

5.2.2 CML Model for PAM Team Organization

A typical PAM Petrologist consists of an X-ray worker, a Near Infrared worker, a Gamma-ray worker, a Thermal IR worker, and a wide field imager worker separated by tens of kilometers. The target or goals of the Petrologist team include determination of the abundances and distribution of elements, minerals, and rocks present, from which the nature of geochemical differentiation, origin, and history of the object, and its relationship to a 'parent body' could be inferred. The team also has one worker acting as a team messenger, which both communicates and cooperates with the sciencecraft. The team is part of a sub-swarm that has a ruler and a messenger spacecraft outside the team level. This section includes a model of the Petrologist team constructed using the Diagram construct grammar in section 4.7. Figure 5.4 includes the CML graphical model for PAM Petrologist team organization scenario.

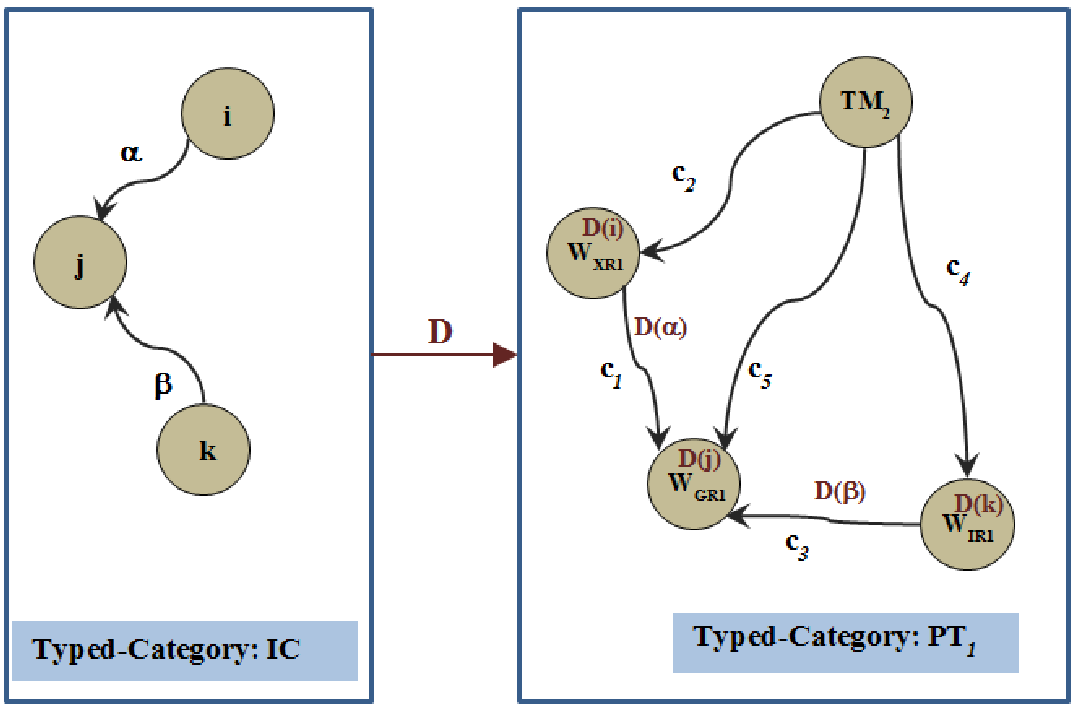


Figure 5.4: CML Graphical Model of the Petrologist Team Organization Scenario

Categories:

TYPED-CATEGORY

Petrologist Team (PT_1)

Types of Objects

Object_Type:	Messenger (M)
Object_Type_Instances:	Team Messenger (TM),
Object_Type:	Worker (W)
Object_Type_Instances:	X-Ray (W_{XR}), Gamma Ray(W_{GR}), Infra-Red(W_{IR})

Objects: Obj(PT_1)

TM: TM_2
W_{XR}: W_{XR1}
W_{GR} : W_{GR1}
W_{IR} : W_{IR1}

Type of Morphisms

Morphism_Type: Cooperation (c):

$W \rightarrow W$, $TM \rightarrow W$

Morphisms: Mor(PT_1)

$c_1 (W_{XR1}) = W_{GR1}$, $c_2 (TM_2) = W_{XR1}$,
$c_3 (W_{IR1}) = W_{GR1}$, $c_4 (TM_2) = W_{IR1}$,
$c_5 (TM_2) = W_{GR1}$

Identity: Identity(PT_1)

$Id(TM_2): TM_2 \rightarrow TM_2$, $Id(W_{IR1}): W_{IR1} \rightarrow W_{IR1}$,
$Id(W_{XR1}): W_{XR1} \rightarrow W_{XR1}$, $Id(W_{GR1}): W_{GR1} \rightarrow W_{GR1}$

Composition

$(c_3 \text{ o } c_4) = c_5$, $(c_1 \text{ o } c_2) = c_5$

Axioms

Identity: $\forall\, x \in Identity(PT_1)$, $y \in Mor(PT_1)$,

$x \text{ o } y = y = y \text{ o } x$

TYPED-CATEGORY

Index Category (IC)

Types of Objects

Object_Type: Index (I)

Objects: Obj(IC)

I: i, j, k

Type of Morphisms

Morphism_Type:

Index(ind): $I \rightarrow I$

Morphisms: Mor(IC)

$\alpha (i) = j$, $\beta (k) = j$

Identity: Identity(IC)

$Id(i): i \rightarrow i$, $Id(j): j \rightarrow j$, $Id(k): k \rightarrow k$

Axioms

Identity: $\forall\, x \in Identity(IC)$, $y \in Mor(IC)$,

$x \text{ o } y = y = y \text{ o } x$

Category Source: IC

Category Target: PT_1

DIAGRAM (D, IC, PT_1)

Diagram Objects

Gamma Ray: $D(i): IC(i) \rightarrow PT_1(D(W_{XR1}))$

X-Ray: $D(j): IC(j) \rightarrow PT_1(D(W_{GR1}))$

Infra-red: $D(k): IC(k) \rightarrow PT_1(D(W_{IR1}))$

Diagram Morphisms

Cooperation:

$D(\alpha)$: $IC(j, i, \alpha) \rightarrow PT_1(D(W_{XR1}), D(W_{GR1}))$

$D(\beta)$: $IC(j, k, \beta) \rightarrow PT_1(D(W_{IR1}), D(W_{GR1}))$

Figure 5.5: CML Specification of the Petrologist Team Organization Scenario

The typed category in the CML model listed in Figure 5.5 is the PAM Petrologist Sub-swarm category. To avoid repetition, wherever applicable the definition of a typed-category is followed by (…) referring to the models constructed in the previous sections.

5.2.3 PAM Self-Configuration / Team Relocation Scenario

The virtual teams of spacecraft are configured to carry out optimal science operations on the target asteroids. When the operations are complete, the team breaks up for possible reconfiguration at another asteroid site. This reconfiguring continues throughout the life of the swarm. Reconfiguring may also be required as the result of a failure or anomaly of some sort. For example, when a worker's instrument is damaged or the team is without a messenger. This section includes a model that represents the Petrologist team's reconfiguration or relocation to a new sub-swarm where some of the spacecraft have been damaged or sent to accomplish new mission goals. The specification in Figure 5.6 captures the behavior of a team relocating to a new position in the sub-swarm.

<u>Categories:</u>

TYPED-CATEGORY

Petrologist Team (PT_1) …

TYPED-CATEGORY

PAM Sub-swarm (S_2)

Types of Objects

Object_Type:	Ruler (R)
Object_Type_Instances:	Leader (L)
Object_Type:	Messenger (M)
Object_Type_Instances:	Team Messenger (TM), Sub-Swarm Messenger (SM)
Object_Type:	Worker (W)
Object_Type_Instances:	Radio Sound (W_{RS}), Imager(W_{IM}), Infra-Red(W_{IR}), Helper(W_H)

Objects: Obj(S_2)

R: L_1

SM: SM_1, SM_3

W_{RS}: W_{RS1}

W_{IM} : W_{IM3}

W_{IR} : W_{IR3}

W_H : W_{H1}

Type of Morphisms

Morphism_Type: Data Update (du):

$W \rightarrow L$

Morphism_Type: Management (m):

$L \rightarrow SM$

Morphism_Type: Cooperation (c):

$W \rightarrow W$

Morphism_Type: Communication (cu):

$SM \rightarrow SM$, $W \rightarrow SM$, $SM \rightarrow L$

Morphisms: Mor(S_2)

$m_7 (L_1) = SM_1$, $c_{11} (W_{H1}) = W_{IR3}$,

$c_{12} (W_{RS1}) = W_{IR3}$, $c_9 (W_{H1}) = W_{RS1}$,

$c_{10} (W_{H1}) = W_{IM3}$, $c_8 (W_{RS1}) = W_{IM3}$,

$cu_{10} (W_{RS1}) = SM_1$, $cu_{11} (W_{RS1}) = SM_3$,

$cu_{12} (SM_3) = SM_1$, $cu_{13} (W_{H1}) = SM_1$

Identity: Identity(S_2)

$Id(L_1): L_1 \rightarrow L_1$, $Id(SM_1): SM_1 \rightarrow SM_1$,

$Id(SM_3): SM_3 \rightarrow SM_3$, $Id(W_{RS1}): W_{RS1} \rightarrow W_{RS1}$,

$Id(W_{H1}): W_{H1} \rightarrow W_{H1}$, $Id(W_{IM3}): W_{IM3} \rightarrow W_{IM3}$,

$Id(W_{IR3}): W_{IR3} \rightarrow W_{IR3}$

Composition

$(c_8 \circ c_9) = c_{10}$, $(c_{12} \circ c_9) = c_{11}$,

$(m_7 \circ du_1) = cu_{13}$, $(cu_{10} \circ c_9) = cu_{13}$,

$(cu_{11} \circ c_9) = cu_{14}$, $(cu_{12} \circ cu_{11}) = cu_{10}$,

$(cu_{12} \circ cu_{14}) = cu_{13}$, $(cu_{15} \circ cu_{11}) = cu_{13}$,

$(cu_{13} \circ cu_9) = cu_{14}$, $(c_{13} \circ c_9) = c_{16}$,

$(cu_{15} \circ cu_{14}) = cu_{16}$

Axioms

Identity: $\forall\, x \in Identity(S_2)$, $y \in Mor(S_2)$,

$$x \circ y = y = y \circ x$$

Associativity:

$cu_{12} \circ (cu_{11} \circ c_9) = (cu_{12} \circ cu_{11}) \circ c_9$

$cu_{15} \circ (cu_{11} \circ c_9) = (cu_{15} \circ cu_{11}) \circ c_9$

Category Source: PT_1

Category Target : S_2

FUNCTOR (Team Relocation, R, PT_1, S_2)

Functor Objects

Messenger:
$R(TM_2): PT_1 (TM_2) \rightarrow S_2 (R(TM_2):W_{H1})$
X-Ray:
$R(W_{XR1}): PT_1 (W_{XR1}) \rightarrow S_2 (R(W_{XR1}) :W_{RS1})$
Infra-red:
$R (W_{IR1}): PT_1 (W_{IR1}) \rightarrow S_2 (R(W_{IR1}) :W_{IR3})$
Gamma Ray:
$R(W_{GR1}): PT_1 (W_{GR1}) \rightarrow S_2 (R(W_{GR1}) :W_{IR3})$

Functor Morphisms

Cooperation:

$R(c_1): PT_1 (W_{XR1}, W_{GR1}, c_1) \rightarrow$
$S_2(R(W_{XR1}): W_{RS1}, R(W_{GR1}): W_{IR3}, R(c_1): c_{12}),$

$R(c_2): PT_1 (TM_2, W_{XR1}, c_2) \rightarrow$
$S_2(R(TM_2): W_{H1}, R (W_{XR1}) :W_{RS1}, R(c_2): c_9),$

$R(c_3): PT_1 (W_{IR1}, W_{GR1}, c_3) \rightarrow$
$S_2(R(W_{IR1}): W_{IR3}, R(W_{GR1}) : W_{IR3}, R(c_3): Id(W_{IR3})),$

$R(c_4): PT_1 (TM_2, W_{IR1}, c_4) \rightarrow$
$S_2(R(TM_2): W_{H1} , R(W_{IR1}): W_{IR3}, R(c_4): c_{11}),$

$R(c_5): PT_1 (TM_2, W_{GR1}, c_5) \rightarrow$
$S_2(R(TM_2): W_{H1}, R(W_{GR1}): W_{IR3}, R(c_5): c_{11})$

Functor Composition

$R (c_3 \circ c_4) = R (c_3) \circ R(c_4) = R(c_5)$
$R (c_1 \circ c_2) = R (c_1) \circ R(c_2) = R(c_5)$

Functor Axioms

Identity: $R(Id(TM_2)) = Id(R(TM_2))$
$R(Id(W_{XR1})) = Id(R(W_{XR1}))$
$R(Id(W_{IR1})) = Id(R(W_{IR1}))$
$R(Id(W_{GR1})) = Id(R(W_{GR1}))$

Figure 5.6: CML Model for Team Relocation Scenario

The model makes use of the functor construct grammar given in section 4.5 to specify the explained scenario. The graphical model for the specification in Figure 5.6 is given in Figure 5.7.

5.2.4 PAM Spacecraft Role Change Scenario

As the teams change from one configuration to another, the responsibilities of the spacecraft in that team could possibly change. This could be specified as the change of role that is a result of the reconfigurations. Figure 5.9 includes a CML model that specifies this scenario using the natural transformation grammar discussed in section 4.6. Natural transformation as explained in chapter 3 is the relationship between two functors where the source and target categories are the same for each functor.

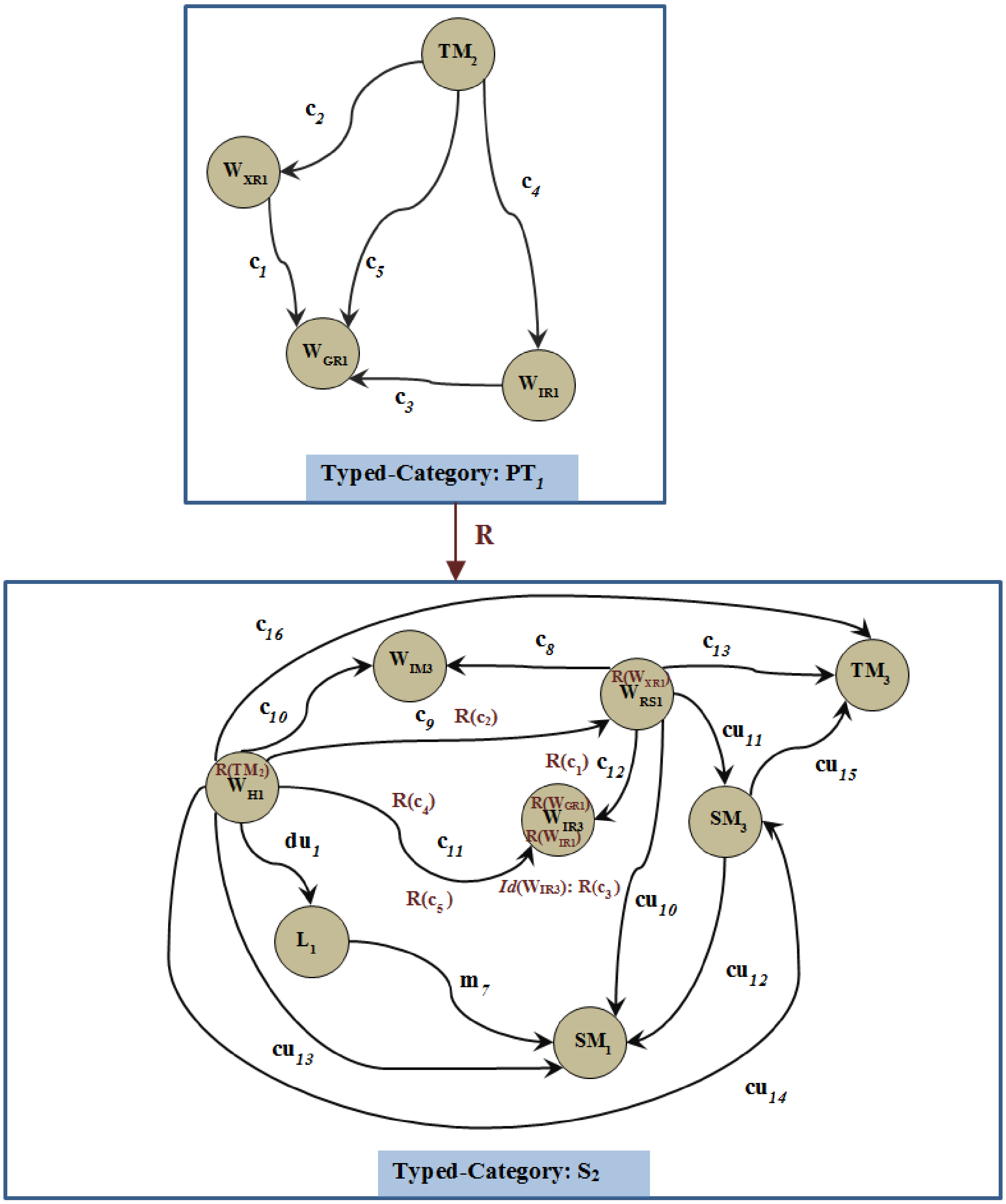


Figure 5.7: CML Model for Team Relocation Scenario

The graphical model for the specification in Figure 5.9 is given in Figure 5.8.

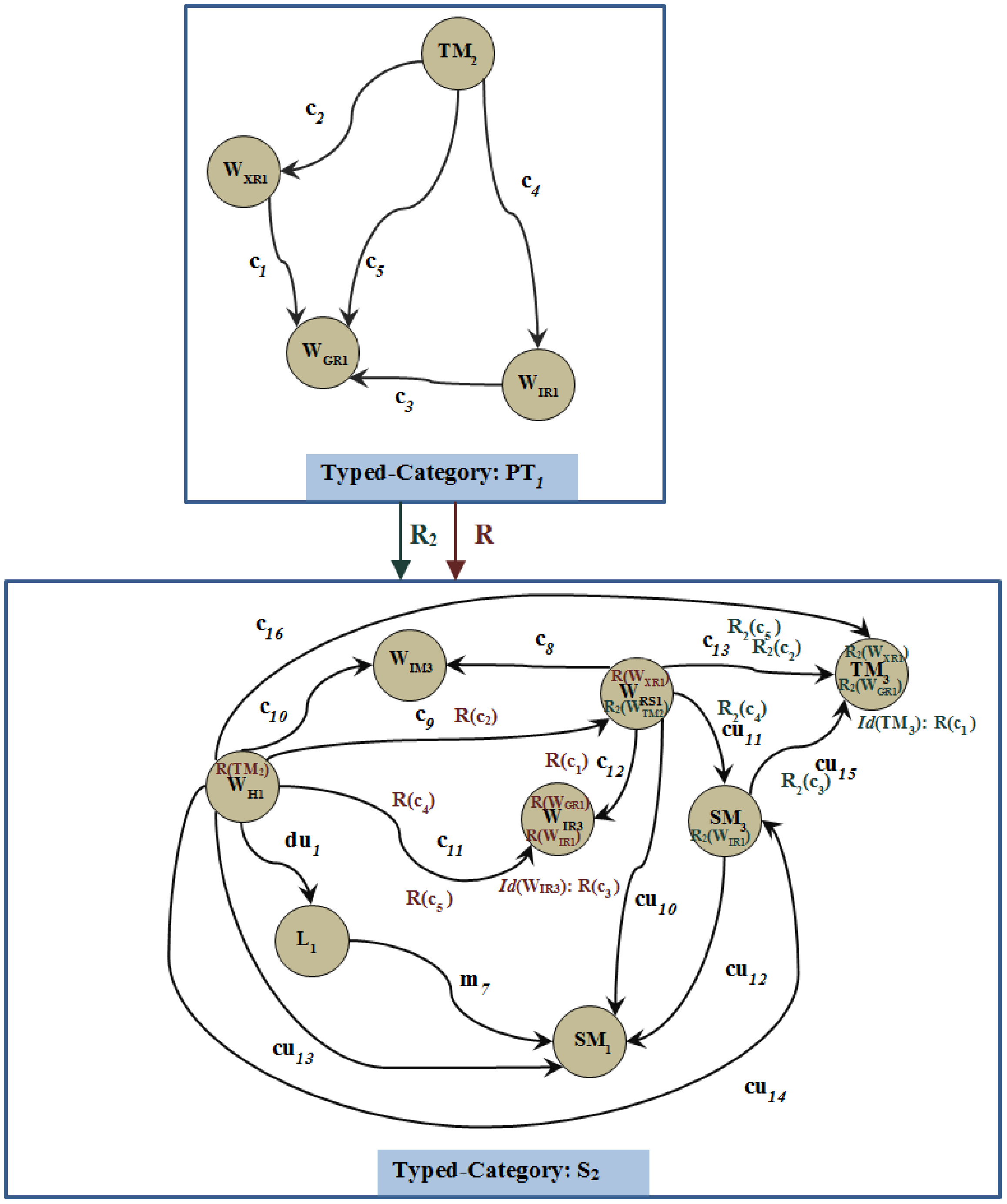


Figure 5.8: CML Model for Spacecraft Role Change Scenario

Categories:

TYPED-CATEGORY

Petrologist Team (PT_1) …

TYPED-CATEGORY

PAM Sub-Swarm (S_2) …

Category Source: PT_1
Category Target : S_2

Functors:

FUNCTOR (Reconfiguration, R, PT_1, S_2)…

FUNCTOR (Team Relocation, R_2, PT_1, S_2)

Functor Objects

Messenger:
R_2 (TM_2): PT_1 (TM_2) → S_2 (R_2(TM_2):W_{RS1})
X-Ray:
R_2 (W_{XR1}): PT_1 (W_{XR1}) → S_2 (R_2 (W_{XR1}):TM_3)
Infra-red:
R_2 (W_{IR1}): PT_1 (W_{IR1}) → S_2 (R_2 (W_{IR1}) :SM_3)
Gamma Ray:
R_2 (W_{GR1}): PT_1 (W_{GR1}) → S_2 (R_2 (W_{GR1}) :TM_3)

Functor Morphisms

Cooperation:

R_2 (c_1): PT_1 (W_{XR1}, W_{GR1}, c_1) →
S_2(R_2 (W_{XR1}): TM_3, R_2 (W_{GR1}): TM_3,
R_2 (c_1): *Id*(TM_3)),

R_2 (c_2): PT_1 (TM_2, W_{XR1}, c_2) →

S_2(R_2 (TM_2): W_{RS1}, R_2 (W_{XR1}) : TM_3, R_2 (c_2): c_{13}),

R_2 (c_3): PT_1 (W_{IR1}, W_{GR1}, c_3) →

S_2(R_2 (W_{IR1}): SM_3, R_2 (W_{GR1}) : TM_3, R_2 (c_3): cu_{15}),

R_2 (c_4): PT_1 (TM_2, W_{IR1}, c_4) →
S_2(R_2 (TM_2): W_{RS1} , R_2 (W_{IR1}): SM_3, R_2 (c_4): cu_{11}),

R_2 (c_5): PT_1 (TM_2, W_{GR1}, c_5) →
S_2(R_2 (TM_2): W_{RS1}, R_2 (W_{GR1}): TM_3, R_2 (c_5): c_{13})

Composition

R_2 (c_3 o c_4) = R_2 (c_3) o R_2 (c_4) = R_2 (c_5),
R_2 (c_1 o c_2) = R_2 (c_1) o R_2 (c_2) = R_2 (c_5)

Functor Axioms

Identity: R_2 (*Id*(TM_2)) = *Id*(R_2 (TM_2))
R_2 (*Id*(W_{XR1})) = *Id*(R_2 (W_{XR1}))
R_2 (*Id*(W_{IR1})) = *Id*(R_2 (W_{IR1}))
R_2 (*Id*(W_{GR1})) = *Id*(R_2 (W_{GR1}))

Functor Ids: R, R_2

NAT_TRANSFORMATION (Role Change, σ)

NTrans Functors

Reconfiguration:
Relocation (R): $PT_1 \Rightarrow S_2$,
Relocation (R_2): $PT_1 \Rightarrow S_2$

NTrans Objects

PT_1:
Messenger: TM_2,
Worker: W_{XR1}, W_{IR1}, W_{GR1}

NTrans Mapping Function

σ(TM_2) : R(TM_2) ⇒ R_2(TM_2)
σ(W_{XR1}) : R(W_{XR1}) ⇒ R_2(W_{XR1})
σ(W_{IR1}) : R(W_{IR1}) ⇒ R_2(W_{IR1})
σ(W_{GR1}) : R(W_{GR1}) ⇒ R_2(W_{GR1})

NTrans Morphisms

Cooperation: R(c_1): R(W_{XR1}) → R(W_{GR1}),
R(c_2): R(TM_2) → R(W_{XR1}),
R(c_3): R(W_{IR1}) → R(W_{GR1}),
R(c_4): R(TM_2) → R(W_{IR1}),
R(c_5): R(TM_2) → R(W_{GR1}),
R_2(c_1): R_2 (W_{XR1}) → R_2 (W_{GR1}),
R_2 (c_2): R_2 (TM_2) → R_2 (W_{XR1}),
R_2 (c_3): R_2 (W_{IR1}) → R_2 (W_{GR1}),
R_2 (c_4): R_2 (TM_2) → R_2 (W_{IR1}),
R_2 (c_5): R_2 (TM_2) → R_2 (W_{GR1})

NTrans Axioms Commutativity: $\forall$ x, y $\in$ ***Obj(PT$_1$)***, f : x $\rightarrow$ y $\in$ ***Mor(PT$_1$),*** $R_2(f) \circ \sigma(x) = \sigma(y) \circ R(f)$	

Figure 5.9: CML Model for Role Change Scenario in PAM

5.2.5 PAM Team Messenger Cooperation

The cooperation of the PAM Petrologist Team Messenger cooperating with the team workers is specified using the cone construct grammar included in section 4.8. The inverse scenario could be modeled using the co-cone construct grammar listed in section 4.9. Together the two models represent the complete working scenario of a Petrologist team messenger, or any team messenger for that matter. Figures 5.10 and 5.12 include the CML specification of a typical PAM team messenger cooperation scenario. The specification in Figure 5.10 captures the behavior of the team messenger communicating information to the team while Figure 5.12 specifies the behavior of a team messenger receiving data from the team.

Diagram: **DIAGRAM** (D, IC, PT$_1$) … ***Category Id:*** PT$_1$ ***Diagram Id:*** D **CONE** (Object: TM$_2$)	***Cone Objects*** D(i), D(j), D(k) ***Cone Morphisms*** Cooperation: TM$_2$(k): TM$_2$ $\rightarrow$ D(k), TM$_2$(j): TM$_2$ $\rightarrow$ D(j), TM$_2$(i): TM$_2$ $\rightarrow$ D(i) Cooperation: D(α): D(i) $\rightarrow$ D(j) D(β): D(k) $\rightarrow$ D(j)

Figure 5.10: PAM Team Messenger Communicating to the Team

Figure 5.11 and 5.13 include the graphical models for the specification in Figure 5.10 and 5.12.

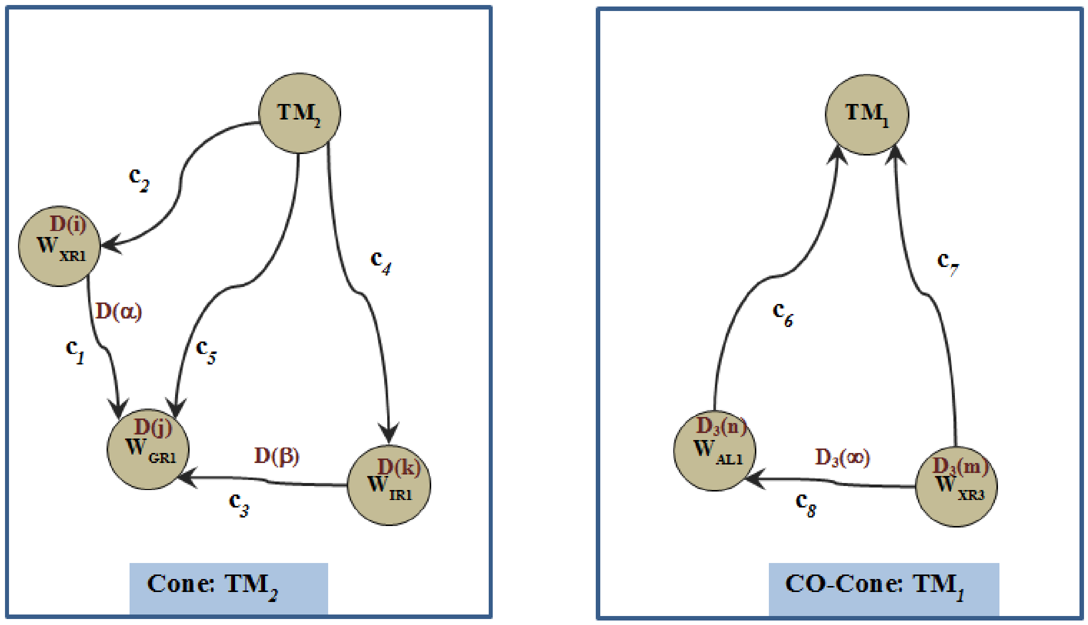


Figure 5.11: CML Graphical Model of a PAM Team Messenger Communication

Diagram:

Category Source: IC_3

Category Target: S_1

DIAGRAM (D_3, IC_3, S_1)

Diagram Id: D_3

Diagram Objects

X-Ray: $D_3(i): IC(i) \rightarrow S_1(D_3(W_{XR3}))$

Altimeter: $D_3(j): IC(j) \rightarrow S_1(D_3(W_{AL1}))$

Diagram Morphisms

Cooperation:

$D_3(\alpha)$: $IC(i, j, \alpha) \rightarrow S_1(D_3(W_{XR3}), D_3(W_{AL1}))$

CO-CONE (Object: TM_1)

Co-Cone Objects

$D_3(i), D_3(j)$

Co-Cone Morphisms

Cooperation: $TM_1(j)$: $D_3(j) \rightarrow TM_1$,
$TM_1(i)$: $D_3(i) \rightarrow TM_1$,
Cooperation: $D_3(\alpha)$: $D_3(i) \rightarrow D_3(j)$

Figure 5.12: PAM Team Messenger Receiving Data from the Team

5.2.6 PAM Self-Protection Scenario

Besides avoiding collisions with asteroids and other spacecraft, PAM teams must protect themselves from solar storms, where charged particles can destroy the sensors and electronic mechanisms, and damage the solar sails. In such situations, PAM spacecraft must re-organize their trajectories, or, in worst-case scenarios, must go into the “stand by” mode to protect their sails and instruments and other subsystems.

After receiving a confirmation from a sub-swarm leader regarding a solar storm, a sub-swarm messenger communicates this information to the other sub-swarm messengers. All sub-swarm messengers inform their team messengers which in turn inform all the workers in the team. Each spacecraft after receiving a warning message and performing necessary communication puts itself to a “stand by” mode. Figure 5.13 includes a CML model for this scenario constructed using the limit construct grammar discussed in chapter 4 section 4.10. The graphical model for the specification in Figure 5.13 is given in Figure 5.14.

<u>Diagram:</u>

DIAGRAM (D, IC, S_1)…

<u>Cones:</u>

CONE (Object: TM_2)…
CONE (Object: L_1)

Co-Cone Objects
D(k), D(j), D(i),

Co-Cone Morphisms

Management: $L_1(k): L_1 \rightarrow D(k)$,
$L_1(j): L_1 \rightarrow D(j)$,
$L_1(i): L_1 \rightarrow D(i)$

Communication: $D(\alpha): D(k) \rightarrow D(l)$
$D(\beta): D(j) \rightarrow D(k)$

Category Id: S_1
Diagram Id: D
Cone Ids: TM_2, L_1

LIMIT
Terminal Object: TM_2
Unique Morphism(u): $m_1: L_1 \rightarrow TM_2$

Limit Objects
D(i), D(j), D(k)

Limit Morphisms		***Limit Axioms***
Management:	$L_1(k): L_1 \rightarrow D(k)$, $L_1(j): L_1 \rightarrow D(j)$, $L_1(i): L_1 \rightarrow D(i)$	$m_1 \circ L_1(i) = TM_2(i)$ $m_1 \circ L_1(j) = TM_2(j)$ $m_1 \circ L_1(k) = TM_2(k)$
Cooperation:	$TM_2(k): TM_2 \rightarrow D(k)$, $TM_2(j): TM_2 \rightarrow D(j)$, $TM_2(i): TM_2 \rightarrow D(i)$, $D(\alpha): D(i) \rightarrow D(j)$, $D(\beta): D(j) \rightarrow D(k)$	

Figure 5.13: CML Model for PAM Self-Protection

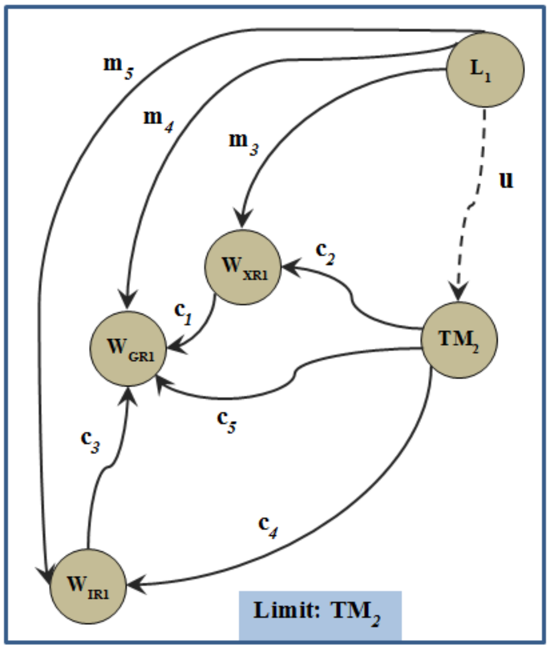


Figure 5.14: CML Model for PAM Self-Protection

5.2.7 Leader Spacecraft Receiving Data Scenario

This section includes the specification of a scenario for a PAM Sub-Swarm Leader spacecraft receiving data from the team messengers. The scenario is from the sub-swarm S_4. The typed-category specification for S_4 is not included to avoid repetition of the PAM

Sub-swarm specification. The model is constructed using the co-limit construct grammar discussed in section 4.11 in chapter 4. Figure 5.15 includes the complete model of the mentioned scenario. The graphical model for the specification in Figure 5.15 is shown in Figure 5.16.

<u>Diagram:</u>

<u>Categories:</u>

TYPED-CATEGORY

PAM Sub-swarm (S_4)…

TYPED-CATEGORY

Index Category (IC_2)

Types of Objects

Object_Type: Index (I)

Objects: Obj(IC_2)

I: k, l

Type of Morphisms

Morphism_Type:

Index(ind): $I \rightarrow I$

Morphisms: Mor(IC_2)

$\alpha(k) = l$

Identity: Identity(IC_2)

$Id(k): k \rightarrow k$, $Id(l): l \rightarrow l$

Axioms

Identity: $Id(k) \circ \alpha = \alpha = \alpha \circ Id(k)$

Category Source: IC_2

Category Target: S_4

DIAGRAM (D, IC, S_4)

Diagram Objects

Team-Messenger: $D(k): IC_2(k) \rightarrow S_4(D(TM_2))$

Altimeter: $D(l): IC_2(l) \rightarrow S_4(D(TM_1))$

Diagram Morphisms

Cooperation:

$D(\alpha)$: $IC_2(k, l, \alpha) \rightarrow S_4(D(TM_2), D(TM_1))$

Communication: $D(\alpha)$: $D(k) \rightarrow D(l)$

<u>Co-Cones:</u>

CO-CONE (Object: L_1)…

CO-CONE (Object: SM_1)

Co-Cone Objects
D(k), D(l)

Co-Cone Morphisms

Communication: $SM_1(k): D(k) \rightarrow SM_1$,
$SM_1(l): D(l) \rightarrow SM_1$
Communication: $D(\alpha)$: $D(k) \rightarrow D(l)$

Category Id: S_4
Diagram Id: D
Co-Cone Ids: L_1, SM_1

CO-LIMIT
Initial Object: SM_1
Unique Morphism(u): $SM_1 \rightarrow L_1$

Co-Limit Objects
D(k), D(l)

Co-Limit Morphisms	*Co-Limit Axioms*
Cooperation: $L_1(k)$: $D(k) \to L_1$, $L_1(l)$: $D(l) \to L_1$, $SM_1(k)$: $D(k) \to SM_1$, $SM_1(l)$: $D(l) \to SM_1$, $D(\alpha)$: $D(k) \to D(l)$	$L_1(k) \circ u = SM_1(k)$ $L_1(l) \circ u = SM_1(l)$

Figure 5.15: Leader Spacecraft Model in PAM

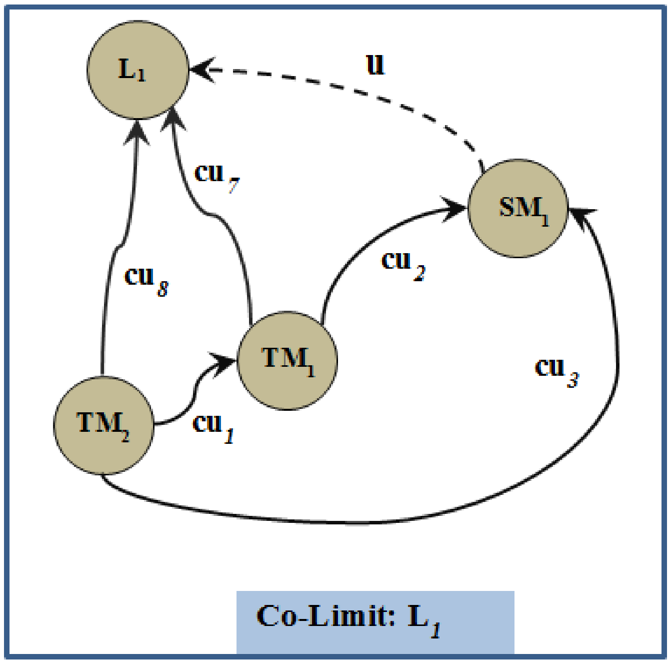


Figure 5.16: Leader Spacecraft Model in a PAM Sub-swarm

5.2.8 PAM Sciencecraft Cooperation

Sciencecraft cooperation to achieve the 3D Model data of an asteroid is shown in Figure 5.17. The model has been constructed using the product construct grammar given in section 4.12.

Category:

TYPED-CATEGORY

PAM Sub-Swarm (S_1) …

Category Id: S_1

PRODUCT (Imaging Cooperation, S_1)

Product Objects

Worker: W_{IM3}, W_{RS1}, W_{H1}
Product: $W_{IM3}\ \Pi\ W_{RS1}$

Product Morphisms

Projection:	$p_{IM3} : W_{IM3}\ \Pi\ W_{RS1} \rightarrow W_{IM3}$ $p_{RS1} : W_{IM3}\ \Pi\ W_{RS1} \rightarrow W_{RS1}$
Cooperation:	c_9: $W_{H1} \rightarrow W_{RS1}$ c_{10}: $W_{H1} \rightarrow W_{IM3}$
Unique:	u: $W_{H1} \rightarrow W_{IM3}\ \Pi\ W_{RS1}$

Product Axioms

Composition:	$p_{IM3}\ o\ u = c_{10}$ $p_{RS1}\ o\ u = c_9$

Figure 5.17: PAM Team Cooperation Behavior Specification

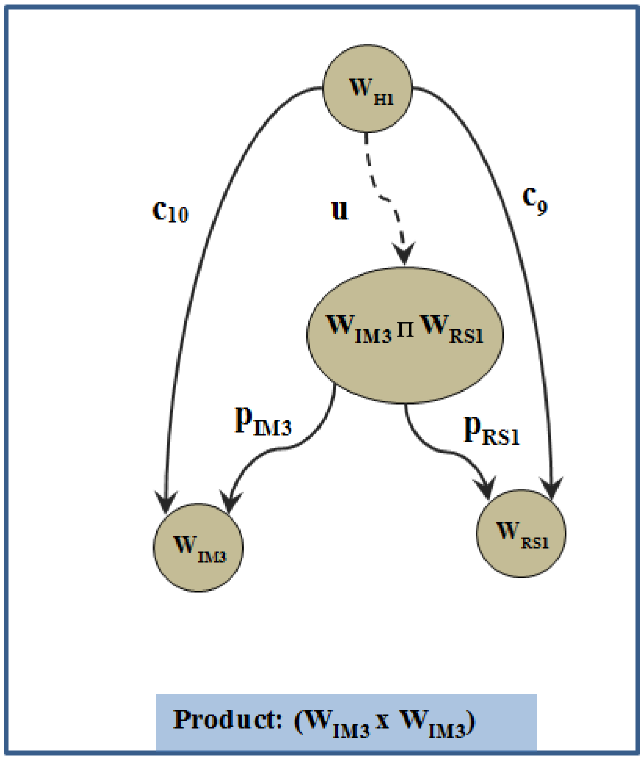


Figure 5.18: PAM Team Cooperation Graphical Model

The unique morphism **u** in the model specifies the role of the sub-swarm Leader in this cooperation scenario where the leader is supervising the cooperation of the sciencecraft to gather the optimal data. The graphical model for the specification in Figure 5.17 is given in Figure 5.18.

5.3 Conclusion

This chapter includes a discussion on different CML models constructed for different mission scenarios of the PAM. The models correspond to the patterns of behavior or high-level behavior policies of the PAM.

6 CATCanvas: CAT Modeling Tool

6.1 Introduction

This chapter includes a concise discussion on the CAT modeling tool written as a part of this research. The name of this modeling tool is CATCanvas inspired from the way the CAT visual models are constructed on a drawing canvas. So far two constructs of CAT have been implemented in CATCanvas, that is, the Category and Functor construct. For every construct there is a separate view and separate drawing canvases. The CAT model could be either drawn manually on the respective canvas or imported from an xml file to the canvas. Similarly, the model could be exported in xml format. The tool is also capable of saving the constructed model as a 'png' image.

The need for implementing CATCanvas is two-fold. First and most important of all reasons is the absence of an existing tool of this nature for constructing CAT diagrams. There is one tool known as Category Theory 3.0 [56] that is also a graphical diagramming tool like CATCanvas with the differences between the two listed in Table 6.1. To summarize, CATCanvas is a UI friendly web-based tool aimed at computer science audience rather than mathematicians. It makes use of the abstraction power of category theory and enables one to create typed categories and functors. The second and most important reason for the implementation of CATCanvas is its use by our research group for easy construction/drawing of the CML-based CAT models and porting these models from this tool to other tools. Figure 6.1 consists of a snapshot of the default view of the CATCanvas.

Table 6.1: CATCanvas vs. Category Theory 3.0 [56]

CATCanvas	Category Theory 3.0
Category is typed and one could construct and work with any type of category	Work with a certain group of Categories in Math (finitely generated Abelian Groups [57], Vector Spaces, Finitely Generated Algebra, Finite Sets
Computer Science Friendly	Very Math Extensive
Powerful Functor Mapping with Source/Target categories in view and UI friendly mapping of objects and morphisms	Functors Generated using Mathematical Calculation
Math symbols available in objects and morphisms properties editor	Formula Editor for Objects and Morphism
Two Category Constructs Available	More Category Constructs Available
Naming Flexibility (Objects, Morphisms, Category, Functor)	Pre-defined Naming (Objects, Morphism, Category, Functor)
Powerful Drawing Canvas with ability to curve the morphism arrows and choose colors for objects/morphisms.	Very Limited drawing Canvas with pre-defined color's and straight lines for morphisms
XML Import Export Available	XML Import/Export Not Available
Ability to Save PNG Images	Cannot Save Image File
Available Online	Desktop Application

CATCanvas is a Flex-based web application running in flash player 10.0. The reason for choosing Adobe Flex [58] for CATCanvas was mainly because of the tools availability online and for the quality purposes of the flex's flash-based graphics/drawing library.

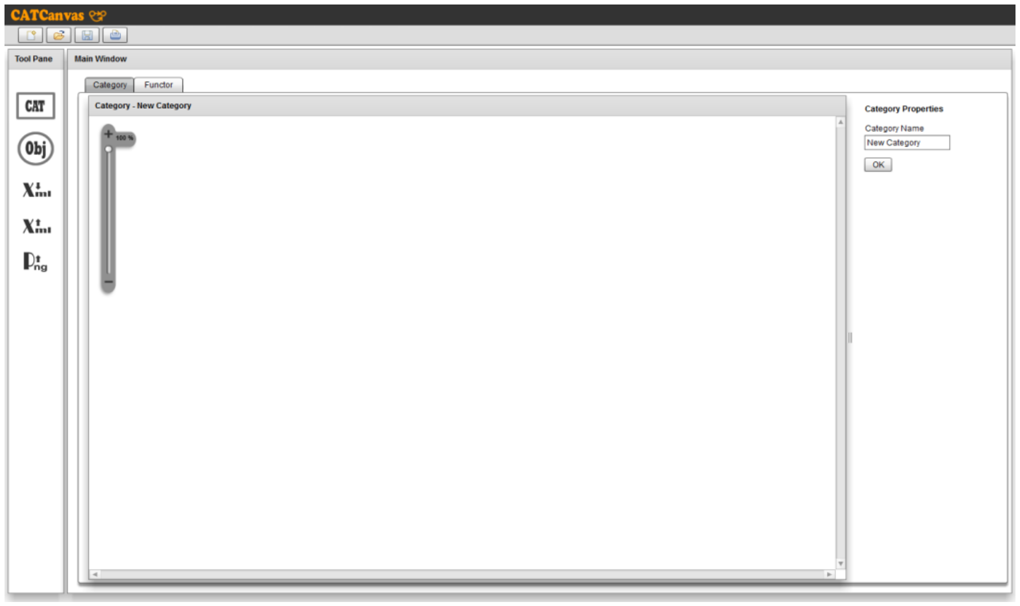


Figure 6.1: CATCanvas Default View

The choice of the development platform was made after writing comparison prototypes in both Java and Flex.

6.2 Architecture of CATCanvas

CATCanvas is a web-based application running in a flash player. The UI is a flex-based Web UI built using mxml controls. The drawing tools on the Web UI use the graphics library for rendering the diagrams on the canvas. There is a 'Rules Engine' in CATCanvas that is responsible for the construction of categorically correct models. The 'Rules Engine' plays an active role when performing functor mappings. For the constructed diagrams/models, the XML generator can generate XML specification and send to the Web UI in order to export the specification to a file. The XML parser can parse an XML file and send the data to the Web UI to render the graphical model with the help of the 'Graphical API'. Figure 6.2 shows a block diagram of the architecture of the tool.

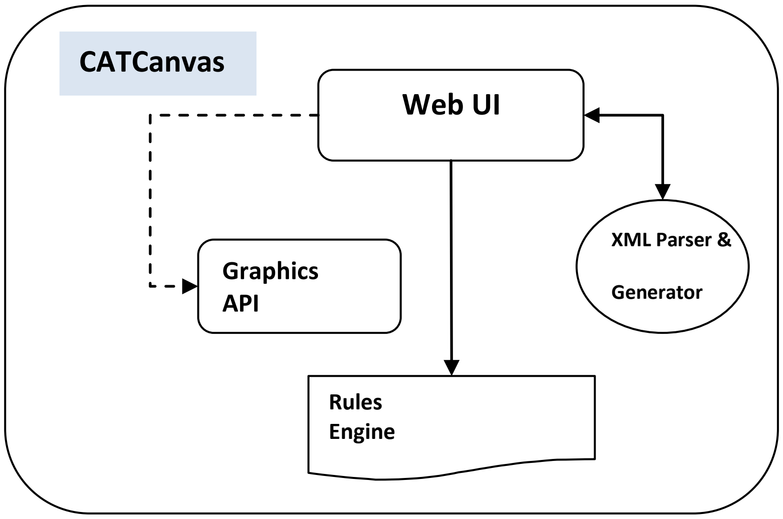


Figure 6.2: Architecture of CATCanvas

6.3 List of Features

In Figure 6.1 we can see the layout of the tool in the form of named views. The look and feel of the application is that of a windows application but these are actually panels/views in Flex. The discussion in this section consists of the explanation of every view along with the list of features contained in that view. Some of the views will in turn have sub-views that would be explained whenever required.

6.3.1 Titlebar and Toolbar

Unlike traditional web application, CATCanvas consists of a title bar and a toolbar. The title bar has a black background with name and icon of the tool in an orange foreground.

Just below the title bar comes the toolbar that consists of the traditional 'New', 'Open', 'Save' and 'Print' buttons for CAT constructs, category and functor.

6.3.2 Main Window

Under the Toolbar, there is a Tool Pane and a Main Window. The Main Window consists of the TabStrip for Category and Functor views. Switching between the Category Tab and the Functor Tab has an impact on the content/views inside the tabs and on the Tool Pane as well. For the functor tab, the layout is very different from that of a category tab.

6.3.3 TabStrip

The TabStrip consists of two tabs, that is, the Category Tab and the Functor Tab. Each tab in turn consists of drawing areas also known as the canvas and a properties panel with different views for each tab based on the requirement of the construct. The views on each tab have horizontal and vertical dividers in between each view that enable resizing of the view as well.

6.3.4 Tool Pane

The Tool Pane consists of a basic set of tools for working in each tab. Figure 6.3 (a) shows the Tool Pane for the Category Tab and Figure 6.3 (b) consists of the Tool Pane for the Functor Tab.

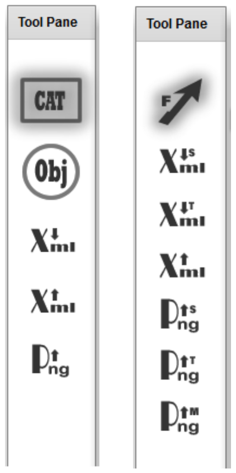


Figure 6.3: (a) Tool Pane Category Tab; (b) Tool Pane Functor Tab

Category Tool Pane

This tool pane consists of five buttons/tools. The shaded ‘Reset/New Category’ tool is used to create a new typed category. Selecting this tool also wipes out the current model under construction. The shaded tool represents the current tool roll-over for scrolling through the tools on the Tool Pane. The next tool is the ‘Add an Object’ tool used to add a new object to the model being constructed on the canvas. ‘Import XML’ from its name is the button used to import a file from the library. Similarly, ‘Export XML’ is used to export the xml for the constructed model. Finally, ‘Export PNG’ is used to export an image of the visual model.

Functor Tool Pane

This tool pane consists of seven buttons/tools in total. ‘Reset/New Functor’ is used to reset the screen and create a new functor. Pressing this button resets all the panes and sets the canvases ready for a new model. ‘Import Source XML’, and ‘Import Target XML’ are used to for importing source and target category xml into the tool. Similarly, ‘Export Functor XML’ is used to export the xml for the constructed functor. , , are used to export png image for source category, target category and the functor mapping table respectively.

6.3.5 Drawing Canvas

The panel window inside the tab window consists of the name of the category on the panel title bar and the model is constructed on the canvas with a white background. For Category tab there is only a single canvas available. For the Functor tab there are two canvases, one for the source category and the other for the target category. It is possible to zoom and pan on the canvases. The canvases also have vertical and horizontal scrollbars to accommodate models requiring more page space.

Zoom In/ Zoom Out

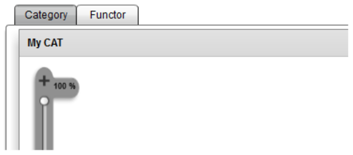
It is possible to zoom-in and zoom-out on the models constructed on the canvas using the lever shown in Figure 6.4. The level sits on top of the canvas and is part of the canvas itself.

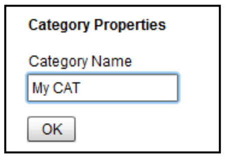
Figure 6.4: Canvas Zoom In/Zoom Out

The default and the maximum zoom is 100% and as the minus ‘-’ button is pressed, the zoom percentage decreases. This is a nifty tool and comes in handy if the canvas is longer/wider than the viewport.

6.3.6 Category Properties

Figure 6.5 (a) shows the canvas panel where the name of the category appears. Tapping once on the canvas panel with a mouse-click shows the Category Properties property window shown in Figure 6.5(b). By default, the name of a category is ‘Category – New Category’.




Category Properties

Category Name

My CAT

OK

Figure 6.5: (a) Canvas Panel w/ Name of the Category in the Title; (b) Property Window Category Properties

6.3.7 Adding an Object

We saw earlier while discussing the Tool Pane for category tab the tool used to add an object. Figure 6.6 shows a snapshot of an object being adding to My CAT category canvas using the ‘Add an Object’ button. The object is added with a default color that is yellow with a black outline.

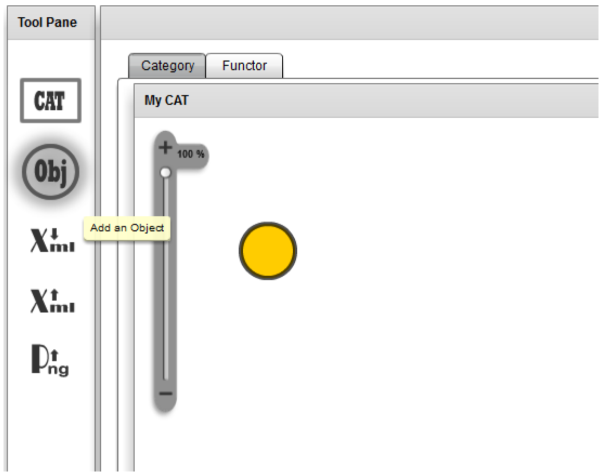


Figure 6.6: Adding an Object to My CAT

6.3.8 Object Properties

In order to customize an object, click on the object on the canvas with a single mouse click which will pop-up the property window 'Object Properties' shown in Figure 6.7 (a).

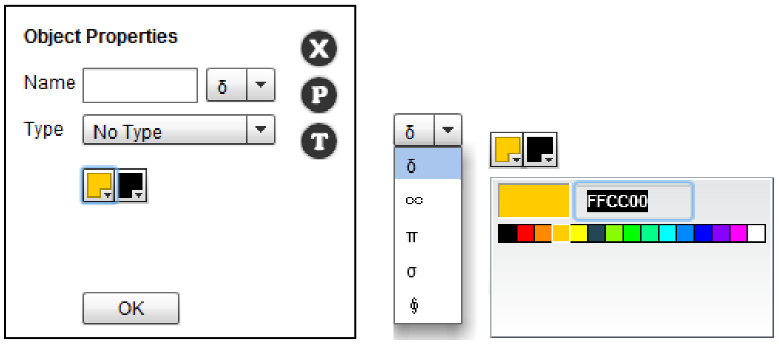


Figure 6.7: (a) Object Properties; (b) List of Symbols; (c) Color-pickers

The name of an object could be up to three characters and can include mathematical symbols as well. The drop-down right next to the name input field consists of a list of some basic mathematical symbols available to be inserted, Figure 6.7 (b). The type field has a drop-down for values. This drop-down is populated using the type creation view. The two color-pickers in the bottom are for background and foreground color selection respectively Figure 6.7 (c). The three buttons on the right present three different views. Ⓧ enables you to delete the object as shown in Figure 6.8(a). Ⓟ is to switch back to ‘Object Properties default view given in Figure 6.7 (a). Finally, Ⓣ is used to select the view used to add a new object type for the model being constructed. The view is shown in Figure 6.8 (b).

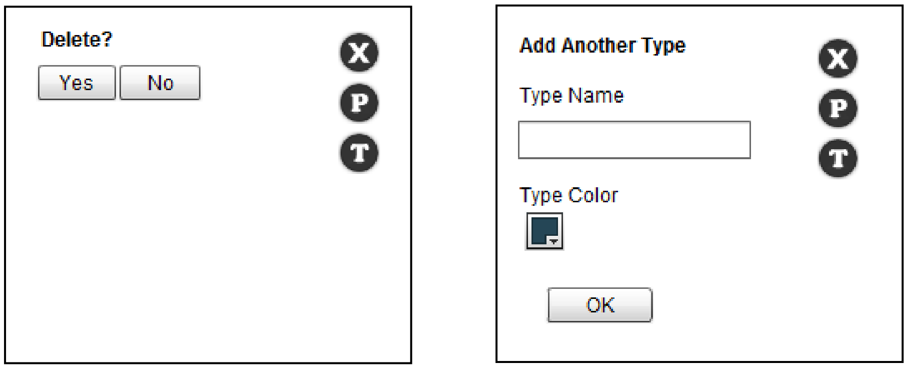


Figure 6.8: (a) Delete Object View; (b) Add Another Type View

To create a new type a type name and a type color is to be assigned/ defined. On pressing ‘OK’ the name and color defined for this type is saved and the type appears in the type drop-down box in ‘Object Properties’. This enables definition of types separately and effectively for separate models.

6.3.9 Adding a Morphism

Self-morphisms are not supported in CATCanvas to avoid cluttering the diagram. So, in order to add a morphism, at least two objects should be present on the canvas. A morphism could be created from the source object, hovering over the source object shows an arrow tool ➔. Figure 6.9(a) shows the popping-up of the arrow tool upon hovering over the source object. In order to draw the morphism, the arrow tool is clicked and pressed till the target object and released. This finishes drawing/adding of the morphism between two objects in a category. Figure 6.9 (b) shows the morphism added using the arrow tool to category My CAT between objects **i** and **j**.

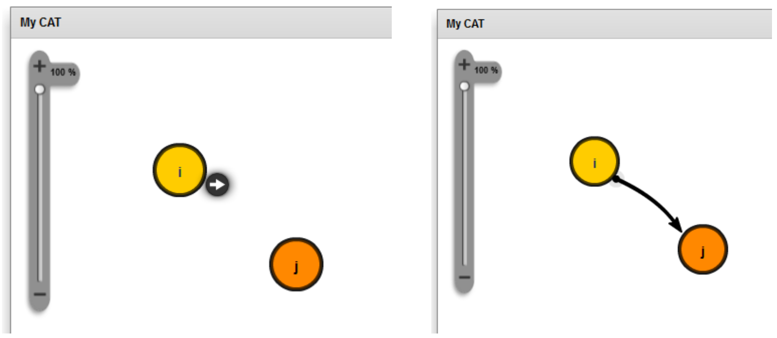


Figure 6.9: (a) Arrow Tool; (b) Morphism b/w Objects i and j

6.3.10 Morphism Properties

Similar to 'Object Properties' morphism properties could also be defined using a property window named 'Morphism Properties' shown in Figure 6.10 (a). The property window will appear whenever a morphism is clicked once. Similar to property window for an object, this also has three buttons for the three different views P 'Morphism Properties', T 'Add a Type' and X 'Delete the Morphism'.

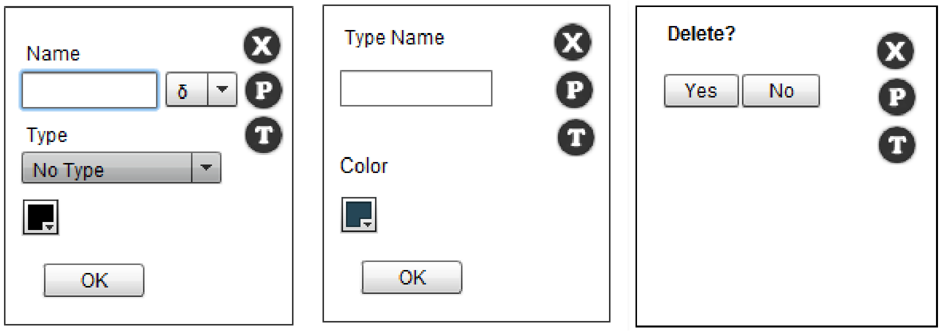


Figure 6.10: (a) Morphism Properties; (b) Morphism Type Definition; (c) Deleting a Morphism

Figure 6.11 includes a customized morphism based on the properties set using the property window 'Morphism Properties' view. The name of the morphism consists of a mathematical symbol π.

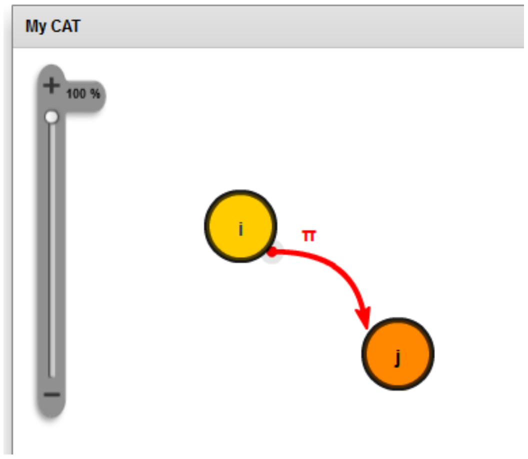


Figure 6.11: Morphism Properties Set Using the Property Window

Deleting a morphism does not delete the source and target objects. Type definition comes in handy when working with typed categories but is not shown in the form of labels in the visual model. The color of the arrow serves the purpose here. Figure 6.12 includes a complete category constructed using CATCanvas on the Category tab. Identity and composition morphisms are not shown in the visual model to avoid cluttering the model but are assumed present. The colors of the objects and morphisms indicate types.

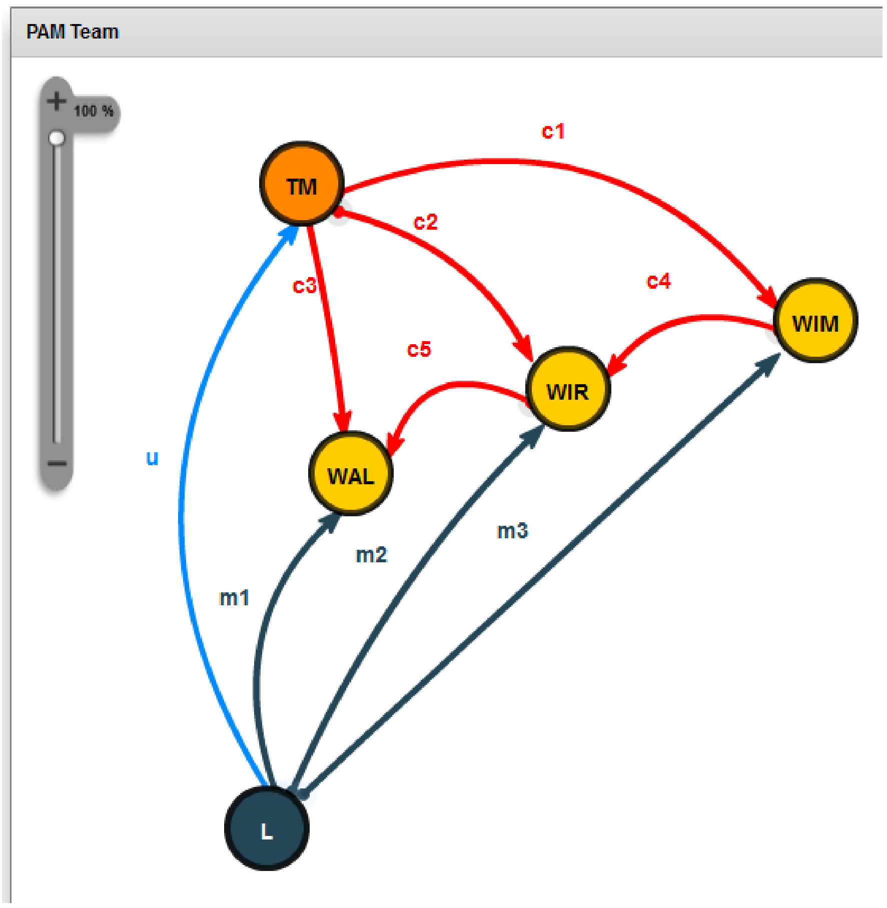
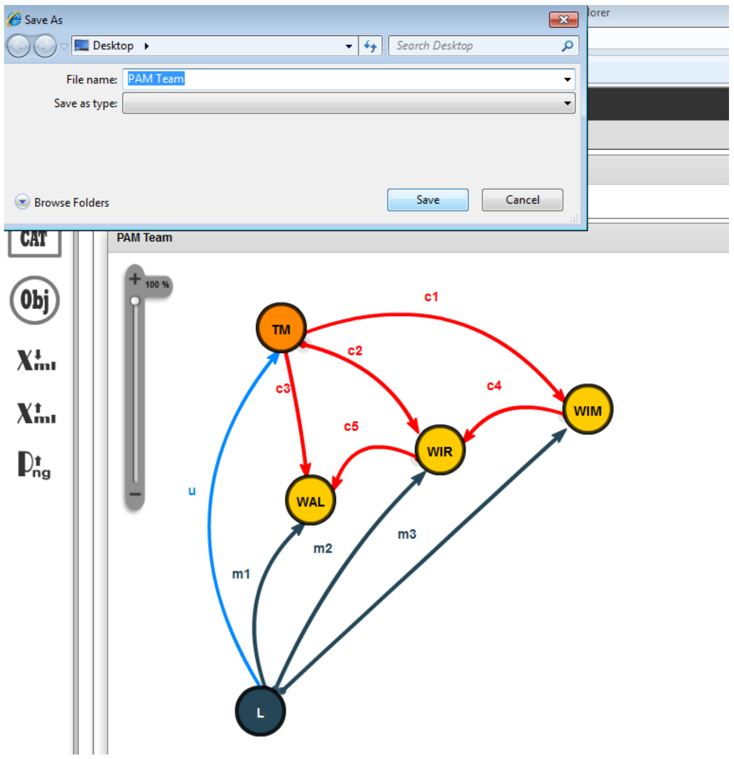


Figure 6.12: Category PAM Team Constructed using CATCanvas

6.3.11 Category XML Export

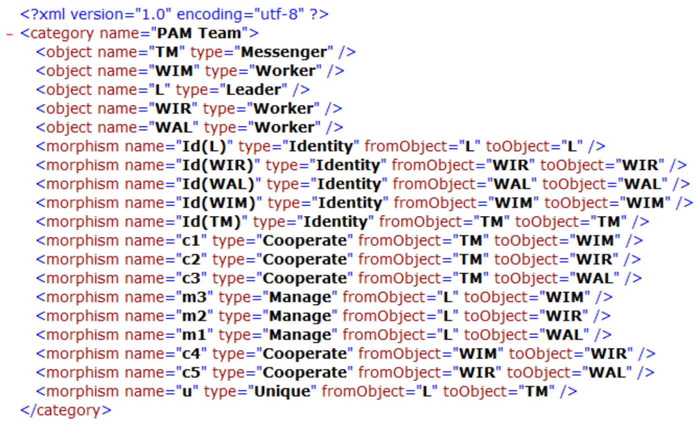
Figure 6.13 shows the XML export scenario for category given in Figure 6.12.


File name: PAM Team

Save as type:

Figure 6.13: PAM Team XML Export – Save File

Figure 6.14 includes a snapshot of the exported XML specification for the PAM model in Figure 6.12.



```
  <?xml version="1.0" encoding="utf-8" ?>
- <category name="PAM Team">
    <object name="TM" type="Messenger" />
    <object name="WIM" type="Worker" />
    <object name="L" type="Leader" />
    <object name="WIR" type="Worker" />
    <object name="WAL" type="Worker" />
    <morphism name="Id(L)" type="Identity" fromObject="L" toObject="L" />
    <morphism name="Id(WIR)" type="Identity" fromObject="WIR" toObject="WIR" />
    <morphism name="Id(WAL)" type="Identity" fromObject="WAL" toObject="WAL" />
    <morphism name="Id(WIM)" type="Identity" fromObject="WIM" toObject="WIM" />
    <morphism name="Id(TM)" type="Identity" fromObject="TM" toObject="TM" />
    <morphism name="c1" type="Cooperate" fromObject="TM" toObject="WIM" />
    <morphism name="c2" type="Cooperate" fromObject="TM" toObject="WIR" />
    <morphism name="c3" type="Cooperate" fromObject="TM" toObject="WAL" />
    <morphism name="m3" type="Manage" fromObject="L" toObject="WIM" />
    <morphism name="m2" type="Manage" fromObject="L" toObject="WIR" />
    <morphism name="m1" type="Manage" fromObject="L" toObject="WAL" />
    <morphism name="c4" type="Cooperate" fromObject="WIM" toObject="WIR" />
    <morphism name="c5" type="Cooperate" fromObject="WIR" toObject="WAL" />
    <morphism name="u" type="Unique" fromObject="L" toObject="TM" />
  </category>
```

Figure 6.14: PAM Team XML Export Output

6.3.12 Functor Tab

The Functor tab as mentioned earlier consists of two canvases, one for the source category and another for the target category. There are no property windows in this tab. The source and target categories are imported from XML files and based on the input, the models are constructed. It is not possible to make changes to the category models in this tab. Figure 6.15 shows the layout of the Functor tab.

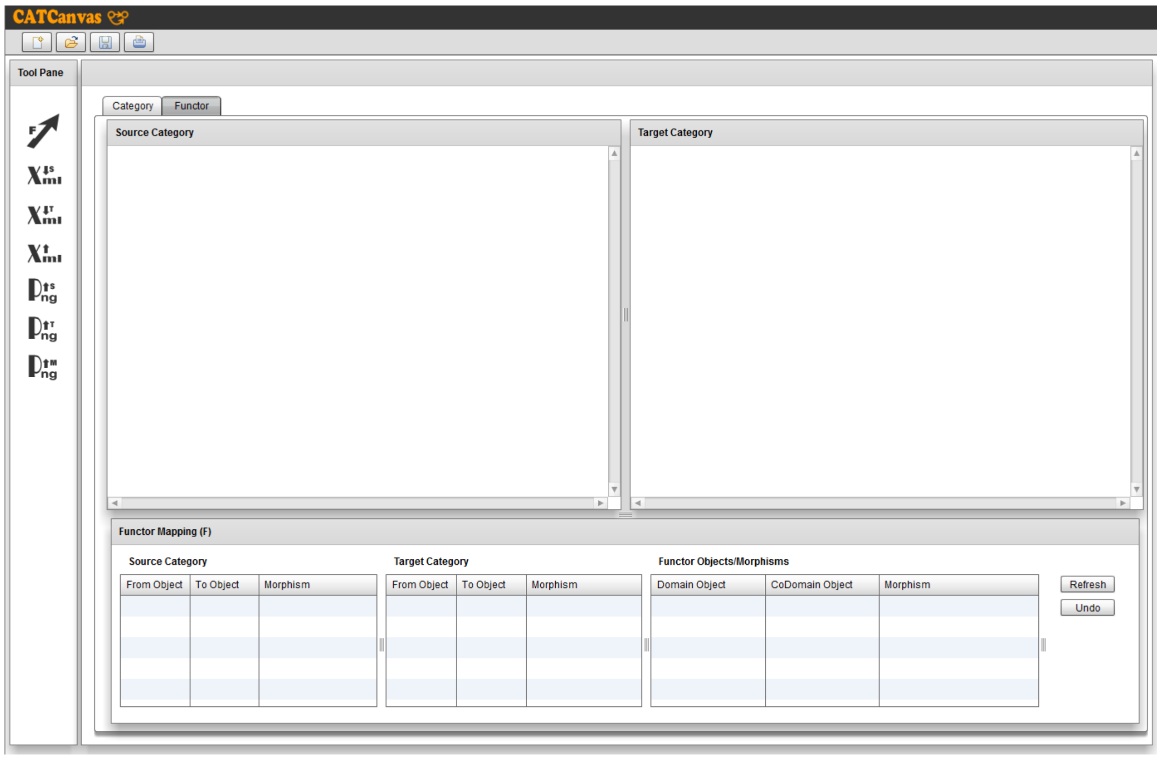
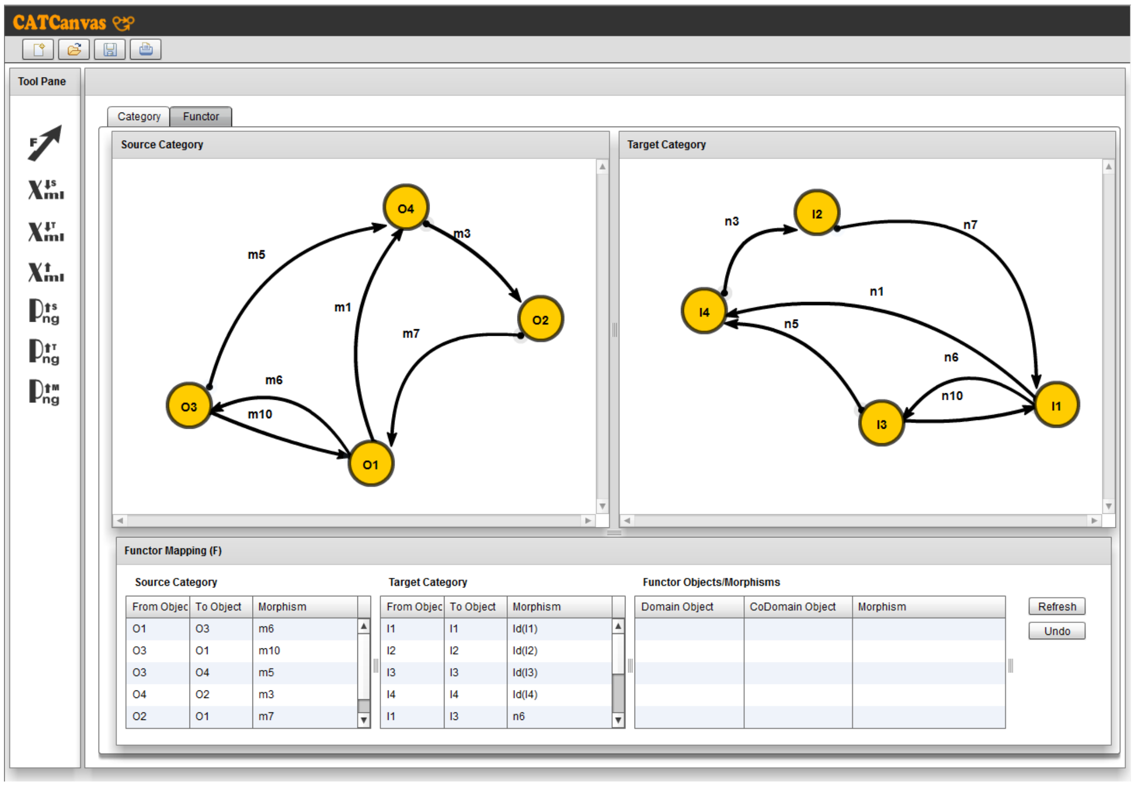


Figure 6.15: CATCanvas Functor Tab

6.3.13 Functor Mapping

From the definition of a Functor in chapter 3, a functor maps all objects in source category to selected objects of the target category. This is done in the panel “Functor Mapping (F)” shown in Figure 6.15. The ‘refresh’ button updates the table with data from the imported files. The purpose of the ‘undo’ button is to revert a wrongly done mapping. ‘Undo’ is like a queue working on the ‘last in last out’ principle until the very first mapping. Figure 6.16 shows the first few steps in construction of a functor.



Source Category

From Objec	To Object	Morphism
O1	O3	m6
O3	O1	m10
O3	O4	m5
O4	O2	m3
O2	O1	m7

Target Category

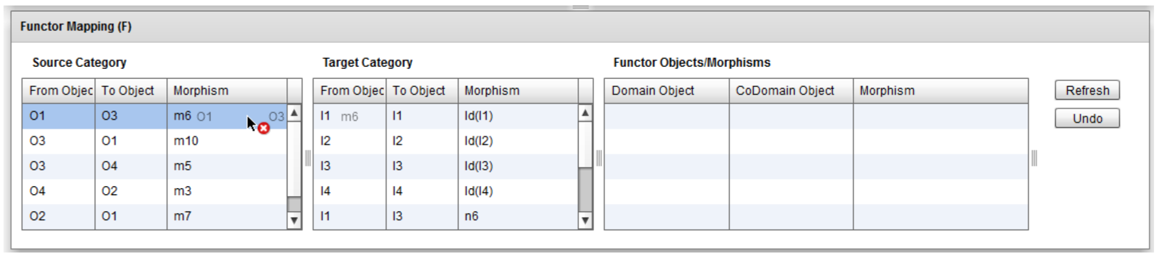
From Objec	To Object	Morphism
I1	I1	Id(I1)
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1	I3	n6

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism

Figure 6.16: Source and Target Categories Imported

To begin the mapping, first the morphism row to be mapped is selected from the ‘Source Category’ table in the Functor Mapping Panel. To perform the mapping the selected row is dragged using the mouse click from the ‘Source Category’ table and dropped onto a morphism row in the ‘Target Category’ table. As soon as the morphism row starts dragging, an indicator icon appears by its side indicating whether it is possible to drop the row at the point it is currently at. A red indicator icon with a white cross indicates it is not possible to drop the row or do the mapping, while a green icon with a white plus indicates the possibility of a mapping.



Functor Mapping (F)

Source Category

From Objec	To Object	Morphism
O1	O3	m6 O1 O3
O3	O1	m10
O3	O4	m5
O4	O2	m3
O2	O1	m7

Target Category

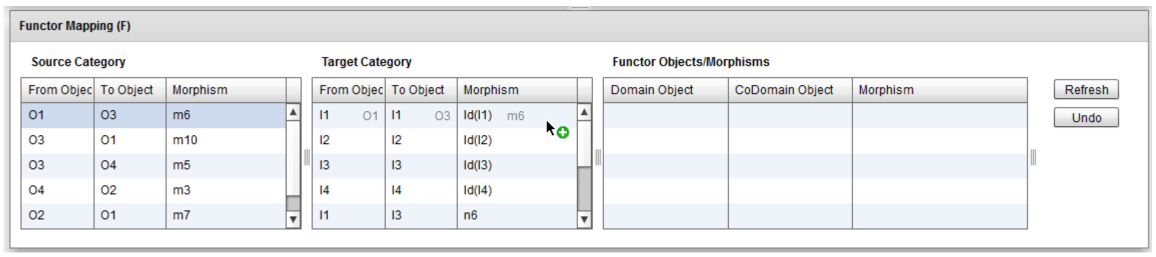
From Objec	To Object	Morphism
I1 m6	I1	Id(I1)
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1	I3	n6

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism

Figure 6.17: Source Morphism Row being Dragged with a Red Indicator Icon

While the morphisms are being mapped in this process, the source and target objects of the morphisms are mapped as well. Figure 6.17 shows the dragging row with a red indicator icon. Figure 6.18 shows a row being dragged with a green indicator icon.



Functor Mapping (F)

Source Category

From Objec	To Object	Morphism
O1	O3	m6
O3	O1	m10
O3	O4	m5
O4	O2	m3
O2	O1	m7

Target Category

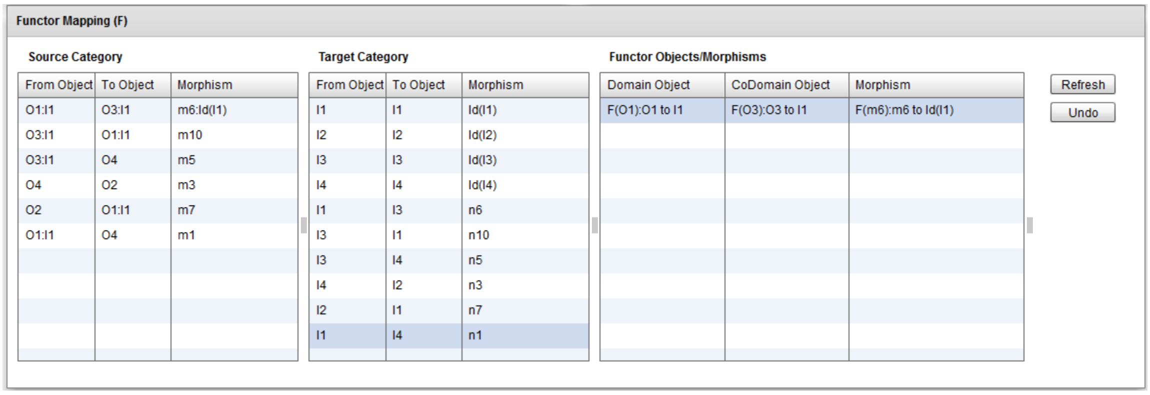
From Objec	To Object	Morphism
I1 O1	I1 O3	Id(I1) m6
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1	I3	n6

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism

Figure 6.18: Source Morphism Row being Dragged with a Green Indicator Icon

As soon as the source category morphism row is dropped onto the target category morphism row, the table Functor Objects/Morphism is instantly populated with the freshly done mapping. Also, the ‘Source Category’ table gets updated with the mapped objects indicating the target category mapped object. Similarly, the mapped morphism in the ‘Source Category’ gets updated with the mapping information similar to this m6:Id(I1). This also helps the user performing the mapping to know what objects and morphism have been mapped already. Figure 6.19 shows a successful morphism/object mapping.



Source Category

From Object	To Object	Morphism
O1:I1	O3:I1	m6:Id(I1)
O3:I1	O1:I1	m10
O3:I1	O4	m5
O4	O2	m3
O2	O1:I1	m7
O1:I1	O4	m1

Target Category

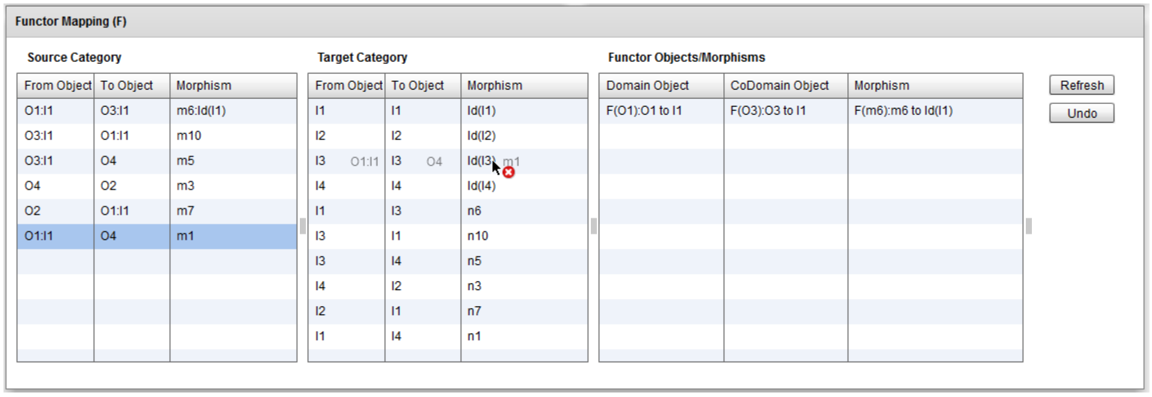
From Object	To Object	Morphism
I1	I1	Id(I1)
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1	I3	n6
I3	I1	n10
I3	I4	n5
I4	I2	n3
I2	I1	n7
I1	I4	n1

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism
F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)

Figure 6.19: Fist Morphism/Object Mapping

For the morphism that has not yet been mapped but the source and/or target objects have been mapped already in the first mapping, the mapping would be intelligent to restrict the mapping wherever necessary. For example, in Figure 6.18 morphism **m1** has not been mapped yet but in the first mapping, object **O1** in source category was mapped to object **I1** in target, so the possible mappings for **m1** would be **n1**, **n6** and **Id(I1)** in the target category. The tool will restrict the mapping for any other morphism as show in Figure 6.20 and Figure 6.21.



Source Category

From Object	To Object	Morphism
O1:I1	O3:I1	m6:Id(I1)
O3:I1	O1:I1	m10
O3:I1	O4	m5
O4	O2	m3
O2	O1:I1	m7
O1:I1	O4	m1

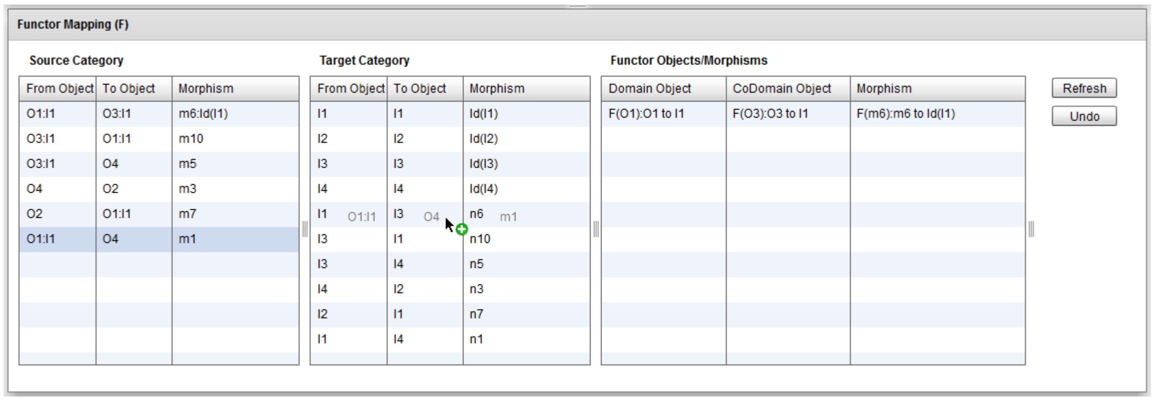
Target Category

From Object	To Object	Morphism
I1	I1	Id(I1)
I2	I2	Id(I2)
I3 O1:I1	I3 O4	Id(I3) m1
I4	I4	Id(I4)
I1	I3	n6
I3	I1	n10
I3	I4	n5
I4	I2	n3
I2	I1	n7
I1	I4	n1

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism
F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)

Figure 6.20: Attempt to Map m1 in Source Category to Id(I3) in Target Category



Source Category

From Object	To Object	Morphism
O1:I1	O3:I1	m6:Id(I1)
O3:I1	O1:I1	m10
O3:I1	O4	m5
O4	O2	m3
O2	O1:I1	m7
O1:I1	O4	m1

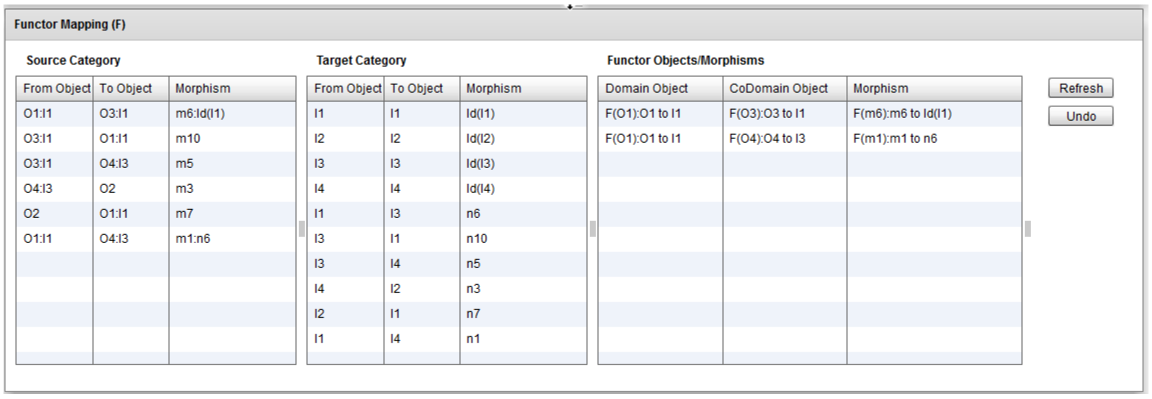
Target Category

From Object	To Object	Morphism
I1	I1	Id(I1)
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1 O1:I1	I3 O4	n6 m1
I3	I1	n10
I3	I4	n5
I4	I2	n3
I2	I1	n7
I1	I4	n1

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism
F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)

Figure 6.21: Mapping m1 in Source Category to n6 in Target Category



Source Category

From Object	To Object	Morphism
O1:I1	O3:I1	m6:Id(I1)
O3:I1	O1:I1	m10
O3:I1	O4:I3	m5
O4:I3	O2	m3
O2	O1:I1	m7
O1:I1	O4:I3	m1:n6

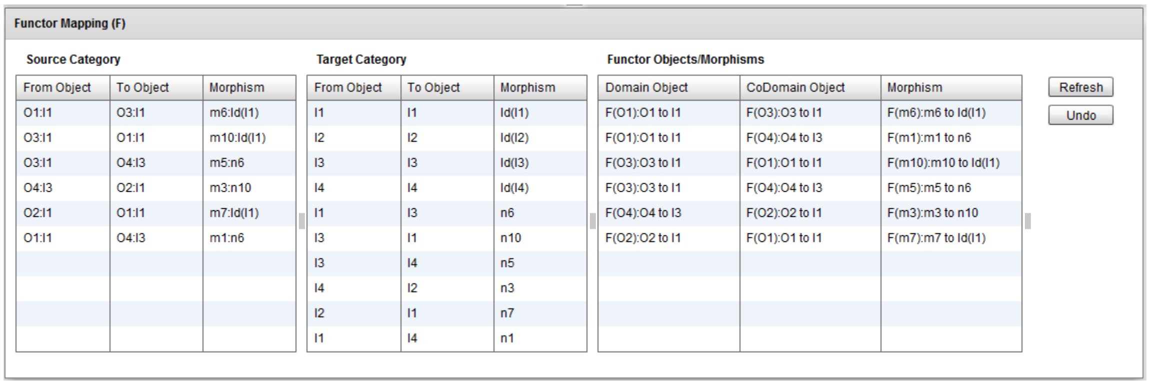
Target Category

From Object	To Object	Morphism
I1	I1	Id(I1)
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1	I3	n6
I3	I1	n10
I3	I4	n5
I4	I2	n3
I2	I1	n7
I1	I4	n1

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism
F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)
F(O1):O1 to I1	F(O4):O4 to I3	F(m1):m1 to n6

Figure 6.22: Successful Mapping of m1 in Source Category to n6 in Target Category



Functor Mapping (F)

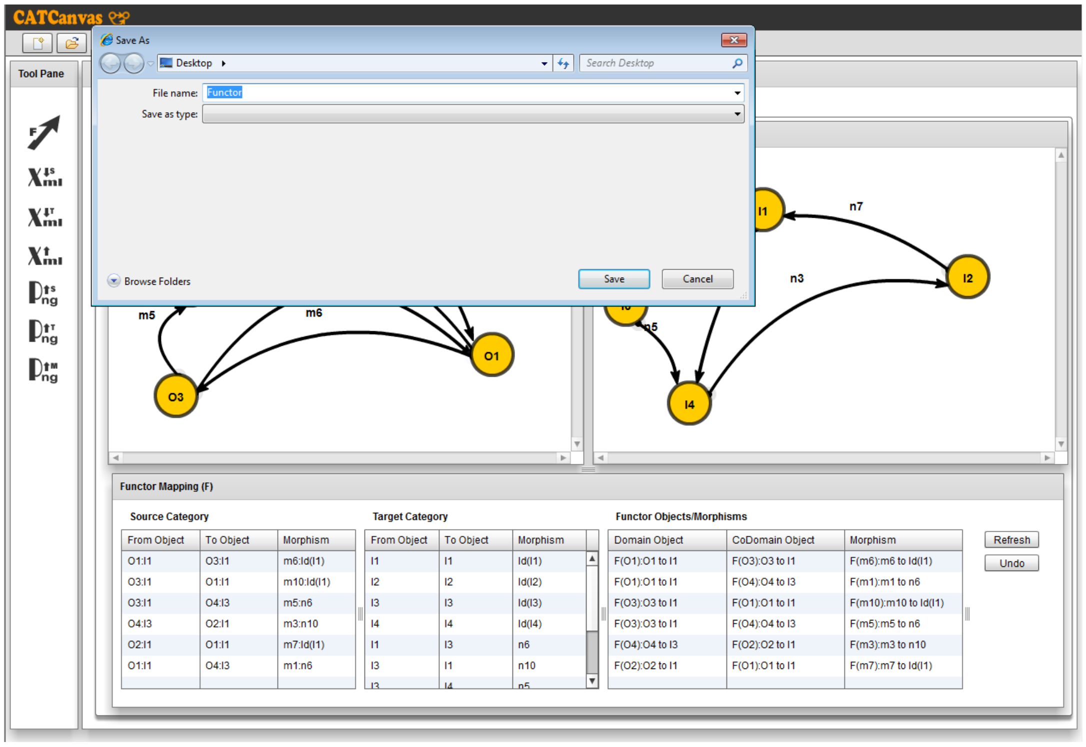
Source Category			Target Category			Functor Objects/Morphisms		
From Object	To Object	Morphism	From Object	To Object	Morphism	Domain Object	CoDomain Object	Morphism
O1:I1	O3:I1	m6:Id(I1)	I1	I1	Id(I1)	F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)
O3:I1	O1:I1	m10:Id(I1)	I2	I2	Id(I2)	F(O1):O1 to I1	F(O4):O4 to I3	F(m1):m1 to n6
O3:I1	O4:I3	m5:n6	I3	I3	Id(I3)	F(O3):O3 to I1	F(O1):O1 to I1	F(m10):m10 to Id(I1)
O4:I3	O2:I1	m3:n10	I4	I4	Id(I4)	F(O3):O3 to I1	F(O4):O4 to I3	F(m5):m5 to n6
O2:I1	O1:I1	m7:Id(I1)	I1	I3	n6	F(O4):O4 to I3	F(O2):O2 to I1	F(m3):m3 to n10
O1:I1	O4:I3	m1:n6	I3	I1	n10	F(O2):O2 to I1	F(O1):O1 to I1	F(m7):m7 to Id(I1)
			I3	I4	n5			
			I4	I2	n3			
			I2	I1	n7			
			I1	I4	n1			

Refresh Undo

Figure 6.23: Functor Mapping Complete

The mapping of the morphisms continues as shown in Figure 6.22 until there are no more morphisms and objects left to be mapped.

6.3.14 Functor XML Export



File name: Functor

Save as type:

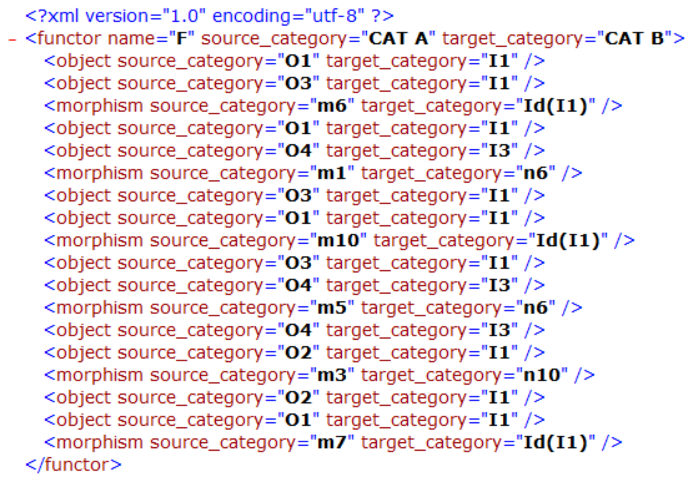
Functor Mapping (F)

Source Category			Target Category			Functor Objects/Morphisms		
From Object	To Object	Morphism	From Object	To Object	Morphism	Domain Object	CoDomain Object	Morphism
O1:I1	O3:I1	m6:Id(I1)	I1	I1	Id(I1)	F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)
O3:I1	O1:I1	m10:Id(I1)	I2	I2	Id(I2)	F(O1):O1 to I1	F(O4):O4 to I3	F(m1):m1 to n6
O3:I1	O4:I3	m5:n6	I3	I3	Id(I3)	F(O3):O3 to I1	F(O1):O1 to I1	F(m10):m10 to Id(I1)
O4:I3	O2:I1	m3:n10	I4	I4	Id(I4)	F(O3):O3 to I1	F(O4):O4 to I3	F(m5):m5 to n6
O2:I1	O1:I1	m7:Id(I1)	I1	I3	n6	F(O4):O4 to I3	F(O2):O2 to I1	F(m3):m3 to n10
O1:I1	O4:I3	m1:n6	I3	I1	n10	F(O2):O2 to I1	F(O1):O1 to I1	F(m7):m7 to Id(I1)
			I3	I4	n5			

Refresh Undo

Figure 6.24: Functor XML Import Save File

Once the mapping is complete, it is possible to generate the XML for the constructed Functor to an output file. Figure 6.24 shows a 'save' dialog box for the generated XML. The XML specification for the constructed functor is included in Figure 6.25.



```
  <?xml version="1.0" encoding="utf-8" ?>
- <functor name="F" source_category="CAT A" target_category="CAT B">
    <object source_category="O1" target_category="I1" />
    <object source_category="O3" target_category="I1" />
    <morphism source_category="m6" target_category="Id(I1)" />
    <object source_category="O1" target_category="I1" />
    <object source_category="O4" target_category="I3" />
    <morphism source_category="m1" target_category="n6" />
    <object source_category="O3" target_category="I1" />
    <object source_category="O1" target_category="I1" />
    <morphism source_category="m10" target_category="Id(I1)" />
    <object source_category="O3" target_category="I1" />
    <object source_category="O4" target_category="I3" />
    <morphism source_category="m5" target_category="n6" />
    <object source_category="O4" target_category="I3" />
    <object source_category="O2" target_category="I1" />
    <morphism source_category="m3" target_category="n10" />
    <object source_category="O2" target_category="I1" />
    <object source_category="O1" target_category="I1" />
    <morphism source_category="m7" target_category="Id(I1)" />
  </functor>
```

Figure 6.25: Functor Exported XML

6.3.15 Saving PNG Image

The tool makes it possible to save a snapshot of the constructed graphical model in .png format for the source category, the target category and the mapping table.

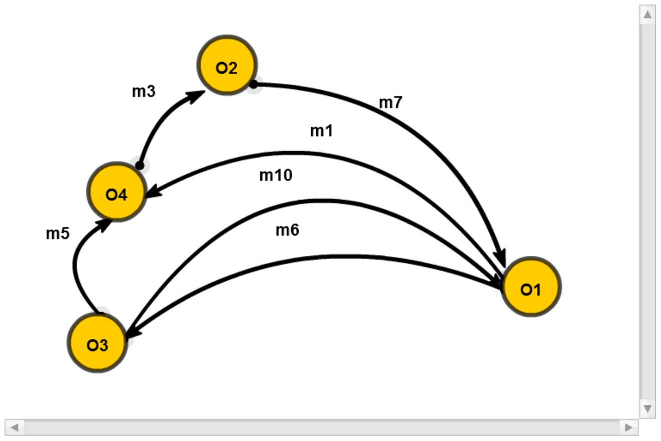


Figure 6.26: Functor – Source Category PNG

Figure 6.26, 6.27 and 6.28 show the captured snapshot for the source category, the target category and the mapping table respectively.

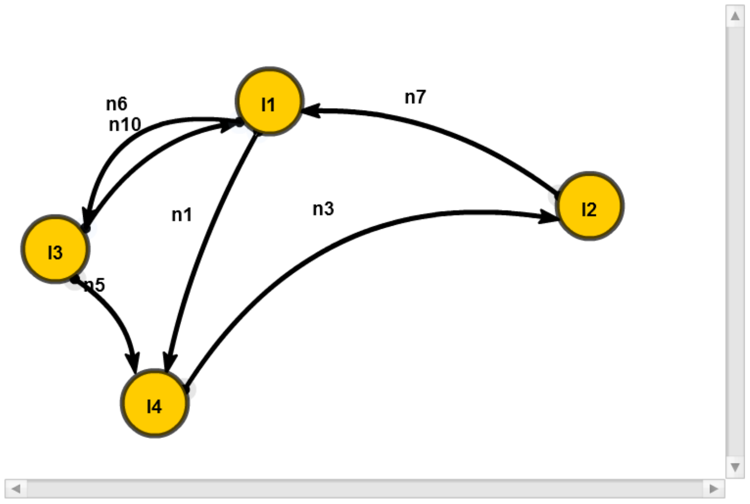
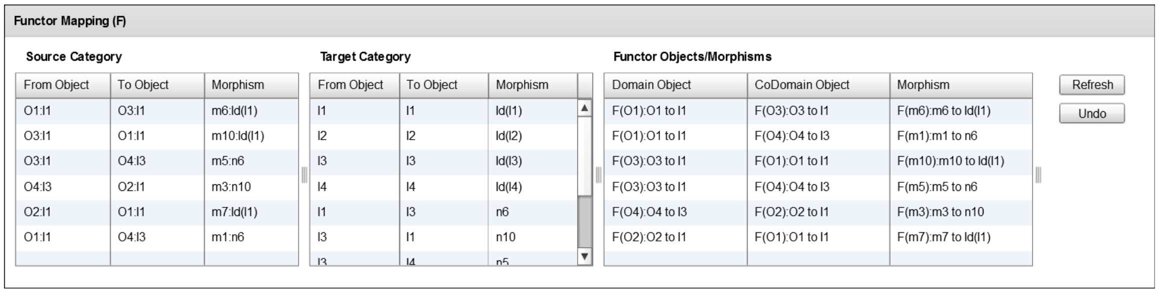


Figure 6.27: Functor – Target Category PNG



Source Category

From Object	To Object	Morphism
O1:I1	O3:I1	m6:Id(I1)
O3:I1	O1:I1	m10:Id(I1)
O3:I1	O4:I3	m5:n6
O4:I3	O2:I1	m3:n10
O2:I1	O1:I1	m7:Id(I1)
O1:I1	O4:I3	m1:n6

Target Category

From Object	To Object	Morphism
I1	I1	Id(I1)
I2	I2	Id(I2)
I3	I3	Id(I3)
I4	I4	Id(I4)
I1	I3	n6
I3	I1	n10
I3	I4	n5

Functor Objects/Morphisms

Domain Object	CoDomain Object	Morphism
F(O1):O1 to I1	F(O3):O3 to I1	F(m6):m6 to Id(I1)
F(O1):O1 to I1	F(O4):O4 to I3	F(m1):m1 to n6
F(O3):O3 to I1	F(O1):O1 to I1	F(m10):m10 to Id(I1)
F(O3):O3 to I1	F(O4):O4 to I3	F(m5):m5 to n6
F(O4):O4 to I3	F(O2):O2 to I1	F(m3):m3 to n10
F(O2):O2 to I1	F(O1):O1 to I1	F(m7):m7 to Id(I1)

Figure 6.28: Functor – Functor Mapping Panel PNG

6.4 Conclusion

The chapter presented a concise discussion of the first release of the CATCanvas, the CAT modeling tool. An attempt has been made to include a list of all available features. The functionality that will be developed in the future releases of the tool along with a list of possible extensions is included in chapter 7.

7 Conclusion & Future Work

Swarm-based systems are based on the concept of an insect swarm depicting entities of the swarm interacting to accomplish a set of goals. To achieve a behavior similar to an insect colony, a swarm-based system is inherently autonomous and autonomic. Because of the behavioral complexity, modeling and specification of such systems is a challenge. This thesis proposes a modeling language termed Categorical Modeling Language or in short, CML for specification of the complex behavior of a swarm-based system, for example, ANTS-based PAM. CML is primarily based on Category Theory (CAT) in mathematics. The contributions of the work presented in this thesis are summarized below:

1. Study the application of CAT to software engineering domain.
2. Application of CAT as a formal method for behavioral specification of swarm-based systems.
3. Proposing a modeling language based on CAT as a formal method.
    a. Construction of a grammar for the specification language.
    b. Defining the visual/graphical model notation.
4. Application of the proposed modeling language to NASAs PAM case study.
5. Development of a modeling tool for the proposed modeling language.

CAT and swarm-based systems have the social life concept in common which laid the foundation for this research. The case study, NASA's PAM concept mission is based on NASA's swarm-based ANTS architecture. The researchers at NASA in [21, 22, 23] also indicated the possibility of CAT to serve as a formal method for the behavioral specification of swarm-based systems including PAM. CAT consists of a set of constructs that are applicable to the concept of a category. Each construct consists of a set of axioms that must hold true for the application of that construct. For the course of this research, only a subset of CAT constructs were studied for application to the case study. CML introduced in this thesis is proposed as a model-based specification language based on CAT as a formal method for behavioral specification. The language consists of visual models as well as specification of the visual models. A significant portion of the work done in this research goes into writing of an EBNF based grammar for the CML specification. Together with the visual models and the specification, CML could prove to be a promising methodology for reasoning about swarm-based systems and for behavioral specification of such systems. As for the future work, the above-mentioned list of goals could be used as a point of reference.

Verifying Emergent Behavior

One of the few challenges of specifying a swarm-based system is specification and verification of the emergent behavior. CML has not yet been studied to verify the specified emergent behavior of a system. This could be a possible path for future research.

CAT as a Formal Method

One of the possible directions for future work could be conducting further study on category theory constructs not included in this thesis.

CML to Include More Constructs

So far CML doesn’t include the grammar for all of the CAT constructs. In future, all of the remaining CAT constructs could be included in CML.

Specifying Autonomic Behavior

In this thesis only two of the self-* autonomic properties of PAM were studied and modelled using CML. In future, the other properties could be studied and modelled using CML.

CATCanvas

The modeling tool presented in this thesis known as CATCanvas has room for additions and improvements:

a. So far only typed category and functor constructs have been implemented. Future work can include implementation of Natural Transformation, Limit and Co-Limit.
b. The tool generates XML specification at this point. In the future, the tool could be extended to include the CML specification as well.
c. Some bugs in the current version could be fixed in the future versions:
    1. Deleting an object
    2. Upon deletion of morphism, the morphism name should be removed as well.

d. Layouts for the visual models

e. The tool could be extended to save project files in addition to xml output.

8 References

[1] J. O. Kephart, D. M. Chess, The Vision of Autonomic Computing, Computer, Volume 36, No. 1, January 2003, pp:41- 43.

[2] P. Horn, Autonomic Computing: IBM's perspective on the state of information technology, Technical report, IBM T. J. Watson Laboratory, October 2001.

[3] A. G. Ganek, T. A. Corbi, The Dawning of the Autonomic Computing Era, IBM Systems Journal, Volume 42, No. 1, January 2003.

[4] Sterritt, R. and Bustard, D., Autonomic Computing - A Means of Achieving Dependability?, Proceedings of IEEE International Conference on the Engineering of Computer Based System (ECBS'03), Huntsville, Alabama, USA; April 2003.

[5] Sterritt, R., Parashar, M., Tianfield, H., and Unland, R. A Concise Introduction to Autonomic Computing. Adv. Eng. Informatics 19, 2005, pp:2.

[6] M. Parashar and S. Hariri. Autonomic Computing: An overview. Hot Topics, Springer LNCS, www.caip.rutgers.edu/TASSL/Papers/automate-upp-overview-05.pdf.

[7] IBM Corporation. An Architectural Blueprint for Autonomic Computing. April 2003.

[8] P. Faratin, C. Sierra and N.R. Jennings, Negotiation decision functions for autonomous agents. Robotics and Autonomous Systems. 1998, pp: 2,3.

[9] P.J. Antsaklis and K.M. Passino. Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues. Journal of Intelligent and Robotic Systems, Vol. 1, 1989.

[10] E. Bonabeau, M. Dorigi, G. Theraulaz, Swarm Intelligence. From Natural to Artificial Systems. Oxford University Press, Oxford, 1999, pp:1-9.

[11] G. Beni. The Concept of Cellular Robotics. In Proc. 1988 IEEE International Symposium on Intelligent Control, Los Alamitos, 1988.

[12] G. Beni and J. Want. Swarm Intelligence. In Proc. Seventh Annual Meeting of the Robotics Society of Japan, Tokyo, 1989, pp:1,2.

[13] P.J. Antsaklis and K.M. Passino. Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues. Journal of Intelligent and Robotic Systems, Vol. 1, 1989, pp:315-320.

[14] E. Bonabeau and G. Théraulaz. Swarm Smarts. Scientific American, March 2000.

[15] W. Truszkowski, M. Hinchey, J. Rash, C. Rouff, NASA's swarm missions: The challenge of building autonomous software. IEEE IT Professional Mag 2004; September/October, pp:51–6.

[16] S. A. Curtis, W. F. Truszkowski, M. L. Rilee, and P. E. Clark, ANTS for the human exploration and development of space. In Proc. IEEE Aerospace Conference, 2003, pp: 3-5.

[17] M.G. Hinchey, J.L. Rash, W.F. Truszkowski, C.A. Rouff and R. Sterritt, Autonomous and Autonomic Swarms, In Proceedings of Autonomic & Autonomous Space Exploration Systems (A&A-SES-1) at 2005 International Conference on Software Engineering Research and Practice (SERP'05), Las Vegas, June, 2005.

[18] W. F. Truszkowski, J. L. Rash, C. A. Rouff, and M. G. Hinchey. Asteroid Exploration with Autonomic Systems. In Proc. 11th IEEE International Conference and

Workshop on the Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe), Brno, May 2004.

[19] A. F. T. Winfield, J. Sa, M-C Fernandez Gago, C. Dixon, and M. Fisher. On formal specification of emergent behaviours in swarm robotic systems. International Journal of Advanced Robotic Systems, December 2005, pp: 364-366.

[20] V. S. Alagar and K. Periyasamy, Specification of Software Systems, New York, Springer, 2nd Edition, 2011, pp: 12,13,38.

[21] M. G. Hinchey, Requirements of an integrated formal method for intelligent swarms, presented at the Proceedings of the 10th international workshop on Formal methods for industrial critical systems, Lisbon, 2005.

[22] Rouff, C., Hinchey, M., Truszkowski, T., and Rash, J. (2004). Formal methods for autonomic and swarm-based systems. In 1st International Symposium on Leveraging Applications of Formal Methods, Cyprus, 2004, pp:2-6.

[23] C. Rouff, W. Truszkowski, J. Rash, and M. Hinchey, Formal approaches to intelligent swarms. In IEEE/NASA Software Engineering Workshop, 2003.

[24] M. Wing. Jeannette, A specifier's introduction to formal methods, September 1990, pp: 10-11.

[25] N. Khurshid, O. Ormandjieva, S. Klasa, Towards a Tool Support for Specifying Complex Software Systems using Categorical Modeling Language, Springer LNCS, 8th ACIS Conference on Software Engineering Research, Management and Applications (SERA'10) Montreal, 2010.

[26] H. Shu and J. Malec, From Process Transition Networks to Behavior Automata. In Proceedings of the IEEE International Symposium on Intelligent Control, Arlington, 1991.

[27] C.A.R. Hoare, Communicating Sequential Processes. Communications of the ACM, August, 1978.

[28] C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash, Properties of a Formal Method for Prediction of Emergent Behaviors in Swarm-based Systems, 2nd IEEE International Conference on Software Engineering and Formal Methods, Beijing, September, 2004, pp:7-9.

[29] C. A. Rouff, W. F. Truszkowski, M. G. Hinchey, and J. L. Rash. Verification of emergent behaviors in swarm based systems. In Proc. 11th IEEE, International Conference on Engineering Computer- Based Systems (ECBS), Workshop on Engineering Autonomic Systems (EASe), Brno, Czech Republic, May 2004. IEEE Computer Society Press, pp:446-448.

[30] M. Kloetzer and C. Belta, Temporal logic planning and control of robotic swarms by hierarchical abstractions, In *IEEE Conference on Decision and Control*, pp: 2619–2624, San Diego, 2006.¸

[31] J. R. Kiniry. The specification of dynamic distributed component systems. Master's thesis, California Institute of Technology, 1998.

[32] Number of Species on Earth, Current Results, January 2007, http://www.currentresults.com/Environment-Facts/Plants-Animals/number-species.php.

[33] Honey Bee Communication, December 2010, http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/B/BeeDances.html

[34] D. J. T. Sumpter, G. B. Blanchard, and D. S. Broomhead, Ants and agents: a process algebra approach to modelling ant colony behaviour, Bulletin of Mathematical Biology, September 2001, pp:951-960.

[35] C. Tofts, Describing social insect behavior using process algebra, Transactions on Social Computing Simulation, 1991, pp:227-283.

[36] S. Eilenberg, Automata, Languages and Machines, Vol. A, Academic press, N.Y. 1974.

[37] E. Vassev, J. Paquet, ASSL - Autonomic System Specification Language, Proceedings of the 31st IEEE Software Engineering Workshop, March 2007, pp:300-309.

[38] E. Vassev, M. Hinchey, and J. Paquet, Towards an ASSL Specification Model for NASA Swarm-Based Exploration Missions, In Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008), ACM, 2008.

[39] Reino Kurki-Suonio. A Practical Theory of Reactive Systems. Springer, 2005.

[40] L. Aceto, A. Ingolfsdottir, K. G. Larsen, and J. Srba. Reactive Systems: Cambridge University Press, 2007.

[41] S. Whitmire. Object Oriented Design Measurement. Whiley C. P. New York, 1997.

[42] M. Barr, C. Wells. Category Theory for Computing Science. Prentice-Hall, NJ, 1990.

[43] J. Fiadeiro. Categories for Software Engineering. Springer. Berlin Heidelberg, 2005.

[44] S. Eilenberg, S. Mac Lane. General Theory of Natural Equivalences. Transactions of the American Mathematical Society, vol. 58, pp. 231–294, 1945.

[45] Mac Lane, Saunders, Basic Category Theory for Computer Scientists, 1998.

[46] Categories for the Working Mathematician. Graduate Texts in Mathematics. Springer-Verlag.

[47] S.A. Curtis, M.L. Rilee, P.E. Clark, and G.C. Marr, Use of Swarm Intelligence in Spacecraft Constellations for the Resource Exploration of the Asteroid Belt, Third International Workshop on Satellite Constellations and Formation Flying, 2003.

[48] Clark, P E, et al. SMART Power Systems for ANT Missions. Chantally, VA: 2004.

[49] M.L. Rilee, S.A. Curtis, P.E. Clark, C.Y. Cheung, W. Truszkowski, From Buses to Bodies: SMART Matter for System Applications, IAC Proceedings, 2004.

[50] M.L. Rilee, S.A. Curtis, P.E. Clark, C.Y. Cheung, W. Truszkowski, An Implementable Pathway to SMART Matter for Adaptive Structures, IAC Proceedings, 2004.

[51] P.E. Clark, J. Iyengar, M.L. Rilee, W. Truszkowski, S.A. Curtis, A Conceptual Framework for Developing Intelligent Software Agents as Space Explorers, 2002.

[52] P.E. Clark, M.L. Rilee, S.A. Curtis, C.Y. Cheung, G. Marr, W. Truszkowski, M. Rudisill, PAM: Biologically Inspired Engineering and Exploration Mission Concept, Components and Requirements for Asteroid Population Survey, IAC Proceedings, IAC-04-Q5.07, 2004.

[53] R. Cervenka, D. Greenwood, and I. Trencansky, The AML Approach to Modeling Autonomic Systems, Proceedings of the 1st International Conference on Autonomic and Autonomous Systems, Silicon Valley, USA, 2006.

[54] S. A. Curtis, J. Mica, J. Nuth, G. Marr, M. L. Rilee, and M. K. Bhat. ANTS (Autonomous Nano-Technology Swarm): An Artificial Intelligence Approach to Asteroid

Belt Resource Exploration, In Proc. Int'l Astronautical Federation, 51st Congress, October 2000.

[55] W. F. Truszkowski, M. G. Hinchey, J. L. Rash, and C. A. Rouff. Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions. IEEE Transactions on Systems, Man and Cybernetics, Part C, 2006.

[56] P. Ivankov, Category Theory, http://www.mathframe.com/downloads/categorytheory /index.html

[57] J.A. Beachy, Abelian Groups, http://www.math.niu.edu/~beachy/rings_modules/notes /03.pdf

[58] Adobe Flex, http://www.adobe.com/products/flex/

[59] M. Holcombe, X-Machines as basis for dynamic systems specification, Software Engineering Journal, Vol. 3, No. 2, 1988.

9 Appendix.

Pull-Back

```
<Pull-Back>::=          Category: <Typed_Category> (<Cat_name> , <Cat_Id>)
                        PULLBACK (Pullback Object: <Obj_Pullback>,
                                  Triplet Object: <Obj_Triplet>,
                                  Pullback Morphisms: <PB_Mor>, <PB_Mo>
                                  Triplet Morphisms: <T_Mor>, <T_Mo>
                                  Unique Morphism: <PB_Mor_Unique>)
                        Diagram Objects <PB_Object><PB_Object><PB_Object>
                        Diagram Morphisms <PB_Morphism><PB_Morphism>
                        Pullback Axiom  {<PB_Axiom>}+

<Obj_Pullback>::=       <Obj_Terminal>

<Obj_Terminal>::=       <Obj_type_Id><Obj_instance_Id>

<Obj_Triplet>::=        <Obj_type_Id><Obj_instance_Id>

<PB_Mor>::=             <Mor_type_Id><Mor_instance_Id>:
                        <Obj_Pullback>  →  <Obj_type_Id> <Obj_instance_Id>

<T_Mor>::=              <Mor_type_Id><Mor_instance_Id>:
                        <Obj_Triplet>  → <Obj_type_Id> <Obj_instance_Id>

<PB_Mor_Unique>::= <Mor_Id>: <Obj_type_Id> <Obj_instance_Id>  →  <Obj_Pullback>

<PB_Object>::=          Obj_Type: <Obj_type> <Obj_type_Id><Obj_instance_Id>
                                  {,<Obj_type_Id><Obj_instance_Id>}*

<PB_Morphism>::=        Mor_Type: <Mor_type> : <Mor_type_Id><Mor_instance_Id>:
                        <Obj_type_Id> <Obj_instance_Id>  →  <Obj_type_Id> <Obj_instance_Id>

<PB_Axiom>:=            <PB_Mor> o <PB_Morphism>  =  <PB_Mor> o <PB_Morphism>
```

PAM Petrologist Team Messenger Scenario Modeled using PULLBACK construct

Category :(Photogeologist Team, PGT)
PULLBACK
Pullback Object: TM_3,
Triplet Object: L_2,
Pullback Morphisms: $tm_1 : TM_3 \rightarrow W_{IR2}$,
$tm_2 : TM_3 \rightarrow W_{IM2}$
Triplet Morphisms: $l_1 : L_2 \rightarrow W_{IR2}$,
$l_2 : L_2 \rightarrow W_{IM2}$
Unique Morphisms: $u: L_2 \rightarrow TM_3$

Diagram Objects
Worker: W_{IM2}, W_{IR2}, W_{RS2}

Diagram Morphisms
Cooperation: $c_{21}: W_{IM2} \rightarrow W_{RS2}$,
$c_{18}: W_{IR2} \rightarrow W_{RS2}$

Pullback Axiom
$tm_1 \circ c_{18} = tm_2 \circ c_{21}$

Pullback TM_3

$$\begin{tikzcd}
 & L_2 \arrow[ddl, "l_2"'] \arrow[ddr, "l_1"] \arrow[d, dashed, "\mathbf{u}"'] & \\
 & TM_3 \arrow[dl, "tm_2"'] \arrow[dr, "tm_1"] & \\
W_{IM2} \arrow[dr, "c_{21}"'] & & W_{IR2} \arrow[dl, "c_{18}"] \\
 & W_{RS2} &
\end{tikzcd}$$

Push-Out

<Push-Out >::= ***Category***: <Typed_Category> (<Cat_name> , <Cat_Id>)
PUSHOUT(Pushout Object: <Obj_Pushout>,
Triplet Object: <Obj_Triplet_PO>,
Pushout Morphisms: <PO_Mor>, <PO_Mo>
Triplet Morphisms: <POT_Mor>, <POT_Mo>
Unique Morphism: <PO_Mor_Unique>)
Diagram Objects <PO_Object><PO_Object><PO_Object>
Diagram Morphisms <PO_Morphism><PO_Morphism>
Pullback Axiom {<PO_Axiom>}$^+$

<Obj_Pushout>::= <Obj_Initial>

<Obj_Initital>::= <Obj_type_Id><Obj_instance_Id>

<Obj_Triplet_PO>::= <Obj_type_Id><Obj_instance_Id>

<PO_Mor>::= <Mor_type_Id><Mor_instance_Id>: <Obj_type_Id> <Obj_instance_Id> →
<Obj_Pushout>

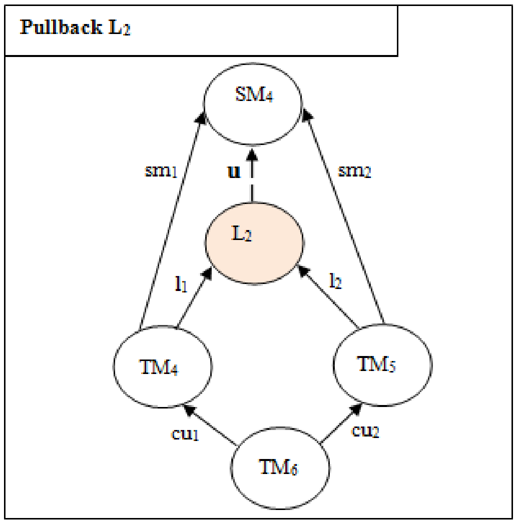
<POT_Mor>::= <Mor_type_Id><Mor_instance_Id>: <Obj_type_Id> <Obj_instance_Id> →
<Obj_Triplet_PO>

<PO_Mor_Unique>::= <Mor_Id>: <Obj_Pushout> → <Obj_type_Id> <Obj_instance_Id>

<PO_Object>::= Obj_Type: <Obj_type> <Obj_type_Id><Obj_instance_Id>
{,<Obj_type_Id><Obj_instance_Id>}*

<PO_Morphism>:= Mor_Type: <Mor_type> : <Mor_type_Id><Mor_instance_Id>:
<Obj_type_Id> <Obj_instance_Id> → <Obj_type_Id> <Obj_instance_Id>

<PO_Axiom>:= <PO_Morphism> o <PO_Mor> = <PO_Morphism> o <PO_Mor>



PAM Sub-Swarm Leader Team Messenger Scenario Modeled using PUSHOUT construct

Category :(Sub-Swarm, S_3)
PUSHOUT
Pushout Object: L_2,
Triplet Object: SM_4,
Pushout Morphisms: $l_1 : TM_4 \rightarrow L_2$,
$l_2 : TM_5 \rightarrow L_2$
Triplet Morphisms: $sm_2 : TM_5 \rightarrow SM_4$,
$sm_1 : TM_4 \rightarrow SM_4$
Unique Morphisms: $u: L_2 \rightarrow SM_4$

Diagram Objects
Messenger: TM_4, TM_5, TM_6

Diagram Morphisms
Communication: $cu_1: TM_6 \rightarrow TM_4$,
$cu_2: TM_6 \rightarrow TM_5$

Pullback Axiom
$cu_2 \ o\ l_2 = cu_1 \ o\ l_1$

Co-Product

```
<Co-Product>::=          Category: <Typed_Category> (<Cat_name> , <Cat_Id>)
                          CO-PRODUCT (<Co-Prod_name> , <Cat_Id>)
                          Co-Product Objects {Obj_Type: <Obj_type>: <Prod_Object>,}+
                                         Co-Product: <Obj_CP>
                          Co-Product Morphisms {<Co-Prod_Morphisms>}+
                          Co-Product Axioms <Co-Prod_Axioms>

<Co-Prod_Object>::=       <Obj_type_Id> <Obj_instance_Id>

<Obj_CP>::=               <<Obj_type_Id><Obj_instance_Id> x <Obj_type_Id><Obj_instance_Id>>

<Co-Prod_Morphisms>::=   Mor_Type: <Mor_type> :<Mor_type_Id> : <Obj_CP>  →  <Co-Prod_Object>
                              |   <Mor_type> :<Mor_type_Id> : <Obj_CP>  →  <Co-Prod_Object>
                              |   <Mor_type> :  <Mor_unique_Id>: <Co-Prod_Object>  →
                                  <Obj_CP>

<Co-Prod_Axioms>::=        Composition:
                          <Mor_Id>  o <Mor_unique_Id> = <Mor_Id>
```

Slice

```
<Slice>::=                Category: <Typed_Category> (<Cat_name> , <Cat_Id>)
                          SLICE CATEGORY (<Cat_name> , <Slice_Object>)
                          Slice Category Objects Obj_Type: {<S_Cat_Object>,}+
                          Slice Category Morphisms {<S_Cat_Morphism>}+
                          Slice Category  Axioms <S_Cat_Axiom>

<Slice_Object>::=      <Cat_Id>/ <Obj_type_Id> <Obj_instance_Id>

<S_Cat_Object>::=    <Obj_type>: (<Obj_type_Id><Obj_instance_Id> ,
                                   <Slice_Object>, <Mor_type_Id><Mor_instance_Id>)

<S_Cat_Morphism>::= Mor_Type: <Mor_type> :<Mor_type_Id><Mor_instance_Id> :
                                   <Obj_type_Id><Obj_instance_Id> →
                                   <Obj_type_Id><Obj_instance_Id>

<S_Cat_Axiom>:=    Composition: <S_Cat_Morphism> o  <S_Cat_Object>  = <S_Cat_Object>
```

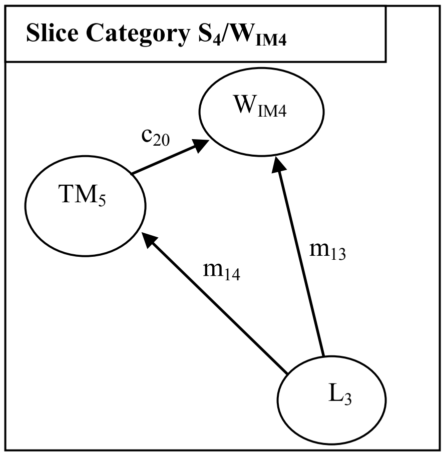
<u>PAM Leader-Messenger-Worker Collaboration Scenario Modeled using SLICE CATEGORY construct</u>

Category**:** (Sub-Swarm, S_4)
SLICE CATEGORY (Imager Collaboration,S_4/W_{IM4})

Slice Category Objects
Obj_Type: Management: (L_3, W_{IM4}, m_{13}),
Cooperation: (TM_5, W_{IM4}, c_{20}),

Slice Category Morphisms
Mor_Type: Management: m_1: $L_1 \rightarrow TM_1$

Slice Category Axioms
Composition: m_{14} o $c_{20} = m_{13}$


Slice Category S4/WIM4
WIM4
c20
TM5
m13
m14
L3