Improvements in Coordination Functionality of Tuple Space model for Mobile Middleware

Suddhasil De

Improvements in Coordination Functionality of Tuple Space model for Mobile Middleware

A Thesis

Submitted in partial fulfilment of the

Requirements for the Degree of

Doctor of Philosophy

by

Suddhasil De

under the Supervision of

Prof. Sukumar Nandi and **Prof. Diganta Goswami**

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Assam – 781039, INDIA

May 2015

Declaration

I hereby declare that,

i. The work contained in this thesis is original and has been done by myself under the supervision of my thesis supervisors.

ii. This thesis, either in full or any part thereof, has not been submitted elsewhere for the award of any degree or diploma.

iii. I have given appropriate credits to the original authors, whenever I used materials (like data, text, figures etc.) from other sources.

iv. Whenever I have used verbatim quotation from other sources, I have put them as per the standard format of quotation, as well as given due credits to the sources by citing them and stating the required details in the references.

v. I have followed the norms and guidelines given in the Ethical Code of Conduct of the Institute.

Suddhasil De

Date:

Dept. of CSE
IIT Guwahati

Certificate

It is to certify that the work contained in this thesis, entitled "**Improvements in Coordination Functionality of Tuple Space model for Mobile Middleware**", being submitted by **Suddhasil De**, who has registered in July 2007 in the Department of Computer Science and Engineering, IIT Guwahati, is a bona fide record of original research work carried out by him under our supervision. This thesis has fulfilled all the requirements as per the regulations of the Institute and is worthy of consideration for the award of the degree of *Doctor of Philosophy* of the Institute.

(Prof. Diganta Goswami)
Supervisor
Professor, Dept. of CSE
IIT Guwahati

Date:

(Prof. Sukumar Nandi)
Supervisor
Professor, Dept. of CSE
IIT Guwahati

Date:

This thesis is dedicated to my family members.

Acknowledgements

This research and thesis could not have been successfully carried out without the support of my supervisors, teachers, friends and family. First and foremost, I owe my profound gratitude to my thesis supervisors, Prof. Sukumar Nandi and Prof. Diganta Goswami. I express my deepest appreciation to them for their careful guidance, constant encouragement, and continuous support and advice. Without their persistent help, this thesis would not have been complete.

I also express my sincere appreciation to Dr. Jatindra Kumar Deka, Dr. Santosh Biswas and Dr. Partha Sarathi Mondal for their thorough yet helpful analysis of my research work from time to time.

I extend my earnest thanks to the Head of the Department of Computer Science and Engineering for providing me with the academic support and the necessary facilities to complete my research work.

I also thank the other faculties, as well as postgraduate and undergraduate students of the Department of Computer Science and Engineering for creating a friendly environment of research.

I like to thank my fellow research scholars, who have later become my fellow labmates, for the support they have extended in many special ways. I gained a lot from them, through personal, scholarly and athletic interactions, during the entire programme.

In particular, the help I have received from Dr. Sandip Chakraborty and Dr. Suchetana Chakraborty are worth mentioning. They have shared their precious times for the stimulating discussions that help me in improving the quality of this thesis. I thank them for their valuable suggestions and insightful comments at different stages of my research.

A special thanks also goes to Dr. Arup Bhattacharjee, Biswanath Dey, Dr. Sraban Mohanty, Tezmani Sinam, Mamata Samal, Dr. Bidyut Patra, Rajendra Pamula, Dr. Ferdous A. Barbhuiya, Dr. Maushumi Barooah, Rohit Tripathi, Dr. Pravati Swain, Niladri Sett, Amrita Bose Paul, Ashok Kumar A. R., Nilkantha Sahu, Madhusudan Paul, Shrishendu Das, Mousum Handique, Shounak Chakraborty, Shrenik A. Surana, Sourabh Prajapati, Arihant Sethia and Arun Bhati.

I extend my sincere thanks to the Government of India and IIT Guwahati for granting me with the assistantship to undergo this research and thesis at IIT Guwahati.

Last but not the least, I also extend my deepest gratitude to my family members for their constant encouragement, support and understanding during the tough times of my research.

Date: (SUDDHASIL DE)

Abstract

Tuple Space based Mobile Middleware (TSMM) platforms emerge to handle coordination of interacting components (called *agents*) of mobile applications in runtime environments of mobile devices (called *hosts*) amidst mobility, dynamics and unreliability in underlying infrastructures. Importance of TSMM platforms are escalating day-by-day with increasing availability of mobile devices and extensive deployment of different forms of wireless networks. Wide variety of mobile applications are being developed for these networks in the direction of providing 'ubiquity' to end-users, and such applications require suitable TSMM platforms for coordination. In spite of the existence of several TSMM platforms to cater such applications, the objective of today's mobile middleware research is to design a single TSMM platform to simultaneously fulfil the desired coordination functionalities, viz. robustness, flexibility, versatility, simplicity, reliability, efficiency and scalability. This thesis has contributed on different aspects of a TSMM platform that directly or indirectly improve its coordination functionalities.

In existing TSMM platforms, a *reference agent* invokes any remote blocking tuple space primitive, and yet is not blocked indefinitely by virtue of *synchronization decoupling*, which requires support of reactivity. That is, reference agent, invoking a blocking tuple-reading/consuming primitive to look up for communicating information from remote tuple space of *target agent*, is relieved from being suspended indefinitely, in case of nonavailability of such information, by registering a reaction at that tuple space. However, when that reaction fires at the tuple space of target agent and starts executing its callback method, availability of reference agent is a necessity to receive the result of execution (i.e. reaction response). In other words, reactivity itself is time-coupled with reference agent. But, underlying dynamics in TSMM platform burdens the availability constraint of reference agent to receive reaction response from target agent, incurring additional network and processing overheads in existing TSMM platforms. Moreover,

coordination in existing TSMM platforms also faces consistency problems due to strict atomicity in remote tuple space operations. However, in presence of underlying dynamics in existing TSMM platforms, the existing approaches of resolving consistency problems incur additional network overheads, and again impose time-coupling in synchronization-decoupled tuple space operations. Also, mobile applications tend to lose robustness due to the coupled behavior, as reaction responses and responses of tuple space operations are not guaranteed to be delivered to reference agent. In order to restore robustness in this scenario, a TSMM platform can augment synchronization decoupling with decoupling of responses of fired reactions from reference agent, and can adopt an appropriate consistency handling mechanism. This thesis has a foremost contribution in these research directions.

Furthermore, in existing TSMM platforms, the tuple/antituple structures and tuple space structures degrade the overall performances of tuple space operations, as well as affect simplicity and versatility in supporting a wide spectrum of applications. The existing tuple/antituple structures lead to trade-off between efficiency/scalability and simplicity/versatility in TSMM platforms. Whereas, the existing tuple space structures directly affect the performances of tuple space operations, further deteriorating efficiency and scalability of TSMM platforms. This thesis has another contribution in enhancing tuple/antituple and tuple space structures of a TSMM platform, so as to improve its efficiency and scalability, without compromising its simplicity and versatility. This thesis has also contributed in resolving heterogeneity of multiple underlying networks, as well as bridging wired networks with wireless networks, which makes a TSMM platform flexible. The existing TSMM platforms have rarely supported heterogeneity in their underlying networks; they are primarily designed targeting a specific network. This thesis has lastly contributed in formal modeling and verification of an entire TSMM platform, using notational constructs and proof logic of the Mobile UNITY model. The modeling approach has the benefits of thoroughly describing the behaviors of its components, and defining precise semantics of agent interactions. The verification strategy proves the correctness of TSMM platform. Finally, this thesis reports about the implementation of a working prototype of TSMM platform, and demonstration of its correct working.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

With the mobile computing paradigm gradually becoming a part of our daily life, the need to enable "ubiquity" to different activities of end-users is gaining primary concern in the present-day mobile computing research. The advancements in wireless communication technologies have resulted in extensive deployment of different forms of wireless networks. The advents of well-equipped mobile computing devices (like smartphones, personal digital assistants, ultra-mobile computers, tablet computers etc.) and portable computing devices with wireless connectivity (like notebooks, laptops etc.) for operating in these networks have stimulated the process of reification of ubiquity to their end-users, and various types of decentralized *mobile applications* are being developed as part of it. These applications, executing in different mobile/portable devices, that may be geographically apart, *interact* with each another or among themselves for *coordination* [74, p. 94] of different cooperative works. A typical mobile application comprises of several active, interacting and distributed components (loosely referred as *agents* [13]), which are the units of execution, mobility and data storage. One striking feature of these agents is that some of them can migrate from one *host* (which is a runtime environment within a mobile/portable device) to another host promoting *logical* mobility (such agents are called *mobile agents* [69]), whereas others remain stationary in their 'home' environment (i.e. *stationary agents*). The hosts provide facilities for execution of these agents and their communication activities, i.e. they simply are the containers of these agents. Also,

the hosts can be mobile, by virtue of device mobility (such hosts are called *mobile hosts*), which is due to *physical* mobility of their end-users.

For supporting different activities of agents of mobile applications in mobile hosts, research has been directed towards the use of middleware platforms in the mobile computing paradigm [42]. A *middleware* [3, 11] is a software interface between 'application' and 'system platform' (i.e. operating system and associated services) of a computing device. It is a layer above operating system services of that device, but below application program interfaces (APIs) that provides a set of common programming abstractions across a distributed system. So, a middleware platform can be viewed as a set of services that are accessible to application programmers through their APIs. The objectives of a middleware platform are (i) to support coordination of interacting agents of decentralized applications, (ii) to resolve heterogeneity in underlying operating systems, architectures and networks transparently, and (iii) to simplify the tasks of application programmers in developing these applications. Typically, a middleware platform incorporates a *coordination model* [91] that manages coordination of supported interacting agents. However, the middleware platform in the mobile computing paradigm not only manages coordination of its agents and heterogeneity in its underlying infrastructure, but also handles the factors of dynamics, unreliability and device mobility in underlying network connectivities. This new genre of middleware platform, commonly termed as *mobile middleware* [9, 14, 78, 79], is the extension of traditional middleware platform for the mobile computing paradigm. The objectives of a traditional middleware platform are equally prevalent in mobile middleware platform, viz. to provide simplified abstractions for building efficient mobile applications and to resolve heterogeneity and distribution for these application, apart from providing coordination functionality for supported agents of these applications etc.

However, the major hindrance in progress of research activities in the area of mobile middleware comes from inherent limitations of wireless communication technologies and pitfalls associated with device mobility. The communication via wireless medium is intrinsically unreliable, intermittent and dynamic, due to the very nature of wireless medium. Again, the mobility of devices poses complications during communication, as

connectivity condition varies with device mobility. Since a mobile middleware platform has to handle such obstacles of underlying infrastructures, tight coupling in interactions of its supported agents has an unsatisfactory effect during coordination of mobile applications. Actually, the term *coupling* [105] refers to a software metric that measures the level of attachment or dependency (called *degree* of coupling) of interacting agents in several aspects (called *dimensions* of coupling [5]), while they participate actively to accomplish their coordination objectives. For instance, tight coupling refers to high dependency or close attachment between interacting agents. Contrary to the popular belief, loose coupling in one/more dimensions in a decentralized system not necessarily precludes less coordination. In fact, coordination achievable through practically no coupling (i.e. "cessation of coupling" or "amelioration of coupling" [113], which is referred as *decoupling*) among interacting agents is ideal in case of any dynamic system, including mobile middleware platform. The term decoupling also refers to a software metric that measures the level of detachment or independency (called *degree* of decoupling) in several *dimensions* of decoupling[1] of interacting agents during their active participation in coordination. The basic need of decoupling in mobile middleware platforms is to separate the concerns of interacting agents in several dimensions during their coordination to overcome the difficulties of underlying dynamic infrastructures. As a result, the mobile middleware research focuses on decoupling the interactions among agents of supported mobile applications using suitable coordination models [49].

Several coordination models exist in literature that can tolerate and operate effectively in the dynamic ambience of mobile middleware platforms [84]. In the taxonomy of coordination models [18, 45], outlined in Figure 1.1, two broad categories of coordination models are indicated; this taxonomy depends on how information of one agent is received by another, viz. direct and indirect. The *direct coordination* model requires temporal and name-spatial coupling of interacting agents, i.e. their 'synchronous' and 'onymous' participation in interactions. The popular client/server paradigm uses this category of coordination, like Sumatra [1], Odyssey, Agent-Tcl [68] etc. The *indirect coordination* model, on the other hand, relaxes the primary coupling restrictions of direct coordination

[1]Chapter 2 reports on the dimensions of decoupling prevalent in coordination models of dynamic systems.

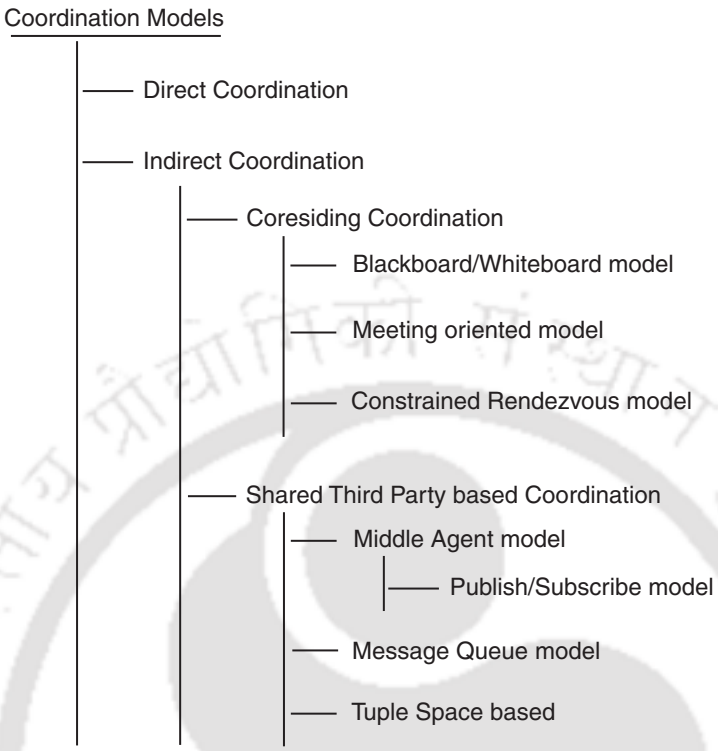


Figure 1.1: Taxonomy of coordination models.

model — either interacting agents need not to be present simultaneously during interaction (i.e. temporal decoupling or 'asynchronous' participation in interactions), or they need not to know the identity of other interacting agents (i.e. name-spatial decoupling or 'anonymous' participation in interactions), or both. The indirect coordination model is further classified into coresiding coordination model and shared third party coordination model.

In the *coresiding coordination* model, agents interact when they coreside on same host, and they either use that host's resources or involve other additional agents within that host to carry out their interactions. It is again classified depending on the nature of resources involved. The *blackboard/whiteboard* model provides a local shared memory (i.e. blackboard or whiteboard) of host for coordination, and interacting agents access the blackboard/whiteboard to write/read their communicating information; e.g. Ambit [20], ffMAIN [41] etc. Despite the fact that this model achieves temporal and name-spatial decoupling of interacting agents during interactions (i.e. asynchronous and

anonymous participation in interactions), they have to a priori decide on structure of communicating information (like, common message identifier), which evidently couples them name-spatially prior to their actual interactions (i.e. 'a priori onymous' communication). The *meeting oriented* model, like Mole [8], Telescript [112] etc., requires support of a host to implement certain meeting points and interacting agents have to register there for coordination. The information, sent by one registered agent of a meeting point, is forwarded to all other registered agents of that meeting point. Even though this model name-spatially decouples the interacting agents during interactions, a priori knowledge of participations in meetings again bring forth name-spatial coupling. Also, interacting agents in meeting points are always temporally coupled (i.e. synchronous participation in interactions). The *constrained rendezvous* model, for instance Ara [92], introduces the involvement of other additional local agents (called coordination servers) within a host for coordination, and these local agents mediate interactions between actual interacting agents. This coordination model operates similar to the blackboard/whiteboard model, but it provides temporal and name-spatial decoupling among interacting agents during as well as prior to interactions.

The *shared third party coordination* model provides different shared third party resources for coordination of interacting agents. This category of coordination model is also capable of achieving temporal and name-spatial decoupling among interacting agents. It is further classified depending on the nature of third party resources used. The *middle agent* model [66] provides a globally accessible broker agent for coordination. The interacting agents communicate by handing over their communicating information to broker agent, as well as accessing it to retrieve them. A consumer expresses its interest in specific communicating information to broker agent, and is notified when such communicating information become available. A producer generates communicating information and hands it over to broker agent for access of consumers. This coordination model is also capable of achieving synchronization decoupling at both producer-side and consumer-side. The publish/subscribe model [43, 58] is related to this category of coordination model. The *message queue* model [12, 40] provides a globally shared message queue for coordination. The communicating information are sent to the shared message

queue, from where consumers can retrieve them (using 'pull' mode of read/withdraw). This coordination model can be implemented by a central message queue manager, or a network of message brokers. Apart from temporally and name-spatially decoupled, this model achieves producer-side synchronization decoupling. It is also persistent, as a message is conserved until explicitly removed from the message queue. The *tuple space* model [21, 50] provides shared tuple space(s) for coordination. This model is not only temporally and name-spatially decoupled, but also decouples producer from its generated information. It additionally achieves producer/consumer-side synchronization decoupling. Moreover, the associative search technique among communicating information in tuple spaces adds another dimension of decoupling and makes this model efficient. Furthermore, the inherent storage capability of this model helps in achieving persistence easily.

The Table 1.1 lists the aforementioned coordination models and their supported dimensions of decoupling. Among these coordination models, the dynamic environment of a mobile middleware platform better suits an asynchronous (i.e. temporally decoupled) model. Along with that, the additional support for other dimensions of decoupling during coordination turns out to be more beneficial for mobile middleware platforms. The coordination models in Table 1.1, which provide temporal decoupling, are the blackboard/whiteboard model, the constrained rendezvous model, the middle agent model, the message queue model, and the tuple space model. Among these five coordination models, the tuple space model has a competitive edge over every other model. It supports an inherent dimension of decoupling (viz. producer decoupling, whereby lifetime of communicating information are decoupled from their producer) that is not provided by other coordination models. Besides, each of these models differ from the tuple space model in several ways. For instance, the coresiding coordination models introduce additional overheads of bringing interacting agents to same host to complete their interactions. Besides, the blackboard/whiteboard model cannot achieve name-spatial decoupling as well as consumer-side synchronization decoupling (as consumers adopt a continuous polling mechanism to know the availability of communicating information). The constrained rendezvous model also cannot achieve synchronization decoupling and

Table 1.1: Coordination models and their decoupling abilities.

Coordination model	Dimension(s) of decoupling supported
Direct coordination	Producer-side synchronization
Blackboard/whiteboard coordination	Temporal, Producer-side synchronization
Meeting oriented coordination	Producer-side synchronization
Constrained rendezvous coordination	Temporal, Name-spatial Producer-side synchronization
Middle agent coordination	Temporal, Name-spatial, Producer-side & consumer-side synchronization
Message queue coordination	Temporal, Name-spatial, Producer-side synchronization
Tuple space coordination	Temporal, Name-spatial, Data-spatial, Producer, Producer-side & consumer-side synchronization

data-spatial decoupling, as well as cannot provide inherent persistency to communicating information.

Even though the shared third party models are capable of coordination among interacting agents without the requirement of their coresiding on same host, most of these models (particularly, the middle agent model and the message queue model) are not capable of achieving producer decoupling as well as data-spatial decoupling. Besides, the middle agent model lacks inherent persistency. The message queue model also cannot achieve consumer-side synchronization decoupling. Moreover, the guarantee of ordered delivery of messages in this model is a much restrictive requirement for the dynamic context, and may limit its scalability in practice. The tuple space model is advantageous over other coordination models, because — (i) it simultaneously supports temporal, name-spatial, data-spatial, producer, and producer-side & consumer-side synchronization decoupling; (ii) it provides inherent persistency; (iii) it does not impose the restriction of coresiding for interacting agents; and (iv) it does not require the guarantee of ordered delivery of communicating information. These advantages make it more suitable as the coordination medium for mobile middleware platforms.

The mobile middleware platform, with the tuple space model as its coordination medium, is termed as the *T*uple *S*pace based *M*obile *M*iddleware (**TSMM**) platform.

The tuple space model of TSMM platform, in reality, came into existence with the proposal of Linda [50]. In fact, literature has already manifested the tuple space model as a popular coordination model for parallel programming, distributed computing and mobile computing paradigms [91, 16], due to its ability in providing multi-dimensional decoupling in coordination of interacting agents. In the tuple space model, a *tuple space* is considered as a repository of tuples, equally accessible to all interacting agents. Any one of them can dynamically deposit new tuples in a tuple space, or read/withdraw existing tuples from it. No update-in-place, however, is defined for tuples, as they are considered as *immutable* units of coordination. A *tuple* is the basic unit of communicating information exchanged during an interaction of involved agents. Each tuple is a sequence of several heterogeneously-typed fields having values (called *actual*), e.g. {1, "Dave", 10.4, "Hello"}. On the other hand, an *antituple* is considered as the basic unit of search key to identify some specific tuples (called *sought tuples* [52]) residing in tuple space. Each antituple is also a sequence of heterogeneously-typed fields, with some fields being *actual*, while other fields have placeholders for *actual* (called *formal*), e.g. {int, "Dave", 10.4, —}[2], {1, string, —, string} etc. The antituple uses associative matching operation to find sought tuple(s) from a tuple space. Typically, the tuple space model includes several primitives to carry out insertion, reading and withdrawal of tuples from tuple spaces. These primitives are: (i) tuple-producing primitives (like `out` and `outg`), (ii) tuple-reading primitives (like `rd`, `rdp`, `rdg` and `rdgp`), and (iii) tuple-consuming primitives (like `in`, `inp`, `ing` and `ingp`). It is to be noted here that the letters '`p`' and '`g`' in the name of these primitives stand for 'probe' and 'group' respectively. In other words, `out`, `rd`, `rdp`, `in` and `inp` are 'single' primitives, whereas `outg`, `rdg`, `rdgp`, `ing` and `ingp` are 'bulk' primitives.

The tuple-producing primitives deposit given tuples in tuple spaces; e.g. to place the tuple {1, "Dave", 10.4, "Hello"} in a tuple space, `out`({1, "Dave", 10.4, "Hello"}) is invoked. The tuple-consuming primitives withdraw one/more sought tuple(s) matching given antituple; e.g. `in`({1, string, —, string}) successfully looks up the above tuple from tuple space, while `inp`({1, int, —, string}) does not match with it. A tuple-antituple matching

[2] In this thesis, '—' in an antituple represents the absence of search value from an antituple field.

algorithm searches the entire tuple space to determine positive match of any tuple with given antituple. In Linda, an antituple positively matches with a tuple, if all the following conditions hold together: (i) both of them must have identical number of constituent fields; (ii) each of their constituent fields must be pairwise of same type; (iii) for *actual* in both cases, the values must be same; and (iv) an antituple *formal* can match any *actual* of same type in tuple. If multiple tuples match positively, any one of them is withdrawn nondeterministically from tuple space as sought tuple and is returned back as the 'result' of invoked operation. The tuple-reading primitives read one/more sought tuples that match given antituple. They operate in the same way as tuple-consuming primitives, except that tuples are not withdrawn from tuple space. If the tuple-antituple matching operation cannot provide a positively-matched tuple, invoking agent gets blocked till any matching tuple becomes available. The primitives `rd`, `rdg`, `in` and `ing` exhibit this blocking behavior, whereas the primitives `rdp`, `rdgp`, `inp` and `ingp` are nonblocking in nature. For the purpose of coordination, system/application programmers distribute multiple tuple spaces across several locations, depending on the nature of computing paradigm. The communicating information are converted to tuples and are inserted in tuple space(s) by one interacting agent (i.e. it invokes `out`/`outg` to insert tuples). The other interacting agent reads or withdraws such data by issuing search keys that are converted to antituples to invoke tuple-reading or -consuming primitives respectively to lookup for sought tuples in tuple space(s).

However, the challenge faced by today's mobile computing research is to design of a single TSMM platform that caters the requirements of end-users over different wireless networks. The challenge becomes the motivating factor for this thesis.

1.2 Motivation

The deployments of different scales of heterogeneous wireless networks, viz. wireless local area networks, mobile ad hoc networks, wireless mesh networks etc., fulfil various purposes, like home/office uses, military applications, handling of disaster situations etc. These wireless networks are characterized by *mobility* of their devices, their sporadic

dynamics, and inherent *unreliability* in their communication links. There is a growing demand for different types of mobile applications to work in these networks to fulfill varied needs of end-users. Despite the presence of several TSMM platforms to support such mobile applications, the challenge still faced by today's mobile computing research in designing a single TSMM platform to cater the requirements of end-users over these multiple networks is due to the dissimilar nature of these networks. Apart from being mobile, dynamic and unreliable, each wireless network also has distinct intrinsic properties that widely vary from other wireless networks. This is also true with their deployed devices, viz. stationary/mobile, resource-limited/resource-rich, having wired/wireless/both connectivities etc. So, a TSMM platform has to be *flexible* so that it can be made compatible with wide variety of mobile/portable devices as well as multiple heterogeneous underlying networks. Also, it has to be *robust* to handle their unreliability issues (like unpredictable data loss in wireless links) and manage dynamic circumstances arising due to mobility of deployed devices. A TSMM platform can become robust, if it can enhance the decoupling abilities of its coordination support towards mobile applications, whereas it can achieve flexibility by resolving its underlying heterogeneity among devices and wireless networks as well as bridging wired-wireless networks. Furthermore, a TSMM platform has to be *versatile* in supporting a wide spectrum of mobile applications, yet it has to be *simple* while providing APIs to application programmers. Lastly, a TSMM platform has to be *efficient* and *scalable* in its performance. The existing TSMM platforms have the shortcomings of not being able to achieve all these characteristics. Some of them focus on robustness and reliability during exchange of communicating information, while others emphasize on versatility/simplicity and efficiency/scalability of coordination. Rarely any existing TSMM platform considers flexibility in coordination. This thesis contributes on different aspects of a TSMM platform that directly or indirectly make it flexible, robust, versatile and simple, as well as reliable, efficient and scalable for coordination of interacting agents.

1.3 Contributions of Thesis

The contributions of this thesis are as follows:

1. The first contribution of this thesis proposes a new approach of time-decoupling of responses corresponding to invokes during agent interactions in a TSMM platform. That is, for each invoke of any tuple space operation by invoker *reference agent*[3], a reaction is registered at the designated tuple space of target agent, in case of nonavailability of sought tuple(s). The registered reaction decouples reference agent from its invoked remote primitive, thereby imposing synchronization decoupling (which turns out to be 'partial'). The proposed notion of decoupling augments it with time-decoupled transfer of response of that reaction back to reference agent. The proposed notion, thus, leads to "complete synchronization decoupling" of agent interactions in TSMM platform. The transfer of reaction response is time-decoupled by inserting an additional tier of tuple spaces at host-level as extra decoupling medium, apart from the existing tuple spaces present within TSMM platform, resulting in a tiered organization of tuple spaces. A new TSMM platform, named $\mathcal{CUTSMuHN}$ (acronym of *C*omplete *U*ncoupling of *T*uple *S*pace in Mobile Middleware over *Mu*ltiple *H*eterogenous *N*etworks), is designed by following this tiered organization of tuple spaces to support the proposed notion of decoupling, in which tiers of tuple spaces are distributed across multiple hosts. Moreover, a new three-way approach of agent interactions is proposed for $\mathcal{CUTSMuHN}$ platform to resolve the consistency problems of tuple space coordination in TSMM platforms. Such consistency problems arise for withdrawal operations during agent interactions, mainly because of the unreliable underlying networks of TSMM platforms. That is, when sought tuples are withdrawn from remote tuple spaces, they are transported to reference agent through unreliable networks. The loss of any sought tuple during this transfer may result in inconsistent coordination. In $\mathcal{CUTSMuHN}$ platform, the proposed approach maintains consistency by including positive/negative

[3]The agent that invokes any tuple space primitive is designated as *reference agent* for that particular invoke, and the destination of that invoke becomes *target agent*. Moreover, the host where reference agent is presently residing is the *reference host*, whereas the *target host* is presently holding target agent.

acknowledgement mechanisms in each remote invoke of tuple-consuming operation. However, the proposed approach operate at agent-level tuple spaces, radically different from the conventional acknowledgement mechanisms operating at network-level. In the proposed approach, reference agent positively acknowledges a target agent, whose received response is accepted as sought tuple(s), and negatively acknowledges otherwise. The negative acknowledgements happen only for remote single tuple-consuming primitives, in which case received responses are sent back to their respective target agents, thus, maintaining consistency. Consequently, the first contribution of this thesis, combining these two proposals, makes a TSMM platform robust.

2. The second contribution of this thesis comprises of the proposals for enhancing tuple/antituple structure and tuple space structure of $\mathcal{CUTSMuHN}$ platform, so that its overall performances improve, thereby making it efficient and scalable. The proposed enhancement of existing unordered tuple/antituple structure upgrades their constituent field substructures as well as quickens tuple-antituple matching operation, so that temporal and spatial complexities of enhanced unordered structure are close to ordered tuple/antituple structure. So, the proposed enhancement reduces the execution overheads of different tuple space primitives of $\mathcal{CUTSMuHN}$ platform, without compromising simplicity and versatility of unordered structure in supporting a wide spectrum of applications. Moreover, the proposed tuple space structure follows a new approach of placement of unordered tuples, so as to expedite tuple space operations in $\mathcal{CUTSMuHN}$ platform. Thus, it further improves the performances of $\mathcal{CUTSMuHN}$ platform, without sacrificing its achieved simplicity and versatility. These proposals also lead to redefinition of entire set of tuple space operations of $\mathcal{CUTSMuHN}$ platform.

3. The third contribution of this thesis proposes the approaches of $\mathcal{CUTSMuHN}$ platform to resolve heterogeneity of multiple underlying networks as well as to bridge wired networks with wireless networks, which makes $\mathcal{CUTSMuHN}$ flexible. For each

supported underlying network, viz. Infrastructure Basic Service Set (iBSS), Independent Basic Service Set (IBSS), Heterogeneous Mobile Ad hoc Network (H-MANET) etc., TSMM-specific discovery and communication mechanisms are proposed for *CUTSMuHN* platform. The proposed discovery mechanisms furnish an updated knowledge of available hosts and their agents that are reachable from source host. On the other hand, the proposed communication mechanisms emphasize on reliable transfer of communicating information of interacting agents from one location to another. These mechanisms, in turn, utilize the services of Internet Protocol (IP) to interface with network interface controllers (NIC) corresponding to the underlying networks.

4. The fourth contribution of this thesis proposes a new approach of formal modeling of entire *CUTSMuHN* platform that can express its coordination ability by simple constructs. In the proposed modeling approach, the behaviors of *CUTSMuHN* platform are captured by the notational constructs of the Mobile UNITY model, a general-purpose formal modeling and verification tool to capture dynamic setting. This modeling not only thoroughly describes the behaviors of different components of *CUTSMuHN* platform over multiple heterogeneous networks, which are ultimately utilized in expressing its coordination functionality and analyzing its correctness, but also defines precise semantics of agent interactions, thereby facilitating the tasks of application programmers. Moreover, it also lays the foundation for implementation of *CUTSMuHN* platform. Additionally, a new strategy of formal verification of correctness of *CUTSMuHN* platform is proposed, that follows the proof logic of the Mobile UNITY model. In the proposed strategy, a set of formal properties representing the behaviors of *CUTSMuHN* platform are defined based on its proposed modeling, and these properties are next proven to verify correctness of its behaviors under common conditions. The proposed verification strategy also helps in reasoning of mobile applications developed for it. Finally, an entire prototype of *CUTSMuHN* platform is implemented in Java™, along with its supported mobile applications, and its working is tested in Java™ Virtual Machine (JVM) version 1.6.0_17.

The above contributions of this thesis demonstrate that *CUTSMuHN* platform is flexible, robust, reliable as well as efficient in coordination over heterogeneous networks that are mobile, dynamic and unreliable.

1.4 Outline of Thesis

This thesis is organized as follows.

CHAPTER 2 reviews the existing TSMM platforms and their approaches of coordination among interacting agents of mobile applications. A comparison of their coordination functionalities are also tabulated in this chapter.

CHAPTER 3 proposes an additional dimension of decoupling for tuple space coordination that is significant for the dynamic environment of TSMM platforms. This chapter also introduces and describes the proposed TSMM platform, named *CUTSMuHN*. The different design aspects of *CUTSMuHN* platform for improving its coordination functionality are also proposed. The complexity analysis of different tuple space primitives of *CUTSMuHN* platform are later presented in this chapter.

CHAPTER 4 proposes the mechanisms of handling heterogeneity in underlying networks of *CUTSMuHN* platform. This chapter concentrates on three different types of networks for *CUTSMuHN* platform (viz. iBSS, IBSS and H-MANET) to demonstrate the proposed approach of handling underlying heterogeneity in a TSMM platform.

CHAPTER 5 proposes an approach of formal modeling of entire *CUTSMuHN* platform using the notations of the Mobile UNITY model. The proposed modeling approach captures the behaviors of *CUTSMuHN* platform as a Mobile UNITY **System**, comprising of a set of **Program**s representing different instances of agents and hosts, which are treated later for verification.

CHAPTER 6 proposes a strategy of verifying correctness in the behaviors of *CUTSMuHN* platform using the proof logic of the Mobile UNITY model. The formal modeling of *CUTSMuHN* platform, specified in the preceding chapter, is used during the verification process.

CHAPTER 7 describes the implementation strategy of *CUTSMuHN* platform for developing its working prototype. The mobile applications of *CUTSMuHN* platform are also developed to demonstrate its correct working. Moreover, the experiments showing performance improvements of tuple space primitives of *CUTSMuHN* platform are also presented in this chapter.

CHAPTER 8 summarizes the contributions of this thesis, and explores the directions of future works in tuple space coordination for mobile middleware platforms.

Chapter 2

Background and Related Work

2.1 Introduction

The middleware technology has been the focus of research since inception due to their multiple advantages. However, with the rise in complexity of application programs as well as increase in variety of underlying infrastructures (at both device-level and network-level), the need for more simple but efficient and versatile coordination models for emergent middleware platforms has been felt. This is the reason why research in mobile middleware platforms has predominantly pioneered the investigation of the tuple space model as their coordination media.

The importance of TSMM platform is escalating with the increasing popularity of host mobility. Variety of decentralized applications are being developed for end-users that fulfil this host behaviour, and they require suitable TSMM platform for their execution. As such, a thorough review of the research avenues of TSMM platform is needed for its wide deployment. Literature shows that, since inception, the tuple space model has served as coordination model for the parallel programming, distributed computing and mobile computing paradigms, to name a few. In [91], a survey of different tuple space platforms for coordination in parallel programming and distributed computing paradigms is presented. However, these platforms are not capable of coordinating applications over mobile, dynamic and unreliable networks, and as such this survey has not fully revealed the research avenues of TSMM platforms. In [18], several coordination paradigms, including the tuple space model, have been presented with illustrations.

The exemplifications of tuple space platforms in this work can be considered as preliminary reviews of some TSMM platforms. Similarly, [78], [48], [79] and [14] have surveyed different coordination models, including the tuple space model, for mobile computing paradigm, and have reviewed TSMM platforms inceptively. The survey given by [100] is a dedicated review of the tuple space model in the context of coordination. However, this survey compares the existing tuple space platforms, starting with that of the parallel programming paradigm, and aims to provide research guidelines to Internet-based multi-agent tuple space platforms.

A better survey of the tuple space paradigm for mobile computing has been presented in [16]. The tuple space platforms, cited in this survey, are considered as TSMM, and this work is most closely related to the survey of this thesis. Even though this survey is relatively contemporary compared to other existing surveys, still it has mainly focused on providing a framework for classifying existing TSMM platforms, and compares their characteristics from the classification scheme. None of these surveys provide proper research guidelines about TSMM platforms. Consequently, a structured and comprehensive review about the research avenues of TSMM platforms is much needed for this thesis. This chapter reviews the research directions of different existing TSMM platforms and paves the background for this thesis.

2.2 Basic Concepts of Tuple Space based Mobile Middleware Platforms

The term "middleware" is first coined by Alex d'Agapeyeff in 1968 [88], with the aim of simplifying design of application programs, coordinating their interactions, as well as resolving heterogeneity and distribution. The objectives of today's middleware platforms (like, mobile middleware) still remain the same. However, a few of the plethora of coordination models can handle application/system/hardware-level complexities of mobile middleware platforms. The tuple space model is one such coordination model, which has been well-accepted due to its multiple advantages, and literature reports the existence of several mobile middleware platforms incorporating the tuple space model.

The use of the tuple space model as coordination media dated back to the parallel programming paradigm, where one/more tuple spaces are placed at specific centralized locations to carry out the task of coordination among interacting parallel processes [2, 51, 59]. The requirement of parallel programming, viz. efficient execution of communicating parallel processes of a program, is fulfilled by the tuple space model. When the tuple space model is ported to middleware platforms of the distributed computing paradigm [81, 25] to act as their coordination media, multiple independent tuple spaces are distributed across distinct locations. In each location, one/more tuple space(s) are shared to decouple interacting agents during coordination. The decentralized implementations of the tuple space model favor better coordination ability of respective middleware platforms. However, both parallel programming and distributed computing paradigms provide stable environments during coordination.

The decoupling ability of the tuple space model also attracts researchers of the mobile computing paradigm. The low connectivity bandwidth, frequent, unavoidable and unpredictable disconnection during communication, as well as high latency of data transmission in wireless link suggest a decoupled nature of system architecture for supporting coordination of mobile applications. So, decentralized implementations of the tuple space model is suitable for mobile computing paradigm, leading to the genre of TSMM platform. The TSMM platforms are popular due to their abilities in providing multiple dimensions of decoupling during coordination of interacting agents.

2.2.1 Dimensions of Decoupling in Coordination

Since the conception of research on decoupling in coordination, three fundamental dimensions of decoupling have been observed in shared variable models, message passing models, remote operation models, brokerage models and shared space models, viz. 'temporal decoupling', 'name-spatial decoupling' and 'data-spatial decoupling'. With the introduction of the tuple space model, 'producer decoupling' is added as another dimension of decoupling in coordination. The decentralized implementation of the tuple space model results in 'synchronization decoupling' in coordination. Recently, one more dimension of decoupling is included in coordination, particularly for TSMM platforms,

viz. 'engagement decoupling'. These dimensions of decoupling are briefly described as follows.

(1) Temporal decoupling: It refers to the decoupling of participation of interacting agents in time, i.e. interacting agents are not required to be available simultaneously for active participation in interactions. The availability of coordination medium, however, is required during interaction phase. Hence, this form of decoupling results in *asynchronous communication* of interacting agents.

(2) Name-Spatial decoupling: It refers to the decoupling of participation of interacting agents in knowledge (i.e. name/identity) of their source/destination agents, i.e. reference agent is not required to know the identity of target agent(s) to actively participate in interaction and vice versa. Only prior knowledge of coordination medium is required to successfully wind up interaction phase. Consequently, this form of decoupling gives rise to *anonymous communication* of interacting agents.

(3) Data-Spatial decoupling: It refers to the decoupling of storage/access of communicating information of agents in data-space (i.e. coordination medium). This dimension of decoupling is possible only in case of shared space models, where shared coordination medium is not accessed by location of its stored communicating information; rather, it is accessed based on its content. The content-based addressing scheme of coordination medium results in decoupling of accesses of communicating information from their stored locations in coordination medium, which gives rise to *associative communication* of interacting agents.

(4) Producer decoupling: It refers to the decoupling of communicating information from their producer agents in time, i.e. communicating information have independent existences in coordination medium and their lifespans are independent of the lifetime of their producers. Moreover, till the communicating information reach their destinations, their existences are also independent of their consumer agents. So, this notion gives rise to *generative communication* of interacting agents.

(5) Synchronization decoupling: It refers to decoupling of production/consumption of communicating information from agents in synchronization, i.e. producer agents

are not suspended while generating their communicating information and consumer agents have not yet attained the proper state to consume the generated information. In fact, consumer agents can get notified first about the presence of communicating information, while performing other concurrent activities, and consumed them afterwards. As a result, this form of decoupling brings forth *nonblocking communication* of interacting agents.

(6) Engagement decoupling: It refers to the decoupling of interacting agents in participation of interactions. However, this dimension of decoupling is external, but related to, a coordination model. According to this dimension, reference agent can selectively decide its relevant set of target agents for interaction, whereas target agent can choose to accept/deny any interaction initiated by reference agent. The set of selective relevant agents for interaction is called acquaintance set, and decision of their selection can be enforced by customized acquaintance policies. In other words, target agent can decide its engagement in participation of each interaction of reference agent. This is contrary to the situation where every interacting agent has to respond to interactions from every other interacting agent. So, this form of decoupling gives rise to *asymmetric communication* of interacting agents.

An important feature of these dimensions of decoupling are that they are all *orthogonal* to each other. In other words, these six dimensions of decoupling can be supported by any coordination model, without sacrificing their innate nature. However, the tuple space model is among the few coordination models to support them simultaneously. The next section focuses on one such dimension of decoupling, where the existing TSMM platforms have some limitations.

2.2.2 Synchronization Decoupling and Reactivity

The tuple space model attains synchronization decoupling by introducing *reactivity* [39, 38]. Reactivity is an ability of the tuple space model to monitor and respond to different circumstances (called *events*) during its course of execution. It provides powerful

programming constructs to facilitate application programmers in developing robust applications for TSMM. It follows *Event-Condition-Action* or *ECA* rule [34]. The reactive tuple space model provides reactions to decouple reference agents from their invoked remote primitives. For monitoring and responding to an event (like, presence of a particular sought tuple in tuple space), a reaction is generated and registered there. For recognizing a related event, reaction expects some condition to be specified as antituple. If condition gets satisfied, desired event is said to happen and corresponding (active) registered reaction fires. The firing of a reaction signifies that some actions (called *reactive codes*) are going to be executed subsequently, like notifying the presence of sought tuple(s) etc., and responses of reactions are sent back to reference agent.

The TSMM platforms support reactivity in either rudimentary or refined form of reactions. A reaction is a high-level abstraction for specifying appropriate actions that need to be performed atomically in response to a change in state of tuple space. Typically, a reaction comprises of antituple, reactive code, tuple space identity(ies), reaction lifetime, user identity etc. The antituple specifies conditions to be satisfied, while reactive code defines corresponding actions on tuple space. The tuple space identity(ies) indicates a list of target agents having destination tuple space(s). The list is generated as per the interest of reference agent, where reaction is to be forwarded. The reaction has to be registered with each target agent to get the desired information. The registration process involves storage and update of registered data, lifetime, scope and distribution policy of registered reaction, firing process of reaction etc. The scope refers to a range of applicability of reaction, while distribution policy defines the rules of propagation of the same. The reaction lifetime is decided to be either owner specific, lease based, network connectivity based, or mode based. The mode of a reaction specifies its active period, called *mode*, which is conventionally of two types, viz. ONCE and ONCE/TUPLE [94]. With ONCE modality, a reaction fires once and immediately gets deregistered, while reaction with ONCE/TUPLE modality continues firing for each positively-matched tuple of tuple space, until deregistered explicitly by other method. This type of reaction fires once for each tuple and hence it requires unique tuple identification. However, simultaneous

firing of reaction with ONCE mode is possible, and one outcome is chosen nondeterministically. The TSMM platform is responsible for lease based life of reaction (with a predetermined maximum lease period). If lease period is over and registration of reaction is not renewed, it indicates end of reaction's lifetime. In case of network connectivity based reaction lifetime, reaction's lifetime ends when communication link of target host gets disconnected, and once the link is restored, explicit registration of that reaction is required.

A TSMM platform has the reactive tuple space model at the core of its architecture. The next section presents a generic TSMM architecture to highlight the supporting components.

2.2.3 Generic TSMM architecture

The generic architecture of a TSMM platform typically contains several indispensable components for coordination, communication, discovery, bookkeeping etc., which may have to be arranged in a tiered manner. In the subsequent sections of this chapter, nature of these components in existing TSMM architectures are going to be surveyed.

The most fundamental component for coordination, viz. the tuple space model, has a distribution in the TSMM architecture that vary from platform to platform. However, TSMM platforms support the use of multiple tuple spaces for coordination. Even though the multiplicity of tuple spaces in TSMM is beneficial in several ways [51, 59], their geographically-distributed placements possibly bring forth new difficulties in coordination, like the consistency problem during coordination.

With the distribution of tuple spaces, there is an obvious need on how to discover them to carry out transfer of communicating information. The communication module fulfills transmission of communicating information, whereas discovery module looks up for different locations of tuple spaces in target agents. These modules carry out the tasks of marshalling/unmarshalling[1] of communicating information and neighborhood discovery that are transmitted/received, formation of suitable message structure for transmission,

[1] *Marshalling* is the process of transforming objects and their references to suitable data format for transmission to remote location (see [53] for details). The reverse process of marshalling is called *unmarshalling*.

handling reliability in message delivery, correctness in message receipt and support for heterogeneity. These modules assume the underlying network connectivity model, two aspects of which are of main concern — dynamics of connectivity, and reliability of network links.

The bookkeeping tasks involve maintaining local active agents, preserving information of remote available agents, accounting to data transmission in dynamic environment, enabling migration of its local agents to remote locations, interfacing with applications as well as system routines etc.

2.2.4 Consistency in TSMM

In tuple space coordination, consistency problem has been studied since tuple spaces are geographically scattered to different locations and end-user still gets the idea of logically-shared common tuple space. In such cases, two facets of consistency problem have been identified in different remote tuple-reading/consuming primitives, viz. *OUT-consistency* and *IN-consistency* [44]. OUT-consistency refers to the situation, where antituple of single tuple-consuming primitive (viz. `in` or `inp`) matches more than one tuple from tuple spaces of different target agents, and only one tuple is actually withdrawn as sought tuple, while other matched tuples are retained in their respective tuple spaces. On the other hand, IN-consistency refers to the situation, where newly written tuple matches multiple pending tuple-consuming primitives, and only one of them actually withdraws that tuple as sought tuple, while others are still kept blocked. It is to be noted that OUT-consistency problem is not an issue for bulk tuple-consuming operations (viz. `ing` and `ingp`), whereas the tuple-producing and tuple-reading primitives never create inconsistencies in tuple space coordination. Furthermore, IN-consistency problem is limited within a particular tuple space. Different mechanisms exist in literature to handle OUT-consistency and IN-consistency problems in decentralized tuple space implementations, viz. *strict/exclusive*, *nonexclusive* and *weak* approaches of executing tuple-consuming primitives to resolve OUT-consistency, whereas *strict/exclusive* way of executing different primitives to resolve IN-consistency [26]. The degree of weakness of these approaches vary in decreasing order.

The consistency problems are also prevalent in the recent tuple space implementations too, including TSMM. These problems become more prominent in TSMM platforms, once their reactive tuple space model provide synchronization decoupling. That is, when a reaction fires at tuple space of a target agent, it responds back to reference agent synchronously. As such, OUT-consistency problem has been handled in TSMM platforms by allowing the sought tuple from a target agent, which responds first, to fulfil the requirement of reference agent, for which the availability of reference agent is a necessity. However, amidst the realistic assumption of underlying infrastructure of TSMM platforms to have unpredictable host mobility, sporadic network dynamics, and unreliability in wireless communication links, the existing approaches of resolving consistency problems during agent interactions incur additional overheads. Not only the existing TSMM platforms impose additional time-coupling in synchronization-decoupled tuple space operations, they tend to loose robustness in their coordination functionalities.

2.2.5 Tuple/Antituple/Tuple Space Structuring in TSMM

Apart from resolving consistency problem, a TSMM platform also has to settle the structures of tuples, antituples and tuple spaces.

The *tuple/antituple structure* controls arity of tuple/antituple, nature of their constituent fields, and arrangement pattern of these fields within tuple/antituple. The tuple/antituple structure in existing TSMM is of two types, *ordered* and *unordered*. In the ordered structure, arity, nature and arrangement of constituent fields of tuple/antituple are in a specific predefined pattern, and each *actual* or *formal* of tuple/antituple consists of a pair, viz. NAME and VALUE/SEARCHVALUE. During tuple-antituple matching operation, tuples and antituples match if their arity, arrangement patterns and respective constituent fields match positively. A serious deficiency of ordered tuple/antituple structure is lack of simplicity and versatility of TSMM for its supported applications. The need for predefined arity and ordering of tuple/antituple fields restricts the ability of TSMM platform and its supported applications, thereby reducing its versatility. This is because, same tuple/antituple structure for different types of information shared by different agents of an application is a restraint of versatility. Moreover, the different

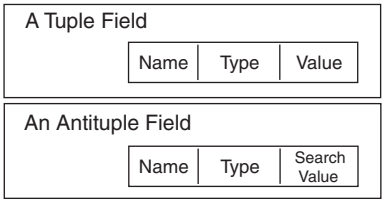


Figure 2.1: Structure of a typical unordered tuple and antituple field.

applications running over same platform possibly have different tuple/antituple structures. Allowing their coordination through fixed and predefined ordered tuple/antituple structure further reduce versatility of TSMM platform.

The TSMM platforms having unordered tuple/antituple structure emerge to tackle these shortcomings. In unordered structure, tuple/antituple do not follow any specific predefined arity, nature and arrangement pattern in constituent fields, thereby improving simplicity and versatility. A typical unordered tuple/antituple field is a triplet of NAME, TYPE and VALUE/SEARCHVALUE, as shown in Figure 2.1. NAME is unique to each tuple/antituple field, as each field has no fixed location inside the respective structure, and it has to be looked up depending on NAME. TYPE stores type information, while VALUE stores *actual* in tuple. The presence or absence of SEARCHVALUE makes an antituple field *actual* or *formal* respectively. For example, {("UserID",String,"Alice"),("MsgSeqNo",Long,20),("Msg",String,"I'm chatting.")} is an unordered tuple, while the unordered antituple {("UserID",String,"Alice"), ("MsgSeqNo", Long,—)} is used to select that tuple; its first field is *actual* and the second one is *formal*. Another possible antituple to select that tuple is {("Msg",String,—)}, which comprises of a single field. This is possible as the arity of tuple and antituple may differ in unordered structure. However, antituple arity must be at most equal to the arity of sought tuple.

A significant drawback of existing unordered tuple/antituple structure is that of additional overheads of *tuple lookup* operations before the tuple-antituple matching operation. This is due to the fact that basic comparison of tuple/antituple fields must be preceded by a lookup operation to ascertain the presence of a related tuple field for comparison against an antituple field, thereby forming an *apposite pair*. For instance, the tuple field `X` and antituple field `Y` form an apposite pair (`X`, `Y`) during comparison, if they fulfil

the lookup operation. Each such tuple in tuple space, whose constituent fields successfully form apposite pairs with all constituent fields of given antituple for comparison, is an *apposite tuple*. Only the apposite tuples can match positively with given antituple; positively-matched apposite tuples are, in fact, sought tuples. The existing approach to reduce lookup overhead in unordered structure is to store the constituent fields of each tuple in an expeditious manner, e.g. hashing. Each tuple incorporates a hash table for that purpose. The process of hashing of constituent fields to organize them within hash table is the *tuple generation* operation. The TSMM platforms follow hashing approaches (like, separate-chained hashing [67]) for generation and lookup of unordered tuple/antituple. The use of hashing scheme though reduces tuple lookup overhead, but creates an additional overhead of tuple generation. In fact, there is a tradeoff between tuple lookup time and tuple generation time in unordered structure: *minimizing lookup time escalates generation time and vice versa*. This tradeoff, in turn, results in the tradeoff between performances of tuple-producing primitives, and tuple-reading/consuming primitives: *improving performance of tuple-producing primitives degrades performance of tuple-reading/consuming primitives, and vice versa*.

In case of unordered tuple/antituple structure, all tuples of tuple space may not be apposite to given antituple, due to vary in arity, nature and arrangement patterns. Hence, it is beneficial to identify the unordered apposite tuples before searching the sought tuples. The structuring of tuple spaces can provide an expeditious way of finding apposite tuples in tuple spaces, which, in turn, can improve performances of tuple-reading/consuming primitives without degrading performances of tuple-producing primitives. The *tuple space structure* controls placement of tuples in tuple space in such a way so as to expedite their operations. However, the TSMM platforms have not much explored this aspect of the tuple space model. Since its inception, tuple space continues to have a *sequential* structure. The new tuples are inserted consecutively in this tuple space structure. The tuple-antituple matching operation compares all tuples of entire tuple space, instead of finding just the apposite tuples, while empty space(s) from withdrawn tuple(s) are subsequently adjusted by shrinking it. Though the sequential structure

benefits tuple-producing primitives, overall performance of the tuple space model centers around frequently-invoked tuple-reading/consuming primitives, a behavior similar to any data storage/retrieval system. These primitives incur significant overhead in sequential structure, due to the expensive comparison overhead of all tuples of entire tuple space with given antituple. In case of *single* tuple-reading/consuming primitives, where only one sought tuple is expected as outcome, if multiple tuples match positively, one of them is chosen nondeterministically. For *bulk* primitives, all positively-matched tuples are returned as sought tuples. Repeating these operations often in sequential structure yields substantial overhead on performances of the tuple space model and reduces TSMM's scalability. The trouble further worsens when TSMM incorporates unordered tuple/antituple structure.

For understanding the impact of sequential tuple space structure in TSMM platform, an example of multiple chat applications at the same time is considered. In this example, the different end-users are chatting and their chat messages are exchanged using a tuple space. The three distinct chat applications are inserting/reading/withdrawing eight different tuple structures simultaneously. Each tuple structure corresponds to a particular type of chat information. A schematic snapshot of the contents of this tuple space at a particular instant is shown in Figure 2.2 to demonstrate expensiveness of tuple-reading/consuming primitives in sequential structure. It can be seen from this figure that apposite tuples for antituple {("UserID",String,—),("MsgSeqNo",Long,—)} are at positions 0, 1, 3, 4, 7, 9 and 10. Again, for antituple {("Sender",String,—),("Receiver",String,—)}, apposite tuples are at 2, 5 and 8. However, in sequential structure, for each of these two antituples, all 11 tuples of tuple space in Figure 2.2 are compared to return one/more sought tuple(s). The additional overhead definitely comes, if tuples at 2, 5, 6 and 8 are futilely compared with first antituple. In the second case, additional overhead is even more, as tuples at 0, 1, 3, 4, 6, 7, 9 and 10 are fruitlessly compared. These additional overheads sum up to deteriorate performances of tuple-reading/consuming primitives, and decrease scalability of TSMM platform.

In some of the existing TSMM platforms having the unordered tuple/antituple structure, the tuple-antituple matching operation does not compare all apposite tuples to

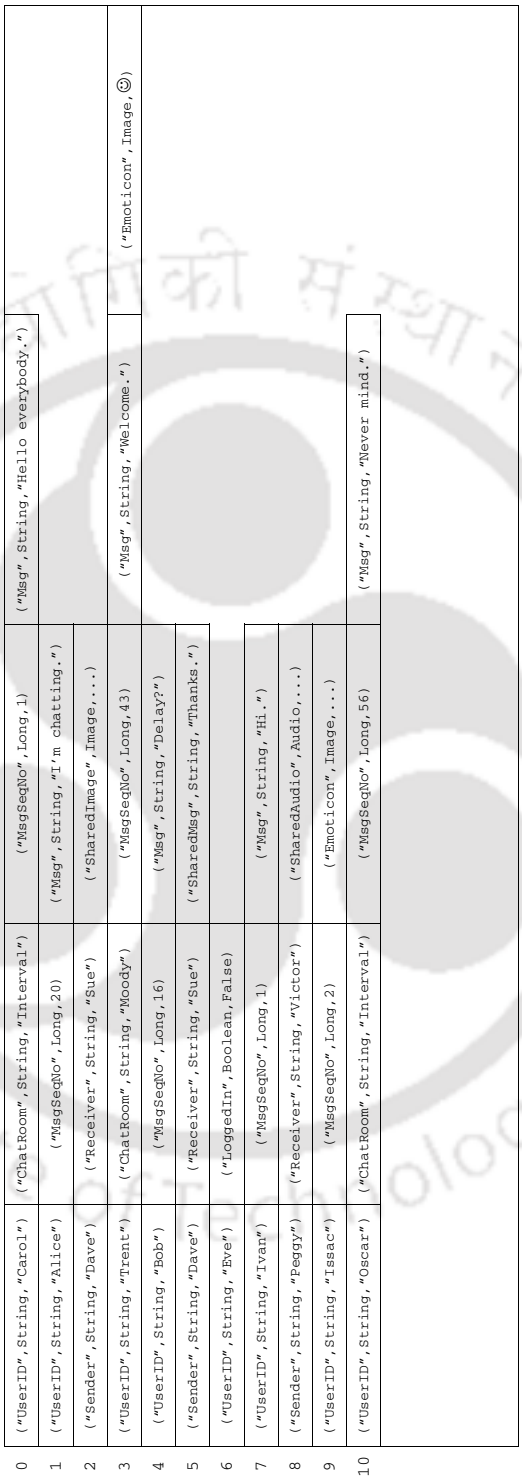


Figure 2.2: A snapshot of sequential tuple space structure for multiple chat applications in TSMM platform.

determine a sought tuple for a single primitive; rather, the comparison stops once a tuple matches positively, and first positively-matched tuple is returned as sought tuple. In this approach, successive invoke of `rd` or `rdp` returns same sought tuple repeatedly, unless interleaved by some other tuple-consuming operations. For example, with antituple {("UserID",String,—), ("MsgSeqNo",Long,—)}, successive `rdp` invokes return the same tuple of position 0 as sought tuple. Furthermore, `rdg`, `rdgp`, `ing` and `ingp` are not benefitted from this approach, as all tuples, irrespective of being apposite or not, are compared. For instance, antituple {("Sender",String,—), ("Receiver",String,—)} in any bulk primitive compares all the eleven tuples of tuple space, even though only tuples at position 2, 5 and 8 are apposite tuples. Similar behavior of returning the first positively-matched tuple as sought tuple in case of single primitive is also found in TSMM platforms having ordered tuple/antituple structure.

2.3 Existing TSMM Platforms

The primary concern in incorporating the tuple space model as coordination media in a TSMM platform is the distribution of (multiple) tuple spaces in its architecture. Based on the distribution pattern, two types of TSMM architecture predominates. In the *centralised* TSMM architecture, all tuple spaces are placed within a central host, whereas in the *decentralised* TSMM architecture, tuple space distributions are not limited by host boundaries. In centralised architecture, a central host (called central server) always holds tuple spaces, and tuple are allowed to move from agents to these tuple space(s) and vice versa. On the other hand, in decentralised architecture, either host or agent or both of them simultaneously can hold tuple spaces, and both tuple movement as well as tuple space mobility are possible. This section thoroughly reviews a few specific existing TSMM platforms that either has commercial importance or directly supports dynamic circumstances, and reports briefly about other existing TSMM platforms.

2.3.1 TSpaces

TSpaces [70, 114] is the IBM's centralized TSMM platform to support coordination and data management of its applications. Such applications are split as 'clients' and 'servers', each being termed as its 'agent'. The clients are having small footprints, as their resource requirements are comparatively less. However, the servers are having heavy footprints with much demand of resources. Figure 2.3 depicts the architecture of TSpaces platform showing its significant components. Apart from supporting coordination and data sharing, TSpaces platform has additional data management functionalities, similar to that of a database system, including support of indexing and different query capabilities.

Communication: The underlying network connectivity of TSpaces platform is assumed to be reliable, and underlying environment is considered to be fairly static.

Coordination: TSpaces follows object-oriented concept during tuple/antituple construction, i.e. ordered tuples and antituples are objects of their respective classes and their fields are also objects of their respective classes. Each tuple field can be of two types — (i) it is a 'value-only' field, where *Value* is only present holding *actual* of some type, or (ii) it is a 'named' field, having two components *Name* and *Value*, which holds type of data and its value as name and *actual* respectively. In an antituple, filled-in search values and `NULL` value are possible. The antituple structure varies with the nature of query. The basic tuple-antituple matching query, indexing query and other complex queries (AND/OR queries) are supported. The AND/OR queries are the combination of basic and indexing queries with logical AND and OR. In TSpaces, multiple tuple spaces, each of which can be uniquely identifiable, are positioned at a centralised location at server-end. These tuple spaces enforce decoupling of interactions between clients and servers. A mobile application can create new tuple spaces or delete existing user-defined tuple spaces, with the help of a special system tuple space called 'Galaxy'. A tuple of Galaxy specifies all parameters related to a new or existing tuple space and is withdrawn to perform an operation related to that tuple space.

The supported tuple space primitives are remote, due to its centralised architecture; these primitives are invoked from client-end and operated on tuple spaces at server-end. Both blocking and nonblocking tuple-reading/consuming primitives are supported; when

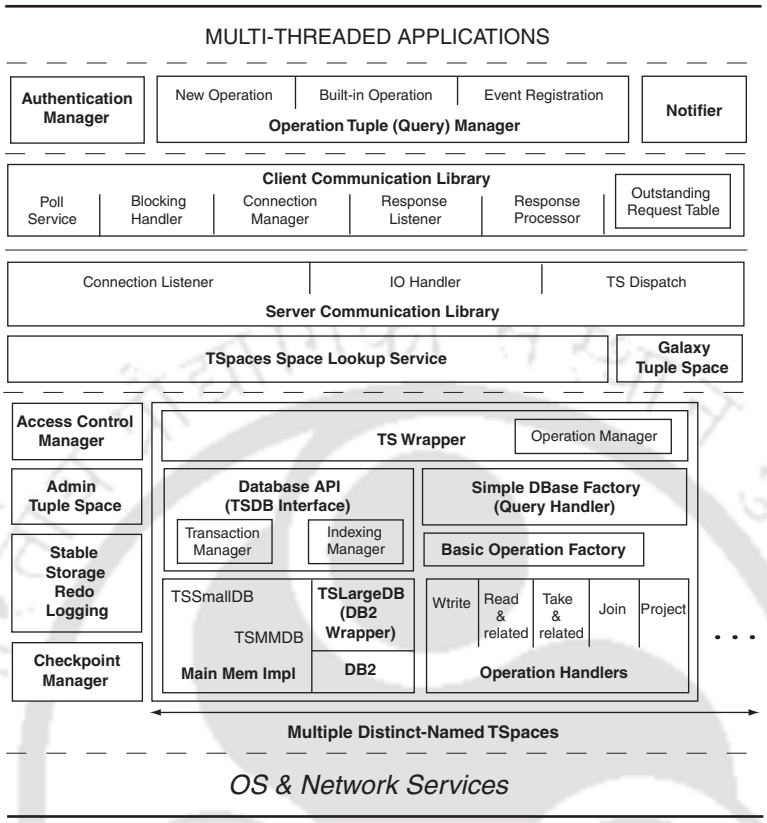


Figure 2.3: Architecture of TSpaces platform showing its significant components.

blocking primitives are invoked, they are blocked directly at client-end, and operations are performed at server-end in nonblocking manner. The supported tuple space primitives are `write`, `read`, `take`, `waittoread`, `waittotake`, `scan`, `consumingscan`, `count` and `rhonda`. Among these primitives, `count` and `rhonda` primitives are newly introduced in TSpaces. Moreover, TSpaces platform allows inclusion of new user-defined tuple space primitives at runtime, provided they can be defined from existing primitives. The first positively-matched tuple is returned as sought tuple of any single tuple-reading/consuming primitive.

TSpaces platform supports reactivity in its rudimentary form. Reactivity is enforced through reactions, which are registered at server-end, and reaction management is also handled there. The clients need to provide required information for registering a reaction, viz. identity of tuple space, nature of operation, antituple, handle of callback object etc.

The given antituple specifies the event of interest, and when that event happens, corresponding reaction fires generating event notification tuples. Subsequently, call method of given callback object is invoked for notifying respective client. There are special tuple spaces in TSpaces platform for storage and maintenance of reactions (stored as tuples) as well as for holding reaction outcome. However, reaction lifetime is decided solely by user. TSpaces does not support multi-destined tuple space operations, and as a result inconsistency problems of tuple space coordination do not arise.

The logical mobility of clients can be accomplished using suitable third-party *M*obile *A*gent *S*ystem (MAS) [65].

Discovery: The discovery mechanism of TSpaces platform follows a two-tier setting — client and server. The Galaxy tuple space of server is used as a lookup database to get location of any inquired tuple space. The discovered data is then stored at client-end for future reference. The identities of server location and Galaxy tuple space are well-known and predetermined. The discovery mechanism performs a unicast communication with the TSpaces server to look up for Galaxy tuple space.

Implementation: TSpaces platform is implemented in Java™ to resolve underlying (device-level) heterogeneity. It utilizes remote method invocation (RMI) of Java™ to manage marshalling/unmarshalling of communicating information.

2.3.2 JavaSpaces

JavaSpaces™ [47, 106, 107] is the Sun's specification of centralized TSMM platform, proposed as a standard service within the Jini framework [111], through which distributed clients and servers are coordinated. In JavaSpaces, both clients and servers are basically agents (of some specific mobile application). Though clients have less resource requirements (i.e. they are small footprints), resource requirements of servers are high for supporting persistent storage and transactional operations. Also, the support of several basic Jini services is required for working of JavaSpaces, like *L*ook*U*p *S*ervice (LUS) for locating tuple spaces, transaction service for transactional operations, event service for supporting reactions, lease service for associating lifetime to different entities etc. Figure 2.4 depicts the architecture of JavaSpaces service showing its significant

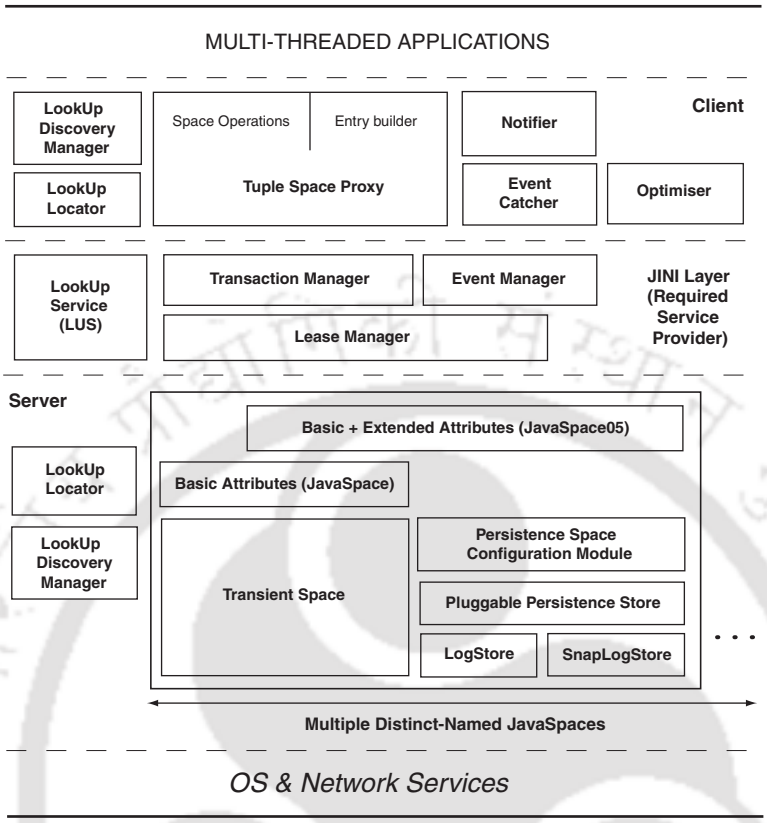


Figure 2.4: Architecture of TSpaces platform showing its significant components.

components.

Communication: In JavaSpaces, the underlying network connectivity is considered to be unreliable, but not dynamic. Instead, a fairly static underlying environment is considered. For handling the unreliability condition, transactional operations are used, such that successful receipts of withdrawn tuples from tuple spaces by clients only commit their withdrawal to servers.

Coordination: In JavaSpaces, each tuple (called 'entry'), antituple (called 'template') and tuple space are viewed as first-class objects, inheriting the advantages of object-oriented paradigm. Each named field of ordered tuple/antituple comprises of two components, viz. *Name* and *Value*. *Name* refers to type of object (viz. name of class, whose instance is that object), and its *Value* contains the object itself. Thus, a tuple becomes a sequence of Java objects. In each antituple field, *Value* has either filled-in

values (specified by the application) or NULL where there is no specific value. However, each antituple field must have a name. The multiple tuple spaces of JavaSpaces are placed at a centralised location, each tuple space being uniquely named. The tuple space operations are remotely invoked from client-end and performed at server-end. The supported tuple space operations include `write`, `read`, `take`, `readIfExists` and `takeIfExists`. The first positively-matching tuple is returned as sought tuple of any single tuple-reading/consuming primitive. The inconsistency problems of tuple space coordination in JavaSpaces, due to multi-destined tuple space operations, are resolved by incorporating such operations within transactional operations.

The reactivity in JavaSpaces is rudimentary in nature, and is implemented as an event, sometimes under a transaction. An event catcher, which is a form of reactive code, is present at client-end to inform the notifier. The event service of Jini framework is used to support reactivity. When a registered reaction fires, it generates an alert and respective event catcher captures and forwards it to notifier. The reactions under a NULL transaction are operated over entire tuple space, while that performed under a non-NULL transaction cover the writes performed within that transaction, in addition to that of entire tuple space. The storage and maintenance of registered reaction is the responsibility of event service. The registered reaction has lease-based lifetime, provided by lease service. If its lease period expires without renewal, it is deregistered. However, its explicit deregistration by owner as well as deregistration on successful completion, when issued under a transaction, are also allowed.

The logical mobility of clients can be made possible in JavaSpaces by applying suitable third-party MAS. However, JavaSpaces has an inherent feature of logical mobility, where proxy of a particular JavaSpaces service is migrated to a client to provide interface for tuple spaces and their supported operations.

Discovery: The discovery mechanism in JavaSpaces involves a three-tier setting — client, Jini distribution layer, and server (i.e. JavaSpaces service). The Jini distribution layer has LUS, which helps in locating required service and providing location information to inquiring client. JavaSpaces locates LUS using its own lookup manager, and registers itself there by sending its proxy. This proxy stores location of sender JavaSpaces service.

The location of LUS is well-known and pre-determined, and clients obtain location of required JavaSpaces proxy from there. Both the location of servers and identification of tuple spaces get discovered via the proxy. The discovery mechanism broadcasts using the IP Multicasting protocol [37, 115] with address 224.0.1.85. Also, LUS itself carries out broadcasting (using IP Multicasting with address 224.0.1.84) in two occasions — initially at its starting and when network gets restored after failure.

Implementation: It is needless to say that JavaSpaces is implemented in Java™. The support of JVM and RMI helps to manage marshalling/unmarshalling of communicating information, and resolve underlying (device-level) heterogeneity.

2.3.3 LIME

LIME [94, 87] (i.e. *L*inda *I*n a *M*obile *E*nvironment) is a decentralized TSMM platform that facilitates distribution of tuple spaces across multiple hosts and agents, as well as transparent merging of available shared tuple spaces to produce a view of 'global virtual data structure' (called 'federated tuple space'). Figure 2.5 shows the architecture of LIME and its significant components.

Communication: The underlying network connectivity of LIME is assumed to be reliable for tuple space operations (using some connection-oriented transport protocol for communication) and unreliable for beacon broadcast (using some connection-less transport protocol for broadcasting). The process of broadcasting is implemented using the IP Multicasting protocol. However, underlying environment of LIME is assumed to be dynamic. This is due to the fact that LIME aims to support its mobile applications over single-hop wireless networks, which can be extended to wired networks as well.

Coordination: The tuple/antituple structure of LIME is ordered in nature, and follows object-oriented concept during construction. In fact, each tuple, antituple and tuple space in LIME can be viewed as a first-class object. Each constituent field of tuple/antituple comprises of two components — *Name* and *Value*. *Name* contains name of a class (considered as type), whose instance is the object (i.e. *actual*), and its *Value* is the object itself. In antituple, only filled-in fields have search values, whereas NULL is used as placeholder for no search value. Tuple uniqueness is achieved by including

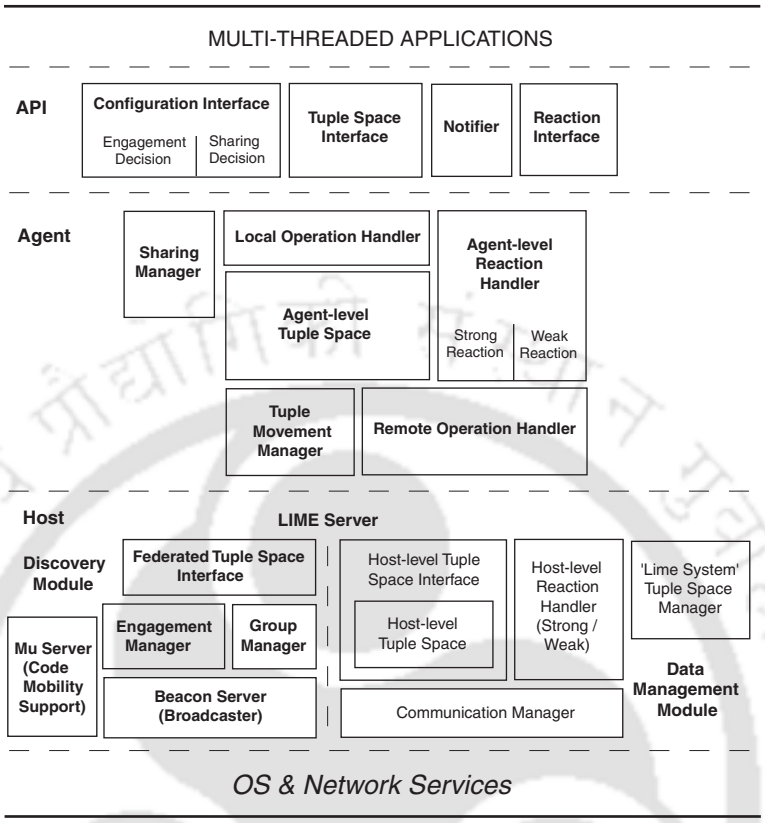


Figure 2.5: Architecture of LIME platform showing its significant components.

extra identity fields, viz. tupleID, hostID and agentID, where hostID and agentID refer to source host and source agent respectively. It is to be noted here that inclusion of location fields in a tuple redefines its boundary of operation domain. In LIME platform, multiple tuple spaces are distributed to different locations. Each such tuple space is identified by a unique name, and can be attached to either agent or host. The supported tuple space operations, which can be local/remote and blocking/non-blocking, are `out`, `rd`, `in`, `rdp`, `inp`, `rdg`, `ing`, `rdgp` and `ingp`. The local operations can be blocking as well as nonblocking, while remote operations are always nonblocking. The local operations are limited within domain of reference agent or reference host. That is, such operations are for its own (local) tuple space, which can be at agent-level or host-level, depending on sharing. The time complexity of `out` for a single tuple as input argument is $O(1)$, while time complexity of `out` for inserting m' input tuples is $O(m')$. The time complexities of all tuple-reading/consuming primitives are asymptotically similar, viz. $O(n'm\hat{n})$, where

n' is the arity of input antituple for such primitives, m is the number of tuples present inside tuple space during matching operation, and $\hat{n}$ is the average tuple arity of residing tuples.

The remote operations span multiple hosts. The first positively-matched tuple of single tuple-reading/consuming primitive is returned as response to reference agent, and the inconsistency problem of multi-destined tuple space coordination is resolved by accepting first received response as sought tuple of invoked single primitive. However, a remote tuple-consuming operation requires at least 5 network transmission to complete. The message overhead complexities for tuple-consuming operations are significant compared to other tuple space operations. As such, their overheads for multi-destined tuple-consuming operations are determined. When a multi-destined `in` primitive is invoked, r messages containing reaction details are sent to the target agents, where r is the total number of target agents chosen for that operation. The target agents return the positively-matched tuples back to reference agent, leading to a worst-case message overhead of r. Among the received responses, the withdrawn process of the first responded tuple is attempted, which may fail as the chosen tuple may be withdrawn in the meantime from its respective tuple space at target agent by other operation. Considering this failure, the worst-case message overhead of issuing actual withdrawal and retrieving the tuple is $2r$. The last stage of remote operation is to issue r reaction-deregister messages to those target agents. So, the message overhead of invoking a multi-destined `in` primitive to r target agents is at most $5r$. In case of the invoke of a multi-destined `inp` primitive, considering that initial p invokes are unsuccessful, and the message overhead due to these occurrences is $2pr$, where r number of invoke messages are sent to the target agents and r number of messages carrying NULL tuples are sent back in each invoke. When successfully invoked, r number of invoke messages are sent, while r' number of messages carrying positively-matched tuples and $(r - r')$ number of messages with NULL tuples are returned. At the best, the message overhead of withdrawing the first-received tuple is 2. On the other hand, at the worst, the message overhead of withdrawing any of the r' matched tuples is $2r'$, due to the possibility of its removal in the meantime. So, the message overhead of invoking a multi-destined `inp` primitive to r target agents

is between $2(p+1)r+2$ and $2(p+1)r+2r'$.

With the invoke of multi-destined `ing` primitive, r reaction messages are sent to the target agents. Assuming that on an average $\bar{\bar{m}}$ number of positively-matched tuples are returned by the target agents, the worst-case message overhead becomes $\bar{\bar{m}}r$ (considering that each message carries a single tuple). The withdrawn process of these $\bar{\bar{m}}r$ matched tuples creates the worst-case message overhead of $2\bar{\bar{m}}r$, due to its possibility of removal. Moreover, sending of r reaction-deregister messages to those target agents results in the worst-case message overhead of invoking a multi-destined `ing` primitive to r target agents is $(3\bar{\bar{m}}+2)r$. In case of multi-destined invoke of `ingp` primitive, the message overhead due to p unsuccessful invokes is $2pr$. When the invoke finds matching tuple, r number of invoke messages are sent, while $\bar{\bar{m}}r'$ number of messages carrying positively-matched tuples and $(r-r')$ number of messages with NULL tuples are returned. The message overhead of withdrawing all the $\bar{\bar{m}}r'$ matched tuples is $2\bar{\bar{m}}r'$. So, the message overhead of invoking a multi-destined `ingp` primitive to r target agents is $2(p+1)r+(3\bar{\bar{m}}-1)r'$.

A reaction of LIME comprises of antituple, handle of callback method, reaction mode, host/agent identity etc. After every regular tuple space operation, all active registered reactions of individual tuple spaces are selected one-by-one nondeterministically and evaluated for the possibility of firing. When a tuple, positively matching a reaction's antituple, is found in a tuple space, the corresponding reaction fires, and its reactive code, in ideal case, gets executed after tuple space operation atomically. This restrictive requirement results in two types of reactions in LIME — 'strong' and 'weak'. Only strong reactions comply to this restrictive atomic condition. Such reactions are limited only within a host boundary, and require knowledge of agent/host locations. The weak reactions form the basis of 'federated tuple space' view spanning beyond host boundaries and is implemented as a collection of strong reactions in multiple hosts. When a tuple matching given antituple is found in respective tuple space, a registered reaction fires. For a strong reaction, firing and invocation of callback method handle is performed atomically. For a weak reaction, a copy of matching tuple is sent to reference agent/host, who then sends back desired operation parameters for performing them at respective target agent/host, and finally sending back result tuples. The special data structures,

termed as 'reaction lists', are present for storage and maintenance of reactions. The reaction lists at hosts are used for storing both strong and weak reactions, while that at agents are only used for strong reactions. Both ONCE and ONCE-PER-TUPLE modes of reaction lifetime are supported. Also, reactions can be explicitly registered and deregistered by user. Moreover, weak reactions can get registered and deregistered based on connectivity of hosts. The scope of a LIME reaction also depends on its type — strong reactions are limited to host boundaries, while weak reactions have no boundaries and are propagated to target hosts that are identified by specific parameters. During propagation, entire reaction structure is sent to target hosts.

The transient community of agents and their hosts, due to their migration and connectivity variation, results into dynamic grouping of tuple spaces. The 'federated tuple space' view is the outcome of group membership function of LIME, whereby neighbor hosts (along with their agents) form a group when they possess identically-named tuple spaces. Through the merged view of tuple space, individual agents can interact among themselves. The tuples from agent-level tuple spaces are moved to respective host-level tuple spaces on getting shared and moved back on becoming private. Reactions are provided to notify about any desired change in the content in a particular tuple space or in a formed group. LIME platform relies on transaction to ensure atomicity in federated tuple space operations. The tuple-producing operation to a disconnected remote tuple space leads to the notion of 'misplaced' tuples, whereby its argument(s) (i.e. tuple(s)) are stored locally. When target tuple space becomes available, all misplaced tuples destined to that tuple space, are transferred to it. This behaviour hides the blocking nature of remote tuple-producing primitives due to disconnection.

LIME platform supports agent mobility using μ-Code [93] toolkit. Only agent-level tuple spaces are migrated along with respective mobile agents.

Discovery: The discovery mechanism of LIME platform eventually leads to discovery of tuple spaces of 'shared' agents and 'engaged' hosts. The process of discovery is carried out by every host broadcasting its availability condition in the form of 'beacon' messages. A beacon message comprises of the identity of a host (and optionally its active agents),

parameters about location (usually representation of physical location, like latitude-longitude pair) and stopping value. The stopping value is a boolean argument indicating a condition of host, viz. whether it is shutting down or not. On receipt of a beacon message, details of corresponding host are stored in a local data structure of recipient, commonly called 'neighbor list'. This list is used by host's group manager to contact other hosts. The discovery mechanism broadcasts to address 230.0.0.1 at port 6000.

Implementation: LIME is implemented in Java™ to handle marshalling and unmarshalling of communicating information and to resolve underlying (device-level) heterogeneity. A striking feature about LIME platform is the addition of another level of transiently-shared tuple spaces (called LIME SYSTEM tuple spaces), through which an agent can get information about system configurations. These system-level tuple spaces are operating alongside actual middleware at the core level. They keep up-to-date information about system changes (like, mobile agents presently residing in a particular host), their group formation, sharing of tuples spaces etc. Only middleware has an exclusive right to operate (like insertion/withdrawal of tuples) on LIME SYSTEM tuple spaces. However, agents can read tuples and register reactions to get updated system configurations.

2.3.4 LIMONE

LIMONE [46] (i.e. *LI*ghtly coordinated *MO*bile *NE*tworks) is similar to LIME, but with less restrictive coordination activities between interacting agents (i.e. not considering strong atomic criteria and stable connectivity), thereby making it lightweight. Figure 2.6 shows the architecture of LIMONE and its significant components.

Communication: The assumptions about underlying environment are similar to that of LIME platform, i.e. dynamic as well as reliable network connectivity for tuple space operations and unreliable for beacon broadcast.

Coordination: The tuple/antituple structure of LIMONE is unordered in nature, and its approach of construction also follows the object-oriented concept. In fact, each constituent field of tuple/antituple comprises of two components — *Name*, *Type* and *Value*. *Name* contains a unique name given to a class, whereas *Type* actually stores the

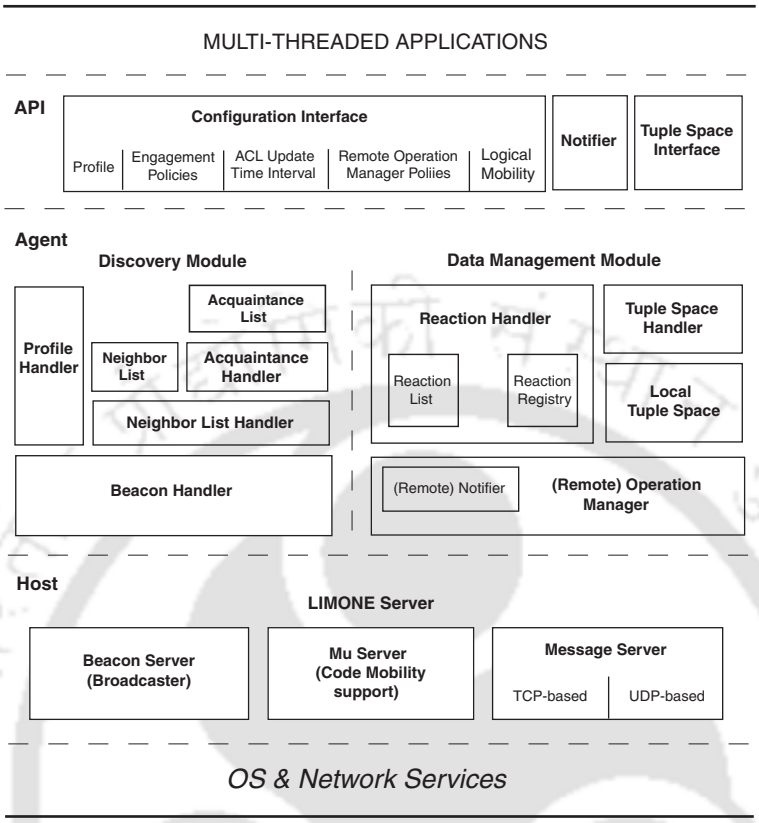


Figure 2.6: Architecture of LIMONE platform showing its significant components.

classname (considered as type), whose instance is the object (i.e. *actual*), and its *Value* is the object itself. The tuples are unique, and uniqueness is implemented by including extra identity fields (having similar triplet structure), viz. tupleID, hostID and agentID, in each tuple. The antituples have named constraints and corresponding constraint functions, resulting in similar triplet structure. The constraint function operates over field type and constrained value. However, in LIMONE, multiple tuple spaces are distributed to different agents only, with each agent holding one tuple space. Thus, both logical mobility and physical mobility of tuple space can be achieved. The supported tuple space primitives, which can be local/remote as well as blocking/non-blocking, are `out`, `rd`, `rdp`, `rdg`, `rdgp`, `in`, `inp`, `ing` and `ingp`. The remote primitives are always blocking on reference agent, and nonblocking as well as atomic on target agent. However, a target agent may accept or deny any requested tuple space operation. The timeout scheme is provided for blocking operations to prevent deadlock. Also, an identification scheme

is included to distinguish results of invoked operations in case of delayed transmission. The tuple-antituple matching operation has the following conditions: (i) each antituple constraint must match a particular tuple field; (ii) constraint *Name* must match a tuple field *Name*, and constraint function must satisfy corresponding (*Type*,*Value*) pair of that field. The time complexities for supported primitives are asymptotically similar to LIME. The first positively-matched tuple of single tuple-reading/consuming primitive is returned to reference agent as response, and the inconsistency problem of multi-destined tuple space coordination is resolved by accepting first received response as sought tuple.

Each reaction in LIMONE comprises of a reactive pattern and a handle of callback function. The reactive pattern, in turn, consists of an antituple and a list of profile selectors. The profile selectors are matched against existing profiles of remote agents to select the set of target agents, where reaction is to be forwarded. During transfer, only reactive pattern is propagated, while callback function handle is retained by reference agent. Like LIME, reactions are evaluated after every tuple space operation. On getting one/more positively matched tuple in any target agent, reaction fires. The firing of a reaction and its successive processes are similar to weak reaction concept of LIME. The reference agent, on receiving copy of positively matched tuple(s), invokes reaction's callback function atomically to read/withdraw that tuple from tuple space of target agent in a non-blocking manner. Two special data structures, viz. 'reaction registry' and 'reaction list', are present in every agent for storage and maintenance of reactions. The reaction registry only stores reactions of reference agent, whereas reaction list contains reactive patterns of other agents registering their reactions in local tuple space. The reaction list gets updated by acquaintance handler on explicit withdrawal of reaction or change in network connectivity. No information about source of registered reactions are maintained to keep LIMONE lightweight. Hence, reregistration of active reactions is required when connectivity is restored. The lifetime of a reaction is similar to that of LIME, viz. ONCE/ONCE-PER-TUPLE mode or explicit by user.

The message overhead complexities of multi-destined tuple-consuming operations are determined, due to their significance compared to tuple-producing/reading operations. When a multi-destined `in` primitive is invoked, r messages are sent to the target agents

(r being the number of target agents chosen for that operation), and at most r messages are received containing positively-matched tuples. The withdrawal of sought tuple from the received tuples produces at most $2r$ messages, due to the possibility of their removal in the meantime. So, the message overhead of invoking a multi-destined `in` primitive to r target agents is at most $4r$. In case of the invoke of a multi-destined `inp` primitive, the message overhead due to unsuccessful invokes is $2pr$, where r number of invoke messages and r number of messages carrying NULL tuples are transmitted in each invoke. When invoke becomes successful, r number of invoke messages are sent, whereas r' number of messages carrying positively-matched tuples and $(r - r')$ number of messages having NULL tuples are returned. The withdrawal process produces the message overhead of at most $2r'$, due to the possibility of removal of matched tuples in the meantime. So, the message overhead of invoking a multi-destined `inp` primitive to r target agents is at most $2(p + 1)r + 2r'$. With the invoke of multi-destined `ing` primitive, r messages are sent to the target agents, and at most $\bar{\bar{m}}r$ number of messages carrying positively-matched tuples are returned by them, considering that each message carries a single tuple ($\bar{\bar{m}}$ being the average number of positively-matched tuples returned by a target agent). The withdrawn process of these tuples produces the message overhead of $2\bar{\bar{m}}r$. So, the message overhead of invoking a multi-destined `ing` primitive to r target agents is $(3\bar{\bar{m}} + 1)r$. In case of multi-destined invoke of `ingp` primitive, the message overhead due to p unsuccessful invokes is $2pr$. When the invoke is successful, r number of invoke messages are sent, whereas $\bar{\bar{m}}r'$ number of messages carrying positively-matched tuples and $(r - r')$ number of messages with NULL tuples are received by reference agent. The message overhead of withdrawing $\bar{\bar{m}}r'$ matched tuples is $2\bar{\bar{m}}r'$. So, the message overhead of invoking a multi-destined `ingp` primitive to r target agents is $2(p + 1)r + (3\bar{\bar{m}} - 1)r'$.

In LIMONE, any pair of agent-level tuple spaces maintains asymmetric and intransitive relations between them. Through its engagement policies, each agent selects a list of interested remote tuple spaces from its available neighbors to decide its list of acquaintances. So, it may be possible that one tuple space is interested in communication with another, while the vice versa is not true. Like LIME, μ-Code also handles agent migration in LIMONE.

Discovery: In the discovery mechanism of LIMONE platform, discovery of host eventually leads to discovery of available agents and their tuple spaces. The discovery mechanism is responsibility of host only, in the form of broadcasting beacon messages. The agents are subsequently informed about available agents from received beacons. A beacon message comprises of a set of profiles of available agents, running in a particular host, along with information about working protocol for data transfer. Each profile corresponds to an agent and its structure resembles a tuple — `[<Property name,type,value>,...]`. There are two mandatory entries in a profile, viz. `Profile[ <"hostID",String,"HostIP">, <"agentID",String,"Unique Number">, ...]`, while other fields depend on application's nature. The profiles from received beacons are copied to active agents of receiving host. Each agent has a 'neighbor list' to store others' profiles. From this list, an agent prepares its 'acquaintance list'. The update interval of acquaintance list is user-specified; if a profile in a particular entry is not updated within that interval, that entry is removed from the list.

Implementation: Like LIME, LIMONE platform is also implemented in Java™ to handle marshalling/unmarshalling of communicating information and resolve underlying (device-level) heterogeneity.

2.3.5 EgoSpaces

EgoSpaces [64] is another decentralized TSMM platform that allows interactions of agents in wireless ad hoc environments using egocentric notion of context. The egocentric notion is achieved through 'view' concept, abstracting context sensed by an agent in presence of its mobility. Figure 2.7 shows the architecture of EgoSpaces and its significant components. The tuple space model of EgoSpaces platform, viz. ELights, supports strict and relaxed look up of sought tuples in an agent-defined 'view', created by conglomerating multiple remote tuple spaces in proximity. Behaviors can be assigned to a 'view', whereby certain actions are to be automatically performed in response to specified changes in view.

Communication: The assumptions about dynamics in underlying environment of EgoSpaces platform are similar to that of LIME and LIMONE platforms.

Coordination: The tuple/antituple structure of EgoSpaces platform is also similar

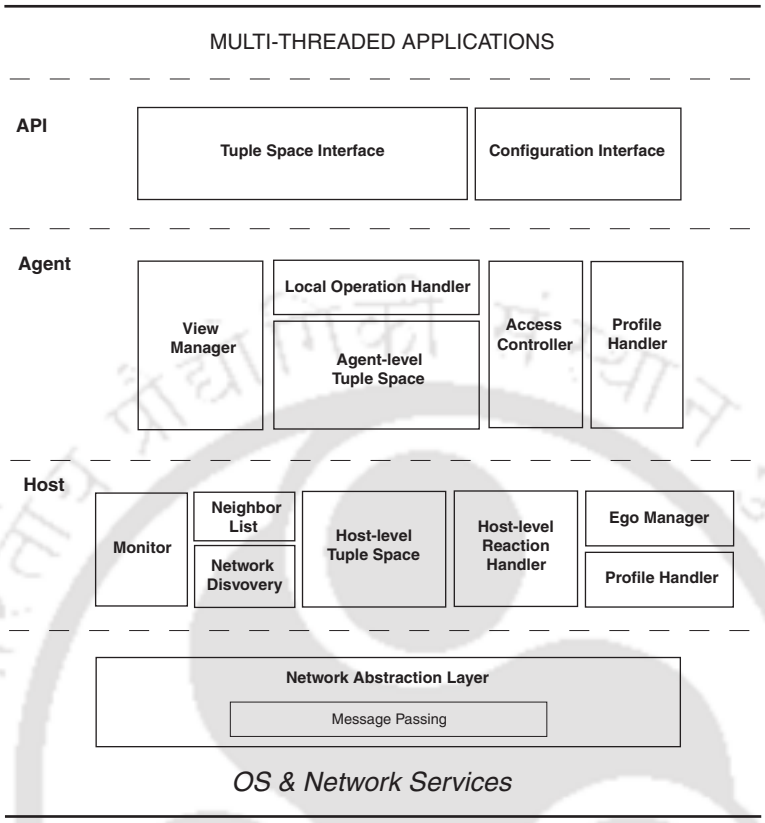


Figure 2.7: Architecture of EgoSpaces platform showing its significant components.

to LIMONE, i.e. unordered structure, viz. *Name*, *Type* and *Value*/*Constraint*. However, in EgoSpaces, multiple tuple spaces are distributed to both agents as well as hosts, with each agent and host holding exactly one tuple space. Like LIME, each shared tuple is transferred from agent's tuple space to host's tuple space for remote access. The supported tuple space primitives, which can be local/remote and blocking/non-blocking, are `out`, `rd`, `rdp`, `rdg`, `rdgp`, `in`, `inp`, `ing`, `ingp`, `rdsp`, `insp`, `rdgsp` and `ingsp`. The last four primitives are introduced to ELights of EgoSpaces for scattered probing operations, which are weaker versions of `rdp`, `inp`, `rdgp` and `ingp` respectively. The local primitives are always nonblocking, while remote primitives can be blocking and nonblocking on reference agent, as well as atomic on target agent. The tuple-antituple matching operation is similar to that of LIMONE platform. Thus, the time complexities of supported primitives are asymptotically similar to LIME. The first positively-matched tuple of single tuple-reading/consuming primitive is returned to reference agent as response,

and the inconsistency problem of multi-destined tuple space coordination is resolved by accepting the first received response as sought tuple. Like LIME platform, a remote tuple-consuming operation requires at least 5 network transmission to complete.

A reaction of EgoSpaces comprises of a reactive pattern, optional indication of its withdrawal from respective tuple space, and optional action of its modification and reinsertion into a possibly different tuple space. The reactive pattern is like that of LIMONE. The reactions are to be registered in remote tuple spaces that are included in the 'view' of reference agent. The reaction firing and its consecutive actions are similar to weak reaction concept of LIME. However, reaction storage and maintenance are through 'reaction registry' and 'reaction list', similar to LIMONE. The reaction list gets updated on explicit de/re-registration of reaction or due to change in network connectivity. The lifetime of a reaction is ONCE-PER-TUPLE mode or explicit by user. Moreover, operating modality of a reaction in EgoSpaces can be 'eager' or 'lazy', a notion which has been implemented using some priority scheme.

The message overhead complexities for tuple-consuming operations are significant, when compared to other tuple space primitives. When a multi-destined `in` primitive is invoked in EgoSpaces, r messages containing its arguments are sent to r target agents chosen for the operation. The target agents return the positively-matched tuples back to reference agent, leading to a worst-case message overhead of r. Among the received responses, a nondeterministically-chosen tuple is withdrawn from tuple space of the respective target agent, and the same is again responded back to reference agent, leading to a message overhead of 2. The last stage of remote `in` operation is to sent r op-deregister messages to r target agents. So, the message overhead of invoking a multi-destined `in` primitive to r target agents is at most $3r+2$. In case of the invoke of a multi-destined `inp` primitive, the initial steps towards the operation (viz. data gathering from r target agents and their ordered locking) take a one-time message overhead of $4r$. Once the operation starts, considering that first p invokes are unsuccessful, and the message overhead due to these occurrences is $2pr$. When successfully invoked, r number of invoke messages are sent, while r' number of messages carrying positively-matched tuples and $(r-r')$ number of messages with NULL tuples are returned. Then, the message overhead of withdrawing

the first-received tuple is 2. At the last, r unlock messages are sent to complete the operation. So, the message overhead of invoking a multi-destined `inp` primitive to r target agents is $(2p + 7)r + 2$.

With the invoke of multi-destined `ing` primitive, r operation messages are sent to r target agents. Assuming that on an average $\bar{\bar{m}}$ number of positively-matched tuples are returned by the target agents, the worst-case message overhead becomes $\bar{\bar{m}}r$ (considering that each message carries a single tuple). The withdrawn process of these $\bar{\bar{m}}r$ matched tuples creates the worst-case message overhead of $2\bar{\bar{m}}r$. Moreover, r op-deregister messages are sent to those target agents. So, the worst-case message overhead of invoking a multi-destined `ing` primitive to r target agents is $(3\bar{\bar{m}} + 2)r$. In case of multi-destined invoke of `ingp` primitive, the initial steps are similar to the invoke of `inp`, resulting in a message overhead of $4r$. With the start of the operation, the message overhead due to p unsuccessful invokes is $2pr$. When the invoke finds matching tuples, r number of invoke messages are sent, while $\bar{\bar{m}}r'$ number of messages carrying positively-matched tuples and $(r - r')$ number of messages with `NULL` tuples are returned. The message overhead of withdrawing all the $\bar{\bar{m}}r'$ matched tuples is $2\bar{\bar{m}}r'$, followed by r unlock messages to the target agents. So, the message overhead of invoking a multi-destined `ingp` primitive to r target agents is $(2p + 7)r + (3\bar{\bar{m}} - 1)r'$.

Like LIMONE, a pair of agent-level tuple spaces of EgoSpaces is asymmetrically and intransitively related. Each agent prepares its 'view' of global (virtual) tuple space from its list of acquainted and interested remote tuple spaces, which is decided by its engagement policies. Any tuple space operation on a view is a logical notion, actually carried by forwarding that operation to individual target tuple space present in the view. Like LIME and LIMONE, agent migration is also possible in EgoSpaces.

Discovery: The discovery mechanism of EgoSpaces platform ia again similar to that of LIME and LIMONE platforms, viz. periodic beacon broadcast. The composition of beacon message is similar to LIMONE. The discovery mechanism eventually leads to the construction of a 'view'.

Implementation: Like LIME and LIMONE platforms, EgoSpaces is also implemented in Java™ for marshalling/unmarshalling of communicating information and resolving underlying (device-level) heterogeneity.

2.3.6 Other TSMM Platforms

The remaining TSMM platforms are non-commercial as well as not much closely related to this thesis. So, a brief study about their nature related to the above-discussed factors of a TSMM platform are presented next.

Communication: The underlying network connectivity of some TSMM platforms, like PageSpace [28, 29], L²imbo [33, 32, 110] etc., are assumed to be static (i.e. deployed in wired networks), while for others, like TOTA [77, 75, 76] (i.e. *T*uples *O*n *T*he *A*ir), are based on dynamic connectivity (i.e. deployed in wireless networks). Again, PageSpace, L²imbo etc., like TSpaces, require reliable connectivity links for their functioning, while TOTA consider unreliable network connectivity models. The other TSMM platforms, like TuCSoN [89] (i.e. *Tu*ple *C*entres *S*pread *o*ver *N*etworks), MARS [17] (i.e. *M*obile *A*gent *R*eactive *S*paces) etc., rely on third party communication modules for data transmission.

Coordination: The tuple/antituple structure of PageSpace, L²imbo, TuCSoN, MARS, TOTA etc. are ordered in nature. The ordered structure of PageSpace is composite, as it is a combination of Linda, Laura [108] and ShaDe [22]. Such structure can be similar to simple Linda structure, or service forms or objects. The ordered tuple/antituple structure of L²imbo, TuCSoN, MARS etc. are like that of TSpaces. The tuple/antituple structure of TOTA comprises of ordered set of data fields (like Linda), along with two additional fields carrying propagation rule and maintenance rule. The tuple space structure of TSpaces, JavaSpaces, LIME, LIMONE, EgoSpaces, PageSpace, L²imbo, TuCSoN, MARS, TOTA etc. is sequential in nature. The supported tuple space primitives are only local in TuCSoN and MARS, while they are both local and remote in other TSMM platforms. All TSMM platforms, except TOTA, support both blocking and nonblocking primitives, while TOTA supports only nonblocking primitives. The tuple space operations in all the surveyed TSMM platforms are performed atomically. All

TSMM platforms surveyed have multiple tuple spaces in their architecture, distributed to agents or hosts or both. However, based on their distribution, PageSpace, TuCSoN, MARS, TOTA etc. follow decentralized TSMM architecture, whereas architecture of L^{2}imbo has both centralized and decentralized distribution. Any pair of such tuple spaces maintains symmetric and transitive relation between them in PageSpace, L^{2}imbo etc., whereas asymmetric and intransitive relation exist between a pair of tuple spaces in TOTA. TuCSoN, MARS etc. isolate each tuple space from others, thereby not following any relational model among them. Also, PageSpace, L^{2}imbo, TuCSoN, MARS, TOTA etc. achieve tuple space mobility only through physical mobility of hosts.

Discovery: In the discovery mechanism of PageSpace, a host component (viz. epsilon module) is responsible for discovery of remote tuple spaces in other hosts, and store discovered data in their local host-level tuple spaces. Similarly, L^{2}imbo has 'Quality of Service Monitoring' component for discovering remote hosts and their tuple spaces and storing locally. In TuCSoN, MARS etc., discovery functionality is handled by additional MAS layer. The discovery mechanism in TOTA is similar to LIME. In these TSMM platforms, discovery mechanism proceeds in a proactive manner, with each host periodically broadcasting the availability data of its active agents in the form of beacons.

Implementation: Since almost all the TSMM platforms surveyed require support of object-oriented concept for their functioning, they are implemented using some object-oriented programming language like Java^{TM}. In fact, the TSMM platforms covered here, except L^{2}imbo, are implemented in Java^{TM} to handle marshalling/unmarshalling of communicating information and to resolve underlying (device-level) heterogeneity. L^{2}imbo, being implemented in C language, incorporates a separate network abstraction layer to manage data marshalling/unmarshalling and resolving heterogeneity.

Other coordination platforms, developed predominantly for stable underlying environments include Ara [92], KLAIM/Klava [35], XMIDDLE [80], EventHeap [62], Anthill [6], XSpace [72], SwarmLinda [85], PeerSpaces [15], CAST [96], xSpace [10], $\sigma\tau$-LINDA [109] etc. These platforms typically follow the traditional middleware architecture, which is

not closely related to this thesis. Few recent TSMM platforms, viz. MESH*Mdl* [55], FactSpaces [86] etc., have also been designed for dynamic underlying environments; however, their inherent behaviors are similar to LIME, LIMONE and EgoSpaces platforms.

In the next section, a summary of the findings after literature survey is tabulated, focussing particularly on the motivating factors of this thesis.

2.4 Comparative Analysis of Existing TSMM Platforms

Among the TSMM platforms surveyed, some platforms require reliable/stable underlying network connectivity (like TSpaces, JavaSpaces, PageSpace, L^{2}imbo etc.). Since motivation for this thesis leads to assumption that underlying network connectivity of $\mathcal{CUTSMuHN}$ platform is dynamic as well as unreliable, above TSMM platforms are not been considered for comparative analyses in later chapters. Other TSMM platforms, like TuCSoN, MARS etc. are bare frameworks of tuple space coordination, as many components are not incorporated for their full functioning. Such platforms are also omitted during comparative analyses. Even though some aspects of TOTA platform are related to this thesis, its research directions focus predominantly on different aspects of autonomic/adaptive computing, which is not within the scope of this thesis.

LIME, LIMONE and EgoSpaces attempt to support tuple space coordination in dynamic environment of wireless ad hoc networks. However, LIME provides a view of global tuple space by federating multiple tuple spaces available within a single-hop communication range. While doing so, it provides strict atomicity during tuple space operations, for which it requires resourceful processing and predictable network for proper working. EgoSpaces behave much similar to LIME, except that the global federating view of all available tuple spaces (within 1-hop) is restricted to only 'interested' tuple spaces. Imposing interest is a form of egocentric notion, so that agents can focus only on relevant data. So, EgoSpaces also requires relatively resourceful hosts and less dynamic network connectivity, which is rare in wireless networks. LIMONE, on the other hand, is designed in a lightweight manner to eliminate all underlying assumptions while enable

Table 2.1: Approaches of handling coordination attributes in different TSMM platforms.

Coordination Attributes	**Approaches of handling in different TSMM platforms**		
Tuple/antituple structure	*Ordered* TSpaces, JavaSpaces, LIME, PageSpace, L^{2}imbo, TuCSoN, MARS, TOTA.		*Unordered* LIMONE, EgoSpaces.
Tuple space structure	*Sequential* TSpaces, JavaSpaces, LIME, LIMONE, EgoSpaces, PageSpace, L^{2}imbo, TuCSoN, MARS, TOTA.		
Atomicity in tuple space operation	*Strictly atomic* TSpaces, JavaSpaces, LIME, LIMONE, EgoSpaces, PageSpace, L^{2}imbo, TuCSoN, MARS, TOTA.		
Tuple space architecture	*Centralized* TSpaces, JavaSpaces.	*Decentralized* LIME, LIMONE, EgoSpaces, PageSpace, TuCSoN, MARS, TOTA.	*Hybrid* L^{2}imbo.
Tuple space relational model	*Symmetric & Transitive* TSpaces, JavaSpaces, LIME, PageSpace, L^{2}imbo.	*Asymmetric & Intransitive* LIMONE, EgoSpaces, TOTA.	*Isolated* TuCSoN, MARS.
Underlying network connectivity	*Reliable* TSpaces, LIME (data), LIMONE (data), EgoSpaces (data), PageSpace, L^{2}imbo.	*Unreliable* JavaSpaces, LIME (beacon), LIMONE (beacon), EgoSpaces (beacon), L^{2}imbo, TOTA.	*Third party MAS* TuCSoN, MARS.
	Stable TSpaces, JavaSpaces, PageSpace.	*Dynamic* LIME, LIMONE, EgoSpaces, TOTA.	
Underlying networks	*Wired network* TSpaces, JavaSpaces, PageSpace, L^{2}imbo, TuCSoN, MARS, TOTA.	*Single-hop wireless ad hoc network* LIMONE, EgoSpaces.	*Both* LIME.

location-dependent interactions among agents; so, during coordination, it provides less strict guarantees.

The Table 2.1 tabulates the comparative observations after surveying the existing TSMM platforms. In this table, the coordination attributes compared are the significant aspects while designing any TSMM platform. In later chapters, these observations are being compared with *CUTSMuHN* platform to indicate its improvements in tuple space coordination.

2.5 Conclusion

This chapter has presented a structured and comprehensive survey of TSMM platform, a promising genre of mobile middleware platform, to investigate its future research directions. Based on the observations from this survey, several limitations have been found in the design factors of TSMM platform, which become the motivating factors for this thesis. Among them, foremost limitation lies in the strict atomicity of tuple space operations in existing TSMM platforms. Their strict atomic nature has the tendency to tight couple interacting agents during remote operations. Moreover, strict atomicity also increases network overhead in order to maintain consistency during such operations. The other limitations in existing TSMM platforms concern with their tuple/antituple and tuple space structures. The benefits of unordered tuple/antituple structure and sequential tuple space structure are curbed by increased performance overheads. Lastly, there is a limited support of network infrastructure and practically no measure to resolve underlying heterogeneity in existing TSMM platforms. In the successive chapters, these problems are being settled individually for betterment of tuple space coordination of TSMM platforms in dynamic environments.

Chapter 3

Improvements in Tuple Space Coordination of *CUTSMuHN*

3.1 Introduction

Chapter 2 has affirmed that the tuple space model has been introduced for multi-dimensional decoupling in coordination among distributed and interacting agents of an application in the parallel programming, distributed computing and mobile computing paradigms. This chapter proposes a new dimension of decoupling for tuple space coordination that is beneficial while using the tuple space model in dynamic environments, like mobile middleware platforms. This chapter also focuses on the design of a new TSMM platform, named *CUTSMuHN*, which implements the proposed notion of decoupling of tuple space coordination. This chapter additionally proposes several improvements in other deficiencies of tuple space coordination of TSMM platform, like consistency problem in agent interactions, and limitations in tuple/antituple as well as tuple space structures. These proposals are also integrated in the design of *CUTSMuHN* platform to demonstrate the improvements of tuple space coordination in dynamic environments.

3.2 Proposed Decoupling Notion in Tuple Space Coordination

This section proposes a new dimension of decoupling in tuple space coordination of TSMM platform. The proposed dimension of decoupling, together with synchronization decoupling, benefit agent interactions in dynamic environments. The assumption about

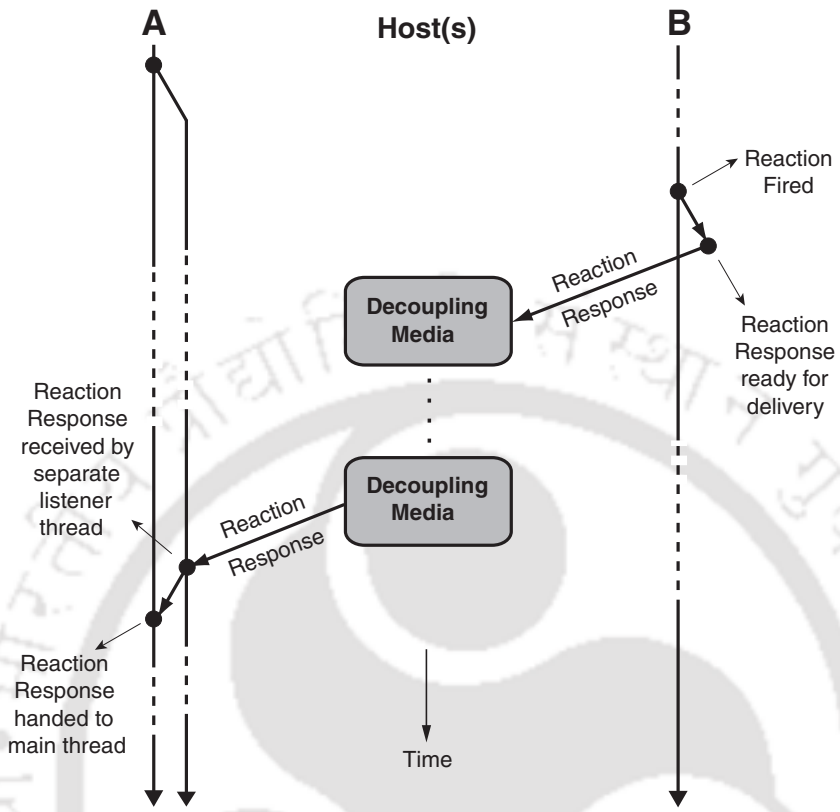


Figure 3.1: Proposed time-decoupling of reaction response of target agent B from reference agent A using decoupling media at host-level.

the working environment of a TSMM platform to be dynamic is realistic, as any mobile middleware platform has to deal with device mobility and wireless connectivity in its underlying networks.

The proposed dimension of decoupling introduces time-decoupling of response of a reaction (which is registered in lieu of invoked tuple space primitive) from reference agent, as shown in Figure 3.1. This figure picturizes the time-decoupled receipt of reaction response by reference agent A from target agent B. The two arrows of A signifies its two threads of execution, its listener thread being forked out of its main thread at initiation. B also has its listener thread, but it is not shown in Figure 3.1 (only its main thread is shown) to highlight on the proposed notion of decoupling. The reaction originally imposes (partial) synchronization decoupling between A and its invoked primitive, and the proposed notion of decoupling augments it with time-decoupled transfer of reaction response back to A, leading to "complete synchronization decoupling" of invoke of tuple

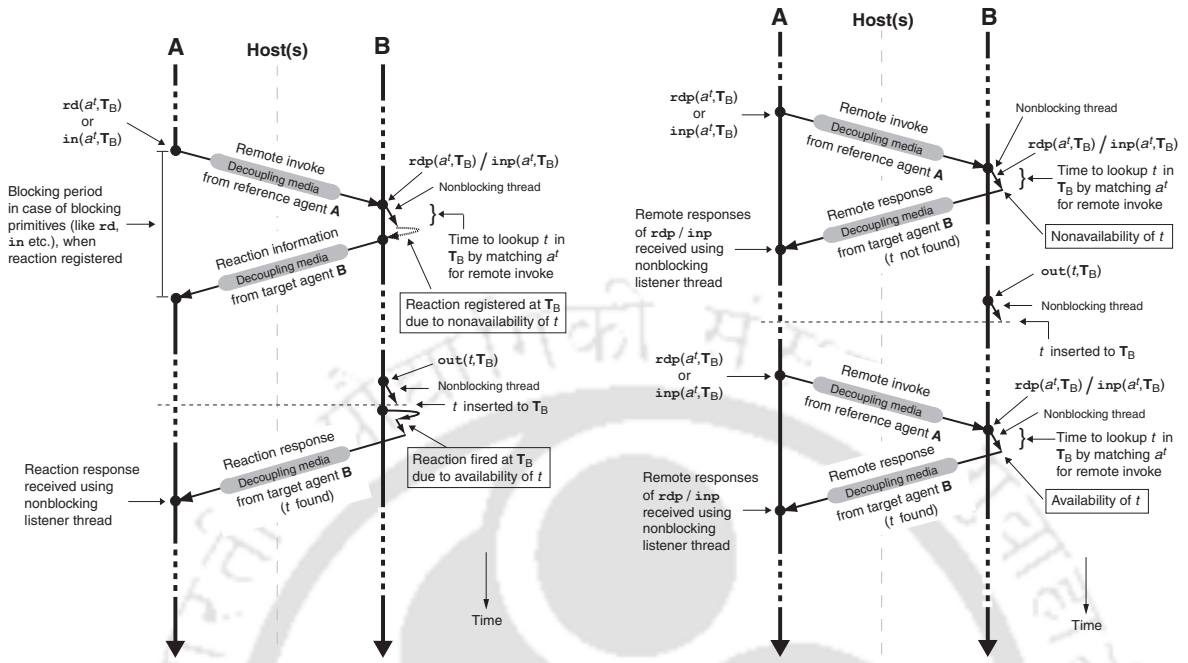
space primitives. The time-decoupled transfer of reaction responses from target agent to reference agent is proposed to be accomplished by inserting an additional tier of tuple spaces as decoupling media in the paths of reaction responses and imposing time-decoupling through them. This results in a tiered organization of tuple spaces in TSMM architecture. Each such tuple space is placed as decoupling medium at each mobile host having interacting agents, which not only time-decouples reaction responses, but also makes the invokes of reference agent and responses thereof by target agent asynchronous at their respective hosts.

The basic concept behind complete synchronization decoupling is that of 'truly' non-blocking communication for interacting agents. The nonblocking communication allows agents to concurrently carry out their processing and network activities, not only while invoking any remote (tuple-reading/consuming) primitive, but also while receiving response from that invoked primitive. There are three aspects that directly affect non-blocking communication in interacting agents —

- use of reactions to relieve reference agent from getting suspended indefinitely while invoking any of the remote blocking primitives (viz. `rd`, `rdg`, `in` and `ing`);
- receiving responses of fired reactions asynchronously while performing some other concurrent activity; and
- use of `rdp`, `rdgp`, `inp` and `ingp` in place of `rd`, `rdg`, `in` and `ing` respectively as nonblocking (i.e. probe) remote invokes of tuple space operations.

Achieving these three aspects of nonblocking communication together result in complete synchronization decoupling.

Figure 3.2(a), Figure 3.2(b), Figure 3.3(a), Figure 3.3(b), Figure 3.4(a) and Figure 3.4(b) express various scenarios of nonblocking communication, required for complete synchronization decoupling. Figure 3.2(a) depicts the first two aspects of nonblocking communication. For each remote blocking invoke of `rd`, `rdg`, `in` or `ing` by reference agent A, corresponding nonblocking primitive (viz. `rdp`, `rdgp`, `inp` or `ingp`) is actually executed at destination tuple space of target agent B, as local invoke of any blocking primitive at B is deadlock-prone for it. If sought tuple(s) are not immediately available



(a) Case of remote blocking primitives.

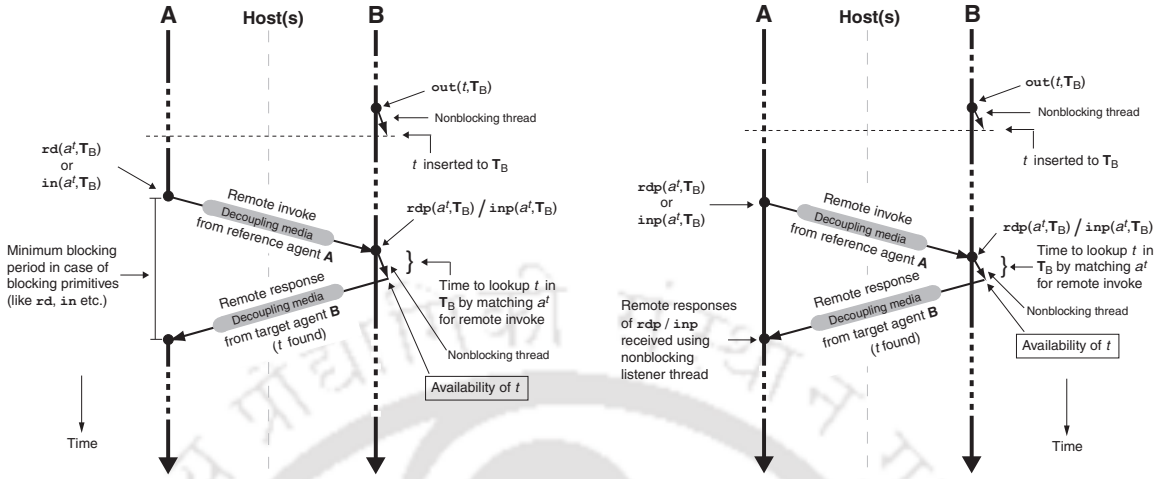
(b) Case of remote probe primitives.

Figure 3.2: Complete synchronization decoupling of remote tuple-reading/consuming primitives from A, when sought tuple is not available in B.

for such invoke, a reaction is registered at destination tuple space of B. When A actually invokes any such blocking primitive, it remains suspended till it receives sought tuple(s) or information of reaction registration, in case of absence of sought tuple(s)[1]. When any local/remote tuple-producing operation deposits the sought tuple(s) in destination tuple space of B, an active registered reaction fires due to the availability of sought tuple(s) and reactive codes execute at B to read/withdraw them. The output of execution of reactive codes becomes the response of fired reaction, which is dispatched back to A. Since the time of firing of reaction is uncertain, time-decoupled delivery of its response is necessary and is achieved by imposing the delivery of response through an additional tier of tuple spaces. They are represented combinedly as one 'grey channel' at host-level in Figure 3.2(a) – Figure 3.4(b).

Figure 3.2(b) depicts the scenario of nonblocking communication using invoke of `rdp`, `rdgp`, `inp` or `ingp` by A, instead of `rd`, `rdg`, `in` or `ing` for reading/withdrawing of sought

[1]Some period of suspension for reference agent while invoking remote blocking tuple-reading/consuming primitives (as shown in Figure 3.2(a) and Figure 3.3(a)) is inevitable, due to the inherent nature of these primitives.



(a) Case of remote blocking primitives.

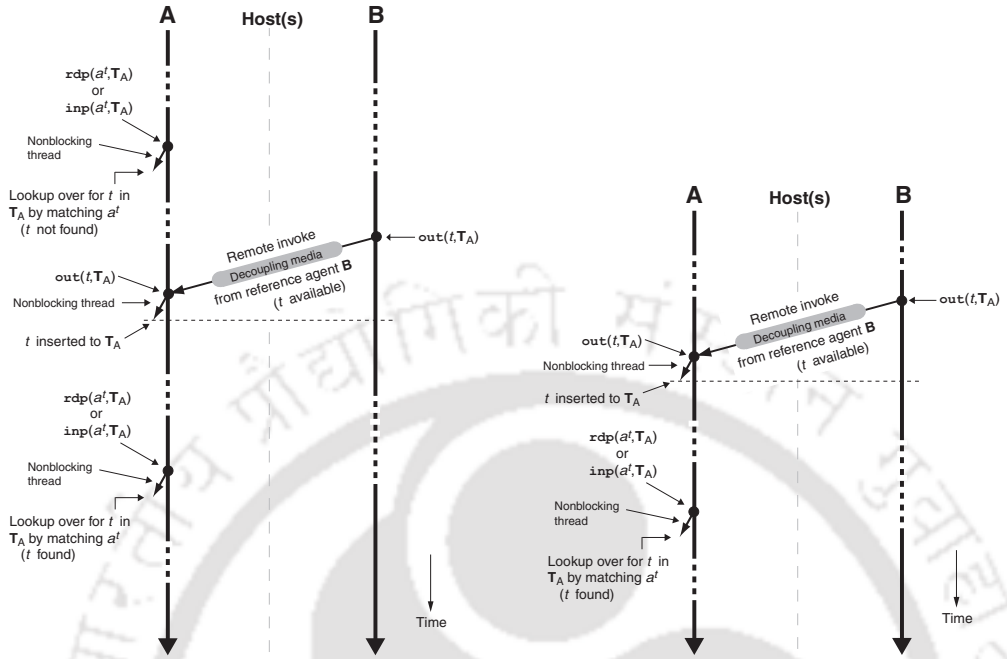
(b) Case of remote probe primitives.

Figure 3.3: Complete synchronization decoupling of remote tuple-reading/consuming primitives from A, when sought tuple is available in B.

tuple(s) in the previous scenario. As shown in this figure, A has to repeatedly invoke these probe primitives till it reads/withdraws sought tuple(s) from destination tuple space at B. Each of such invokes and responses are also time-decoupled by tuple spaces at respective host(s). Figure 3.3(a) and Figure 3.3(b) portray the scenarios of nonblocking communication in case of invokes of blocking and probe tuple-reading/consuming primitives respectively by A, where such invokes happen after sought tuple(s) are deposited in destination tuple space of B. As such, there is no need for generation and registration of reactions corresponding to invokes of blocking primitives in Figure 3.3(a). These invokes and their responses are also time-decoupled through host-level tuple spaces.

Figure 3.4(a) and Figure 3.4(b) exhibit the functioning of host-level tuple spaces in case of remote tuple-producing primitives (viz. `out` and `outg`). In these figures, B becomes reference agent, as it invokes remote `out` and `outg` to insert given tuple(s) in destination tuple space of target agent A. A may invoke any local probe tuple-reading/consuming primitive repeatedly till sought tuple(s) are inserted in its tuple space by B, when they are finally read/withdrawn.

Advantages of host-level decoupling media: The use of additional tier of tuple spaces as host-level decoupling media has multiple benefits:



(a) Case of periodic invokes till sought tuple is available.

(b) Case of invokes after sought tuple is available.

Figure 3.4: Complete synchronization decoupling of remote tuple-producing primitives from B, when local tuple-reading/consuming primitives are invoked in A.

1. primary benefit is to introduce time-decoupling of each reaction response (corresponding to each invoke of any remote tuple-reading/consuming primitive) from reference agent;

2. availability constraint of reference agent, while interacting with target agent(s) in mobile, dynamic and unreliable infrastructure, is relaxed;

3. network and processing overheads, due to repeated attempts of transmission of communicating information through such infrastructure, are reduced;

4. delivery of every invoke/response of tuple space primitive as well as reaction response to appropriate destination agents are assured; thus, these tuple spaces are used for achieving robustness in agent interactions in TSMM platform over dynamic environment;

5. these tuple spaces also time-decouple invokes of tuple space primitives by reference agent from target agent and their responses by target agent from reference agent. So,

interacting agents can concurrently carry out their processing and network activities while dealing with invoking/responding to tuple space operations.

The above benefits combinedly establish that the additional tier of host-level tuple spaces is effective in leading to "complete synchronization decoupling" of agent interactions in TSMM platforms. This can be demonstrated by designing a new TSMM platform that follows the tiered organization of tuple spaces to support the proposed notion of decoupling.

3.3 Design of $\mathcal{CUTSMuHN}$ as Tuple Space Coordination Platform

A new TSMM platform, named $\mathcal{CUTSMuHN}$, is designed to overcome the shortcomings in tuple space coordination of existing TSMM platforms. $\mathcal{CUTSMuHN}$ platform, in essence, comprises of multiple software processes, which are the instances of *hosts* or *agents*. A host is a runtime environment of a mobile/portable device deployed in dynamic environment. Each mobile/portable device houses a single host, which is uniquely identified by combining a predefined port with device identity (like, IP address, physical address etc.). Each host represents the unit of computation, containment and distribution. The host is designed as a layer lying immediately above 'operating system and related network services'. On the other hand, each agent corresponds to a user-defined component of mobile application and it represents the unit of modularity and mobility. One/more agents typically reside in each host, and they are layered above host. The host provides an operating environment for their execution. The mobile agents can also migrate from one host to another. $\mathcal{CUTSMuHN}$ platform integrates the tuple space model for supporting coordination. Every communicating information of an agent is coordinated through tuple spaces, distributed across the multi-hosted $\mathcal{CUTSMuHN}$ platform. It supports the proposed dimension of decoupling (presented in Section 3.2), thereby becoming the foremost TSMM platform to entirely time-decouple coordination activities of its supported applications. Rest of the existing dimensions of decoupling in tuple space coordination (mentioned in Section 2.2.1) are also supported as usual. Consequently, $\mathcal{CUTSMuHN}$ becomes a very loosely-coupled form of coordination platform, which not only facilitates

development of large-scale dynamic applications in unreliable environments, but also stimulates improvements of its robustness, versatility, simplicity, efficiency and scalability as well.

CUTSMuHN platform follows the agent-centric view of coordination, whereby each agent can selectively identify its relevant (in other words, friendly) agents from a list of its available neighbor agents for interaction. A minimalist set of tuple space primitives are supported by *CUTSMuHN* platform, which agents invoke to exchange their communicating information. Each such information is encapsulated as a tuple, whereas, search for any specific kind of tuple is accomplished by encapsulating a search key as an antituple. The tuple space primitives are provided as APIs to application programmers; more complicated operations can be derived using them. This also shows the expressiveness of *CUTSMuHN* platform in serving any degree of coordination needed by end-users. If a remote blocking primitive does not find sought tuple(s) for reading/withdrawal in a remote tuple space, synchronization decoupling is enforced by registering a reaction on that tuple space to carry out the search and retrieval of sought tuple(s), while reference agent is relieved from suspension. The whole remote operation, including firing of reaction, is entirely time-decoupled by installing a special tuple space at every host, which enforces complete synchronization decoupling. The invoke of remote operation is marshalled as *reaction tuple* (denoted hereon as reaction-tuple), whereas responses from execution of this operation is marshalled back as *response tuple* (denoted hereon as response-tuple). The host-level tuple spaces ephemerally store reaction-tuples and/or response-tuples on their way of transit till they are delivered to their respective destinations.

3.3.1 *CUTSMuHN* Architecture

CUTSMuHN platform consists of several components, each of which is specific to either agent or host. The architecture of *CUTSMuHN* platform, with its significant components, is shown in Figure 3.5. Each agent contains (i) an Agent Tuple Space (ATS) and its interface, (ii) a Local Operation Manager, (iii) a Remote Operation Manager, (iv) an ATS Reaction Manager, (v) an Engagement Manager etc. ATS is the local tuple space of an agent. Local Operation Manager handles the invokes of local primitives on its ATS,

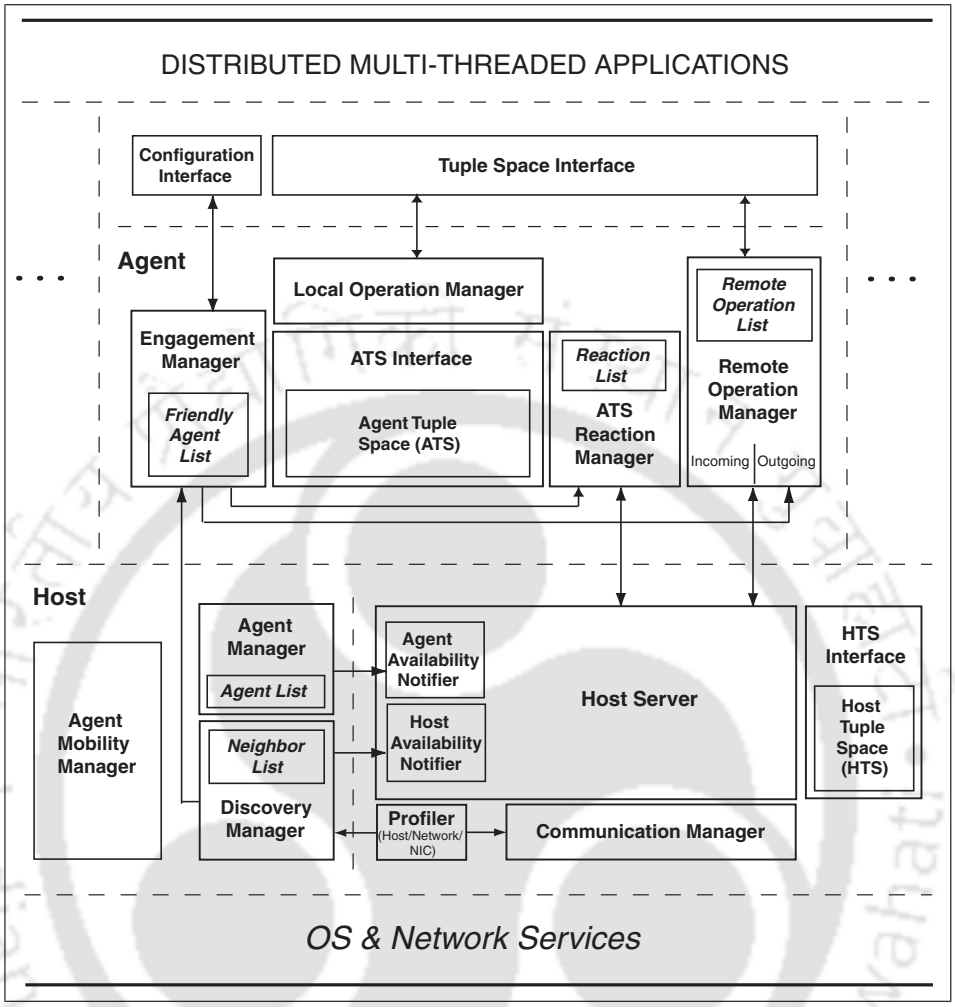


Figure 3.5: Architecture of $\mathcal{CUTSMuHN}$ platform, showing its significant components.

while Remote Operation Manager and ATS Reaction Manager jointly handle the invokes of remote primitives on ATS of other distant friendly agents. Engagement Manager enforces engagement policy to maintain the list of friendly agents. On the other hand, each host comprises of (i) a Host Server, (ii) a Host Tuple Space (HTS) and its interface, (iii) an Agent Manager, (iv) a Discovery Manager, (v) a Communication Manager, (vi) an Agent Migration Manager etc. Host Server decouples every agent interaction using its HTS. Agent Manager and Agent Migration Manager control registration/deregistration of agents and their logical mobility respectively. Discovery Manager collects neighborhood information, while Communication Manager transfers reaction/response-tuples from one host to another.

The architecture of *CUTSMuHN* platform presents a new type of tuple space distribution among hosts and agents, different from that of existing TSMM platforms. In this distribution, each agent holds a single tuple space (viz. ATS) for coordinating agent interactions, whereas each host holds another single tuple space (viz. HTS) to transfer coordinating information among the agents asynchronously. For time-decoupling of the invoke of reference agent from its invoked remote blocking primitive (which results in partial synchronization decoupling), reactivity is added to ATS, which is implemented by generating and registering reaction in ATS for every such invoke. The registered reactions monitor and respond to different types of events; one such event is the presence of a particular sought tuple in tuple space. For recognizing any relevant event, a reaction expects an antituple to specify some condition. If the condition gets satisfied, the corresponding reaction fires, which signifies that application-defined reactive code is executed subsequently (like reading/withdrawing new sought tuples from ATS), and responses are sent back to reference agent. In *CUTSMuHN* platform, a reaction is also allowed to specify its mode as ONCE or ONCE/TUPLE. Thus, a typical reaction comprises of (i) antituple, (ii) name of invoked primitive, (iii) reactive code, (iv) identity of ATS, (v) mode, and (vi) user identity. HTS is utilized as additional tier of decoupling medium to accomplish time-decoupling of reaction responses from reference agent. For achieving this dimension of decoupling, reaction responses are stored in HTS, till delivered to reference agent. The two types of special tuples, introduced to store and withdraw invokes and responses of remote tuple space operations in HTS, are reaction-tuple and response-tuple respectively. The reaction/response-tuples are unordered tuples, and so their arity as well as nature of their constituent fields vary with the nature of invoked remote primitives. Both the tuples are stored in HTS till delivery to their destinations. The two special primitives, `inject` and `eject`, are provided for managing operations (viz. storage and withdrawal) of reaction/response-tuples in HTS. The reaction-tuple, which is used for shipping the parameters of any remote primitive invoked by reference agent, is generated from the marshalled value of these parameters. A constituent of reference agent, viz. Remote Operation Manager, is responsible for shipping them to a different component of target agent, viz. ATS Reaction Manager.

The reaction-tuple is first inserted into HTS of reference host using `inject` primitive. On availability of target host, it is withdrawn from HTS of reference host using `eject`, passed as message payload through communication links to reach target host, and subsequently inserted into its HTS. Eventually, reaction-tuple is withdrawn from HTS of target host, while delivering to target agent. It is next processed to unmarshall the parameters of invoked primitive, and execution of invoked primitive starts at ATS of target agent. Once the execution of tuple-reading/consuming primitive or its corresponding reaction is over, result of execution (viz. sought tuple(s) from ATS of target agent and other necessary parameters) are marshalled into response-tuples. ATS Reaction Manager of target agent is responsible for initiating the shipment of response-tuples. Following the previous approach (i.e. via HTS of target and reference hosts successively), a response-tuple eventually reaches reference agent, and sought tuple(s) are extracted from it to complete the remote invoke. This entire procedure of remote invoke fulfils two separate dimension of decoupling, viz. synchronization decoupling during invoke phase and proposed time-decoupling in response phase, thereby attaining "complete synchronization decoupling" in agent interactions. The entire remote invoke process in *CUTSMuHN* architecture is portrayed using Figure 3.6.

However, since *CUTSMuHN* platform also allows *multi-destined* remote tuple space operations, additional consistency handling mechanism is a necessity, particularly with invoke of remote tuple-consuming primitives in dynamic environments.

3.3.2 Proposed Consistency Mechanisms

In *CUTSMuHN* platform, the proposed consistency mechanisms resolve both the OUT-consistency and IN-consistency problems, while performing different local/remote tuple-reading/consuming operations to support a wide variety of mobile applications. The proposed mechanisms ascertain consistency even when agent interactions have been entirely time-decoupled. The proposed mechanism to handle the OUT-consistency problem in coordination follows a new *three-way approach of agent interaction*, which includes an additional course of positive/negative acknowledgement schemes. However, these acknowledgement schemes operate at agent-level tuple spaces, which makes them radically

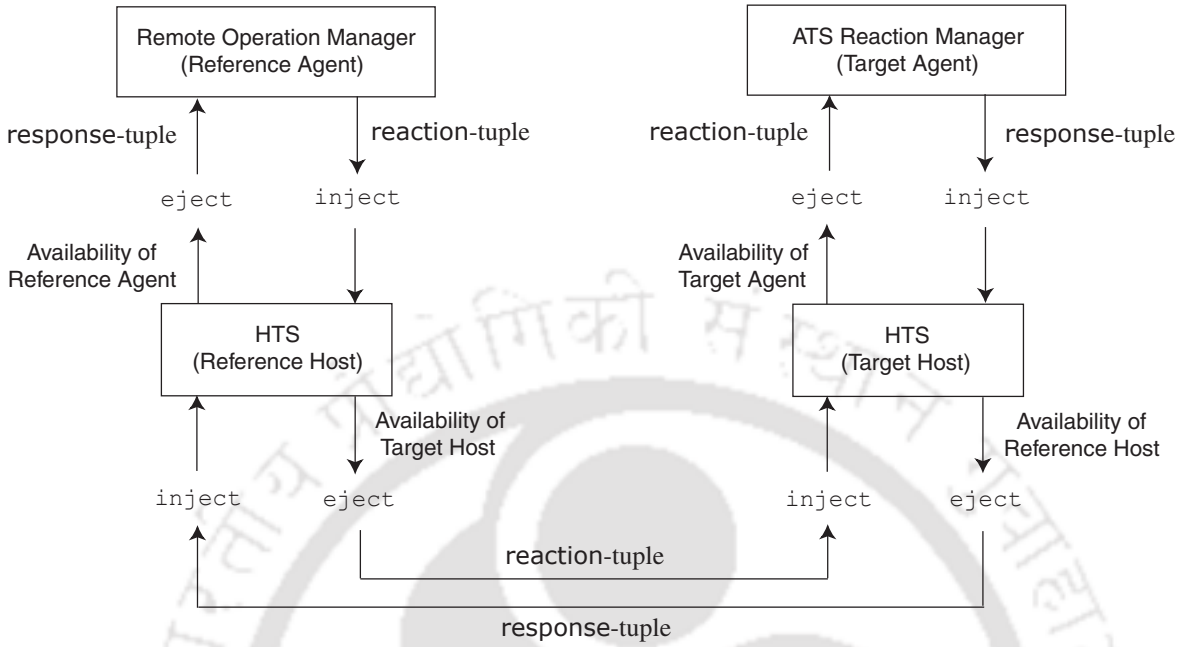
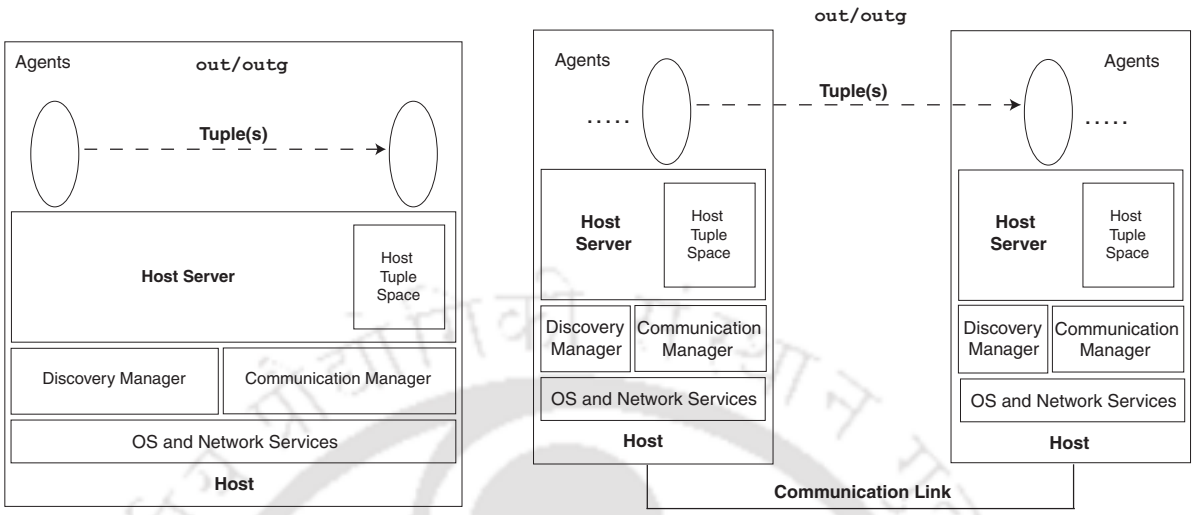


Figure 3.6: Entire invoke process of remote tuple space operations through additional host-level tuple spaces, that results in complete synchronization decoupling in *CUTSMuHN* platform.

different from conventional acknowledgement schemes that operate at network-level.

Figure 3.7(a), Figure 3.8(a) and Figure 3.9(a) show how a pair of interacting agents behave during tuple-producing, tuple-reading and tuple-consuming operations respectively in intra-host agent interactions in *CUTSMuHN* platform, whereas Figure 3.7(b), Figure 3.8(b) and Figure 3.9(b) show the same behavior in inter-host agent interactions. Figure 3.9(a) and Figure 3.9(b) enforce the proposed mechanism to resolve the OUT-consistency problem. It is noteworthy that tuple-producing and -reading operations are always OUT-consistent. In case of invoke of any single remote multi-destined tuple-reading/consuming primitive, the proposed mechanism works as follows —

- The given antituple of invoked primitive is distributed to the designated target agents to carry out the operations in their respective tuple spaces. In case of any positive match, the matched tuple is read (for tuple-reading operation) or marked as "reserved" for reference agent (in case of tuple-consuming operation) in its respective tuple space at target agent. In the latter case, a copy of the tuple is actually transmitted. It is to be noted here that such tuple is unavailable for lookup in any



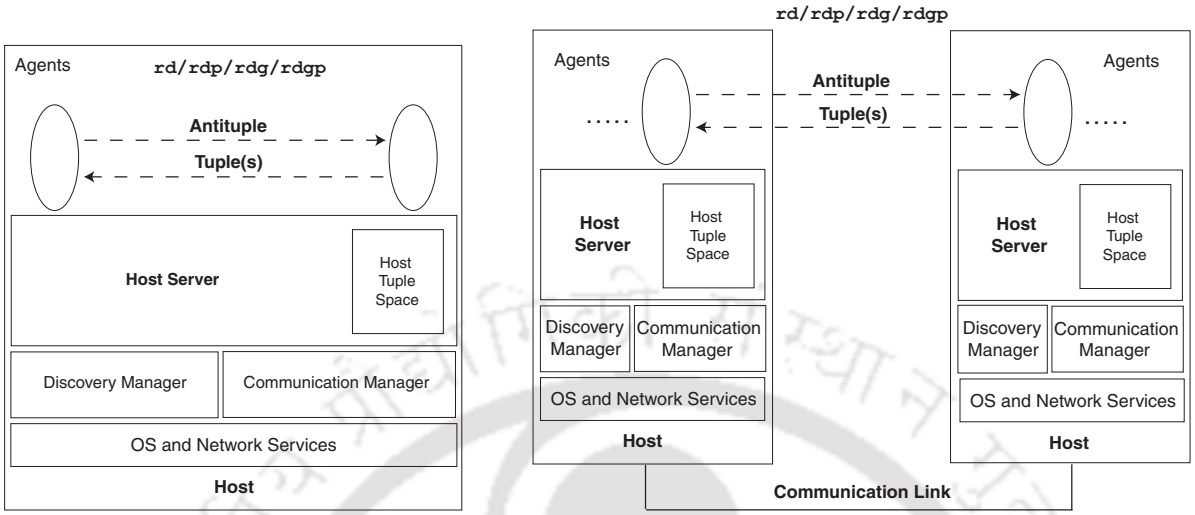
(a) In case of intra-host agent interaction.

(b) In case of inter-host agent interaction.

Figure 3.7: Behavior of remote tuple-producing operations, amidst proposed OUT-consistency mechanism, in CUTSMuHN platform.

further tuple space operations, till it is marked as reserved.

- Considering that the antituple matches more than one tuple from multiple tuple spaces of specified target agents, all the responses (i.e. positively-matched tuples) from these target agents are first preserved in a 'sought tuple set' at reference agent.
- Subsequently, a single sought tuple is nondeterministically chosen from that set.
- Once a sought tuple is chosen, however, single tuple-reading primitive and single tuple-consuming primitive behave differently:
  1. In case of a tuple-reading primitive, all remaining (i.e. non-selected) tuples of the 'sought tuple set' are subsequently discarded by reference agent;
  2. For a single tuple-consuming primitive, reference agent sends back positive/ negative acknowledgements to each target agent, whoever has responded. Positive acknowledgement (in the form of ack-tuple) is sent to that target agent, whose tuple is chosen as sought tuple, whereas negative acknowledgements (in the form of nack-tuples) are sent to other target agents who have responded. Each positive/negative acknowledgement carries with it the identity of the corresponding tuple, sent by respective target agent. Based on the tuple's identity,



(a) In case of intra-host agent interaction.

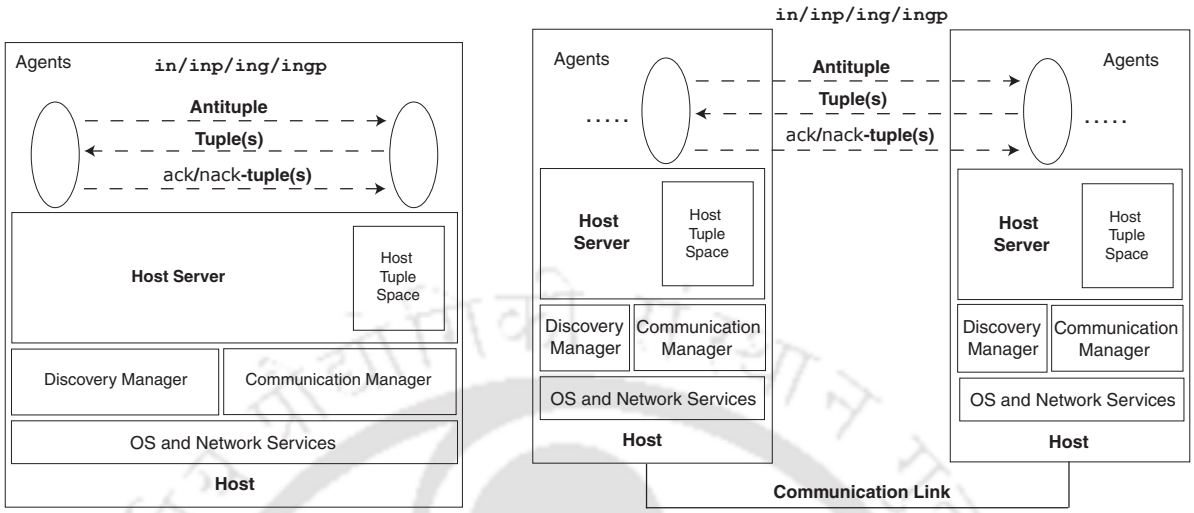
(b) In case of inter-host agent interaction.

Figure 3.8: Behavior of remote tuple-reading operations, following proposed OUT-consistency mechanism, in CUTSMuHN platform.

when ack-tuple arrives informing the tuple's selection, owning target agent actually withdraws the reserved tuple from its tuple space and discards it before carrying out its next operation. When nack-tuple arrives informing the tuple's non-selection, it is unmarked (i.e. "reserved" label is removed) and made available for lookup in subsequent tuple space operations.

So, the proposed mechanism becomes a two-way approach during remote multi-destined tuple-reading operation, whereas it is a three-way approach for tuple-consuming operation. Remote Operation Manager of reference agent is responsible for ensuring OUT-consistency, once it initiates the remote tuple space operation that reference agent intends to execute on target agent(s). At the target agent, its ATS Reaction Manager works together with Remote Operation Manager of reference agent to resolve the OUT-consistency problem.

A target agent, on receiving a nack-tuple, extracts the identity of its responded tuple (which is not selected by reference agent), and removes its "reserved" label in its own ATS. However, a target agent, which receives ack-tuple carrying identity of the chosen tuple, needs to remove that tuple from its ATS. Effectively, that target agent actually



(a) In case of intra-host agent interaction.

(b) In case of inter-host agent interaction.

Figure 3.9: Behavior of remote tuple-consuming operations, amidst proposed OUT-consistency mechanism, in CUTSMuHN platform.

performs the withdrawal operation. For bulk tuple-reading/consuming primitives, all responses from specified target agents are kept as sought tuples, and particularly for bulk tuple-consuming primitives, only ack-tuples are sent back to target agents, whoever have responded to complete the withdrawal operations. It is noteworthy that Figure 3.9(a) and Figure 3.9(b) show the three courses of proposed three-way mechanism (as dashed arrows) to handle the OUT-consistency problem. The ack-tuples and nack-tuples are considered as special system tuples, and they are also converted to response-tuples accordingly while being handed over to Host Server for shipment from one agent to another.

The proposed mechanism to resolve the IN-consistency problem in CUTSMuHN platform follows a relatively simpler approach. Since the IN-consistency problem is restricted to a particular ATS, the proposed mechanism imposes a *predefined ordering* scheme in that tuple space, and enforces each operation to happen in a *strict/exclusive* way of processing. The predefined ordering in ATS implies that execution of all tuple space primitives, including pending reactions of blocked tuple-reading/consuming primitives, are scheduled in a particular (predefined) order (say, first-in-first-out or FIFO order). The strict/exclusive way of accessing ATS is illustrated in Figure 3.10, whereby access of ATS is required to be acquired and released for each operation. In CUTSMuHN plat-

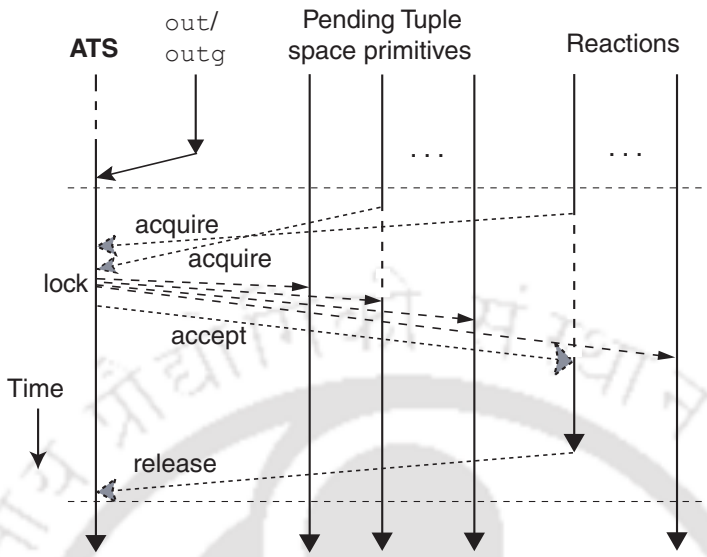


Figure 3.10: Strict/exclusive processing of tuple space operations to resolve IN-consistency problem in CUTSMuHN platform.

form, while inserting a particular tuple in ATS, a tuple-producing primitive first acquires access of that ATS, inserts given tuple, and subsequently releases that access. Once tuple insertion is over, all registered reactions are served one-by-one (according to the predefined order) by following the same approach of acquiring and releasing access of ATS. In each such occasion of accessing tuple space, tuple space is first locked, and subsequently insertion/reading/withdrawal operation is performed to completion before releasing the lock. Both the predefined ordering and strict/exclusive way of accessing ATS together resolves the IN-consistency problem in CUTSMuHN platform. The proposed IN-consistency mechanism is integrated within each tuple space primitive and also within each registered reaction.

Although the proposed mechanisms of handling consistency problems in CUTSMuHN platform during multi-destined tuple space operations improve its nature of coordination, performances of individual tuple space operations also significantly affect coordination quality. The next section focuses on factors of tuple space coordination that impart performance improvement of tuple space operations.

3.3.3 Proposed Structuring of Tuples, Antituples and Tuple Spaces

The objective of proposing modifications to existing structures of tuples, antituples and tuple spaces is to improve the overall performances of CUTSMuHN platform. The proposed enhancements on existing tuple/antituple structure reduce the execution overheads of tuple space primitives, thereby improving efficiency as well as scalability of CUTSMuHN platform, without sacrificing its simplicity and versatility in supporting a wide spectrum of applications. Additionally, the proposed tuple space structure further improves its efficiency and scalability, without compromising its achieved simplicity and versatility.

Proposed Enhancements in Tuple/Antituple Structure

The objective in enhancing the existing unordered tuple/antituple structure is to design an unordered tuple/antituple structure of CUTSMuHN platform, whose temporal and spatial complexities are close to ordered tuple/antituple structure, and yet which is scalable and versatile in supporting a variety of applications. Several notable characteristics of the tuple space model motivate these proposals. Foremost, a tuple is generated once in its lifetime and stored in a tuple space, where it is looked up multiple times and finally withdrawn at the end of its lifetime. Moreover, a tuple is never modified while residing inside a tuple space[2]. So, in the performance of CUTSMuHN platform, significance of lookup time of a tuple is more than its generation time, due to the possibility of multiple searches in its lifetime. In fact, a highly inexpensive tuple lookup time can be afforded against a comparatively expensive tuple generation time. Furthermore, an antituple is a form of search key, whose lifetime ends with the matching operation for which it is created. It is never stored in tuple spaces, but distributed to different locations for matching purpose. So, an antituple's size is a noteworthy factor, which impacts network overhead during coordination. The proposed enhancements for unordered tuple/antituple structure are as follows.

(i) The first proposal enhances the substructure of a tuple field, where each tuple field comprises of only one component, viz. VALUE, from which other components, viz. NAME

[2]A tuple is immutable. For updating/modifying a tuple, it is first withdrawn from tuple space and changed subsequently, thereby giving rise to a new tuple

and TYPE, are extracted as required. This proposal requires support of the object-oriented paradigm.

(ii) The second proposal uses the *cuckoo hashing* mechanism [90] for handling generation and lookup of tuples, as cuckoo hashing provides faster lookup time. It uses two hash tables to store the constituent tuple fields. Each tuple field is inserted in anyone of the two tables. The two hash functions, selected randomly from a set of universal hash functions **H**, are bound to the two tables.

Thus, the proposed tuple structure is represented by $\{\mathtt{HT}_1, \mathtt{HT}_2, \mathfrak{E}\}$, where $\mathtt{HT}_1$ and $\mathtt{HT}_2$ are two hash tables of same length L (where, $L = \hat{n} + 1$, $\hat{n}$ being tuple arity), and $\mathfrak{E}$ corresponds to extra information required in a tuple's lifetime, including the two hash functions used (viz. $\mathtt{h}_1$ and $\mathtt{h}_2$ for $\mathtt{HT}_1$ and $\mathtt{HT}_2$ respectively) and some middleware-specific data. Each of $\mathtt{HT}_1$ and $\mathtt{HT}_2$ initially contains L empty entries; as tuple generation continues, the insertion method is invoked iteratively for inserting each generated tuple field in one of these entries. During insertion, $\mathtt{h}_1$ or $\mathtt{h}_2$ determines the hash index for $\mathtt{HT}_1$ or $\mathtt{HT}_2$ respectively based on the NAME of that field. In the proposed structure, an additional step is included in the insertion method, where a tuple field is first attempted to be directly inserted in $\mathtt{HT}_2$, bypassing cuckoo hashing. In this attempt, the entry of $\mathtt{HT}_2$ is determined by hashing that field using $\mathtt{h}_2$. If this insertion fails due to the presence of some existing field in that entry, then cuckoo hashing is initiated. The advantage of this modification is to restore a balance in distribution of tuple fields between $\mathtt{HT}_1$ and $\mathtt{HT}_2$, as cuckoo hashing is biased towards $\mathtt{HT}_1$. In the proposed structure, an antituple field has two components, viz. NAME and SEARCHVALUE; TYPE is extracted from NAME with support of the object-oriented paradigm. These two components are included in an antituple field to enable tuple-antituple matching operation by *actual* and *formal*. Thus, the proposed antituple structure is represented by $\{<N_i, SV_i>\}$, where $(1 \leq i \leq n')$, n' is antituple's arity and $<N_i, SV_i>$ is an antituple field having some search constraint. All such fields are simply stored consecutively inside antituple.

The tuple-antituple matching algorithm, for the proposed tuple/antituple structure, takes an antituple as argument, and compares each antituple field with its apposite paired tuple field to decide a match. For pairing, the lookup method of cuckoo hashing

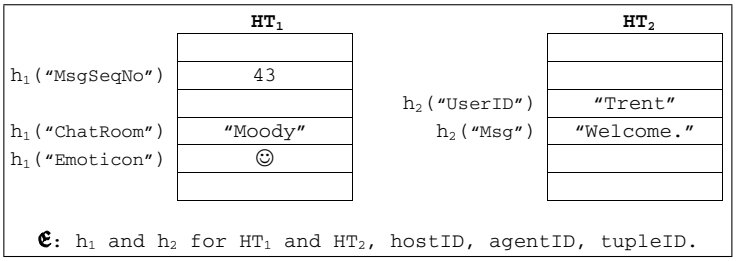


Figure 3.11: Example of proposed tuple structure of $\mathcal{CUTSMuHN}$ platform.

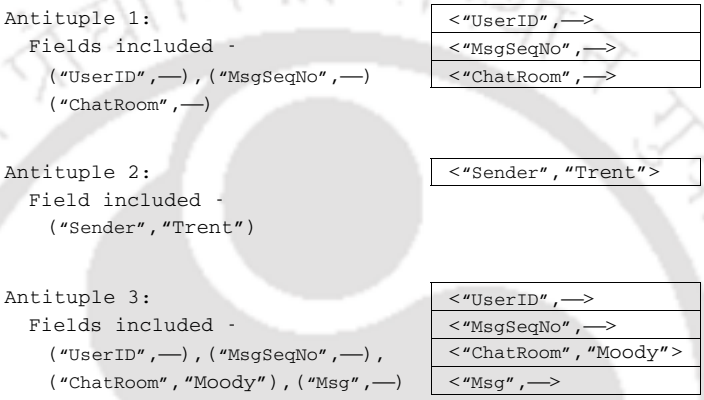


Figure 3.12: Examples of proposed antituple structure of $\mathcal{CUTSMuHN}$ platform.

is invoked. The following example gives a better view of the proposed tuple/antituple structure. Consider that a user wants to generate a tuple, comprising of the following five fields: {("UserID", String, "Trent"), ("ChatRoom", String, "Moody"), ("MsgSeqNo", Long, 43), ("Msg", String, "Welcome."), ("Emoticon", Image, ☺)}. Here, ☺ denotes a binary representation (like, an image file) of some emoticon. In the proposed tuple structure, these fields become: {"Trent", "Moody", 43, "Welcome.", ☺} (note that this particular tuple corresponds to index 3 of the tuple space in Figure 2.2). Once these fields are inserted in $\mathtt{HT}_1$ and $\mathtt{HT}_2$, it has the structure as in Figure 3.11. For searching that tuple, let another user creates several antituples of variable arity, shown in Figure 3.12. From these figures, it is clear that antituple 1 and antituple 3 successfully lookup the tuple shown in Figure 3.11, while antituple 2 cannot, as the field with NAME "Sender", which is present in antituple 2, is not present in that tuple.

The benefits of the proposed tuple/antituple structure can be fully exploited if tuple

space structure has also been simultaneously enhanced.

Proposed Tuple Space Structure

The objective of restructuring a tuple space is to place tuples efficiently within a tuple space so as to lookup apposite tuples quickly, thereby expediting tuple space operations in CUISMuHN platform. This issue becomes significant with the unordered tuple/antituple structure, as, unlike the ordered structure, not all tuples of a tuple space are apposite to an antituple. As such, there is a good reason to identify those tuples that are by no means apposite and are bound to mismatch during tuple-antituple matching operation.

In CUISMuHN platform, an indexed tuple space structure is proposed that incorporates a modified form of 'array subscripting' to identify apposite tuples for a given antituple. The key idea of the proposed tuple space structure is to index all tuples of a tuple space based on their constituent field types. For this reason, a tuple space is partitioned into two regions, viz. Preamble and Tuple Store. Preamble holds all index tables corresponding to different constituent fields of all tuples present in tuple space, while Tuple Store is the actual storehouse of these tuples. Each index table is a list of indices of those tuples in Tuple Store, each of which contains at least one constituent field having NAME or TYPE identical/polymorphically-related to the index table name.

An index table is accessed either during tuple-antituple matching operation or while withdrawing one/more sought tuples. For speeding up the access of dynamic set of index tables, any hashing mechanism that handles dynamic (*key*, *element*) set can be applied. The meta-data of all index tables are maintained in another table in Preamble, called Meta index table. Meta index table can also use hashing, with name of the index table as key, to improve its lookup process; it is also capable of dynamic resizing to cope up with the changing set of index tables. Moreover, for faster access of tuple indices within an index table for reading/withdrawal purposes, each index in the index table can be hashed, based on its value as key, using similar hashing mechanism. The index tables are also capable of dynamic resizing to manage the changing set of indices. Furthermore, free spaces, resulting due to withdrawn tuples from Tuple Store and removed indices from index tables, are handled by maintaining another table in Preamble, called Free List. It

enlists all empty indices of Tuple Store. While inserting a new tuple, the Free List is looked up for an empty index of the Tuple Store. Also, any index table can become empty, once all tuple indices are removed from it. A periodic garbage collection mechanism is, thus, employed to eventually remove the empty index tables from the Preamble.

For an easy understanding of the proposed tuple space structure, the earlier example of chat applications is continued. Figure 3.13 shows the same snapshot of tuples in the proposed tuple space structure, as shown in Figure 2.2 for existing structure (note that tuples are projected here as a series of *actual*, for better understanding; actually, each of them has a structure as shown in Figure 3.11). Any single/bulk tuple-reading/consuming primitive, using the antituple {("UserID", —), ("MsgSeqNo", —)}, matches the tuples present in the set of apposite tuples (viz. tuples at positions 0, 1, 3, 4, 7, 9 and 10 in Figure 3.13), and accordingly returns single/multiple positively-matched tuples as outcome. In this particular case, all apposite tuples match positively. From these positively-matched tuples, in case of single primitives, one sought tuple is chosen nondeterministically, as defined by [50]. Similarly, the tuples at positions 2, 5 and 8 are compared for the antituple {("Sender", —), ("Receiver", —)}. If one sought tuple is desired, it is subsequently chosen nondeterministically from them. Consequently, the proposed tuple space structure not only follows the original behaviour of the tuple space model, but also deals with selective tuple(s) of a tuple space that are apposite to given antituple.

The next section presents the semantics of supported tuple space primitives that cope up with the proposed tuple/antituple and tuple space structures of *CUTSMuHN* platform.

3.3.4 Tuple Space Primitives of *CUTSMuHN*

The semantics of different tuple space primitives of *CUTSMuHN* platform are defined as follows.

- `out`: Figure 3.14 presents the semantics of `out` primitive. The `out` primitive invokes the `store` method, shown in Figure 3.15, to look up an empty index of Tuple Store for inserting the given tuple. In case the Free List is empty, given tuple is inserted at the

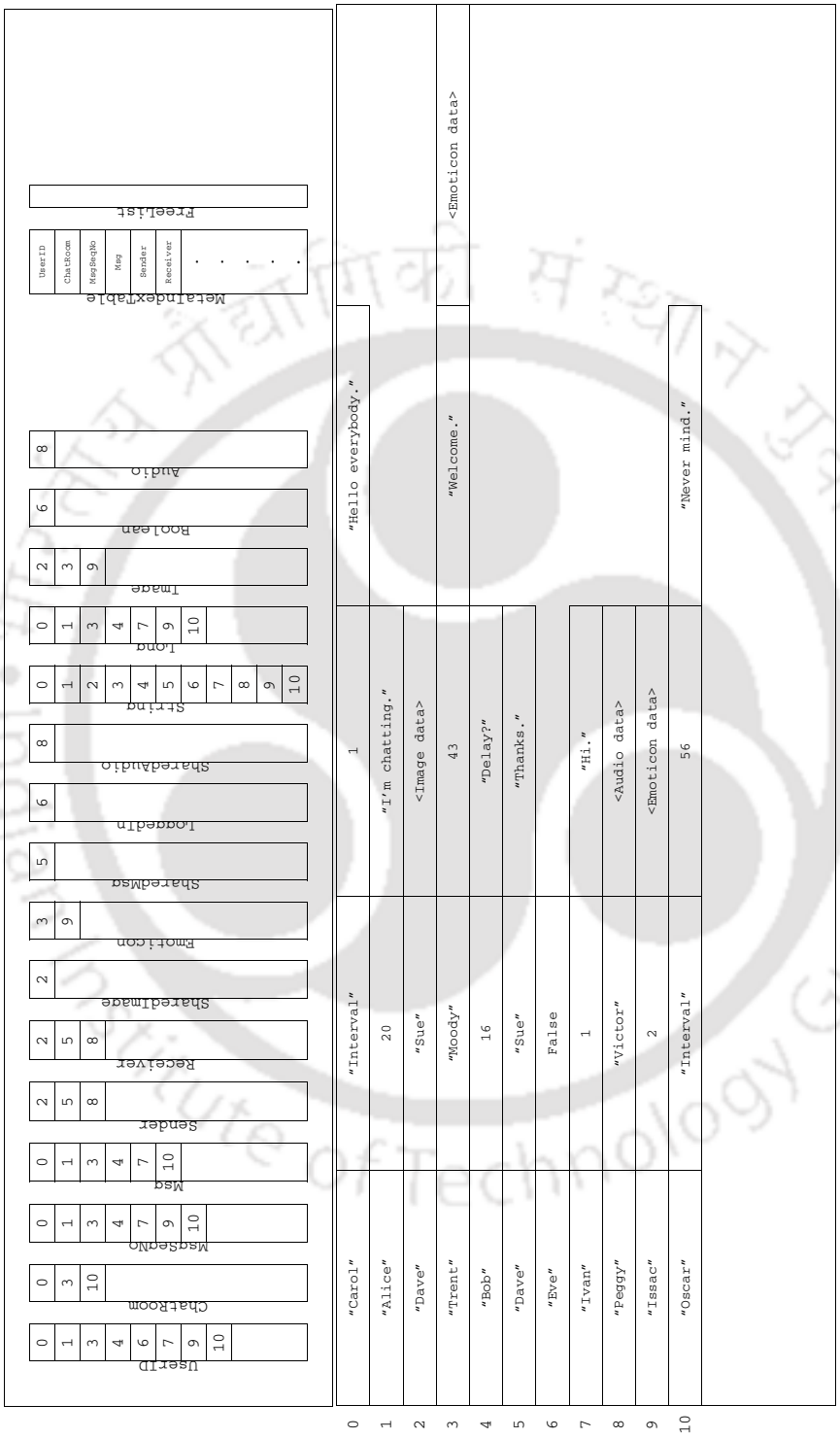


Figure 3.13: A snapshot of proposed tuple space structure of *CUTSMuHN* platform, showing tuples of chat applications.

```
Input: Tuple t, tuple space T.
Output: Tuple space T.
begin
    Call store(t), and return result to ind.
    For each tuple field <VALUE> do
        TypeVal ← name(VALUE).
        If an index table of name matching TypeVal is not present in Meta index table M then
            Create a new index table of name TypeVal in T^p and enter its meta-data in M.
        Hash ind in index table TypeVal.
        PolymorphicTypesVal ← type(VALUE).
        For each type Typename present in list PolymorphicTypesVal do
            If an index table of name matching Typename is not present in M then
                Create a new index table of name Typename in T^p and enter its meta-data in M.
            Hash ind in index table Typename.
        end
    end
    Return T, containing given tuple t.
end
```

Figure 3.14: Primitive `out` of $\mathcal{CUTSMuHN}$ platform.

```
store(t)
Input: Tuple t.
Output: Index ind' of tuple space entry storing t.
  begin
      If Free List F is not empty then
          ind' ← extract a free index from F.
      Else
          ind' ← index value to store at the rear end of T^s.
      Insert t in T^s at index ind'.
      Return ind'.
  end
```

Figure 3.15: Method `store`, invoked by `out` primitive.

rear end of Tuple Store. This is followed by hashing of the index of inserted tuple in all related index tables, viz. tables corresponding to names, types and possible subtypes (i.e. polymorphic types) of each constituent field of that tuple. The method `name`, invoked by `out`, returns the name and type of given *actual*; other tuple space primitives of $\mathcal{CUTSMuHN}$ platform has also used this method. Similarly, the method `type` returns possible subtypes of given *actual*.

- `outg`: The semantics of `outg` primitive is simply the repetitive calls of `out` primitive to store each tuple that is present in the input set of tuples. The number of repetitions is same as arity of this input set.
- `rdp`: Figure 3.16 presents the semantics of `rdp` primitive. It invokes the `lookupIndTab` method, shown in Figure 3.17, to find out all positively-matched tuples from the specified index table. The `lookupIndTab` method scans the selective tuple indices of that table, which are apposite, for comparison with given antituple. This method, in turn, calls the

Input: Antituple a, tuple space **T**.
Output: Sought tuple t or NULL tuple ε.

begin
  If arity of $a = 0$ **then**
    Choose a tuple t from $\mathbf{T}^s$ nondeterministically, and return t as sought tuple.
  $\mathbf{S} \leftarrow \emptyset$.
  Choose a field <NAME, SEARCHVALUE> from a.
  Call **lookupIndTab**(NAME, a), and return result to *SoughtTuples*.
  $\mathbf{S} \leftarrow \mathbf{S} \bigcup$ *SoughtTuples*.
  PolymorphicTypes $\leftarrow$ **type**(NAME).
  For each type *Typename* present in list *PolymorphicTypes* **do**
    Call **lookupIndTab**(*Typename*, a), and return result to *SoughtTuples*.
    $\mathbf{S} \leftarrow \mathbf{S} \bigcup$ *SoughtTuples*.
  end
  If $\mathbf{S} \neq \emptyset$ **then**
    Choose a tuple t from **S** nondeterministically, and return t as sought tuple.
  Else
    Return ε as sought tuple.
end

Figure 3.16: Primitive `rdp` of *CUTSMuHN* platform.

`lookup` method, shown in Figure 3.18, to determine a positive match. If the specified index table is not present in Preamble, then the `lookupIndTab` method returns an empty set of matched tuples, as no tuples of tuple space positively match given antituple.

- `rdgp`: Figure 3.19 presents the semantics of `rdgp` primitive. It is similar to the `rdp` primitive of Figure 3.16, except that all sought tuples found inside a tuple space are returned, instead of nondeterministically choosing one of the sought tuples.
- `rd` and `rdg`: These two primitives have the semantics similar to that of `rdp` and `rdgp` primitives respectively, except that when the set of sought tuples are empty, `rd` and `rdg` primitives register reactions in tuple spaces to return one/more non-NULL sought tuples, as and when such tuples are available in that tuple space.
- `inp`: The semantics of `inp` primitive is presented in Figure 3.20. Its semantics is similar to the `rdp` primitive except the withdrawal part, for which two additional steps are included in the `inp` primitive. The method `updateIndTab`, shown in Figure 3.21, is also included for same purpose. This method removes the index of each withdrawn tuple from related index tables, and includes that index in the Free List to mark it as empty.
- `ingp`: Figure 3.22 presents the semantics of `ingp` primitive. It is similar to that of `inp` primitive, except that all sought tuples found inside tuple space are withdrawn and returned, instead of a nondeterministically chosen sought tuple. Also, for each withdrawn tuple, its index is removed from related index tables in Preamble of tuple space, and include that index in the Free List.

```
lookupIndTab(TypeName, a)
Input: Type TypeName, antituple a.
Output: Set of sought tuples SotTuples.
  begin
      SotTuples ← ∅.
      If an index table of name matching TypeName is not present in Meta index table M then
          Return SotTuples.
      While not all indices stored in index table TypeName have been read do
          flag ← success.
          ind ← read next index from TypeName.
          t ← T^s_ind.
          While flag = success do
              Choose a field <FIELDNAME, SEARCHVALUE> from a that is not already chosen.
              Call lookup(FIELDNAME, t), and return result to flag and VALUE.
              If flag = success then
                  Apply matching conditions on VALUE and SEARCHVALUE,
                      and return result (success or failure) to flag.
          end
          If flag = success then
              SotTuples ← SotTuples ⋃ {t, ind}.
      end
      Return SotTuples.
  end
```

Figure 3.17: Method `lookupIndTab`, invoked by `rdp` primitive.

```
lookup(NAME, t)
Input: Type NAME, tuple t.
Output: Boolean flag, tuple field actual VALUE or NULL.
  begin
      key ← NAME.
      Obtain HT1, HT2, h1, and h2 from t.
      If key = name(HT1[h1(key)]) then
          Return success and HT1[h1(key)].
      Elseif key = name(HT2[h2(key)]) then
          Return success and HT2[h2(key)].
      Else
          Return failure and NULL.
  end
```

Figure 3.18: Method `lookup`, invoked by `lookupIndTab` method.

- `in` and `ing`: The semantics of these two primitives are similar to that of `inp` and `ingp` primitives respectively, except that in the case of empty set of sought tuples, `in` and `ing` primitives also register reactions in tuple space so that some non-NULL sought tuples are eventually withdrawn and returned, once such tuples become available in tuple spaces.

3.4 Analysis of Tuple Space Coordination of $\mathcal{CUTSMuHN}$

The time complexity analyses of tuple space primitives of $\mathcal{CUTSMuHN}$ platform are required to manifest efficiency of its tuple space model in comparison to existing TSMM platforms. The different symbols used in these analyses are listed in Table 3.1. It is to

Input: Antituple *a*, tuple space **T**.
Output: Set of sought tuples **S** or NULL tuple ε.

begin
  If arity of *a* = 0 **then**
    S ← **T**^{s}; return **S** as the set of multiple sought tuples.
  S ← $\emptyset$.
  Choose a field <NAME, SEARCHVALUE> from *a*.
  Call **lookupIndTab**(NAME, *a*), and return result to *SoughtTuples*.
  S ← **S** $\bigcup$ *SoughtTuples*.
  PolymorphicTypes ← **type**(NAME).
  For each type *Typename* present in list *PolymorphicTypes* **do**
    Call **lookupIndTab**(*Typename*, *a*), and return result to *SoughtTuples*.
    S ← **S** $\bigcup$ *SoughtTuples*.
  end
  If S $\neq \emptyset$ **then**
    Return **S** as the set of multiple sought tuples.
  Else
    Return ε as sought tuple.
end

Figure 3.19: Primitive `rdgp` of *CUTSMuHN* platform.

Table 3.1: Symbols used in analyzing tuple space model of *CUTSMuHN* platform.

Symbols	Meaning
m	Number of tuples present inside tuple space
$\hat{n}$	Average tuple arity
α	Load factor of 'Meta index table'; due to process of resizing value of α is limited to 1, i.e. $0 < \alpha < 1$
ρ	Average degree of subtyping for a tuple; typical values of ρ range from 1 to 5
m'	Number of input tuples (in case of `outg` primitive)
n'	Arity of input antituple (for tuple-reading/consuming primitives)
$\bar{m}$	Number of positively-matched tuples in any tuple-reading/consuming operation
ι	Number of entries of any particular 'index table' that is looked upon

be noted here that in these analyses, any form of 'c' is used to define a constant, viz. c_1, c_2, ..., c_{20}, $\acute{c}$, $\grave{c}$, $\bar{c}$, c_{S}, c_{H}, c_{T}, c_{L}, c_{R}, c'' etc. represent some form of constants.

3.4.1 Time Complexities of `out` & `outg` of *CUTSMuHN*

The time complexity analysis of `out` primitive of *CUTSMuHN* platform is based on its algorithm, which is shown in Figure 3.14. The symbols involved in this analysis, viz. $\hat{n}$, α and ρ, are already declared in Table 3.1. The first statement of `out` primitive calls the `store` method, which is shown in Figure 3.15. As visible in this figure, the "If-Else"

Input: Antituple *a*, tuple space **T**.
Output: Sought tuple *t* or NULL tuple ε.

begin
  If arity of *a* = 0 **then**
    Choose a tuple *t* from $\mathbf{T}^{\mathbf{S}}$ nondeterministically, and keep its index in *ind*.
    Call **updateIndTab**(*t*, *ind*) to remove *ind* from related index tables.
    Store *ind* as free index in Free List $\mathcal{F}$.
    Return *t* as sought tuple.
  S $\leftarrow \emptyset$.
  Choose a field <NAME, SEARCHVALUE> from *a*.
  Call **lookupIndTab**(NAME, *a*), and return result to *SoughtTuples*.
  S $\leftarrow$ **S** $\bigcup$ *SoughtTuples*.
  PolymorphicTypes $\leftarrow$ **type**(NAME).
  For each type *Typename* present in list *PolymorphicTypes* **do**
    Call **lookupIndTab**(*Typename*, *a*), and return result to *SoughtTuples*.
    S $\leftarrow$ **S** $\bigcup$ *SoughtTuples*.
  end
  If S $\neq \emptyset$ **then**
    Choose a tuple *t* from **S** nondeterministically, and keep its index in *ind*.
    Call **updateIndTab**(*t*, *ind*) to remove *ind* from related index tables.
    Store *ind* as free index in $\mathcal{F}$.
    Return *t* as sought tuple.
  Else
    Return ε as sought tuple.
end

Figure 3.20: Primitive `inp` of *CUISMuHN* platform.

block, and tuple insertion as well as index return of the `store` method take constant amounts of execution time, which result in a constant execution time for the `store` method itself (say, c_{S}). So, the first statement of `out` primitive takes constant time c_{S}. Next in the algorithm of `out` is the "For" block that repeats for $\hat{n}$ times to assess each case of constituent tuple field. The first statement inside the "For" block takes constant time (say, c_1). The second statement is a "If" block, whose condition is evaluated in $(c_{\text{H}} + \bar{c}\alpha)$ amount of time. This is due to the fact that while evaluating the condition of "If", hashing is first required (that is assumed to be over by constant time c_{H}) and then lookup is followed in Meta index table (which is assumed to take some time $\bar{c}\alpha$, where $\bar{c}$ is a constant). The consequent (i.e. truth) statement of this "If" block takes constant execution time of $(c_{\text{T}} + c_{\text{H}})$, where c_{T} is the time to create a new index table in Preamble. The third and fourth statements of "For" block also take constant time (say, c_{H} and c_2 respectively). Lastly, there ia a nested "For" loop that repeats ρ times to handle each subtype of the indicated value. The first statement inside the inner "For" loop is again a "If" block, which is similar to the "If" block of outer "For" loop. So, it also takes $(c_{\text{H}} + \bar{c}\alpha)$ time to evaluate the condition and $(c_{\text{T}} + c_{\text{H}})$ time to execute the consequent statement. The last statement in the inner "For" loop takes constant time c_{H}. Finally,

```
updateIndTab(t, ind)
Input: Tuple t, index of tuple space ind.
Output: Void.
  begin
      For each tuple field <VALUE> of t do
          TypeVal ← name(VALUE).
          If an index table of name matching TypeVal is present in Meta index table M then
              Remove ind from index table TypeVal.
              If index table TypeVal becomes empty then
                  Remove table TypeVal from T^p.
          PolymorphicTypesVal ← type(VALUE).
          For each type Typename present in the list PolymorphicTypesVal do
              If an index table of name matching Typename is present in M then
                  Remove ind from index table Typename.
                  If index table Typename becomes empty then
                      Remove table Typename from T^p.
          end
      end
  end
```

Figure 3.21: Method `updateIndTab`, invoked by `inp` primitive.

the last statement of `out` primitive takes constant time c_3. Collectively, the execution time of `out` primitive, after simplification, becomes $O(\hat{n})$, which is shown in the following.

$$
\begin{aligned}
& c_{\text{S}} + \hat{n}\Big(c_1 + (c_{\text{H}} + \bar{c}\alpha) + (c_{\text{T}} + c_{\text{H}}) + c_{\text{H}} + c_2 + \rho\big((c_{\text{H}} + \bar{c}\alpha) + (c_{\text{T}} + c_{\text{H}}) + c_{\text{H}}\big)\Big) + c_3 \\
&= k_1\hat{n} + \bar{c}\alpha\hat{n} + k_2\rho\hat{n} + \bar{c}\alpha\rho\hat{n} + c_{\text{S}} + c_3 \\
&\qquad \text{where } k_1 = (c_1 + c_2 + 3c_{\text{H}} + c_{\text{T}}) \text{ and } k_2 = (3c_{\text{H}} + c_{\text{T}}) \\
&= O(\hat{n} + \alpha\hat{n} + \rho\hat{n} + \alpha\rho\hat{n}), \qquad \text{removing constant addends and constant multipliers} \\
&= O(\alpha\rho\hat{n}) \;=\; O(\rho\hat{n}), \qquad \text{ignoring lower-order terms and insignificant multipliers} \\
&= O(\hat{n}), \qquad \text{ignoring } \rho, \text{ as it typically varies from 1 to 5}
\end{aligned} \tag{3.1}
$$

The time complexity of `outg` primitive of *CUTSMuHN* platform can be determined by including the number of tuples given as input (i.e. m') in the time complexity of `out` primitive of Equation (3.1). In other words, the time complexity of `outg` primitive can be shown to be $O(m'\hat{n})$.

3.4.2 Time Complexities of `rdp`, `rdgp`, `rd` & `rdg` of *CUTSMuHN*

The time complexity analysis of `rdp` primitive of *CUTSMuHN* platform is also based on its algorithm that is given in Figure 3.16. For this analysis, n', ρ and ι carry their usual

```
Input: Antituple a, tuple space T.
Output: Set of sought tuples S or NULL tuple ε.
begin
    S ← ∅.
    If arity of a = 0 then
        S ← T^s.
        For each tuple t present in S do
            Keep index of t in ind.
            Call updateIndTab(t, ind) to remove ind from related index tables.
            Store ind as free index in Free List F.
        end
        Return S as the set of multiple sought tuples.
    end
    Choose a field <NAME, SEARCHVALUE> from a.
    Call lookupIndTab(NAME, a), and return result to SoughtTuples.
    S ← S ⋃ SoughtTuples.
    PolymorphicTypes ← type(NAME).
    For each type Typename present in list PolymorphicTypes do
        Call lookupIndTab(Typename, a), and return result to SoughtTuples.
        S ← S ⋃ SoughtTuples.
    end
    If S ≠ ∅ then
        For each tuple t present in S do
            Keep index of t in ind.
            Call updateIndTab(t, ind) to remove ind from related index tables.
            Store ind as free index in Free List F.
        end
        Return S as the set of multiple sought tuples.
    Else
        Return ε as sought tuple.
end
```

Figure 3.22: Primitive `ingp` of $\mathcal{CUTSMuHN}$ platform.

meaning (mentioned in Table 3.1). Also, $\bar{c}\alpha$ is the lookup time in Meta index table.

The first statement of `rdp` is a "If" block, conditioned on arity of given antituple. The condition is evaluated in constant time (say, c_1). However, the statement inside this "If" block is not part of the worst-case and expected-case analyses, as it is an exit statement. The next statement of `rdp` resets the output set of sought tuples in another constant time c_2. The following statement selects a field of given antituple, which is done in some constant time c_3. The time of these three statements is aggregated to $(c_1 + c_2 + c_3)$. The subsequent statement is the call of `lookupIndTab` method, which is given in Figure 3.17. The `lookupIndTab` method, in turn, invokes `lookup` method shown in Figure 3.18. Their time analyses are necessary for the analysis of `rdp`.

The `lookupIndTab` method starts with a reset statement concerning one of its variable, which is evaluated in a constant amount of time, say c_4. The next statement in this method is a "If" block, whose condition is evaluated in $(c_{\text{H}} + \bar{c}\alpha)$ amount of time, like the preceding case in Section 3.4.1. The statement inside this "If" block is not part of the worst-case and expected-case analyses, since it is an exit statement. After the "If"

block, there is a nested "While" block, whose outer loop is to be repeated for ι times. The first statement inside the outer "While" block takes a constant time c_5, while the second statement takes the constant $(c_{\text{H}} + c')$ time. The third statement of this outer loop also takes a constant time, say c_6. Next is the inner "While" loop, which is repeated n' times. The condition of this loop additionally takes a constant time c_7 for each repetition. Inside the inner "While" loop, the first statement takes a constant time of c_8, while the second statement calls the `lookup` method. All the statements of `lookup` method execute in constant time, and thus it is considered to take a overall constant time, say c_{L} (where, $c_{\text{L}} = c_{\text{L}_1} + c_{\text{L}_2} + c_{\text{H}_1} + c_{\text{H}_2} + c_{\text{L}_3}$). The last statement of the inner "While" loop of `lookupIndTab` method is another "If" block, whose condition and consequent statement are evaluated in constant time c_9 and c_{10} respectively. The last statement of outer "While" loop is also an "If" block, whose condition as well as consequent statement take constant time c_{11} and c_{12} respectively. Finally, the last statement of `lookupIndTab` method is a return statement, which is evaluated in a constant time c_{13}. So, the overall time complexity of `lookupIndTab` method becomes:

$$\begin{aligned} & c_4 + c_{\text{H}} + \bar{c}\alpha + \iota\Big(c_5 + c_{\text{H}} + c' + c_6 + n'\big(c_7 + c_8 + c_{\text{L}} + c_9 + c_{10}\big) + c_{11} + c_{12}\Big) + c_{13} \\ = & \, k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n') \end{aligned} \tag{3.2}$$

where, $k_3 = (c_4 + c_{\text{H}} + c_{13})$, $k_4 = (c_5 + c_{\text{H}} + c' + c_6 + c_{11} + c_{12})$, and

$k_5 = (c_7 + c_8 + c_{\text{L}} + c_9 + c_{10})$

Continuing with the time analysis of `rdp` primitive, the two subsequent statements after the call of `lookupIndTab` method are completed in some constant time c_{14} and c_{15} respectively. The following is a "For" loop that repeats ρ times to handle each subtype of the determined value, similar to the complexity analysis of `out` primitive in Section 3.4.1. The first statement inside this "For" loop again calls the `lookupIndTab` method, which is evaluated according to Equation (3.2). The only remaining statement inside the "For" loop is evaluated in a constant time c_{14}. The last statement of `rdp` is an "If-Else" block, whose condition is evaluated in another constant time c_{16}. The consequent part of the "If-Else" block takes some constant time c_{R} (which takes into account the use of a random

algorithm to select a tuple from the set of tuples), whereas the alternative (i.e. false) part takes a constant time c''. It is to be noted here that $c_{\text{R}} > c''$. So, the aggregate execution time of `rdp` primitive, after simplification, becomes $O(n'\iota)$, as shown in the following.

$$
\begin{aligned}
& c_1 + c_2 + c_3 + k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n') + c_{14} + c_{15} + \rho\big(k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n') + c_{14}\big) \\
& + c_{16} + c_{\text{R}} \;\; = k_6 + \bar{c}\alpha + k_4\iota + k_5 n'\iota + k_7\rho + \bar{c}\alpha\rho + k_4\iota\rho + k_5 n'\iota\rho \\
& \qquad \text{where, } k_6 = (c_1 + c_2 + c_3 + k_3 + c_{14} + c_{15} + c_{16} + c_{\text{R}}) \text{ and } k_7 = k_3 + c_{14} \\
&= O(\alpha + \iota + n'\iota + \rho + \alpha\rho + \iota\rho + n'\iota\rho), \quad \text{removing constant addends and multipliers} \\
&= O(n'\iota\rho), \qquad \text{ignoring lower-order terms and insignificant multipliers} \\
&= O(n'\iota), \qquad \text{ignoring } \rho \text{ as it typically varies from 1 to 5}
\end{aligned} \tag{3.3}
$$

The time complexity of `rdgp` primitive of CUTSMuHN platform can be similarly shown to be asymptotically same as that of the time complexity of `rdp`, i.e. $O(n'\iota)$. During the time complexity analyses of `rd` and `rdg` primitives, the waiting time due to reactions, which are registered in absence of sought tuples, are ignored. Consequently, the time complexities of `rd` and `rdg` primitives of CUTSMuHN platform are asymptotically same as that of the time complexities of `rdp` and `rdgp`, viz. $O(n'\iota)$.

3.4.3 Time Complexities of `inp`, `ingp`, `in` & `ing` of CUTSMuHN

The time complexity analysis of `inp` primitive of CUTSMuHN platform is also based on its algorithm shown in Figure 3.20. Since the algorithms of `inp` and `rdp` are much similar, the steps of analysis of `inp` closely follows that of `rdp`. So, for this analysis, n' is considered to be the arity of input antituple to `inp`. Also, ρ is the average degree of subtyping for a tuple in the proposed tuple space structure, whereas $\hat{n}$ is the average tuple arity. Additionally, $\bar{c}\alpha$ still denotes the lookup time in Meta index table, while ι is considered to be the number of entries of any particular index table that is looked up.

The first statement of `inp` is a "If" block, which is conditioned on the arity of given antituple and is evaluated in a constant time c_1. Also, the statements inside this "If"

block are part of exit statements and are not considered in this analysis. The following statement of `inp` is evaluated in constant time c_2. The next statement selects a field of given antituple in a constant time c_3. The subsequent statement is the call of `lookupIndTab` method of Figure 3.17, which is evaluated in $\big(k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n')\big)$ according to Equation (3.2). The next two statements after the call of `lookupIndTab` method are completed in constant time c_{14} and c_{15} respectively. The following "For" loop repeats ρ times to complete in $\rho\big(k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n') + c_{14}\big)$, same as the "For" loop in `rdp`. The last statement of `inp` is the "If-Else" block, whose condition is evaluated in a constant time c_{16}. The consequent part of this block comprises of several statements, viz. (i) the statement that uses a random algorithm to select a tuple from the set of tuples and takes some constant time c_{R}, (ii) the call to `updateIndTab` method, which is given in Figure 3.21 and is analyzed next, and (iii) the two subsequent statements that take constant time of $\bar{c}_{\text{H}}$ and c_{20}. Whereas the alternative part takes a constant time c'' and is ignored compared to the consequent part of this block.

The `updateIndTab` method comprises of an external "For" loop, which repeats for $\hat{n}$ times. Inside this loop, the first statement evaluates in a constant time c_{17}. The next statement is an "If" block, whose condition is evaluated in $(c_{\text{H}} + \bar{c}\alpha)$, like the analysis in Section 3.4.1. The first consequent statement of this "If" block takes constant execution time of $(c'_{\text{H}} + \acute{c})$. This is because, hashing is first required to locate *ind* in index table and is assumed to be over by a constant time c'_{H}. The last consequent statement of this "If" block is a nested "If" block, whose condition is evaluated in some constant time c_{18}. The only consequent statement of the inner "If" block is evaluated in a constant time $(c''_{\text{H}} + \grave{c})$, as it also involves a hashing mechanism. The third statement of the outer "For" loop is completed in a constant time c_{19}. Lastly, there ia a nested "For" loop that repeats ρ times to handle each subtype. The statement inside the inner "For" loop is a nested "If" block, which is completed in $(c_{\text{H}} + \bar{c}\alpha + c'_{\text{H}} + \acute{c} + c_{18} + c''_{\text{H}} + \grave{c})$. So, the overall time complexity of `updateIndTab` method becomes:

$$\begin{aligned} &\hat{n}\big(c_{17} + c_{\text{H}} + \bar{c}\alpha + c'_{\text{H}} + \acute{c} + c_{18} + c''_{\text{H}} + \grave{c} + c_{19} + \rho(c_{\text{H}} + \bar{c}\alpha + c'_{\text{H}} + \acute{c} + c_{18} + c''_{\text{H}} + \grave{c})\big) \\ &= \hat{n}\big(k_8 + \bar{c}\alpha + \bar{c}\alpha\rho + k_9\rho\big) \end{aligned} \tag{3.4}$$

$$\text{where, } k_8 = (c_{17} + c_{\text{H}} + c'_{\text{H}} + \acute{c} + c_{18} + c''_{\text{H}} + \grave{c} + c_{19}) \text{ and}$$

$$k_9 = (c_{\text{H}} + c'_{\text{H}} + \acute{c} + c_{18} + c''_{\text{H}} + \grave{c})$$

Consequently, the aggregate execution time of `inp` primitive, after simplification, becomes $O(n'\iota)$, as shown in the following.

$$\begin{aligned}
&c_1 + c_2 + c_3 + k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n') + c_{14} + c_{15} + \rho\big(k_3 + \bar{c}\alpha + \iota(k_4 + k_5 n') + c_{14}\big) \\
&\quad + c_{16} + c_{\text{R}} + \hat{n}\big(k_8 + \bar{c}\alpha + \bar{c}\alpha\rho + k_9\rho\big) + \bar{c}_{\text{H}} + c_{20} \\
&= k_{10} + \bar{c}\alpha + k_4\iota + k_5 n'\iota + k_7\rho + \bar{c}\alpha\rho + k_4\iota\rho + k_5 n'\iota\rho + k_8\hat{n} + \bar{c}\alpha\hat{n} + \rho\bar{c}\alpha\hat{n} + k_9\rho\hat{n} \\
&\quad \text{where, } k_{10} = (c_1 + c_2 + c_3 + k_3 + c_{14} + c_{15} + c_{16} + c_{\text{R}} + \bar{c}_{\text{H}} + c_{20}) \text{ and } k_7 = k_3 + c_{14} \\
&= O(\alpha + \iota + n'\iota + \rho + \alpha\rho + \rho\iota + \rho n'\iota + \hat{n} + \alpha\hat{n} + \alpha\rho\hat{n} + \rho\hat{n}) \\
&\qquad\qquad \text{removing constant addends and constant multipliers} \\
&= O\big(\rho(n'\iota + \hat{n})\big), \qquad \text{ignoring lower-order terms and insignificant multipliers} \\
&= O(n'\iota), \qquad \text{ignoring } \rho \text{ as it typically varies from 1 to 5, and } \hat{n} \text{ as } \hat{n} < n'\iota
\end{aligned} \tag{3.5}$$

The time complexity of `ingp` primitive of *CUTSMuHN* platform can be similarly shown to be $O(n'\iota + \hat{n}\bar{m})$. Like the analyses of `rd` and `rdg` primitives, the waiting time due to reactions in case of `in` and `ing` primitives are also ignored. Consequently, the time complexities of `in` and `ing` primitives of *CUTSMuHN* platform are asymptotically same as that of the time complexities of `inp` and `ingp`, viz. $O(n'\iota)$ and $O(n'\iota + \hat{n}\bar{m})$ respectively.

3.4.4 Space Complexity of Tuple Spaces of *CUTSMuHN*

The space complexity of the proposed tuple space structure of *CUTSMuHN* platform (presented in Section 3.3.3) is $O(m\hat{n})$. This can be argued as follows. In the proposed tuple structure, the space overhead of $\texttt{HT}_1$ and $\texttt{HT}_2$ together in each tuple is $2(\hat{n} + 1)$, whereas that of $\mathfrak{E}$ is a constant (say, K). So, the Tuple Store has a space complexity of $m(2\hat{n} + K + 2) = (2m\hat{n} + K'm)$ for storing m tuples having average tuple arity of $\hat{n}$, where $K' = K + 2$. The total number of entries from all index tables together is $K''m\hat{n}$

(considering polymorphism), while Meta index table contributes $K'''m\hat{n}$ entries, where K'' and K''' are constants. The overhead of Free List is independent of the number of tuples in tuple space and can be considered as constant $\bar{K}$. Thus, the space complexity of proposed tuple space structure of *CUTSMuHN* platform becomes:

$$
\begin{aligned}
&(2m\hat{n} + K'm) + K''m\hat{n} + K'''m\hat{n} + \bar{K} = (K'' + K''' + 2)m\hat{n} + K'm + \bar{K} \\
&= O(m\hat{n} + m), && \text{removing constant addends and constant multipliers} \\
&= O(m\hat{n}), && \text{ignoring lower-order term } m, \text{ as } \hat{n} \geq 1
\end{aligned} \tag{3.6}
$$

Thus, the space complexity of proposed tuple space structure is asymptotically similar to that of the space complexity of tuple spaces of existing TSMM platforms.

3.4.5 Message Complexity of *CUTSMuHN*

The analyses of message overhead complexities are required to justify that the proposed approaches (viz. the notion of decoupling as well as the consistency handling mechanisms) in *CUTSMuHN* have reduced its network overhead. Since, the message overhead complexities due to tuple-consuming primitives have more effect than any other tuple space primitive, their analyses are presented in this section. The message overhead for multi-destined invokes of `in` and `inp` primitives are first determined, followed by `ing` and `ingp` primitives. The message overhead complexities of tuple-producing/reading primitives can be similarly analyzed. The different symbols used in these analyses are listed in Table 3.2.

In case of the multi-destined invoke of `in` primitive, the number of messages carrying antituple to (inter-host) target agents is r (where r is the number of target agents chosen by reference agent for the multi-destined tuple space operation). In absence of any positively-matched tuples, the reactions are registered in each of the designated target agents, and the information about these reaction registrations are transmitted back to reference agent. The number of messages carrying such information is at most r.

Table 3.2: Symbols used in analyzing message overhead complexities of proposed consistency handling mechanisms of CUTSMuHN platform.

Symbols	Meaning
r	Number of (inter-host) target agents chosen for any multi-destined tuple space operation
r'	Number of target agents in any multi-destined probe operation, which have positively-matched tuples and responded with them back to reference agent
$\bar{\bar{m}}$	Average number of positively-matched tuples returned by any target agent in any multi-destined tuple-reading/consuming operation
p	Number of unsuccessful "invokes" (where, no matching tuples exist in all designated target agents) in any probe tuple-reading/consuming operation

From all designated target agents, the number of response messages carrying positively-matched tuples to reference agent is at most $\bar{\bar{m}}r$ (assuming that each message carries a single tuple). Here, $\bar{\bar{m}}$ is the average number of positively-matched tuples returned by any target agent in any tuple-reading/consuming operation. For each response message, another message, containing either ack-tuple or nack-tuple, is sent back to the respective target agent. The number of such messages is also at most $\bar{\bar{m}}r$. Combinedly, the message overhead of invoking multi-destined `in` primitive for r target agents becomes at most:

$$r + r + \bar{\bar{m}}r + \bar{\bar{m}}r = 2(\bar{\bar{m}} + 1)r \tag{3.7}$$

However, the complexity analysis for multi-destined invoke of `inp` primitive is different. Unless the target agents possess positively-matched tuples, they respond with NULL tuples. Such invokes turn out to be unsuccessful for `inp` primitive[3]. Assuming p number of unsuccessful invokes, the number of messages carrying antituple to target agents during unsuccessful invokes is pr, whereas the number of response messages carrying NULL tuples back to reference agent during these invokes is also pr. In case of the successful invoke, considering that only r' out of r target agents (where, $r' \leq r$) have positively-matched tuples, they only respond back with those matched tuples, while the remaining agents send back NULL tuples. So, the number of messages carrying antituple to target agents during successful invoke is r, whereas the number of response

[3]The notion of unsuccessful invokes can be similarly observed in other probe tuple-reading/consuming primitives.

messages carrying positively-matched tuples back to reference agent is $\bar{\bar{m}}r'$ and number of messages carrying NULL tuples this time is $(r - r')$. Also, the number of messages, containing ack/nack-tuples, from reference agent to target agents (which have responded with positively-matched tuples) is $\bar{\bar{m}}r'$. Combinedly, the message overhead of invoking multi-destined `inp` primitive for r target agents becomes:

$$pr + pr + r + \bar{\bar{m}}r' + r - r' + \bar{\bar{m}}r' = 2(p+1)r + (2\bar{\bar{m}} - 1)r' \tag{3.8}$$

The process of determining the message overhead for multi-destined invokes of `ing` and `ingp` primitives are more or less alike. For instance, in case of `ing` primitive, all positively-matched tuples from designated tuple spaces are received via response messages like `in` primitive. However, no nack-tuples are sent back by reference agent to responded target agents. Instead, only ack-tuples are sent back through those messages. So, the message overhead of invoking multi-destined `ing` primitive for r target agents becomes at most:

$$r + r + \bar{\bar{m}}r + \bar{\bar{m}}r = 2(\bar{\bar{m}} + 1)r \tag{3.9}$$

For the case of multi-destined invokes of `ingp` primitive, the analysis is similar to that of `inp` primitive. Apart from the similarity in unsuccessful probes, in case of the successful probe, all positively-matched tuples are received from r' target agents. However, only the ack-tuples are sent back to them. So, the message overhead of invoking multi-destined `ingp` primitive for r target agents becomes:

$$pr + pr + r + \bar{\bar{m}}r' + r - r' + \bar{\bar{m}}r' = 2(p+1)r + (2\bar{\bar{m}} - 1)r' \tag{3.10}$$

The preceding complexity analyses of the metrics associated with tuple space coordination are used in the next section for a comparative evaluation of *CUTSMuHN* platform with the existing TSMM platforms.

3.5 Comparative Analysis

Even though the existing TSMM platforms (reported in Chapter 2) support different dimensions of decoupling individually in their tuple space coordination, few of them support these dimensions of decoupling together. In fact, many TSMM platforms provide tighter coupling between interacting agents, as they are not been able to achieve complete synchronization decoupling of tuple space coordination. Moreover, these TSMM platforms have not incorporated any consistency handling mechanism, even though they support multi-destined remote tuple space operations in unreliable environments. These platforms also have limitations in their tuple/antituple and tuple space structures, which reduce their performances.

Compared to existing TSMM platforms, *CUTSMuHN* platform improves the nature of tuple space coordination in several aspects. (1) *CUTSMuHN* platform achieves complete synchronization decoupling, as it provides the proposed notion of entire time-decoupling of every interaction of coordinating agents. After including the proposed consistency handling mechanisms in agent interactions, *CUTSMuHN* platform is able to handle the unreliability issues in data transmission and dynamic circumstances arising due to device mobility, which, in turn, improves robustness of its tuple space coordination. Moreover, reduced network overhead in the proposed consistency handling mechanisms also improves efficiency of *CUTSMuHN* platform. (2) The enhancements in proposed tuple/antituple and tuple space structures of *CUTSMuHN* platform improve the performances of different tuple space operations, as well as support wide spectrum of dynamic applications, which, in turn, makes it simple, versatile, efficient and scalable in its coordination.

Table 3.3 compares the performances of *CUTSMuHN* platform with existing TSMM platforms, particularly LIMONE, EgoSpaces and LIME. These three platforms are considered to be the representatives of other existing TSMM platforms, and they are the close contenders of *CUTSMuHN* platform. It is revealed from Table 3.3 that —

- *CUTSMuHN* platform is more robust compared to existing TSMM platforms due to its support for a new dimension of decoupling (resulting in complete synchronization decoupling) and new consistency handling mechanisms. Also, it is more efficient compared to others due to reduced network overhead of its consistency mechanisms.

- *CUTSMuHN* platform provides the unordered tuple/antituple structure, which makes it simple but versatile like other TSMM platforms having the unordered structure. Also, it incorporates the indexed tuple space structure having space complexity of tuple spaces asymptotically similar to that of existing TSMM platforms.
- Indexed tuple space structure of *CUTSMuHN* platform makes it more efficient and scalable by improving performances of tuple space coordination. In fact, performances of tuple-reading/consuming operations improves manyfold, which have significant impacts in comparison to performances of tuple-producing operations. Considering the fact that tuple-producing operations occur less often compared to tuple-reading/consuming operations in most applications, overall performance enhancement of the tuple space model of *CUTSMuHN* platform has found to be substantial.
- The message complexities of *CUTSMuHN* platform are better than the existing TSMM platforms. The cases, where message complexities are comparable, are justified by the fact that *CUTSMuHN* platform introduces an additional level of messaging to reduce the periods of suspension of the blocking primitives (as shown in Figure 3.2(a)).

3.6 Conclusion

The existing dimensions of decoupling in tuple space coordination of TSMM platforms have certain shortcomings in dynamic environments. In particular, synchronization decoupling in tuple space coordination has turned out to be partial in nature. This chapter has proposed an additional dimension of decoupling in tuple space coordination, that augments the existing (partial) synchronization decoupling to achieve complete synchronization decoupling in TSMM platforms. This chapter has also proposed new consistency handling mechanisms to achieve OUT-consistency and IN-consistency in multi-destined tuple space coordination in unreliable environments. Moreover, this chapter has proposed enhancements to the existing tuple/antituple and tuple space structures, which radically impact efficiency of TSMM platform. Combining the benefits of these contributions, this chapter has concluded that *CUTSMuHN* platform improves the nature of

Table 3.3: Comparison of tuple space coordination of $\mathcal{CUTSMuHN}$ platform with existing TSMM platforms.

Coordination Attributes		$\mathcal{CUTSMuHN}$	Existing TSMM platforms		
			LIMONE	EgoSpaces	LIME
Proposed Dimension of Decoupling		Supported	No	No	No
Complete Synchronization Decoupling		Supported	Partial	Partial	Partial
Response Atomicity of Blocking Tuple-Consuming Primitive		Relaxed	Strict	Strict	Strict
Nature of Consistency		3-way	4-way	5/7-way	5-way
Nature of Tuple/Antituple structure		Unordered	Unordered	Unordered	Ordered
Nature of Tuple Space structure		Indexed	Sequential	Sequential	Sequential
Nature of Tuple/Antituple comparison		Apposite	Apposite & Non-apposite	Apposite & Non-apposite	Apposite & Non-apposite
Space complexity of Tuple Space		$O(m\hat{n})$	$O(m\hat{n})$	$O(m\hat{n})$	$O(m\hat{n})$
Original nature of Tuple Space primitives followed		Yes	No	No	No
Time complexity of Tuple Space Primitives:					
	`out`	$O(\hat{n})$	$O(1)$	$O(1)$	$O(1)$
	`outg`	$O(m'\hat{n})$	$O(m')$	$O(m')$	$O(m')$
	`rdp`	$O(n'\iota)$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`rdgp`	$O(n'\iota)$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`rd`	$O(n'\iota)$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`rdg`	$O(n'\iota)$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`inp`	$O(n'\iota)$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`ingp`	$O(n'\iota + \hat{n}\bar{m})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`in`	$O(n'\iota)$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
	`ing`	$O(n'\iota + \hat{n}\bar{m})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$	$O(n'm\hat{n})$
Message complexity of multi-destined Tuple Space Primitives:					
	`in`	$2(\bar{\bar{m}}+1)r$	$4r$	$3r+2$	$5r$
	`inp`	$2(p+1)r+$ $(2\bar{\bar{m}}-1)r'$	$2(p+1)r$ $+2r'$	$(2p+7)r$ $+2$	$2(p+1)r$ $+2r'$
	`ing`	$2(\bar{\bar{m}}+1)r$	$(3\bar{\bar{m}}+1)r$	$(3\bar{\bar{m}}+2)r$	$(3\bar{\bar{m}}+2)r$
	`ingp`	$2(p+1)r+$ $(2\bar{\bar{m}}-1)r'$	$2(p+1)r+$ $(3\bar{\bar{m}}-1)r'$	$(2p+7)r+$ $(3\bar{\bar{m}}-1)r'$	$2(p+1)r+$ $(3\bar{\bar{m}}-1)r'$

tuple space coordination in many aspects. However, there is a need to support the tuple space coordination of CUTSMuHN platform over multiple heterogeneous underlying networks to benefit the end-users. The next chapter focuses on the approaches to resolve heterogeneity of underlying networks.

Chapter 4

Discovery and Communication of *CUTSMuHN*

4.1 Introduction

An important objective of developing *CUTSMuHN* platform is to provide "ubiquity" to the wide variety of activities of end-users through different mobile applications. Any such application, depending on its design and applicability, may have to execute over different types of networks (wired or wireless) having diverged characteristics. Developing a single TSMM (like, *CUTSMuHN*) to resolve heterogeneity of multiple underlying networks is challenging, due to the disparate nature of each network. Each type of wired network is broadly distinguishable from the other one. This is also true with wireless networks. Apart from being mobile, dynamic and unreliable, each wireless network has some distinct intrinsic properties that are radically different from other wireless networks. So, designing a single TSMM platform to withstand multiple heterogeneous underlying networks is difficult. This is the reason why existing TSMM platforms, like TSpaces, JavaSpaces, LIME, LIMONE, EgoSpaces, PageSpace, L^2imbo, TuCSoN, MARS, TOTA etc., focus on specific underlying networks, viz. wired network or ad hoc network. These shortcomings of existing TSMM platforms are overcome in this chapter by integrating additional discovery and communication functionalities in *CUTSMuHN* platform, which facilitates complete synchronization decoupling of its mobile applications over multiple heterogeneous networks.

4.2 Heterogeneity in Underlying Networks of CUISMuHN

The recent research advancements in wireless communication technologies and mobile/ portable computing devices enable the end-users to operate their mobile applications in different types of networks. This leads to underlying heterogeneity for mobile middleware platforms (like CUISMuHN) running in mobile/portable devices of end-users. This chapter focuses on different types of wireless networks, which CUISMuHN platform supports for its end-user's applications. These networks include Infrastructure Basic Service Set, Independent Basic Service Set, and Heterogeneous Mobile Ad hoc Network. The following sections briefly overview these networks.

4.2.1 Infrastructure Basic Service Set

Infrastructure Basic Service Set (iBSS) [60] is a form of IEEE 802.11 Wireless Local Area Network, which comprises of a *Distribution System* (DS) and multiple connected *Basic Service Set*s (BSS), as shown in Figure 4.1. Typically, iBSS allows three distinct nature of devices as its building blocks, viz. stationary devices, mobile devices and access points. The stationary devices operate within the DS, while mobile devices are limited within

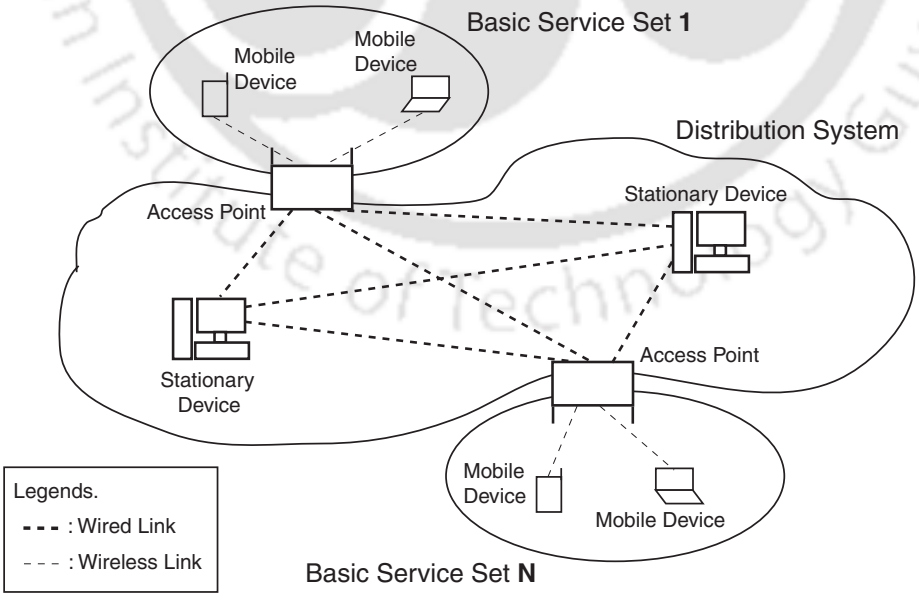


Figure 4.1: Schematic view of different components of iBSS.

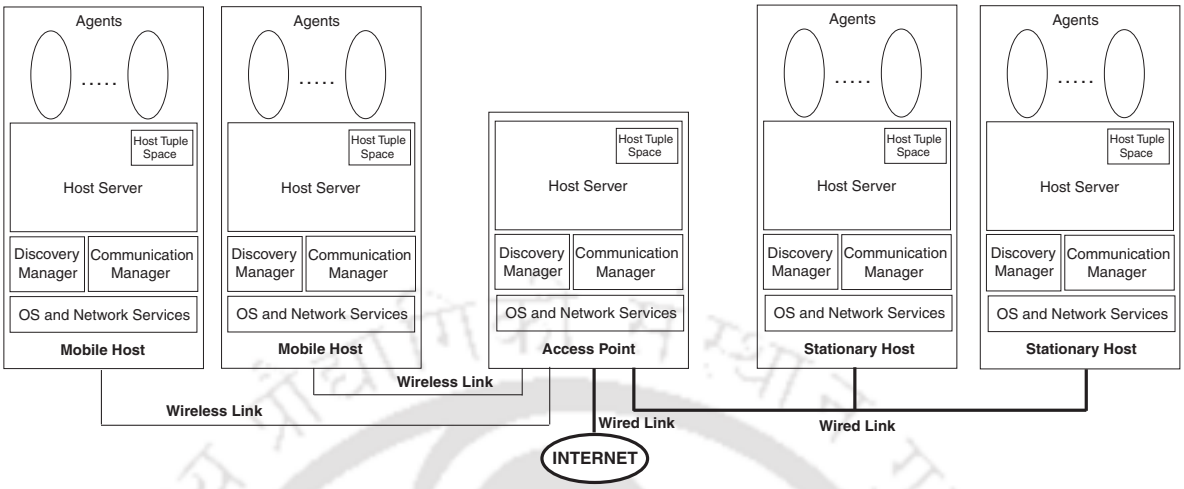


Figure 4.2: Connectivity of different hosts of *CUTSMuHN* over iBSS.

each *Basic Service Area* (BSA) of BSS. Only the access points are positioned at both DS as well as BSA simultaneously to arbitrate between them. In other words, the stationary devices are provided with only wired network connectivity, whereas mobile devices are only having wireless network connectivity. The access points act as "mediator" either between a pair of mobile devices, or between a mobile device and a stationary device, as it contains both wired and wireless network interfaces.

Accordingly, three categories of hosts are earmarked in *CUTSMuHN* platform for supporting its deployment over iBSS, viz. *stationary host* (SH), *mobile host* (MH) and *access point* (AP). The agents of different applications can be present in stationary host and/or mobile host only; no agents are allowed to be executed in access point. Figure 4.2 presents the connectivity pattern among different category of hosts of *CUTSMuHN* platform over iBSS; stationary hosts and access points are connected through wired network connectivity of DS, while mobile hosts are connected through wireless network connectivity to associated access points in BSA. A possibility is also considered in this scenario, where a mobile host can move out of the BSA cluster into "no network" or "void" zone, making it standalone and disabling its communication functionality.

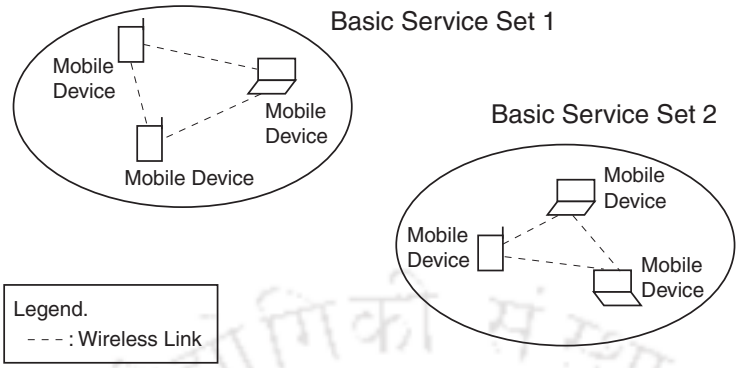


Figure 4.3: Schematic view of different components of IBSS.

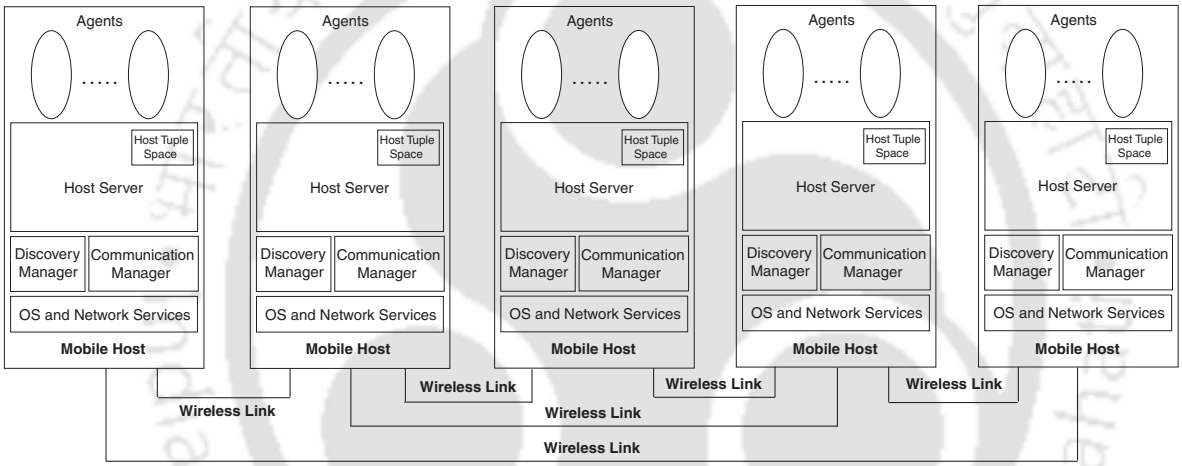


Figure 4.4: Connectivity of different hosts of *CUTSMuHN* platform over IBSS.

4.2.2 Independent Basic Service Set

Independent Basic Service Set (IBSS) [60] is also another form of IEEE 802.11 Wireless Local Area Network. However, unlike iBSS, it comprises of multiple disjoint BSSs and allows only one nature of devices, viz. mobile devices, as shown in Figure 4.3. The mobile devices are those devices with wireless network connectivity, including portable devices with wireless network connectivity, and their coverage is confined within any BSA. Consequently, only one category of host is earmarked in *CUTSMuHN* platform for supporting its deployment over IBSS, viz. *mobile host*. The agents of different applications can be present in any mobile host. Figure 4.4 presents the connectivity pattern among different hosts of *CUTSMuHN* platform over IBSS. As in iBSS, the possibility of any mobile host moving out of a BSA into "no network" zone is also considered in IBSS network, making

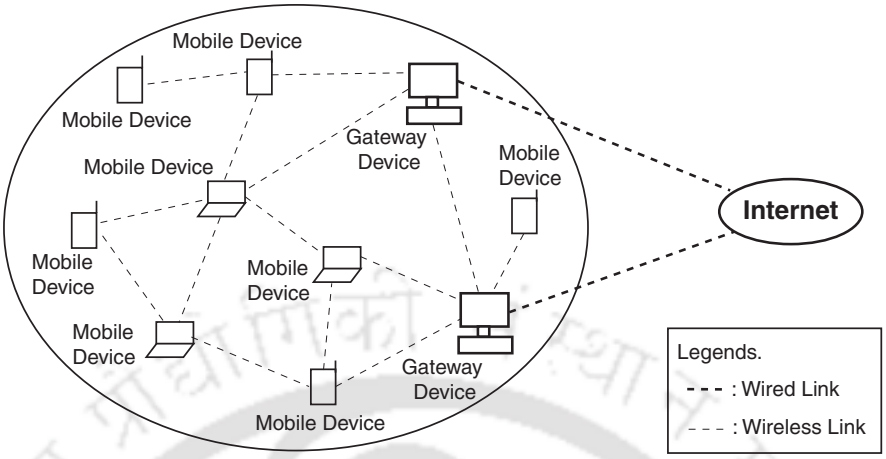


Figure 4.5: Schematic view of different components of H-MANET.

it standalone and disabling its communication functionality.

4.2.3 Heterogeneous Mobile Ad Hoc Network

Heterogeneous Mobile Ad hoc Network (H-MANET) [57] comprises of two kind of nodes, viz. mobile devices and gateway devices, as shown in Figure 4.5. So, two categories of hosts are earmarked in $\mathcal{CUTSMuHN}$ platform when it is deployed over H-MANET, viz. *gateway host* (GH) and *mobile host*. The gateway hosts have wireless network connectivity and additionally may have wired network connectivity to provide the gateway functionality. The gateway hosts are also resourceful, strategically-positioned and practically stationary. Moreover, each gateway host in H-MANET is assumed to be always directly-connected to other gateway hosts via wireless network connectivity. On the other hand, mobile hosts are the hosts with only wireless network connectivity. The agents of different applications are present in mobile hosts only. $\mathcal{CUTSMuHN}$ platform enables each gateway host to act as mediator only for two different mobile hosts that are not directly connected; the same is also true for mobile hosts. Thus, a pair of mobile hosts are either directly connected (viz. single hop) or connected via other mobile/gateway hosts (viz. multiple hops). Figure 4.6 presents the connectivity pattern among different category of hosts of $\mathcal{CUTSMuHN}$ platform over H-MANET. Moreover, as in the cases of iBSS and IBSS, the possibility is considered, where a mobile host can move out of the coverage

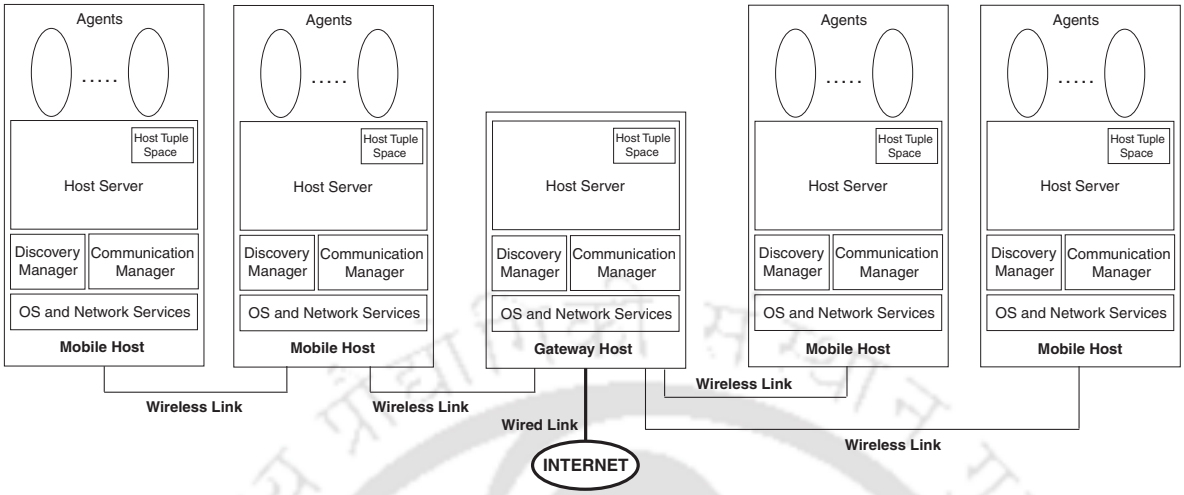


Figure 4.6: Connectivity of different hosts of $\mathcal{CUISMuHN}$ platform over H-MANET.

area of other mobile/gateway hosts into "no network" zone, making it standalone and disabling its communication functionality.

The succeeding sections propose mechanisms to handle discovery and communication functionalities of $\mathcal{CUISMuHN}$ platform in these underlying networks.

4.3 Proposed Discovery Mechanisms of $\mathcal{CUISMuHN}$

The proposed discovery mechanisms furnish an updated knowledge of available agents (and their hosts) that are reachable from (i.e. neighbors of) source host. In the following subsections, the proposed discovery mechanisms of $\mathcal{CUISMuHN}$ platform are presented for iBSS, IBSS and H-MANET.

4.3.1 Discovery Mechanism for iBSS

In case of iBSS, the proposed discovery mechanism attains the availability information by broadcasting and receiving *beacons* at periodic intervals. Typically, a beacon comprises of the identity of source host and an updated list of identities of agents presently active in source host. In $\mathcal{CUISMuHN}$ platform, the beacons are implemented as special tuples. The interval of beacon-broadcasting is decided based on several parameters, like

Discovery Process for Stationary Host over iBSS

I. [*Broadcast* Beacon *message*]
The Discovery Manager of a stationary host (say, X_{SH}) broadcasts $Beacon_{X_{SH}}$ messages to other stationary hosts and access points in DS at regular predefined time intervals, informing about its active agents; such information are preserved in *Agent List*.

II. [*Receive* Beacon *message*]
It builds its *Neighbor List* with the availability information from beacon messages received from other stationary hosts or access points through DS.

III. [*No* Beacon *message from a particular host*]
If it does not receive beacon messages from a particular stationary host or access point, their availability information in *Neighbor List* becomes stale, and are going to be removed/replaced eventually.

Figure 4.7: Rules of proposed discovery mechanism of $\mathcal{CUTSMuHN}$ platform for stationary host over iBSS.

reliability, throughput, delay etc., of underlying iBSS. The information from received beacons are preserved in a data structure called *Neighbor List*; each entry of *Neighbor List* specifies a neighbor host, its active agents, and a lifespan of that entry called the *extant*. *Neighbor List* is periodically updated on receipt of new beacons from different hosts, and also on expiry of extant values of existing entries. For the expired entries, if no new beacons have been received from the corresponding hosts, those entries are considered as stale and can be overwritten by new entries when needed. Additionally, each mobile host and access point also has another data structure, called *assoc*, to store the identity of its associated host(s). The rules of proposed discovery mechanism of $\mathcal{CUTSMuHN}$ platform for stationary host, mobile host and access point over iBSS are shown in Figure 4.7, Figure 4.8 and Figure 4.9 respectively.

4.3.2 Discovery Mechanism for IBSS

The objective of proposed discovery mechanism remains the same (i.e. providing updated neighborhood by broadcasting beacons at periodic intervals), when $\mathcal{CUTSMuHN}$ platform is deployed over IBSS. Only, the rules of discovery mechanism for mobile host over IBSS,

Discovery Process for Mobile Host over iBSS

I. [*Broadcast* Beacon *message*]

The Discovery Manager of a mobile host (say, X_{MH}) broadcasts $\text{Beacon}_{X_{MH}}$ messages to access points within its range at regular predefined time intervals, irrespective of being 'associated' or 'standalone', informing about its active agents; such information are preserved in *Agent List*.

II. [*Receive* Beacon *message*]

i. [*Received in standalone mode*]

If it receives $\text{Beacon}_{Z_{AP}}$ message from an access point Z_{AP} in 'standalone' condition, it becomes 'associated' to Z_{AP} and updates the availability information of Z_{AP} in its *Neighbor List*.

It optionally informs Communication Manager to resume data communication.

ii. [*Received in associated mode*]

If it receives beacon messages from access points in 'associated' condition, it simply updates their availability information in its *Neighbor List*.

III. [*No* Beacon *message*]

If it does not receive any beacon message from its associated access point Z_{AP}, availability information of Z_{AP} in *Neighbor List* become stale, and is going to be removed/replaced eventually. Moreover, it disassociates from Z_{AP} and searches *Neighbor List* for active presence of any other access points within range.

If any such information is provided by *Neighbor List* (i.e. another access point Z'_{AP} is available), it associates to Z'_{AP} and informs Communication Manager about the change in association with access point.

Otherwise, it becomes 'standalone', and continues Step (I.) while waiting for beacon messages from any access point. If it receives any such message, it follows Step (II.i.).

It optionally informs Communication Manager to stop data communication.

Figure 4.8: Rules of proposed discovery mechanism of *CUTSMuHN* platform for mobile host over iBSS.

Discovery Process for Access Point over iBSS

I. [*Broadcast* Beacon *message*]
The Discovery Manager of an access point (say, X_{AP}) broadcasts $Beacon_{X_{AP}}$ messages to other stationary hosts, mobile hosts and access points in DS and BSA at regular predefined time intervals, informing about the available agents of neighboring stationary hosts/mobile hosts/access points; such information are preserved in its *Neighbor List*.

II. [*Receive* Beacon *message*]

i. [*Received from DS*]
If it receives beacon messages from other stationary hosts or access points in DS, it simply updates their availability information in its *Neighbor List*.

ii. [*Received from BSA*]
If it receives beacon messages from mobile hosts in its own BSA, it also updates its *Neighbor List* with their availability information. Furthermore, it updates *assoc* to enlist each such mobile host as 'associated' to X_{AP}.

III. [*No* Beacon *message from a particular host*]
If no beacon messages are received from a particular stationary host or mobile host or access point, then their availability information in *Neighbor List* becomes stale, and are going to be removed/replaced eventually.

Figure 4.9: Rules of proposed discovery mechanism of *CUTSMuHN* platform for access point over iBSS.

Discovery Process for Mobile Host over IBSS

I. [*Broadcast* Beacon *message*]
The Discovery Manager of a mobile host (say, X_{MH}) broadcasts $Beacon_{X_{MH}}$ messages to other mobile hosts within its range at regular predefined time intervals, irrespective of 'connected' or 'standalone', informing about its active agents; such information are preserved in *Agent List*.

II. [*Receive* Beacon *message*]

i. [*Received in standalone mode*]
If it receives $Beacon_{Y_{MH}}$ message from another mobile host Y_{MH} in 'standalone' condition, it becomes 'connected', and updates availability information of Y_{MH} in its *Neighbor List*.
It optionally informs Communication Manager to start data communication.

ii. [*Received in connected mode*]
If it receives beacon messages from other mobile hosts in 'connected' condition, it simply updates their availability information in its *Neighbor List*

III. [*No* Beacon *message from a particular mobile host*]
If it does not receive $Beacon_{Y_{MH}}$ message from Y_{MH}, availability information of Y_{MH} in *Neighbor List* become stale, and is going to be removed/replaced eventually.

IV. [*All information in* Neighbor List *expired*]
If the availability information of all mobile hosts in *Neighbor List* become stale, X_{MH} becomes 'standalone', and continues Step (I.) while waiting for beacon messages from other mobile hosts. On receiving any such message, it follows Step (II.i.).
It optionally informs Communication Manager to stop data communication.

Figure 4.10: Rules of proposed discovery mechanism of *CUTSMuHN* platform for mobile host over IBSS.

Discovery Process for Gateway Host over H-MANET

I. [*Broadcast* Beacon *message*]
The Discovery Manager of a gateway host (say, X_{GH}) broadcasts $\text{Beacon}_{X_{GH}}$ messages to other mobile hosts and gateway hosts within its range at regular predefined time intervals, informing about the available agents of neighbor mobile hosts that are reachable within 2 hops; such information are preserved in its *Neighbor List*.

II. [*Receive* Beacon *message*]
If it receives beacon messages from other mobile hosts or gateway hosts, it preserves their availability information in its *Neighbor List*.

III. [*No* Beacon *message from a particular host*]
If it does not receive beacon messages from a particular host, availability information of that host in *Neighbor List* becomes stale, and is going to be removed/replaced eventually.

Figure 4.11: Rules of proposed discovery mechanism of $\mathcal{CUTSMuHN}$ platform for gateway host over H-MANET.

which are shown in shown in Figure 4.10, differ from that of mobile host over iBSS.

4.3.3 Discovery Mechanism for H-MANET

Like the two preceding scenarios, the proposed discovery mechanism for H-MANET provides an up-to-date information of availability of gateway hosts and mobile hosts by broadcasting and receiving beacons. The interval of beacon-broadcasting is decided by reliability, throughput, delay etc. in underlying H-MANET. The beacons carry identities of hosts (and their agents) that are within 2 hops of the broadcaster. Figure 4.11 and Figure 4.12 show the rules of proposed discovery mechanism of $\mathcal{CUTSMuHN}$ platform for gateway host and mobile host respectively over H-MANET.

The aforementioned proposed discovery mechanisms for iBSS, IBSS and H-MANET are realized through Discovery Manager of $\mathcal{CUTSMuHN}$ platform. Their knowledge of availability are utilized by other components of $\mathcal{CUTSMuHN}$ platform, including its communication mechanisms.

Discovery Process for Mobile Host over H-MANET

I. [*Broadcast* Beacon *message*]
The Discovery Manager of a mobile host (say, X_{MH}), irrespective of being connected or standalone, broadcasts $\text{Beacon}_{X_{MH}}$ messages to other mobile hosts and gateway hosts within its range at regular predefined time intervals, informing about its active agents as well as available agents of its neighbor mobile hosts that are reachable within 2 hops; such information are preserved in data structures *Agent List* and *Neighbor List* respectively.

II. [*Receive* Beacon *message*]

i. [*Received in standalone mode*]
If it receives $\text{Beacon}_{Y_{MH}}$ message from another mobile host Y_{MH} or $\text{Beacon}_{Z_{GH}}$ message from a gateway host Z_{GH} in 'standalone' condition, it becomes 'connected', and updates the availability information about Y_{MH} or Z_{GH} in its *Neighbor List* from their respective received messages.
It optionally informs Communication Manager to resume data communication.

ii. [*Received in connected mode*]
If it receives $\text{Beacon}_{Y_{MH}}$ message or $\text{Beacon}_{Z_{GH}}$ message from Y_{MH} or Z_{GH} respectively while 'connected', it simply updates their availability information from the received messages in its *Neighbor List*.

III. [*No* Beacon *message from a particular host*]
If it does not receive $\text{Beacon}_{Y_{MH}}$ message or $\text{Beacon}_{Z_{GH}}$ message from Y_{MH} or Z_{GH} respectively, their availability information in *Neighbor List* become stale, and are going to be removed/replaced eventually.

IV. [*All information in Neighbor List expired*]
If the availability information of all mobile/gateway hosts in *Neighbor List* become stale, X_{MH} enter the 'standalone' mode, and continues Step (I.) while waiting for beacon messages from other mobile/gateway hosts. If any such message is received, Step (II.i.) is followed.
It optionally informs its Communication Manager to stop data communication.

Figure 4.12: Rules of proposed discovery mechanism of *CUTSMuHN* platform for mobile host over H-MANET.

4.4 Proposed Communication Mechanisms of *CUTSMuHN*

The proposed communication mechanisms of *CUTSMuHN* platform primarily emphasize on reliably transferring reaction/response-tuples (collectively referred in this chapter as RT for convenience) from one host to another. In the following subsections, these mechanisms are presented for iBSS, IBSS and H-MANET.

4.4.1 Communication Mechanism for iBSS

In iBSS, the reliable transfer of RT is proposed to be accomplished by a host-level acknowledgement mechanism with timeout. However, the proposed communication mechanism regulates the use of acknowledgement depending on the nature of network interfaces. For instance, the acknowledgement mechanism is not involved while transferring RT through wired network interface, because wired connectivity of DS can be considered as reliable. The acknowledgement mechanism is used only when wireless network interface is employed, i.e. when access points and their associated mobile hosts are communicating. The need for reliability in this case is due to the fact that wireless connections are intrinsically unreliable, and mobile hosts can move from one BSA to other or to a void zone. The rules of proposed communication mechanism of *CUTSMuHN* platform for stationary host, mobile host and access point over iBSS are shown in Figure 4.13, Figure 4.14, Figure 4.15, Figure 4.16, Figure 4.17, Figure 4.18 and Figure 4.19 respectively.

Each RT, handed over by Host Server, is converted to an RT-message before being given to the transport service of that host for onward transmission to a target host. An RT-message comprises of its identity (which includes its creation timestamp), identity of source host, identity of destination host, the RT itself, and type of network interface used for transfer (viz. wired or wireless). It is to be noted here that the information about target agents are incorporated within the RT itself. In case of mobile hosts or access points (i.e. when transmission proceeds via wireless network interfaces), a copy of the RT-message is also preserved in a special data structure called *Comm Stash* (short form of 'communication stash'). Also at destination host, identity of message and identity of

Communication Process for Stationary Host over iBSS

I. [*Generate* RT_S *message for transfer*]
The Communication Manager of stationary host $\mathbf{X}_{\mathrm{SH}}$ generates an RT_S message, and executes the following steps depending on the destination of RT_S message.

II. [*Send* RT_S *message*; *Destination*: *stationary host*]
If the destination of RT_S message is another stationary host (say, $\mathbf{Y}_{\mathrm{SH}}$), it sends the RT_S message to $\mathbf{Y}_{\mathrm{SH}}$ through DS.

III. [*Send* RT_S *message*; *Destination*: *mobile host*]

i. [*Send* RT_S *message*]
If the destination of RT_S message is a mobile host (say, $\mathbf{Y}_{\mathrm{MH}}$), it checks for the presence of any active path to $\mathbf{Y}_{\mathrm{MH}}$ in its *History*.
If an active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *History* through an access point (say, $\mathbf{Z}^1_{\mathrm{AP}}$) presently associating $\mathbf{Y}_{\mathrm{MH}}$, it sends the RT_S message to $\mathbf{Z}^1_{\mathrm{AP}}$ through DS.
If no active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *History*, then it looks up its *Neighbor List* for any active path to $\mathbf{Y}_{\mathrm{MH}}$.
If an active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *Neighbor List* via another access point (say, $\mathbf{Z}^2_{\mathrm{AP}}$) that presently associates $\mathbf{Y}_{\mathrm{MH}}$, it sends the RT_S message to $\mathbf{Z}^2_{\mathrm{AP}}$ through DS.
If no active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *History* and *Neighbor List*, then it follows Step (III.ii.) and Step (III.iii.).

ii. [*Broadcast* $\mathsf{locateMH}_{\mathbf{Y}_{\mathrm{MH}}}$ *message in* DS]
It broadcasts a $\mathsf{locateMH}_{\mathbf{Y}_{\mathrm{MH}}}$ message to all access points in DS asking about the presence of $\mathbf{Y}_{\mathrm{MH}}$ in their BSA, and waits for the corresponding $\mathsf{foundMH}_{\mathbf{Y}_{\mathrm{MH}}}$ message for a predefined time interval. Also, it preserves RT from the RT_S message and other details of the search information in *Loc Stash*.

Figure 4.13: Rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for stationary host over iBSS: Part 1.

iii. [*Receive* foundMH$_{\mathbf{Y}_{MH}}$ *message*]
If it receives a foundMH$_{\mathbf{Y}_{MH}}$ message from another access point (say, $\mathbf{Z}'_{AP}$, that is presently associating $\mathbf{Y}_{MH}$) within the predefined time interval, it regenerates RT$_S$ message, with RT from *Loc Stash*, sends this message to $\mathbf{Z}'_{AP}$ through DS, and updates the corresponding path record of $\mathbf{Y}_{MH}$ in *History*.
Otherwise, it returns back RT from *Loc Stash* to Host Server.

IV. [*Receive* RT$_R$ *message*]
If it receives an RT$_R$ message from some other stationary host or access point through DS, it checks the destination of RT$_R$ message.
If the destination of received RT$_R$ message matches with itself, it passes RT from that message to Host Server.
Otherwise, it ignores that message.

Figure 4.14: Rules of proposed communication mechanism of *CUTSMuHN* platform for stationary host over iBSS: Part 2.

source host of the last received message (which is received via wireless network interface) are preserved in another data structure called *Last RT*, before an acknowledge message (henceforth referred as ACK-message) is sent back to source host. Each ACK-message comprises of its identity (including its creation timestamp), identity of source host, identity of destination host, identity of received RT-message, and type of network interface to be used for transmission. So, an ACK-message contains the receiving information of a new RT-message since the receipt of last ACK-message.

4.4.2 Communication Mechanism for IBSS

The proposed communication mechanism for IBSS also emphasizes on reliable transfer of each RT, but from one mobile host to another mobile host. Consequently, the need for acknowledgement mechanism is equally prevalent here, like it exists for iBSS. The rules of proposed communication mechanism for mobile host over IBSS are shown in Figure 4.20 and Figure 4.21.

Communication Process for Mobile Host over iBSS

I. [*Generate* RT_S *message for transfer*]
The Communication Manager of mobile host X_{MH} generates an RT_S message, and preserves a copy of it in *Comm Stash*.

II. [*Send* RT_S *message*]

i. [*Wait during standalone mode*]
If X_{MH} is in the 'standalone' condition, it waits to get associated with any access point, and then follows Step (II.ii.).

ii. [*Send* RT_S *message to associated access point*]
If X_{MH} is presently associated with an access point (say, Z_{AP}), it sends the RT_S message to Z_{AP}, and waits for the corresponding ACK_{RT_S} message for a predefined time interval.

iii. [*Receive* ACK_{RT_S} *message within timeout*]
If it receives an ACK_{RT_S} message from Z_{AP} within timeout interval, it erases the RT_S message in *Comm Stash*.
Moreover, it optionally informs the availability information of Z_{AP} to Discovery Manager of X_{MH}.

iv. [*No* ACK_{RT_S} *message within timeout*]
If it does not receive an ACK_{RT_S} message from Z_{AP} within present timeout interval, it resends the RT_S message (that is preserved in *Comm Stash*) to Z_{AP}, and waits for the corresponding ACK_{RT_S} message for twice the previous time interval.

v. [*No* ACK_{RT_S} *message within timeout repeatedly*]
If it has repeated Step (II.iv.) for 3 successive times, and still has not received ACK_{RT_S} message within updated time intervals, then it returns back RT from the RT_S message to Host Server of X_{MH}, and informs the "not reachable" status of Z_{AP} to Discovery Manager.

Figure 4.15: Rules of proposed communication mechanism of *CUTSMuHN* platform for mobile host over iBSS: Part 1.

III. [*Receive* RT_R *message; Send* $\mathsf{ACK}_{\mathsf{RT}_R}$ *message*]
If it receives an RT_R message from $\mathbf{Z}_{\mathrm{AP}}$, it checks the destination of RT_R message.
If $\mathbf{X}_{\mathrm{MH}}$ is itself the destination of that message, it next checks *Last RT* to ascertain duplicate receipt.
If RT_R message is not duplicated, it hands over RT from the RT_R message to Host Server, records the message identity of that message in *Last RT*, and then sends an $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\mathbf{Z}_{\mathrm{AP}}$. Otherwise, it discards RT_R message and resends a new $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\mathbf{Z}_{\mathrm{AP}}$.
If the destination of RT_R message does not match with itself, it ignores that message.

Figure 4.16: Rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for mobile host over iBSS: Part 2.

4.4.3 Communication Mechanism for H-MANET

Like the cases of iBSS and IBSS networks, the primary emphasis of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for H-MANET is also to reliably transfer different RT-messages from one mobile host to another reachable mobile host, directly or via other mobile/gateway hosts, using acknowledgement mechanism. The mediator hosts forward the received RT-messages to eventually deliver to target mobile host. Figure 4.22 and Figure 4.23 combinedly show the rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform aggregated for both mobile host and gateway host over H-MANET.

The proposed communication mechanisms for iBSS, IBSS and H-MANET are realized through Communication Manager of $\mathcal{CUTSMuHN}$ platform. Their capability of reliable transfer of RT from one host to another are primarily utilized by Host Server to effectuate coordination of interacting agents.

4.4.4 Proposed Mechanisms and Three-way Agent Coordination

In Figure 4.24(a), Figure 4.24(b) and Figure 4.24(c), it has been shown that the proposed communication mechanisms are transferring data messages as part of the proposed three-way approach of performing remote multi-destined tuple-producing/reading/consuming

Communication Process for Access Point over iBSS

I. [*Generate* RT_S *message for transfer*]
The Communication Manager of access point $\mathbf{X}_{\mathrm{AP}}$ generates an RT_S message, and executes the following steps according to the destination of RT_S message.

II. [*Send* RT_S *message*; *Destination*: *stationary host*]
If the destination of RT_S message is a stationary host (say, $\mathbf{Y}_{\mathrm{SH}}$), it sends the RT_S message to $\mathbf{Y}_{\mathrm{SH}}$ through DS.

III. [*Send* RT_S *message*; *Destination*: *mobile host*, *in own* BSA]

i. [*Send* RT_S *message to associated mobile host*]
If the destination of RT_S message is a mobile host (say, $\mathbf{Y}_{\mathrm{MH}}$) and is presently associated with $\mathbf{X}_{\mathrm{AP}}$, it first preserves a copy of the RT_S message in *Comm Stash*, sends that message to $\mathbf{Y}_{\mathrm{MH}}$ in BSA, and waits for the corresponding $\mathsf{ACK}_{\mathsf{RT}_S}$ message for a predefined time interval.

ii. [*Receive* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout*]
If it receives an $\mathsf{ACK}_{\mathsf{RT}_S}$ message from $\mathbf{Y}_{\mathrm{MH}}$ within time interval, it erases the RT_S message from *Comm Stash*.
Moreover, it optionally informs the availability information of $\mathbf{Y}_{\mathrm{MH}}$ to Discovery Manager of $\mathbf{X}_{\mathrm{AP}}$.

iii. [*No* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout*]
If it does not receive an $\mathsf{ACK}_{\mathsf{RT}_S}$ message from $\mathbf{Y}_{\mathrm{MH}}$ within present timeout interval, it resends the RT_S message (that is preserved in *Comm Stash*) to $\mathbf{Y}_{\mathrm{MH}}$ in BSA, and waits for the corresponding $\mathsf{ACK}_{\mathsf{RT}_S}$ message for twice the previous time interval.

iv. [*No* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout repeatedly*]
If it has repeated Step (III.iii.) for 3 successive times and still has not received the $\mathsf{ACK}_{\mathsf{RT}_S}$ message within updated time intervals, then it follows Step (IV.).

Figure 4.17: Rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for access point over iBSS: Part 1.

IV. [*Send* RT_S *message*; *Destination*: *mobile host*, *not in own* BSA]
If the destination of RT_S message, $\mathbf{Y}_{\mathrm{MH}}$, is not presently associated with it, it checks for the presence of any active path to $\mathbf{Y}_{\mathrm{MH}}$ in its *History*.
If an active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *History* through another access point (say, $\mathbf{Z}^{\mathbf{1}}_{\mathrm{AP}}$) that presently associates $\mathbf{Y}_{\mathrm{MH}}$, it sends RT_S message to $\mathbf{Z}^{\mathbf{1}}_{\mathrm{AP}}$ through DS.
If no active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *History*, then it looks up its *Neighbor List* for any active path to $\mathbf{Y}_{\mathrm{MH}}$.
If an active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *Neighbor List* via some other access point (say, $\mathbf{Z}^{\mathbf{2}}_{\mathrm{AP}}$) presently associating $\mathbf{Y}_{\mathrm{MH}}$, it sends RT_S message to $\mathbf{Z}^{\mathbf{2}}_{\mathrm{AP}}$ through DS.
If no active path to $\mathbf{Y}_{\mathrm{MH}}$ is found in *History* and *Neighbor List*, it follows Step (V.i.) and Step (V.ii.).

V. [*Identify location of destination mobile host*]

i. [*Broadcast* $\mathsf{locateMH}_{\mathbf{Y}_{\mathrm{MH}}}$ *message in* DS]
It broadcasts a $\mathsf{locateMH}_{\mathbf{Y}_{\mathrm{MH}}}$ message to other access points in DS, asking for the presence of $\mathbf{Y}_{\mathrm{MH}}$ in their BSA, and waits for the corresponding $\mathsf{foundMH}_{\mathbf{Y}_{\mathrm{MH}}}$ message for another predefined time interval. Also, it preserves RT from the RT_S message and other details of search information in *Loc Stash*.

ii. [*Receive* $\mathsf{foundMH}_{\mathbf{Y}_{\mathrm{MH}}}$ *message*]
If it receives a $\mathsf{foundMH}_{\mathbf{Y}_{\mathrm{MH}}}$ message from another access point (say, $\mathbf{Z}_{\mathrm{AP}}$), which is presently associating $\mathbf{Y}_{\mathrm{MH}}$), within the time interval, it regenerates the RT_S message, with RT from *Loc Stash*, sends this message to $\mathbf{Z}_{\mathrm{AP}}$ through DS, and updates the corresponding path record of $\mathbf{Y}_{\mathrm{MH}}$ in *History*.
Otherwise, it returns back RT from *Loc Stash* to Host Server.

Figure 4.18: Rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for access point over iBSS: Part 2.

VI. [*Receive* RT_R *message*]

i. [*Source of* RT_R message: *stationary host or access point*]

If it receives an RT_R message from some stationary host or access point through DS, it checks the destination of received message.

If the destination of RT_R message matches with itself, it passes RT from that message to Host Server.

Otherwise, it ignores that message.

ii. [*Source of* RT_R message: *associated mobile host; Send* $\mathsf{ACK}_{\mathsf{RT}_R}$ *message*]

If it receives the RT_R message from an associated mobile host (say, $\mathbf{Y}'_{\mathrm{MH}}$) in BSA, it checks the destination of RT_R message.

If $\mathbf{X}_{\mathrm{AP}}$ is the destination of that message, it next checks *Last RT* to ascertain duplicate receipt.

If RT_R message is not duplicated, it hands over RT from that message to Host Server, records the message identity of that message in *Last RT*, and then sends an $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\mathbf{Y}'_{\mathrm{MH}}$. Otherwise, it discards the RT_R message and resends a new $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\mathbf{Y}'_{\mathrm{MH}}$.

If destination of RT_R message does not match with itself, it ignores that message.

VII. [*Receive* locateMH *message; Send* foundMH *message*]

If it receives a $\mathsf{locateMH}_{\mathbf{Y}''_{\mathrm{MH}}}$ message from some stationary host (say, $\mathbf{Y}'_{\mathrm{SH}}$) or access point (say, $\mathbf{Z}''_{\mathrm{AP}}$) through DS, it checks in *assoc* for $\mathbf{Y}''_{\mathrm{MH}}$.

If such information exists, it generates a $\mathsf{foundMH}_{\mathbf{Y}''_{\mathrm{MH}}}$ message, and sends it to $\mathbf{Y}'_{\mathrm{SH}}$ or $\mathbf{Z}''_{\mathrm{AP}}$ through DS. Otherwise, it ignores that message.

Figure 4.19: Rules of proposed communication mechanism of *CUTSMuHN* platform for access point over iBSS: Part 3.

Communication Process for Mobile Host over IBSS

I. [*Generate* RT_S *message for transfer*]
The Communication Manager of mobile host $\mathbf{X}_{\mathrm{MH}}$ generates an RT_S message, and preserves a copy of it in *Comm Stash*.

II. [*Send* RT_S *message*]

i. [*Wait during standalone mode*]
If $\mathbf{X}_{\mathrm{MH}}$ is in the 'standalone' condition, it waits to get 'connected' with other mobile hosts, and then follows Step (II.ii.).

ii. [*Send* RT_S *message to directly connected mobile host*]
If the destination of RT_S message is another mobile host (say, $\mathbf{Y}_{\mathrm{MH}}$) that is directly connected at present, it sends the RT_S message to $\mathbf{Y}_{\mathrm{MH}}$, and waits for the corresponding $\mathsf{ACK}_{\mathsf{RT}_S}$ message for a predefined time interval.
Otherwise, $\mathbf{X}_{\mathrm{MH}}$ waits to get directly connected with $\mathbf{Y}_{\mathrm{MH}}$, and then repeats this step.

iii. [*Receive* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout*]
If it receives an $\mathsf{ACK}_{\mathsf{RT}_S}$ message from $\mathbf{Y}_{\mathrm{MH}}$ within timeout of the defined interval, it erases the RT_S message from *Comm Stash*.
Moreover, it optionally informs the availability information of $\mathbf{Y}_{\mathrm{MH}}$ to Discovery Manager of $\mathbf{X}_{\mathrm{MH}}$.

iv. [*No* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout*]
If it does not receive an $\mathsf{ACK}_{\mathsf{RT}_S}$ message from $\mathbf{Y}_{\mathrm{MH}}$ within timeout of the present interval, it resends the RT_S message (that is preserved in *Comm Stash*) to $\mathbf{Y}_{\mathrm{MH}}$, and waits for the corresponding $\mathsf{ACK}_{\mathsf{RT}_S}$ message for twice the previous time interval.

v. [*No* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout repeatedly*]
If it has repeated Step (II.iv.) for 3 successive times and still has not received the $\mathsf{ACK}_{\mathsf{RT}_S}$ message within updated time intervals, then it returns back RT from the RT_S message to Host Server of $\mathbf{X}_{\mathrm{MH}}$, and informs the "not reachable" status of $\mathbf{Y}_{\mathrm{MH}}$ to Discovery Manager.

Figure 4.20: Rules of proposed communication mechanism of *CUTSMuHN* platform for mobile host over IBSS: Part 1.

III. [*Receive* RT_R *message; Send* $\mathsf{ACK}_{\mathsf{RT}_R}$ *message*]
If it receives an RT_R message from some other mobile host (say, $\mathbf{Z}_{\mathrm{MH}}$), it checks for the destination of RT_R message.
If $\mathbf{X}_{\mathrm{MH}}$ is itself the destination of that message, it next checks *Last RT* to ascertain duplicate receipt.
If RT_R message is not duplicated, it hands over RT from the RT_R message to Host Server, records the message identity of that message in *Last RT*, and then sends an $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\mathbf{Z}_{\mathrm{MH}}$. Otherwise, it discards RT_R message and resends a new $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\mathbf{Z}_{\mathrm{MH}}$.
If the destination of RT_R message does not match with itself, it ignores that message.

Figure 4.21: Rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for mobile host over IBSS: Part 2.

operations respectively. Each course of interaction in such agent-level approach (shown as solid arrow) is carried out by a pair of message exchanges of RT-message and ACK-message in proposed communication mechanisms at host-level (shown as two successive dashed arrows). This behavior shows that the proposed communication mechanisms are intended to support consistency handling mechanisms of $\mathcal{CUTSMuHN}$ platform (presented in Section 3.3.2 of preceding chapter).

The next section describes the internal architecture of Discovery Manager and Communication Manager of $\mathcal{CUTSMuHN}$ platform to show how the proposed discovery and communication mechanisms are realized in a TSMM.

4.5 Incorporating Proposed Mechanisms in $\mathcal{CUTSMuHN}$

The proposed discovery and communication mechanisms of $\mathcal{CUTSMuHN}$ platform are enforced through Discovery Manager and Communication Manager of $\mathcal{CUTSMuHN}$ platform. Figure 4.25 depicts the generalized view of internal architecture of Discovery Manager and Communication Manager after incorporating the proposed mechanisms. This figure shows the different subcomponents of Discovery Manager and Communication Manager of

Communication Process for Mobile/Gateway Host over H-MANET

I. [*Generate* RT_S *message for transfer*]
The Communication Manager of mobile host $\mathbf{X}_{\mathrm{MH}}$ or gateway host $\mathbf{X}_{\mathrm{GH}}$ generates an RT_S message, and preserves a copy of it in *Comm Stash*. It next checks the destination of RT_S message, and executes the following steps accordingly.

II. [*Send* RT_S *message*]

i. [*Wait during standalone mode of mobile host*]
If $\mathbf{X}_{\mathrm{MH}}$ is in the standalone condition, it waits to get connected with other mobile hosts or gateway hosts.

ii. [*Send* RT_S *message to directly connected host*]
If the destination of RT_S message is a directly connected mobile host $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$, or another mobile host reachable via one/more mobile/gateway hosts with $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$ or $\mathbf{Z}^{\mathbf{1}}_{\mathrm{GH}}$ as immediate host, Communication Manager of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$ sends the RT_S message to $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$ or $\mathbf{Z}^{\mathbf{1}}_{\mathrm{GH}}$, and waits for the corresponding $\mathsf{ACK}_{\mathsf{RT}_S}$ message for a predefined timeout interval.

iii. [*Receive* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout*]
If Communication Manager of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$ receives an $\mathsf{ACK}_{\mathsf{RT}_S}$ message from $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$ or $\mathbf{Z}^{\mathbf{1}}_{\mathrm{GH}}$ within timeout interval, it erases the RT_S message in *Comm Stash*. Moreover, it optionally informs the availability information of $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$ or $\mathbf{Z}^{\mathbf{1}}_{\mathrm{GH}}$ to Discovery Manager of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$.

iv. [*No* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout*]
If it does not receive an $\mathsf{ACK}_{\mathsf{RT}_S}$ message from $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$ or $\mathbf{Z}^{\mathbf{1}}_{\mathrm{GH}}$ within present timeout interval, it resends the RT_S message (that is preserved in *Comm Stash*) to $\mathbf{Y}^{\mathbf{1}}_{\mathrm{MH}}$ or $\mathbf{Z}^{\mathbf{1}}_{\mathrm{GH}}$, and waits for the corresponding $\mathsf{ACK}_{\mathsf{RT}_S}$ message for twice the previous timeout interval.

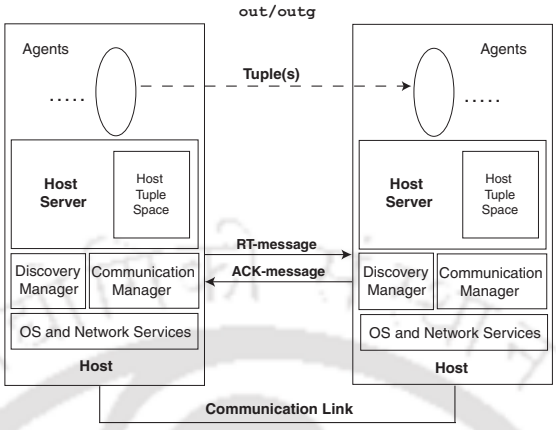
Figure 4.22: Rules of proposed communication mechanism of *CUTSMuHN* platform for mobile/gateway host over H-MANET: Part 1.

v. [*No* $\mathsf{ACK}_{\mathsf{RT}_S}$ *message within timeout repeatedly*]
If it has repeated Step (II.iv.) for 3 successive times, and still has not received an $\mathsf{ACK}_{\mathsf{RT}_S}$ message within updated timeout intervals, then it returns back RT from the RT_S message to Host Server of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$, and informs the "not reachable" status of $\mathbf{Y}^1_{\mathrm{MH}}$ or $\mathbf{Z}^1_{\mathrm{GH}}$ to Discovery Manager.

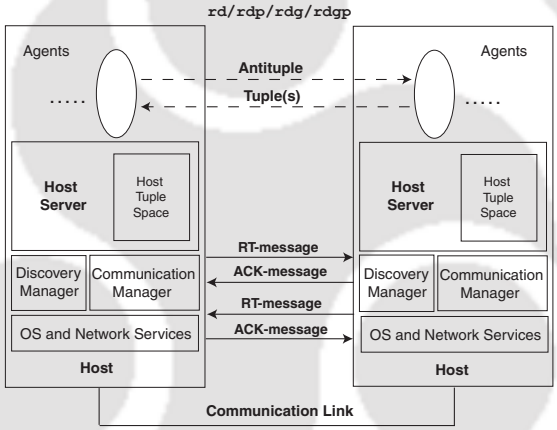
vi. [*Destination mobile host not reachable directly/indirectly*]
If the destination of RT_S message is some other mobile host (say, $\mathbf{Y}''_{\mathrm{MH}}$), which is presently not reachable directly/indirectly, it returns back RT from the RT_S message to Host Server of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$.

III. [*Receive* RT_R *message from mobile/gateway host; Send* $\mathsf{ACK}_{\mathsf{RT}_R}$ *message*]
If Communication Manager of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$ receives an RT_R message from some other mobile host $\widehat{\mathbf{Y}}_{\mathrm{MH}}$ or gateway host $\widehat{\mathbf{Z}}_{\mathrm{GH}}$, it checks the destination of received RT_R message.
If $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$ is itself the destination of that message, it next checks *Last RT* to ascertain duplicate receipt.
If the RT_R message is not duplicated, it hands over RT from the RT_R message to Host Server of $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$, records the message identity of that message in the entry for $\widehat{\mathbf{Y}}_{\mathrm{MH}}$ or $\widehat{\mathbf{Z}}_{\mathrm{GH}}$ in *Last RT*, and then sends an $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\widehat{\mathbf{Y}}_{\mathrm{MH}}$ or $\widehat{\mathbf{Z}}_{\mathrm{GH}}$. Otherwise, it discards the RT_R message and resends a new $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\widehat{\mathbf{Y}}_{\mathrm{MH}}$ or $\widehat{\mathbf{Z}}_{\mathrm{GH}}$.
If $\mathbf{X}_{\mathrm{MH}}/\mathbf{X}_{\mathrm{GH}}$ is not the destination of received RT_R message, it generates a new RT_S message with information from that message, preserves a copy of RT_S message in *Comm Stash*, and then follows Step (II.ii.). Afterwards, it sends an $\mathsf{ACK}_{\mathsf{RT}_R}$ message to $\widehat{\mathbf{Y}}_{\mathrm{MH}}$ or $\widehat{\mathbf{Z}}_{\mathrm{GH}}$.

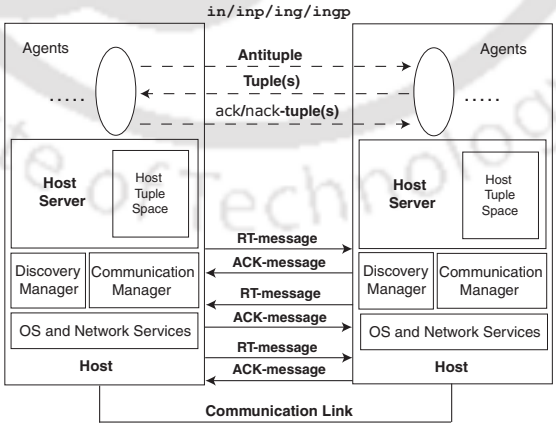
Figure 4.23: Rules of proposed communication mechanism of $\mathcal{CUTSMuHN}$ platform for mobile/gateway host over H-MANET: Part 2.



(a) In case of tuple-producing primitives.



(b) In case of tuple-reading primitives.



(c) In case of tuple-consuming primitives.

Figure 4.24: Host-level communication mechanisms in CUTSMuHN platform to support agent-level three-way approach of remote tuple space operations.

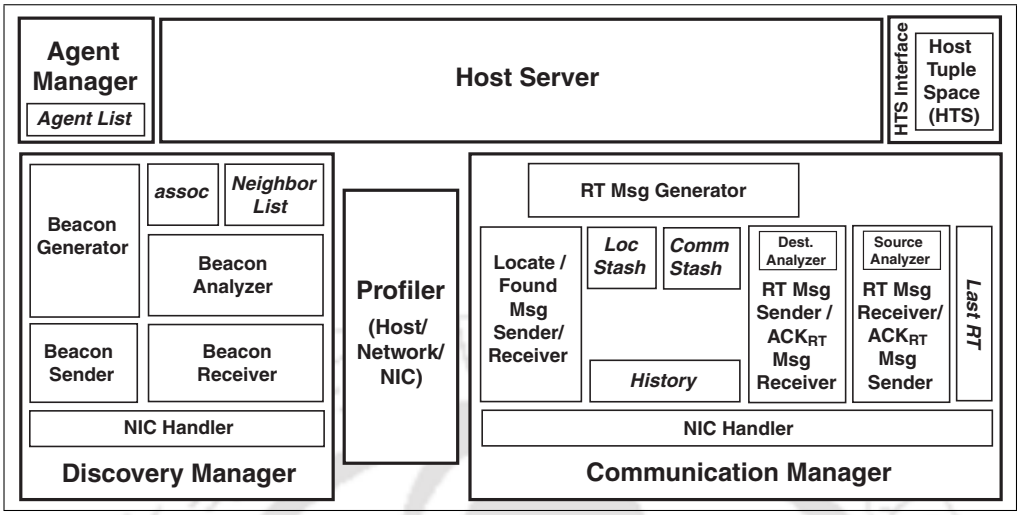


Figure 4.25: Internal architecture of Discovery Manager and Communication Manager of $\mathcal{CUISMuHN}$.

$\mathcal{CUISMuHN}$ platform, each of which implements different rules of the proposed mechanisms. For instance, Beacon Generator of Discovery Manager generates beacons (for stationary host, mobile host, access point and gateway host) to be broadcasted by Beacon Sender. Similarly, Beacon Receiver of Discovery Manager receives the broadcasted beacons, while Beacon Analyzer extracts information from received beacons to update *Neighbor List* and/or *assoc*. Again, RT Msg Generator of Communication Manager generates RT-message (from RT, which it receives from Host Server) to be sent by RT Msg Sender to target host. RT Msg Sender is also responsible to receive an ACK-message corresponding to the sent RT-message. When Communication Manager becomes the destination of data transfer, it uses RT Msg Receiver to receive the RT-message and extract RT from it to hand over to Host Server. It is also responsible to send the corresponding ACK-message (specific to some host, like mobile host, access point, gateway host etc.) back to the source of RT-message. Communication Manager uses *History*, *Loc Stash*, *Comm Stash* and *Last RT* to preserve the different states of data transfer. It also has Locate/Found Msg Sender/Receiver (specific to stationary host and access point) to identify BSA of target mobile host in iBSS for effecting data transfer. Moreover, each of Discovery Manager and Communication Manager also has an NIC Handler to interface with

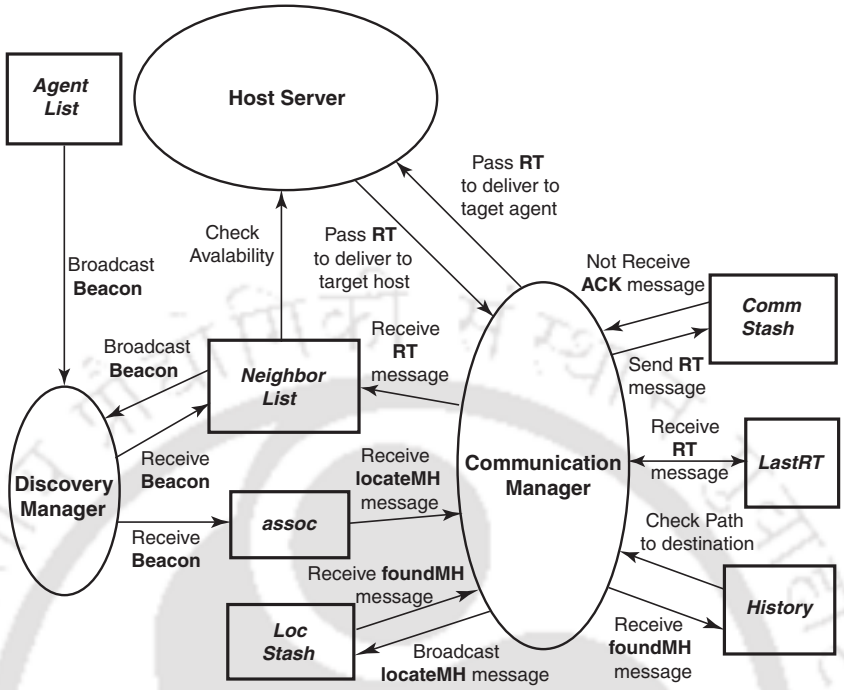


Figure 4.26: Working of Discovery Manager and Communication Manager of *CUTSMuHN*.

the available NIC(s) of device. Figure 4.26 shows the proposed discovery and communication mechanisms being enforced among the subcomponents of Discovery Manager and Communication Manager.

It is to be noted here that some of the subcomponents of Discovery Manager and Communication Manager, shown in the architecture in Figure 4.25, remain deactivated depending on the nature of host. For instance, *assoc* of Discovery Manager as well as *Comm Stash* and *Last RT* of Communication Manager remain deactivated in case of stationary host over iBSS. It is the responsibility of the Profiler of *CUTSMuHN* platform to identify the components and subcomponents in its architecture, which remain deactivated depending on the nature of deployed hosts, networks and NICs.

4.6 Comparative Analysis

The increasing need for coordination support of dynamic applications has led to the emergence of different TSMM platforms. However, some of these TSMM platforms, like TSpaces, JavaSpaces, PageSpace, L^2imbo, TuCSoN, MARS etc., act as coordination

Table 4.1: Comparison of Underlying Heterogeneity of $\mathcal{CUISMuHN}$ with existing TSMM.

Underlying network support	$\mathcal{CUISMuHN}$	**Existing TSMM platforms**		
		LIMONE	EgoSpaces	LIME
Wired network	Yes	—	—	Yes
iBSS	Yes	No	No	No
Wired-wireless bridging	Yes	No	No	No
IBSS or Single-hop wireless ad hoc network	Yes	Yes	Yes	Yes
H-MANET	Yes	No	No	No

platforms for stable networks. In fact, these platforms barely support underlying dynamics, as well as they cannot handle heterogeneity of underlying networks by themselves. Other TSMM platforms, namely LIME, LIMONE, EgoSpaces, TOTA, etc., are, however, developed for mobile and dynamic networks. However, each of them focuses on a specific type of underlying network. In fact, they are developed either for the cellular network or for the single-hop mobile ad hoc network. The multi-hop behavior in underlying networks is not supported in any of them, neither is the support for underlying heterogeneity. Table 4.1 compares the support of underlying networks and network-level heterogeneity of $\mathcal{CUISMuHN}$ platform with existing TSMM platforms, particularly LIMONE, EgoSpaces and LIME.

Compared to existing TSMM platforms, $\mathcal{CUISMuHN}$ platform improves the nature of handling underlying infrastructure in several aspects. (1) The proposed discovery and communication mechanisms of $\mathcal{CUISMuHN}$ platform bridge data communication of wired networks with wireless networks. This functionality is rare among existing TSMM platforms. (2) $\mathcal{CUISMuHN}$ platform supports iBSS, IBSS and H-MANET as its underlying infrastructure, which is non-existent among existing TSMM platforms. $\mathcal{CUISMuHN}$ platform handles unreliability in these networks using its own acknowledgement mechanism, which is included at the middleware-level. (3) Supporting H-MANET suggests that $\mathcal{CUISMuHN}$ platform can handle multi-hop underlying networks. This distinctive ability gives $\mathcal{CUISMuHN}$ platform an edge over existing TSMM platforms. (4) Another distinctive capability of $\mathcal{CUISMuHN}$ platform is to support coordination of interacting

agents over iBSS, IBSS and H-MANET simultaneously, which shows that *CUTSMuHN* platform can support coordination over multiple, heterogeneous, mobile, dynamic and unreliable networks at the same time. (5) Moreover, handling discovery and communication mechanisms of heterogeneous underlying networks at the middleware-level has the advantage of easily extending such mechanisms for newer underlying networks, without disturbing mobile applications as well as network protocol stacks of newer networks. This infers extendibility of *CUTSMuHN* platform. (6) Finally and most importantly, supporting heterogeneity in underlying networks improves flexibility of *CUTSMuHN* platform as well.

4.7 Conclusion

The existing TSMM platforms have the shortcoming of supporting coordination over heterogeneous networks. This chapter has proposed mechanisms of *CUTSMuHN* platform for supporting heterogeneity of unreliable underlying networks. The proposed discovery and communication mechanisms are TSMM-specific, i.e. they are enforced within *CUTSMuHN* platform to handle heterogeneity. This chapter showcases the support of *CUTSMuHN* platform for three distinct underlying networks (viz. iBSS, IBSS and H-MANET) by defining rules of discovery and communication mechanisms for each of them in case of different category of hosts. Combining the benefits of these proposals, this chapter has concluded that *CUTSMuHN* platform not only improves flexibility of TSMM platform, but also shows extendibility of its support for heterogeneity. However, to facilitate application programmers as well as to enable proper analysis of robustness and flexibility of *CUTSMuHN* platform, the formal modeling of its semantics are required to be clearly stated. The modeling also prepares the foundation for its implementation.

Chapter 5

Formal Modeling of *CUTSMuHN*

5.1 Introduction

The preceding chapters (viz. Chapter 3 and Chapter 4) present the design and architecture of *CUTSMuHN* as a TSMM platform that is flexible, robust, versatile, simple, reliable, efficient and scalable during coordination. However, like the design of any software or hardware system, the formal modeling of coordination functionality of *CUTSMuHN* platform is also beneficial in several aspects. Apart from thoroughly describing the behaviors of different constituents of *CUTSMuHN* platform, which are ultimately utilized in expressing its coordination functionality, the formal modeling also defines the precise semantics of agent interactions in *CUTSMuHN* platform, thereby simplifying the task of application programmers. Moreover, the formal modeling of *CUTSMuHN* platform also lays the foundation for its implementation. This chapter proposes an approach of formal modeling of *CUTSMuHN* platform, that can express its coordination ability by simple constructs. The notations of a general-purpose formal reasoning tool, the Mobile UNITY model [97, 82], which is an extension of the well-known UNITY model [24], is used for formal modeling of *CUTSMuHN* platform. Favoring the Mobile UNITY model over other formal tools is due to its suitability in the modeling of inherently nonterminating programs (like any mobile middleware platform), verification and reasoning about the behaviors of their various system components (including agents' temporal behaviors) using its proof rules, and following stepwise specification and refinement.

5.2 Proposed Modeling Approach of $\mathcal{CUTSMuHN}$

The proposed modeling approach principally focuses on representing coordination functionality of $\mathcal{CUTSMuHN}$ platform, and so, some aspects of mobile middleware platforms, not directly related to coordination (like inner details of data transmission in lower layers of network protocol stacks), are abstracted away. In this section, the foundation of proposed modeling of $\mathcal{CUTSMuHN}$ platform is presented first, followed by description of the modeling. The different notations and symbols, used in the proposed modeling approach, are listed in Table 5.1, Table 5.2 and Table 5.3.

Table 5.1: Notations/symbols in proposed modeling of $\mathcal{CUTSMuHN}$, and their meaning.

Notations/Symbols	Meaning
t, τ	Notations of tuple
$\mathbf{t}$	Notation of set of tuples
ε, $\emptyset$	Notations of NULL tuple and empty set of tuples respectively
a	Notation of antituple
f^t, f^a	Notations of constituent fields of tuple and antituple respectively
$\mathsf{f}\cdot$ name, $\mathsf{f}\cdot$ type	Name and Type of tuple/antituple field respectively
$\mathsf{f}\cdot$ val	Value of tuple/antituple field
$\mathfrak{a}(\mathsf{f})$	Predicate to identify *actual* tuple/antituple field)
$\mathfrak{f}(\mathsf{f})$	Predicate to identify *formal* tuple/antituple field)
id	Notation of identity field of tuple/antituple
TupleID	Type of id field of tuple/antituple
$t \uparrow$ id	Indicating identity (i.e. id) field of tuple t
$a \uparrow$ id	Indicating identity (i.e. id) field of antituple a
$\mathfrak{p}(\mathsf{f}^a, \mathsf{f}^t)$	Predicate to specify apposite pairing of f^t and f^a
$\mathfrak{M}(t, a)$	Predicate to express matching of t and a
$\mathtt{mod}(t)$, $\mathtt{mod}(a)$	Macros to return arity of t and a respectively
$\mathbf{T}$	Notation of tuple space
$\mathbf{T}^{\mathsf{p}}$, $\mathbf{T}^{\mathsf{s}}$	Notations of 'Preamble' and 'Tuple Store' of $\mathbf{T}$ respectively
$\mathcal{M}$, $\mathcal{F}$	'Meta index table' and 'Free List' in $\mathbf{T}^{\mathsf{p}}$
ind, ind'	Indices of $\mathbf{T}$ for storing t
$\mathbf{T}^{\mathsf{s}}_{ind}$	Entry of $\mathbf{T}^{\mathsf{s}}$ at index ind
I, $\mathcal{I}$	Notations of 'index table' in $\mathbf{T}^{\mathsf{p}}$
$I\cdot$ name	Name of index table I
$I \bullet ind$	Insertion of ind into index table I
$I.(I \in \mathbf{T}^{\mathsf{p}})$	Nondeterministic selection of index table from $\mathbf{T}^{\mathsf{p}}$

Table 5.2: Notations/symbols in proposed modeling of $\mathcal{CUTSMuHN}$, and their meaning (continued).

Notations/Symbols	Meaning
A, **S**	Notations of sets of apposite tuples and sought tuples respectively
matched(a,**T**)	Predicate to express matching of a with any tuple of **T**
`single`(a,**T**)	Macro to return single sought tuple from **T** matching a
`multiple`(a,**T**)	Macro to return multiple (i.e. bulk) sought tuples from **T** matching a
R_{pr}	Identity of reaction, registered due to primitive `pr` (viz. `rd`, `rdg`, `in`, `ing`)
$\mathsf{m}_{\mathsf{R}_{pr}}$	Mode of reaction R_{pr} (possible values: 'ONCE' and 'ONCE/TUPLE')
$reacted_{\mathsf{R}_{pr}}$	Set of tuples that already reacted to reaction R_{pr} (for 'ONCE/TUPLE' mode)
$enabled_{\mathsf{R}_{pr}}$	Boolean value to enable/disable reaction R_{pr}
`reactrd`(a,**T**)	Macro expressing semantics of reaction registered due to `rd`
`reactrdg`(a,**T**)	Macro expressing semantics of reaction registered due to `rdg`
`reactin`(a,**T**)	Macro expressing semantics of reaction registered due to `in`
`reacting`(a,**T**)	Macro expressing semantics of a reaction registered due to `ing`
$CUTSMuHN$	Mobile UNITY abstraction of $\mathcal{CUTSMuHN}$ (modeled as **System**)
$agent$, $host$	Mobile UNITY abstractions of components of $\mathcal{CUTSMuHN}$: 'agent' and 'host' respectively (each modeled as individual **Program**)
i, k	Representations of particular instances of above components
λ	Notation for location parameter of $agent$ and $host$
$\mathcal{T}_{\mathrm{W}}$, $\mathcal{T}_{\mathrm{WL}}$	Abstractions of transport-layer services of wired and wireless networks
$\mathcal{Q}_{T^S_{a_k}}$, $\mathcal{Q}_{T^R_{a_k}}$	Notations of queues of $agent(k)$ to transfer remote invocation parameters from $agent$ to $host$ and results of such invocation in vice versa
$\mathcal{Q}_{RT_S}$, $\mathcal{Q}_{RT_R}$, $\mathcal{Q}_S$ $\mathcal{Q}_R$, $\mathcal{Q}_{S_{\mathrm{W}}}$, $\mathcal{Q}_{S_{\mathrm{WL}}}$ $\mathcal{Q}_{S_B}$, $\mathcal{Q}_{R_B}$, $\mathcal{Q}_{S_{RT}}$ $\mathcal{Q}_{R_{RT}}$, $\mathcal{Q}_{in}$, $\mathcal{Q}_{out}$	Notations of queues of $host(i)$ for interfacing with their respective discovery module, communication module and transport-layer service abstractions, as well as for specifying registration and deregistration of active agents
`head`($\mathcal{Q}$), `tail`($\mathcal{Q}$)	Methods to respectively return the front element and all elements except the front element of queue $\mathcal{Q}$
$\mathcal{Q} \bullet X$	Abstraction of inserting element X at rear of queue Q
M, m	Notations of messages in $CUTSMuHN$; comprising of: $\{mid, src, dest, kind, data, ni\}$
`newMsg`	Method to generate above message, given its $src, dest, kind, data, ni$
$m{\cdot}\langle fname\rangle$	Notation to access a field $fname$ of message m
BCON, RT, ACK Locate, Found	Possible type of messages used in $CUTSMuHN$
$\uparrow$	Notation to identify/access a field in (ordered/unordered) data structure
$a, aid, taid, \mathcal{A}$, A	Representations of identity of 'agent'
$hid, dstid$	Representations of identity of 'host'
$\perp$	Notation of 'Emptiness' (like, empty queue/list/value)
$nwdeploy$	Representation of deployed network for $CUTSMuHN$
$\rightarrow$	Notation of transient-sharing among different instances of components

Table 5.3: Notations/symbols in proposed modeling of $\mathcal{CUTSMuHN}$, and their meaning (continued).

Notations/Symbols	Meaning
Γ, Γ'	Availability relation of $agent$ in $host$, and connectivity relation of two $host$
$status$	Representation of status of $host$
$\mathcal{N}$, $\mathcal{N}'$, $assoc$	Notations of *Neighbor Lists* and *assoc* of $host$
$\mathcal{H}$, $\mathcal{L}'$, $\mathcal{CS}$	Notations of *History*, *Loc Stash* and *Comm Stash* of $host$
$\mathcal{LRT}$	Notation of *Last RT* of $host$

5.2.1 Modeling of Tuple Space Coordination

The foundation of proposed modeling approach lies in specifying the tuple space model (including its constituents and primitives) of $\mathcal{CUTSMuHN}$ platform.

An unordered tuple of $\mathcal{CUTSMuHN}$ platform is denoted as t, while its unordered antituple is denoted as a. Any constituent field of t or a is denoted as f. A predicate $\mathfrak{field}$ is defined to specify whether f is a field of t or a. That is, if $\mathfrak{field}(\mathsf{f}, t)$ holds, then f is a field of t (denoted as f^t), or $\mathfrak{field}(\mathsf{f}, a)$ holds when f is a field of a (denoted as f^a). Whether a field is *actual*/*formal* is indicated by two predicates: for field f that is *actual*, $\mathfrak{a}(\mathsf{f})$ holds, whereas for a *formal* field f', $\mathfrak{f}(\mathsf{f}')$ holds. Moreover, NAME, TYPE and VALUE (for *actual* field) of f is denoted by $\mathsf{f}\cdot\mathsf{name}$, $\mathsf{f}\cdot\mathsf{type}$ and $\mathsf{f}\cdot\mathsf{val}$ respectively, where type specifies polymorphically-related types of f. For forming a tuple, `buildTuple`() method is called with tuple arity and all constituent fields (i.e. all *actual*) as parameters. Similarly, `buildAntituple`() method forms an antituple, given its constituent fields that carry some search criteria. An additional field (denoted as id) is added to t and a while they are formed. In a tuple space, id of a tuple uniquely identifies it from other tuples. This field is of type $\mathsf{TupleID}$, and it gets its value from `newTupleID`() method before being included in t. In antituples, id is a *formal*. For acquiring the identity value of t, $t \uparrow \mathsf{id}$ is used, whereas $a \uparrow \mathsf{id}$ returns *formal* of id (i.e. $\mathsf{TupleID}$). For matching t and a in $\mathcal{CUTSMuHN}$ platform, all the following conditions must hold:

(1) if arity of a is 0, then a matches t;

(2) arity of a must be less than or equal to that of t;

(3) each f^a must successfully lookup a appositely-paired[1] f^t, such that $\mathsf{f}^a\cdot\mathsf{name} = \mathsf{f}^t\cdot\mathsf{name}$ and $\mathsf{f}^a\cdot\mathsf{type} = \mathsf{f}^t\cdot\mathsf{type}$; successful apposite pairing is specified by predicate $\mathfrak{p}(\mathsf{f}^a, \mathsf{f}^t)$;

(4) if f^a is *actual*, $\mathfrak{p}(\mathsf{f}^a, \mathsf{f}^t)$ (from condition (3)) holds and $\mathsf{f}^a\cdot\mathsf{val} = \mathsf{f}^t\cdot\mathsf{val}$, then f^a matches f^t;

(5) if f^a is *formal* and $\mathfrak{p}(\mathsf{f}^a, \mathsf{f}^t)$ (from condition (3)) holds, then f^a matches f^t;

(6) If "f^a matches f^t" holds for all constituent fields of a, then a matches t.

The predicate $\mathfrak{M}(t, a)$ is next defined to express the above conditions in tuple-antituple matching operation in $\mathcal{CUTSMuHN}$ platform:

$$\begin{aligned}\mathfrak{M}(t,a) \equiv\ & \big(\mathtt{mod}(a) = 0\big)\vee \\ & \Big((\mathtt{mod}(a) > 0) \wedge (\mathtt{mod}(a) \leq \mathtt{mod}(t))\wedge \\ & \big\langle \forall j : 1 \leq j \leq \mathtt{mod}(a) \\ & \quad :: \big\langle \exists i : 1 \leq i \leq \mathtt{mod}(t) :: \mathfrak{p}(\mathsf{f}_j^a, \mathsf{f}_i^t) \wedge \big((\mathfrak{a}(\mathsf{f}_j^a) \wedge (\mathsf{f}_j^a\cdot\mathsf{val} = \mathsf{f}_i^t\cdot\mathsf{val})) \vee \mathfrak{f}(\mathsf{f}_j^a)\big)\big\rangle \ \big\rangle \Big) \end{aligned} \tag{5.1}$$

where, $\mathfrak{p}(\mathsf{f}^a, \mathsf{f}^t) \equiv \big((\mathsf{f}^a\cdot\mathsf{name} = \mathsf{f}^t\cdot\mathsf{name}) \wedge (\mathsf{f}^a\cdot\mathsf{type} = \mathsf{f}^t\cdot\mathsf{type})\big)$, as well as $\mathtt{mod}(t)$ and $\mathtt{mod}(a)$ represent arity of t and a respectively.

A tuple space of $\mathcal{CUTSMuHN}$ platform is denoted as $\mathbf{T}$, whereas Preamble and Tuple Store of $\mathbf{T}$ are denoted as $\mathbf{T}^{\mathsf{p}}$ and $\mathbf{T}^{\mathsf{s}}$ respectively. Also, Meta index table and Free List of Preamble are denoted as $\mathcal{M}$ and $\mathcal{F}$ respectively. The process of tuple insertion in a tuple space, i.e. when t is going to be stored in $\mathbf{T}$ at index ind, is specified as:

$$\begin{aligned} & \mathbf{T}^{\mathsf{s}}_{ind} := t \\ \| \ \big\langle \ \| \ i : 1 \leq i \leq \mathtt{mod}(t) :: \langle \| \ I : I\cdot\mathsf{name} = \mathsf{f}_i^t\cdot\mathsf{name} :: & \quad I := I \bullet ind \qquad \text{if } (I \in \mathbf{T}^{\mathsf{p}}) \\ & \quad \| \ I, \mathbf{T}^{\mathsf{p}} := I \bullet ind, \mathbf{T}^{\mathsf{p}} \cup \{I\} \quad \text{if } \neg(I \in \mathbf{T}^{\mathsf{p}})\rangle\big\rangle \end{aligned} \tag{5.2}$$

where, I denotes any index table, and $I \bullet ind$ specifies the insertion of ind into I. In the above specification, when t is stored at ind of $\mathbf{T}$, all index tables identified by the constituent fields of t are updated with the value of ind. The process of searching tuple(s) from a tuple space (i.e. searching tuple(s) matching a within $\mathbf{T}$) requires tuple-antituple

[1]Section 2.2.5 of Chapter 2 explains about apposite pairing.

matching operation. At the start of tuple-antituple matching operation, a identifies an index table $\mathcal{I}$ in $\mathbf{T}^{\mathsf{p}}$ (where, $I.(I \in \mathbf{T}^{\mathsf{p}})$ specifies the nondeterministic selection [7] of an index table from $\mathbf{T}^{\mathsf{p}}$):

$$\begin{aligned} \mathcal{I} :=& I.(I \in \mathbf{T}^{\mathsf{p}}) && \text{if } (\mathtt{mod}(a) = 0) \ \sim \\ & \langle \exists\ I, i : (1 \leq i \leq \mathtt{mod}(a)) \wedge (I \in \mathbf{T}^{\mathsf{p}}) :: I \quad \text{if } (I.\mathsf{name} = \mathsf{f}_i^a.\mathsf{name}) \rangle \quad \text{if } (\mathtt{mod}(a) > 0) \end{aligned} \tag{5.3}$$

All tuples, indexed by $\mathcal{I}$, form the set of apposite tuples (denoted by $\mathbf{A}$) for a in $\mathbf{T}$:

$$\langle \parallel ind : (ind \in \mathcal{I}) :: \mathbf{A} := (\mathbf{A} \cup \{t\}) \quad \text{if } (t = \mathbf{T}^{\mathsf{s}}_{ind}) \rangle \tag{5.4}$$

In the above three-part notation, operator "||" serves the purpose of creating a quantified statement (unlike its usual purpose of denoting parallel execution of substatements in a Mobile UNITY statement). The set of sought tuples, denoted by $\mathbf{S}$ for a in $\mathbf{T}$, is derived from $\mathbf{A}$ in *CUTSMuHN* platform as:

$$\langle \parallel t : (t \in \mathbf{A}) \wedge \mathfrak{M}(t, a) :: \mathbf{S} := \mathbf{S} \cup \{t\} \rangle \tag{5.5}$$

The predicate $\mathbf{matched}(a, \mathbf{T})$ is defined to express the fact that at least one tuple of $\mathbf{A}$ (i.e. any tuple of $\mathbf{S}$) matches a, where ε represents a NULL tuple:

$$\mathbf{matched}(a, \mathbf{T}) \equiv \langle \exists t :: (t \in \mathbf{A}) \wedge \mathfrak{M}(t, a) \wedge \neg(t = \varepsilon) \rangle \equiv \langle \exists t :: (t \in \mathbf{S}) \wedge \neg(t = \varepsilon) \rangle \tag{5.6}$$

The macro $\mathtt{single}(a, \mathbf{T})$ returns a single sought tuple, as required by single tuple-reading/consuming primitives, where τ represents any tuple of $\mathbf{S}$:

$$\mathtt{single}(a, \mathbf{T}) \triangleq \langle \parallel t : t = \tau.(\tau \in \mathbf{S}) :: t \rangle \tag{5.7}$$

In the above specification, τ is nondeterministically selected from $\mathbf{S}$, binding it to tuple t and quantifying in three-part notation. For cases, where multiple tuples are expected as outcome (like, bulk primitives), the macro $\mathtt{multiple}(a, \mathbf{T})$ is used:

$$\mathtt{multiple}(a, \mathbf{T}) \triangleq \langle \mathbf{set}\ t : t \in \mathbf{S} :: t \rangle \tag{5.8}$$

The tuple space model of *CUTSMuHN* platform provides both local and remote tuple space primitives as APIs to applications. The local primitives are expressed in Equation (5.9), Equation (5.10), Equation (5.11), Equation (5.12), Equation (5.13) and Equation (5.14). From hereon, **t** is used to denote a set of tuples.

$$\begin{aligned}\texttt{out}(t, \mathbf{T}) \triangleq \Big\langle \parallel t :: \ \mathbf{T}^{\mathsf{s}}_{ind} := t \\ \parallel \Big\langle \parallel i : 1 \leq i \leq \texttt{mod}(t) :: \langle \parallel I : I{\cdot}\,\mathsf{name} = \mathsf{f}^t_i{\cdot}\,\mathsf{name} \\ :: \quad I := I \bullet ind \qquad\qquad \text{if } (I \in \mathbf{T}^{\mathsf{p}}) \\ \parallel I, \mathbf{T}^{\mathsf{p}} := I \bullet ind, \mathbf{T}^{\mathsf{p}} \cup \{I\} \quad \text{if } \neg(I \in \mathbf{T}^{\mathsf{p}})\rangle\rangle\Big\rangle \end{aligned} \tag{5.9}$$

Equation (5.9) specifies the operation of `out` primitive, which follows the expression of tuple insertion specified in Equation (5.2). The identities of reference/target agents are not explicitly required in the specification of `out`; they can be suitably added while modeling the agent. Also, the method of determining the value of *ind* as well as the method of creating a new index table I to store *ind* are abstracted away in Equation (5.9) to keep it simple and apprehensible.

Equation (5.10) specifies the operation of `outg` primitive.

$$\begin{aligned}\texttt{outg}(\mathbf{t}, \mathbf{T}) \triangleq \Big\langle \forall t : t \in \mathbf{t} :: \ \mathbf{T}^{\mathsf{s}}_{\mathsf{i}} := t \\ \parallel \Big\langle \parallel i : 1 \leq i \leq \texttt{mod}(t) :: \langle \parallel I : I{\cdot}\,\mathsf{name} = \mathsf{f}^t_i{\cdot}\,\mathsf{name} \\ :: \quad I, \mathbf{T}^{\mathsf{p}} := I \bullet ind, \mathbf{T}^{\mathsf{p}} \cup \{I\} \quad \text{if } \neg(I \in \mathbf{T}^{\mathsf{p}}) \\ \parallel I := I \bullet ind \qquad\qquad \text{if } (I \in \mathbf{T}^{\mathsf{p}})\rangle\rangle\Big\rangle \end{aligned} \tag{5.10}$$

The above expression is similar to Equation (5.9), except that the insertion process repeats for all tuples present in set **t**.

Equation (5.11) specifies the operation of `rdp` primitive.

$$\begin{aligned} t := \texttt{rdp}(a, \mathbf{T}) \triangleq \quad & t := \texttt{single}(a, \mathbf{T}) \quad \text{if } \mathbf{matched}(a, \mathbf{T}) \\ \parallel \ & t := \varepsilon \qquad\qquad \text{if } \neg\mathbf{matched}(a, \mathbf{T}) \end{aligned} \tag{5.11}$$

In the above expression, macro $\texttt{single}(a, \mathbf{T})$ (defined in Equation (5.7)) and predicate $\textbf{matched}(a, \mathbf{T})$ (defined in Equation (5.6)) are used. Both `single` and **matched** macros, in turn, involve the use of $\mathbf{A}$, $\mathbf{S}$ and tuple-antituple matching operation in their definitions.

Equation (5.12) specifies the operation of `rdgp` primitive.

$$\begin{aligned} \mathbf{t} := \texttt{rdgp}(a, \mathbf{T}) \triangleq \quad & \mathbf{t} := \texttt{multiple}(a, \mathbf{T}) && \text{if } \textbf{matched}(a, \mathbf{T}) \\ \| \ & \mathbf{t} := \emptyset && \text{if } \neg\textbf{matched}(a, \mathbf{T}) \end{aligned} \tag{5.12}$$

The above expression differs from Equation (5.11) in the use of $\texttt{multiple}(a, \mathbf{T})$ macro (defined in Equation (5.8)), instead of `single` macro, and receiving the output in set $\mathbf{t}$, instead of a single tuple t. Consequently, Equation (5.11) and Equation (5.12) express single and bulk local tuple-reading primitives respectively.

Equation (5.13) specifies the operation of `inp` primitive.

$$\begin{aligned} t := \texttt{inp}(a, \mathbf{T}) \triangleq \quad & t, \mathbf{T} := \texttt{single}(a, \mathbf{T}), (\mathbf{T} \setminus \texttt{single}(a, \mathbf{T})) && \text{if } \textbf{matched}(a, \mathbf{T}) \\ \| \ & t := \varepsilon && \text{if } \neg\textbf{matched}(a, \mathbf{T}) \end{aligned} \tag{5.13}$$

The above expression is similar to Equation (5.11), except that $\mathbf{T}$ is also updated (thereby expressing tuple withdrawal) in case when predicate **matched** becomes TRUE.

Equation (5.14) specifies the operation of `ingp` primitive.

$$\begin{aligned} \mathbf{t} := \texttt{ingp}(a, \mathbf{T}) \triangleq \quad & \mathbf{t}, \mathbf{T} := \texttt{multiple}(a, \mathbf{T}), (\mathbf{T} \setminus \texttt{multiple}(a, \mathbf{T})) && \text{if } \textbf{matched}(a, \mathbf{T}) \\ \| \ & \mathbf{t} := \emptyset && \text{if } \neg\textbf{matched}(a, \mathbf{T}) \end{aligned} \tag{5.14}$$

The above expression differs from Equation (5.13) in the use of `multiple`, instead of `single`, and receiving the outcome in $\mathbf{t}$, instead of t. So, Equation (5.13) and Equation (5.14) express single and bulk local tuple-consuming primitives respectively.

Before expressing the semantics of remote tuple space primitives of *CUTSMuHN* platform, it is required to model its reactivity. For supporting reactivity, two variables are defined to keep track of tuples that have already reacted and to enable/disable reactions, viz. $reacted_{\mathsf{R}}$ and $enabled_{\mathsf{R}}$ respectively. Adding reaction identity (denoted as R)

to these two variables ensures that they cannot interfere with each other operation. The subscript in the notation of R depends on invoked primitive for which the reaction is generated. For instance, R_{rd} represents a reaction for primitive `rd`, while R_{ing} is due to `ing` and so on. The mode of reaction is denoted as m, and its possible values are ONCE and ONCE/TUPLE. The following macros define the semantics of registered reactions for different blocking operations; these macros, in turn, use the "reacts-to" construct of Mobile UNITY.

$$\texttt{reactrd}(a, \mathbf{T}) \triangleq$$
$$\Big\langle \| \tau : \tau = \tau'. \Big((\tau' \in \mathbf{S}) \wedge \big((\mathsf{m}_{\mathsf{R}_{rd}} = \mathsf{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{rd}} = \emptyset) \big) \wedge \big((\mathsf{m}_{\mathsf{R}_{rd}} = \mathsf{ONCE/TUPLE}) \Rightarrow (\tau' \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{rd}}) \big) \Big) :: \tau \| reacted_{\mathsf{R}_{rd}} := reacted_{\mathsf{R}_{rd}} \cup \{\tau \cdot \mathsf{id}\} \Big\rangle \quad \text{reacts-to } \big(\mathbf{matched}(a, \mathbf{T}) \wedge enabled_{\mathsf{R}_{rd}} \big) \tag{5.15}$$

$$\texttt{reactrdg}(a, \mathbf{T}) \triangleq$$
$$\Big\langle \mathbf{set}\ \tau : (\tau \in \mathbf{S}) \wedge \big((\mathsf{m}_{\mathsf{R}_{rdg}} = \mathsf{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{rdg}} = \emptyset) \big) \wedge \big((\mathsf{m}_{\mathsf{R}_{rdg}} = \mathsf{ONCE/TUPLE}) \Rightarrow (\tau \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{rdg}}) \big) :: \tau \| reacted_{\mathsf{R}_{rdg}} := reacted_{\mathsf{R}_{rdg}} \cup \{\tau \cdot \mathsf{id}\} \Big\rangle \quad \text{reacts-to } \big(\mathbf{matched}(a, \mathbf{T}) \wedge enabled_{\mathsf{R}_{rdg}} \big) \tag{5.16}$$

$$\texttt{reactin}(a, \mathbf{T}) \triangleq$$
$$\Big\langle \| \tau : \tau = \tau'. \Big((\tau' \in \mathbf{S}) \wedge \big((\mathsf{m}_{\mathsf{R}_{in}} = \mathsf{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{in}} = \emptyset) \big) \wedge \big((\mathsf{m}_{\mathsf{R}_{in}} = \mathsf{ONCE/TUPLE}) \Rightarrow (\tau' \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{in}}) \big) \Big) :: \tau \| \mathbf{T}, reacted_{\mathsf{R}_{in}} := \mathbf{T} \setminus \{\tau\}, reacted_{\mathsf{R}_{in}} \cup \{\tau \cdot \mathsf{id}\} \Big\rangle \quad \text{reacts-to } \big(\mathbf{matched}(a, \mathbf{T}) \wedge enabled_{\mathsf{R}_{in}} \big) \tag{5.17}$$

$$\texttt{reacting}(a, \mathbf{T}) \triangleq$$
$$\Big\langle \mathbf{set}\ \tau : (\tau \in \mathbf{S}) \wedge \big((\mathsf{m}_{\mathsf{R}_{ing}} = \mathsf{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{ing}} = \emptyset) \big) \wedge \big((\mathsf{m}_{\mathsf{R}_{ing}} = \mathsf{ONCE/TUPLE}) \Rightarrow (\tau \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{ing}}) \big) :: \tau \| \mathbf{T}, reacted_{\mathsf{R}_{ing}} := \mathbf{T} \setminus \{\tau\}, reacted_{\mathsf{R}_{ing}} \cup \{\tau \cdot \mathsf{id}\} \Big\rangle \quad \text{reacts-to } \big(\mathbf{matched}(a, \mathbf{T}) \wedge enabled_{\mathsf{R}_{ing}} \big) \tag{5.18}$$

The macro `reactrd`$(a,\mathbf{T})$, defined in Equation (5.15), expresses the semantics of a reaction registered on behalf of invoke of `rd`$(a,\mathbf{T})$. Each macro contains a three-part quantified-assignment statement to express reaction firing based on stated conditions. These conditions are justifiably incorporated within the assignment statement as boolean expressions, as well as added to reactive clauses of assignment statement, like, checking different conditions of $reacted_{\mathsf{R}_{rd}}$ for different values of $\mathsf{m}_{\mathsf{R}_{rd}}$. Similarly, the macros `reactrdg`$(a,\mathbf{T})$, `reactin`$(a,\mathbf{T})$, and `reacting`$(a,\mathbf{T})$ are defined in Equation (5.16), Equation (5.17) and Equation (5.18) respectively. The macros `reactin` and `reacting` differ from the macros `reactrd` and `reactrdg` in that the earlier ones additionally update the state of tuple space $\mathbf{T}$ to effectuate tuple withdrawal.

The remote tuple space primitives of *CUTSMuHN* platform are both blocking as well as nonblocking. For instance, the remote variants of `out` and `outg` are nonblocking in nature and thus are expressed like their local variants defined in Equation (5.9) and Equation (5.10) respectively. Similarly, the remote primitives `rdp`, `rdgp`, `inp` and `ingp` are nonblocking and are expressed same as local ones defined in Equation (5.11), Equation (5.12), Equation (5.13) and Equation (5.14) respectively. Only the remote primitives `rd`, `rdg`, `in` and `ing` are blocking and are expressed in the following. The blocking primitives require the support of reactions for their successful execution. These primitives either execute themselves or generate reactions. Equation (5.19) specifies the operation of `rd` primitive.

$$\begin{aligned} t := \texttt{rd}(a,\mathbf{T}) \triangleq \quad & t := \texttt{single}(a,\mathbf{T}) \quad \text{if } \mathbf{matched}(a,\mathbf{T}) \\ \| \; & t := \texttt{reactrd}(a,\mathbf{T}) \quad \text{if } \neg\mathbf{matched}(a,\mathbf{T}) \end{aligned} \tag{5.19}$$

In the above expression, when sought tuple is not available (i.e. predicate **matched** becomes FALSE), a reaction is registered that eventually responds with (available) sought tuple using macro `reactrd`$(a,\mathbf{T})$.

Equation (5.20) specifies the operation of `rdg` primitive.

$$\begin{aligned} \mathbf{t} := \texttt{rdg}(a,\mathbf{T}) \triangleq \quad & \mathbf{t} := \texttt{multiple}(a,\mathbf{T}) \quad \text{if } \mathbf{matched}(a,\mathbf{T}) \\ \| \; & \mathbf{t} := \texttt{reactrdg}(a,\mathbf{T}) \quad \text{if } \neg\mathbf{matched}(a,\mathbf{T}) \end{aligned} \tag{5.20}$$

The above expression differs from Equation (5.19) in the use of `multiple`, instead of `single`, and `reactrdg`, instead of `reactrd`, as well as receiving the result in **t**, instead of t. So, Equation (5.19) and Equation (5.20) express single and bulk blocking tuple-reading primitives respectively.

Equation (5.21) specifies the operation of `in` primitive.

$$t := \texttt{in}(a, \mathbf{T}) \triangleq \begin{array}{ll} t, \mathbf{T} := \texttt{single}(a, \mathbf{T}), (\mathbf{T} \setminus \texttt{single}(a, \mathbf{T})) & \text{if } \mathbf{matched}(a, \mathbf{T}) \\ \| \; t := \texttt{reactin}(a, \mathbf{T}) & \text{if } \neg\mathbf{matched}(a, \mathbf{T}) \end{array} \quad (5.21)$$

The above expression resembles Equation (5.19), except that tuple is withdrawn from **T** when predicate **matched** becomes TRUE and macro `reactin` is used to handle reactivity instead of `reactrd`.

Equation (5.22) specifies the operation of `ing` primitive.

$$\mathbf{t} := \texttt{ing}(a, \mathbf{T}) \triangleq \begin{array}{ll} \mathbf{t}, \mathbf{T} := \texttt{multiple}(a, \mathbf{T}), (\mathbf{T} \setminus \texttt{multiple}(a, \mathbf{T})) & \text{if } \mathbf{matched}(a, \mathbf{T}) \\ \| \; \mathbf{t} := \texttt{reacting}(a, \mathbf{T}) & \text{if } \neg\mathbf{matched}(a, \mathbf{T}) \end{array} \quad (5.22)$$

The above expression is the bulk variant of Equation (5.21), as macro `multiple` is used, instead of `single`, and results are preserved in **t**, instead of t. For verifying the correctness of above specifications of the tuple space model of $\mathcal{CUTSMuHN}$ platform, it is obligatory to prove their safety and progress properties based on the Mobile UNITY model's proof logic. The next chapter presents the verification approaches.

Based on the specifications of $\mathcal{CUTSMuHN}$ platform, next section formally describes the proposed modeling approach, which includes the Mobile UNITY **System** expressing the semantics of $\mathcal{CUTSMuHN}$ platform as well as the constituent Mobile UNITY '**Program**s' expressing its components.

5.2.2 Modeling of Entire Middleware Platform

This section formally expresses $\mathcal{CUTSMuHN}$ platform as a Mobile UNITY '**System**', comprising of a set of formal programs (i.e. Mobile UNITY '**Program**s') representing

System *CUTSMuHN*

Program $host(i)$ **at** λ

$\vdots \quad \vdots$ {Program description of $host(i)$, given separately}

end

Program $agent(k)$ **at** λ

$\vdots \quad \vdots$ {Program description of $agent(k)$, given separately}

end

Components

$\langle [] \; i :: host(i) \rangle \; [] \; \langle [] \; k :: agent(k) \rangle$

Interactions

$\vdots \quad \vdots$ {Statements of **Interactions** section, given separately}

end

Figure 5.1: Mobile UNITY '**System** *CUTSMuHN*'.

multiple instances of its different static and mobile interacting components (i.e. agents and hosts). However, many aspects of *CUTSMuHN* platform are not directly expressed in this modeling approach to keep it simple. Among these aspects, expressing the mechanism to handle agent mobility (i.e. migration of agents from one host to another) is already described in literature [95, 98].

System *CUTSMuHN*

System *CUTSMuHN*, shown in Figure 5.1, models the functionalities of *CUTSMuHN* platform as multiple instances of only two Mobile UNITY **Program**s (viz. *agent* and *host*), and their interactions are specified in its **Interactions** section. In the proposed modeling approach, i-th host is specified by **Program** $host(i)$, whereas k-th agent is represented by **Program** $agent(k)$, where i and k are assumed to be quantified over appropriate ranges. The instantiation of these **Program**s are specified in the **Components** section of **System** *CUTSMuHN*. Each of these **Program**s is parameterized by its location, denoted using 'λ'. The significance of λ lies in the fact that *agent* and *host*

communicate when colocate, or two different *host*s communicate when they are within their transmission ranges. That is, an *agent* can only communicate directly with its supported *host* having same value of λ. Similarly, one *host* can communicate with another reachable *host* having same λ value.

For communication between multiple instances of *host*, the "transient-sharing" construct of the Mobile UNITY model is followed, which is based on transmission range proximity. A transiently-shared variable $\mathcal{T}$ is used to model communication between any pair of *host* instances. Every *host* instance holds $\mathcal{T}$; each $\mathcal{T}$ corresponds to an NIC. For instance, the transiently-shared variable $\mathcal{T}_{\mathrm{W}}$ abstracts the working of underlying transport-layer services of mobile/portable device having a wired NIC. On the other hand, $\mathcal{T}_{\mathrm{WL}}$ abstracts the transport-layer services of wireless NIC. These transiently-shared variables are capable of expressing dynamics, mobility and unreliability issues in a TSMM platform. The proposed modeling approach also captures the realities of dynamic environment in mobile computing paradigm. For instance, the use of network-level addressing is not a obligatory issue of $\mathcal{CUTSMuHN}$ platform, and so communication abstractions are allowed to use any user-defined addressing, including IP addressing scheme, to uniquely identify the components of *CUTSMuHN*.

Different variables are used in this modeling approach to express different behaviors of hosts and agents of $\mathcal{CUTSMuHN}$ platform. For instance, $\mathcal{Q}$ is used to express any queue that is defined to interface different components of *CUTSMuHN*; each of its subscript denotes the purpose of using it. In this modeling approach, $\texttt{head}(\mathcal{Q})$ and $\texttt{last}(\mathcal{Q})$ returns the front element and rear element of $\mathcal{Q}$ respectively, while $\texttt{tail}(\mathcal{Q})$ returns all elements of $\mathcal{Q}$ except the front element. Also, $\mathcal{Q} \bullet X$ is used to denote the insertion of element X in the rear end of $\mathcal{Q}$ and returning the updated $\mathcal{Q}$. For instance, $\mathcal{Q} \bullet m$ updates $\mathcal{Q}$ by inserting message m in its rear end. Each message m comprises of its message identity (denoted as *mid*), source host's identity (denoted as *src*), destination host's identity (denoted as *dest*), type of message (denoted as *kind*), data encapsulated within message (denoted as *data*) and network interface to transmit the message (denoted as *ni*). The message m is generated by calling the method $\texttt{newMsg}(src, dest, kind, data, ni)$, which inserts its *mid* to return the newly-generated complete message. Each field of

m is accessed by appending the field name to m, viz. $m{\cdot}\,src$ or $m{\cdot}\,dest$. The possible type of messages used in this modeling approach include 'BCON', 'RT', 'ACK', 'Locate' and 'Found' messages. The BCON messages are used to abstract path discovery for data transfer between hosts, whereas the RT messages are used to model the carry of communicating information. The ACK messages are used to abstract reliability in data communication through unreliable, dynamic environment. The Locate and Found messages deal with discovery of communicating hosts in different underlying networks. While accessing each entry e of any ordered data structure (like *Neighbor List*), each field of e is specified by $\uparrow$ followed by the position of that field. For instance, the second field of e is accessed using '$e \uparrow 2$'. However, for every unordered data structure (like data tuple), each field is accessed by $\uparrow$, followed by name of that field; for example, the field InvokeResults of tuple t is accessed by '$t \uparrow$ InvokeResults'.

The interactions among **Program**s of **System** *CUTSMuHN* are specified in its **Interactions** section, shown in Figure 5.2. In the **Interactions** section, different conditions for interactions between any pair of these **Program**s are enforced through **when**, **engage** and **disengage** clauses. These clauses, along with **current** and $\perp$ constructs, are used for effecting transient-sharing (its notation being '$\rightarrow$' [82, p. 106]) among different hosts. Also, the first four statements in **Interactions** section, labeled as $shared_{\text{WiBSS}}$, $shared_{\text{WL}_{\text{iBSS}}}$, $shared_{\text{WL}_{\text{IBSS}}}$ and $shared_{\text{WL}_{\text{HMANET}}}$ in Figure 5.2, are reactive statements as they have used "$\approx$" construct of the Mobile UNITY model. The first interaction statement, labeled as $shared_{\text{WiBSS}}$, is used to transfer messages between wired NICs of $host(i)$ and $host(j)$. In this statement, the transfer is only possible, when both $host(i)$ and $host(j)$ are connected in iBSS network, i.e. $host(i)\Gamma' host(j)$. The connectivity relation, denoted by "Γ'", expresses the connection status between two different device environments (in this case, $host(i)$ and $host(j)$), while delivering messages between them. This statement is also predicated on the same deployed network (viz. iBSS). The second interaction statement, labeled as $shared_{\text{WL}_{\text{iBSS}}}$, is similar to the first one, except that data transfer happens through wireless NICs of $host(i)$ and $host(j)$. The third and fourth interaction statements, labeled as $shared_{\text{WL}_{\text{IBSS}}}$ and $shared_{\text{WL}_{\text{HMANET}}}$, are respectively used to transfer messages between wireless NICs of $host(i)$ and $host(j)$ in IBSS and H-MANET.

Interactions

{Attach $\mathcal{T}_{\text{W}}$ of hosts with wired NICs in iBSS as transiently-shared variable when connected}

$shared_{\text{W}_{\text{iBSS}}}$ ::

$\langle [] \; i,j :: host(i).\mathcal{T}_{\text{W}} \approx host(j).\mathcal{T}_{\text{W}}$

when $(host(i).nwdeploy = \text{iBSS}) \wedge (host(j).nwdeploy = \text{iBSS}) \wedge (host(i)\Gamma' host(j))$

$\wedge \big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)$

engage $host(i).\mathcal{T}_{\text{W}}$ disengage current $\| \bot \; \rangle$

{Attach $\mathcal{T}_{\text{WL}}$ of mobile host and access point in iBSS as transiently-shared variable, only when colocated}

$[] \; shared_{\text{WL}_{\text{iBSS}}}$ ::

$\langle [] \; i,j :: host(i).\mathcal{T}_{\text{WL}} \approx host(j).\mathcal{T}_{\text{WL}}$

when $(host(i).nwdeploy = \text{iBSS}) \wedge (host(j).nwdeploy = \text{iBSS}) \wedge (host(i)\Gamma' host(j))$

$\wedge \big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isAP}(host(j))\big) \vee \big(\texttt{isAP}(host(i)) \wedge \texttt{isMH}(host(j))\big)\big)$

engage $host(i).\mathcal{T}_{\text{WL}}$ disengage current $\| \bot \; \rangle$

{Attach $\mathcal{T}_{\text{WL}}$ of mobile hosts in IBSS as transiently-shared variable, only when colocated}

$[] \; shared_{\text{WL}_{\text{IBSS}}}$ ::

$\langle [] \; i,j :: host(i).\mathcal{T}_{\text{WL}} \approx host(j).\mathcal{T}_{\text{WL}}$

when $(host(i).nwdeploy = \text{IBSS}) \wedge (host(j).nwdeploy = \text{IBSS}) \wedge (host(i)\Gamma' host(j))$

$\wedge \texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))$

engage $host(i).\mathcal{T}_{\text{WL}}$ disengage current $\| \bot \; \rangle$

{Attach $\mathcal{T}_{\text{WL}}$ of mobile and gateway hosts in HMANET as transiently-shared variable, only when colocated}

$[] \; shared_{\text{WL}_{\text{HMANET}}}$ ::

$\langle [] \; i,j :: host(i).\mathcal{T}_{\text{WL}} \approx host(j).\mathcal{T}_{\text{WL}}$

when $(host(i).nwdeploy = \text{HMANET}) \wedge (host(j).nwdeploy = \text{HMANET}) \wedge (host(i)\Gamma' host(j))$

$\wedge \big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big)$

$\vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\big)$

engage $host(i).\mathcal{T}_{\text{WL}}$ disengage current $\| \bot \; \rangle$

{Prepare to register active agents in respective hosts}

$[] \; regAgent :: \langle [] \; i,k :: host(i).\mathcal{Q}_{in} := host(i).\mathcal{Q}_{in} \bullet agent(k).aid \quad \text{when } (host(i).\lambda = agent(k).\lambda) \; \rangle$

{Prepare to deregister terminated/migrated agents from respective hosts}

$[] \; dregAgent :: \langle [] \; i,k :: host(i).\mathcal{Q}_{out} := host(i).\mathcal{Q}_{out} \bullet agent(k).aid \quad \text{when } \neg(host(i).\lambda = agent(k).\lambda) \; \rangle$

{Prepare to transfer reaction/response tuple from agent to host}

$[] \; \langle [] \; i,k :: host(i).\mathcal{Q}_{T^S_{a_k}}, agent(k).\mathcal{Q}_{T^S_{a_k}} := host(i).\mathcal{Q}_{T^S_{a_k}} \bullet \texttt{head}(agent(k).\mathcal{Q}_{T^S_{a_k}}), \texttt{tail}(agent(k).\mathcal{Q}_{T^S_{a_k}})$

when $(host(i).\lambda = agent(k).\lambda) \wedge \neg(agent(k).\mathcal{Q}_{T^S_{a_k}} = \bot) \; \rangle$

{Prepare to transfer reaction/response tuple from host to agent}

$[] \; \langle [] \; i,k :: agent(k).\mathcal{Q}_{T^R_{a_k}}, host(i).\mathcal{Q}_{T^R_{a_k}} := agent(k).\mathcal{Q}_{T^R_{a_k}} \bullet \texttt{head}(host(i).\mathcal{Q}_{T^R_{a_k}}), \texttt{tail}(host(i).\mathcal{Q}_{T^R_{a_k}})$

when $(host(i).\lambda = agent(k).\lambda) \wedge \neg(host(i).\mathcal{Q}_{T^R_{a_k}} = \bot) \; \rangle$

Figure 5.2: '**Interactions**' section in **System** *CUTSMuHN*.

The fifth and sixth statements, labeled as *regAgent* and *dregAgent* in Figure 5.2, are respectively used to register new agents in hosts and deregister existing agents from them. The statement, labeled *regAgent*, requires that $agent(k)$ and $host(i)$ are at the same location, whereas the statement, labeled as *dregAgent*, evaluates only when $agent(k)$ and $host(i)$ are not at the same location (denoted by '¬'). It is to be noted here that, when an agent remains active in a host, it remains available for interaction with other components; this availability relation is denoted by "Γ". That is, $\Gamma_{agent(k)}$ represents that $agent(k)$ is registered and active in any host.

Modeling of agent: Program $agent(k)$

An agent (say, $agent(k)$) is modeled as **Program** $agent(k)$, which comprises of ***declare***, ***always***, ***initially*** and ***assign*** sections, as shown in Figure 5.3, Figure 5.4 and Figure 5.5. It is considered as one fundamental component of **System** *CUTSMuHN*, and is modeled as stationary or mobile. The different behaviors of an agent, including the functionalities of ATS, Local Operation Manager, Remote Operation Manager, ATS Reaction Manager etc. are contained in $agent(k)$. Variables are declared in ***declare*** section of $agent(k)$ to capture these agent behaviors, particularly related to invoke of tuple space primitives. These variables include *aid* and *type*, where *aid* represents the identity of $agent(k)$ that uniquely identifies it in a mobile host during interaction, and *type* represents its nature (viz. stationary or mobile). Also, **T** is declared as ATS present in $agent(k)$, and *prid* is declared as the identity of any invoked primitive. Moreover, *ROL* and *RL* are declared as *Remote Operation List* and *Reactive List* of $agent(k)$ respectively. The queues $\mathcal{Q}_{T^S_{a_k}}$ and $\mathcal{Q}_{T^R_{a_k}}$ are declared to model interfacing of agents with their hosts. These queues transfer reaction/response-tuples from agents to hosts and vice versa.

When a user application is generating an event for any tuple space operation during interaction, the corresponding agent must capture different parameters required to perform that operation. In the proposed modeling, readiness of the user application is abstracted by a boolean variable, $UsrRdy4Evt$. Once the user application is ready, capturing values of different parameters are specified by different functions in $agent(k)$. The variables $prType$, $prName$, $tuple/tuples$, $atuple$, $taids$, $mode$ and boolean variables

Program $agent(k)$ **at** λ

declare

$aid, taid, a$: agentid ▯ $type$: $\in${stationary, mobile} ▯ $taids$: sequence of agentid
▯ **T** : tuple space ▯ $t, tuple$: tuple ▯ **t**, $tuples$: set of tuple ▯ **a**, $atuple$: antituple
▯ $prid$: primitiveid ▯ ROL : sequence of (primitiveid, primitivename, set of agentid of target agents)
▯ RL : sequence of (reactionid, primitiveid) ▯ $\mathbb{T}$: set of {agentid, set of tuple}
▯ $prType$: $\in${local, remote} ▯ $prName$: $\in${OUT, OUTG, RD, RDG, RDP, RDGP, IN, ING, INP, INGP}
▯ $mode$: $\in${ONCE, ONCE/TUPLE} ▯ r : RT_{tuple} ▯ $\mathcal{Q}_{T^S_{a_k}}, \mathcal{Q}_{T^R_{a_k}}$: queue of RT_{tuple}
▯ $TAs, rform$: natural ▯ $prBulk, prRdIn, UsrRdy4Evt$: boolean

always

$aid :=$ `getMyAgentID`(k) ▯ $type :=$ `getAgentType`(stationary, mobile)
▯ $\text{isPresent}_{\text{inROL}}(prid, taid) \equiv \langle\ \exists e :: (e \in ROL) \wedge (e \uparrow 1 = prid) \wedge (aid \in e \uparrow 3)\ \rangle$
▯ $\text{isEmpty}_{\text{inROL}}(prid) \equiv \langle\ \exists e :: (e \in ROL) \wedge (e \uparrow 1 = prid) \wedge (e \uparrow 3 = \emptyset)\ \rangle$

initially

$\lambda =$ `Location`(k) ▯ **T** $=\bot$ ▯ $t = \varepsilon$ ▯ $tuple = \varepsilon$ ▯ **t** $= \emptyset$ ▯ $tuples = \emptyset$ ▯ **a** $= \varepsilon$ ▯ $atuple = \varepsilon$
▯ $taid = \varepsilon$ ▯ $a = \varepsilon$ ▯ $taids = \emptyset$ ▯ $TAs = 0$ ▯ $rform = 0$ ▯ $prid = \varepsilon$ ▯ $ROL = \bot$ ▯ $RL = \bot$
▯ $\mathbb{T} = \emptyset$ ▯ $prType = \varepsilon$ ▯ $prName = \varepsilon$ ▯ $mode = \varepsilon$ ▯ $r = \varepsilon$ ▯ $\mathcal{Q}_{T^S_{a_k}} = \bot$ ▯ $\mathcal{Q}_{T^R_{a_k}} = \bot$
▯ $prBulk =$ FALSE ▯ $prRdIn =$ FALSE ▯ $UsrRdy4Evt =$ FALSE

assign

{Migrate to different location}
▯ $\lambda :=$ `Location`(`Move`()) if $(type =$ mobile$)$
{Capture different parameters when user application is ready}
▯ $\langle$ $prType, prName, UsrRdy4Evt :=$ `getPrimType`(), `getPrimName`(), FALSE
‖ $prRdIn, prBulk :=$ `getPrimRDorIN`(), `getPrimBulk`()
‖ $tuple :=$ `getTuple`() if $\big((prRdIn =$ FALSE$) \wedge (prBulk =$ FALSE$)\big)$
‖ $tuples :=$ `getTuples`() if $\big((prRdIn =$ FALSE$) \wedge (prBulk =$ TRUE$)\big)$
‖ $atuple :=$ `getAntiTuple`() if $(prRdIn =$ TRUE$)$
‖ $TAs :=$ `getTargetAgentCount`() if $(prType =$ remote$)$
‖ $\langle \| a : 1 \leq a \leq TAs :: taids[a] :=$ `getTargetAgentID`$(a) \rangle$ if $(prType =$ remote$)$
‖ $mode :=$ `getMode`(ONCE, ONCE/TUPLE) if $\big((prType =$ remote$) \wedge (prRdIn =$ TRUE$)\big)$
$\rangle$ if $(UsrRdy4Evt =$ TRUE$)$
{- - - - - - - - - - Start of Local Operation Manager - - - - - - - - - -}
{Perform different local tuple space primitives}
▯ $\langle$ $\langle$ $t, tuple, prType :=$ `out`$(t,$ **T**$)$ $\rangle$ if $\big((prName =$ OUT$) \wedge \neg(tuple = \varepsilon)\big)$
▯ $\langle$ **t**, $tuples, prType :=$ `outg`$($**t**, **T**$)$ $\rangle$ if $\big((prName =$ OUTG$) \wedge \neg(tuples = \emptyset)\big)$
▯ $\langle$ **a**, $atuple, prType :=$ `rdp`$($**a**, **T**$)$ ‖ `retTuple2Usr`(t) $\rangle$ if $(prName =$ RDP$)$
‖ $\langle$ **t** $:=$ `rdgp`$($**a**, **T**$)$ ‖ `retTuples2Usr`$($**t**$)$ $\rangle$ if $(prName =$ RDGP$)$
‖ $\langle$ $t :=$ `inp`$($**a**, **T**$)$ ‖ `retTuple2Usr`(t) $\rangle$ if $(prName =$ INP$)$
‖ $\langle$ **t** $:=$ `ingp`$($**a**, **T**$)$ ‖ `retTuples2Usr`$($**t**$)$ $\rangle$ if $(prName =$ INGP$)$
$\rangle$ if $\neg(atuple = \varepsilon)$
$\rangle$ if $(prType =$ local$)$
{- - - - - - - - - - End of Local Operation Manager - - - - - - - - - -}

Figure 5.3: Mobile UNITY '**Program** $agent(k)$': part 1.

{- - - - - - - - - - Start of Remote Operation Manager - - - - - - - - - -}

{Initiate (as reference agent) execution of different remote tuple space operations}

$[]\ \langle\ t, tuple, prType, prid, rform := tuple, \varepsilon, \varepsilon, \texttt{getPrID}(prName), 1$

$[]\ \langle \| a : 1 \leq a \leq TAs :: \mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{\mathsf{r}}(rform, prid, prName, \mathbf{t}, mode, aid, taids[a]) \rangle$

$\rangle$ if $\big((prType = \mathsf{remote}) \wedge (prName = \mathsf{OUT}) \wedge \neg(tuple = \varepsilon)\big)$

$[]\ \langle\ \mathbf{t}, tuples, prType, prid, rform := tuples, \emptyset, \varepsilon, \texttt{getPrID}(prName), 1$

$[]\ \langle \| a : 1 \leq a \leq TAs :: \mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{\mathsf{r}}(rform, prid, prName, \mathbf{t}, mode, aid, taids[a]) \rangle$

$\rangle$ if $\big((prType = \mathsf{remote}) \wedge (prName = \mathsf{OUTG}) \wedge \neg(tuples = \emptyset)\big)$

$[]\ \langle\ \mathsf{a}, atuple, prType, prid, rform := atuple, \varepsilon, \varepsilon, \texttt{getPrID}(prName), 1$

$[]\ ROL := ROL \cup \{prid, prName, taids\}$

$[]\ \langle \| a : 1 \leq a \leq TAs :: \mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{\mathsf{r}}(rform, prid, prName, \mathsf{a}, mode, aid, taids[a]) \rangle$

$\rangle$ if $\big((prType = \mathsf{remote}) \wedge (prRdIn = \mathsf{TRUE}) \wedge \neg(atuple = \varepsilon)\big)$

$[]\ \langle\ r, \mathcal{Q}_{T^R_{a_k}} := \texttt{head}(\mathcal{Q}_{T^R_{a_k}}), \texttt{tail}(\mathcal{Q}_{T^R_{a_k}})\quad []\ prid := r \uparrow \text{prid}$

$[]\ \langle\ \mathbb{T}_{prid} := \mathbb{T}_{prid} \cup \{r \uparrow \text{tAid}, r \uparrow \text{data}\} \parallel \langle\ \exists e : (e \in ROL) \wedge (e \uparrow 1 = prid) :: e \uparrow 3 := e \uparrow 3 \setminus r \uparrow \text{tAid}\ \rangle$

$\rangle$ if $\big((r \uparrow \text{rAid} = aid) \wedge \texttt{isPresent}_{\text{in}ROL}(prid, r \uparrow \text{tAid})\big)$ {Handling Response tuple}

$\rangle$ if $\big(\neg(\mathcal{Q}_{T^R_{a_k}} = \perp) \wedge (\texttt{head}(\mathcal{Q}_{T^R_{a_k}}) \uparrow \text{rform} = 2)\big)$

{Return result of execution of remote tuple-reading or -consuming operation to user}

$[]\ \Big\langle [] e : \big((e \in ROL) \wedge (e \uparrow 3 = \emptyset)\big)$

$::\ prid, prName, ROL := e \uparrow 1, e \uparrow 2, ROL \setminus e$

$[]\ \langle\ \langle \| e' : (e' \in \mathbb{T}_{prid}) :: \mathbf{t} := \mathbf{t} \cup e' \uparrow \text{tuples}\ \rangle\quad []\ \texttt{retTuples2Usr}(\mathbf{t})$

$[]\ \langle \| e' : \big((e' \in \mathbb{T}_{prid}) \wedge \big((prName = \mathsf{ING}) \vee (prName = \mathsf{INGP})\big)\big)$

$:: \mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{\mathsf{r}'}(3, prid, prName, aid, e' \uparrow \text{tAid})\ \rangle$

$\rangle$ if $\big((prName = \mathsf{RDG}) \vee (prName = \mathsf{RDGP}) \vee (prName = \mathsf{ING}) \vee (prName = \mathsf{INGP})\big)$

$[]\ \langle\ \langle \| e' : \big(e' = e''.(e'' \in \mathbb{T}_{prid})\big) :: t, taid := e' \uparrow \text{tuple}, e' \uparrow \text{tAid}\ \rangle\quad []\ \texttt{retTuple2Usr}(t)$

$[]\ \mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{\mathsf{r}'}(3, prid, prName, aid, taid)$

if $\big((prName = \mathsf{IN}) \vee (prName = \mathsf{INP})\big)$

$[]\ \langle \| e' : \big((e' \in \mathbb{T}_{prid}) \wedge \neg(e' \uparrow \text{tAid} = taid) \wedge \big((prName = \mathsf{IN}) \vee (prName = \mathsf{INP})\big)\big)$

$:: \mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{\mathsf{r}'}(4, prid, prName, e' \uparrow \text{tuple}, aid, e' \uparrow \text{tAid})\ \rangle$

$\rangle$ if $\big((prName = \mathsf{RD}) \vee (prName = \mathsf{RDP}) \vee (prName = \mathsf{IN}) \vee (prName = \mathsf{INP})\big)$

$\Big\rangle$

{- - - - - - - - - - End of Remote Operation Manager - - - - - - - - - -}

{Discard messages destined for other agents}

$[]\ \mathcal{Q}_{T^R_{a_k}} := \texttt{tail}(\mathcal{Q}_{T^R_{a_k}})$ if $\big(\neg(\mathcal{Q}_{T^R_{a_k}} = \perp) \wedge \neg(\texttt{head}(\mathcal{Q}_{T^R_{a_k}}) \uparrow \text{dstAg} = aid)\big)$

Figure 5.4: Mobile UNITY '**Program** $agent(k)$': part 2.

{- - - - - - - - - - Start of ATS Reaction Manager - - - - - - - - - -}

{Complete execution of different remote tuple space operations}

[] $\Big\langle$ $r, \mathcal{Q}_{T^R_{a_k}} := \texttt{head}(\mathcal{Q}_{T^R_{a_k}}), \texttt{tail}(\mathcal{Q}_{T^R_{a_k}})$ [] $prid, prName := r \uparrow \text{prid}, r \uparrow \text{pName}$

[] $prBulk :=$ TRUE

if $\big((prName = \text{RDG}) \vee (prName = \text{RDGP}) \vee (prName = \text{ING}) \vee (prName = \text{INGP})\big)$

$\sim$ FALSE if $\big((prName = \text{RD}) \vee (prName = \text{RDP}) \vee (prName = \text{IN}) \vee (prName = \text{INP})\big)$

[] $\langle$ $\langle$ $t := r \uparrow \text{data}$ [] $\texttt{out}(t, \mathbf{T})$ $\rangle$ if $(prName = \text{OUT})$

[] $\langle$ $\mathbf{t} := r \uparrow \text{data}$ [] $\texttt{outg}(\mathbf{t}, \mathbf{T})$ $\rangle$ if $(prName = \text{OUTG})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $t := \texttt{rd}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{RD})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $\mathbf{t} := \texttt{rdg}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{RDG})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $t := \texttt{rdp}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{RDP})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $\mathbf{t} := \texttt{rdgp}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{RDGP})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $t := \texttt{in}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{IN})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $\mathbf{t} := \texttt{ing}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{ING})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $t := \texttt{inp}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{INP})$

[] $\langle$ $a := r \uparrow \text{data}$ [] $\mathbf{t} := \texttt{ingp}(a, \mathbf{T})$ $\rangle$ if $(prName = \text{INGP})$ [] $rform := 2$

[] $\mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{r'}(rform, prid, prName, t, aid, r \uparrow \text{rAid})$ if $(prBulk = \text{FALSE})$

[] $\mathcal{Q}_{T^S_{a_k}} := \mathcal{Q}_{T^S_{a_k}} \bullet \texttt{createRTuple}_{r'}(rform, prid, prName, \mathbf{t}, aid, r \uparrow \text{rAid})$ if $(prBulk = \text{TRUE})$

$\rangle$ if $\big((r \uparrow \text{tAid} = aid) \wedge (r \uparrow \text{rform} = 1)\big)$ {Handling of reaction-tuple}

[] $\langle$ $t := r \uparrow \text{data}$ [] $\texttt{out}(t, \mathbf{T})$ $\rangle$ if $\big((r \uparrow \text{tAid} = aid) \wedge (r \uparrow \text{rform} = 4)\big)$ {Handling of nack-tuple}

$\Big\rangle$ if $\big(\neg(\mathcal{Q}_{T^R_{a_k}} = \perp)$

$\wedge \big((\texttt{head}(\mathcal{Q}_{T^R_{a_k}}) \uparrow \text{rform} = 1) \vee (\texttt{head}(\mathcal{Q}_{T^R_{a_k}}) \uparrow \text{rform} = 3) \vee (\texttt{head}(\mathcal{Q}_{T^R_{a_k}}) \uparrow \text{rform} = 4)\big)\big)$

{- - - - - - - - - - End of ATS Reaction Manager - - - - - - - - - -}

end

Figure 5.5: Mobile UNITY '**Program** $agent(k)$': part 3.

$prRdIn$ and $prBulk$ capture these values to carry on the execution of invoked tuple space primitives. The local operations are expressed by invoking these primitives to execute on local **T**, and returning the outcome such operations back to user application by `retTuple2Usr` method.

Modeling of host: Program $host(i)$

Like $agent(k)$, any host (say, $host(i)$) of $\mathcal{CUTSMuHN}$ platform (i.e. host environment of a mobile/portable device) is modeled as **Program** $host(i)$, which also comprises of ***declare***, ***always***, ***initially*** and ***assign*** sections, as shown in Figure 5.6, Figure 5.7, Figure 5.8 and Figure 5.9. The behaviors of hosts are captured by different variables,

Program $host(i)$ **at** λ

declare

$hid, mhid$: hostid [] $type$: $\in$ {stationary, mobile, accesspoint, gateway}

[] $\mathbf{T}'$: tuple space [] r : RT_{tuple} [] $\mathcal{Q}_{T^S_{a_k}}, \mathcal{Q}_{T^R_{a_k}}, \mathcal{Q}_{RT_S}, \mathcal{Q}_{RT_R}$: queue of RT_{tuple}

[] a : agentid [] $\mathcal{A}$, A : set of agentid [] $\mathcal{Q}_{in}, \mathcal{Q}_{out}$: queue of agentid

[] $assoc$: set of hostid [] $\mathcal{H}$: set of ($\text{MH}_{\text{hostid}}$, $\text{AP}_{\text{hostid}}$, timestamp) [] $\mathcal{L}$: set of ($\text{MH}_{\text{hostid}}$, RT_{tuple}, timestamp)

[] $\mathcal{L}RT$: set of ($\text{AP}_{\text{hostid}}$/$\text{MH}_{\text{hostid}}$, RT_{msgid}) [] $\mathcal{N}$: set of ($\text{Host}_{\text{hostid}}$, set of agentid, timestamp, extant)

[] M, m : message [] $\mathcal{CS}$: message [] $\mathcal{T}_{\text{W}}, \mathcal{T}_{\text{WL}}$: message [] $nwdeploy$: $\in$ {iBSS, IBSS, HMANET}

[] $\mathcal{Q}_{S_{\text{W}}}, \mathcal{Q}_{S_{\text{WL}}}, \mathcal{Q}_S, \mathcal{Q}_R, \mathcal{Q}_{S_B}, \mathcal{Q}_{R_B}, \mathcal{Q}_{S_{RT}}, \mathcal{Q}_{R_{RT}}$: queue of message

[] $clock, lastHTSchk, lastRTsent, lastBsent, newRTGap, rtAtmpt$: natural

[] $status$: $\in$ {standalone, connected, associated} [] $\mathcal{N}'$: set of ($\text{DHost}_{\text{hostid}}$, hop, $\text{IHost}_{\text{hostid}}$, timestamp, extant)

always

$B_{iBSS_{\text{W}}} = \textbf{IBSSBroadcastAddress}_{\textbf{DS}}$ [] $B_{iBSS_{\text{WL}}} = \textbf{IBSSBroadcastAddress}_{\textbf{BSA}}$

[] $B_{IBSS_{\text{WL}}} = \textbf{IBSSBroadcastAddress}$ [] $B_{HMANET} = \textbf{HMANETBroadcastAddress}$

[] $type :=$ `getHostType`(stationary, mobile, accesspoint, gateway) [] $bLife =$ SystemBeaconLifetime

[] $\lambda :=$ `Location`(i) [] $hid :=$ `getMyHostID`(i) [] $nwdeploy :=$ `getNetwork`(iBSS, IBSS, HMANET)

[] $mhGap =$ SystemMHValidityInterval [] $HTSaccessGap =$ SystemHTSAccessInterval

[] $locateGap =$ SystemLocateMsgInterval [] $bconGap =$ SystemBeaconInterval

[] `isPresent`$_{\mathcal{H}}(mhid) \equiv \langle\ \exists e : (e \in \mathcal{H}) \wedge (e \uparrow 1 = mhid)\ \rangle$

[] `isPresent`$_{\mathcal{L}}(mhid) \equiv \langle\ \exists e : (e \in \mathcal{L}) \wedge (e \uparrow 1 = mhid)\ \rangle$

[] `isPresent`$_{\mathcal{N}}(hostid) \equiv \langle\ \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = hostid)\ \rangle$

[] `isPresent`$_{\mathcal{N}'}(hostid) \equiv \langle\ \exists e : (e \in \mathcal{N}') \wedge (e \uparrow 1 = hostid)\ \rangle$

[] `isPresent`$_{\mathcal{L}RT}(hostid) \equiv \langle\ \exists e : (e \in \mathcal{L}RT) \wedge (e \uparrow 1 = hostid)\ \rangle$

[] `isRepeat`$_{\mathcal{L}RT}(hostid, msgid) \equiv \langle\ \exists e : (e \in \mathcal{L}RT) \wedge (e \uparrow 1 = hostid) \wedge (e \uparrow 2 = msgid)\ \rangle$

[] `isValid`$_{\mathcal{H}}(e, now) \equiv \big((e \in \mathcal{H}) \wedge ((now - e \uparrow 3) \leq mhGap)\big)$

[] `isValid`$_{\mathcal{L}}(e, now) \equiv \big((e \in \mathcal{L}) \wedge ((now - e \uparrow 3) \leq locateGap)\big)$

[] `isValid`$_{\mathcal{N}}(e, now) \equiv \big((e \in \mathcal{N}) \wedge ((now - e \uparrow 3) \leq e \uparrow 4)\big)$

[] `isMsgBcon`$(msg) \equiv (msg{\cdot}\, kind =$ Beacon) [] `isMsgRT`$(msg) \equiv (msg{\cdot}\, kind =$ RT)

[] `isMsgACK`$(msg) \equiv (msg{\cdot}\, kind =$ ACK) [] `isMsgLocate`$(msg) \equiv (msg{\cdot}\, kind =$ Locate)

[] `isMsgFound`$(msg) \equiv (msg{\cdot}\, kind =$ Found) [] `isNotOwnMsg`$(msg) \equiv \neg(msg{\cdot}\, src = hid)$

[] `isSH`$(host) \equiv (host{\cdot}\, type =$ stationary) [] `isMH`$(host) \equiv (host{\cdot}\, type =$ mobile)

[] `isAP`$(host) \equiv (host{\cdot}\, type =$ accesspoint) [] `isGH`$(host) \equiv (host{\cdot}\, type =$ gateway)

initially

$clock = 0$ [] $lastHTSchk = 0$ [] $lastRTsent = 0$ [] $lastBsent = 0$ [] $status = standalone$

[] $assoc = \emptyset$ [] $\mathcal{H} = \emptyset$ [] $\mathcal{L} = \emptyset$ [] $\mathcal{L}RT = \emptyset$ [] $\mathcal{A} = \emptyset$ [] A $= \emptyset$ [] $\mathcal{N} = \emptyset$ [] $\mathcal{N}' = \emptyset$

[] $\mathbf{T}' = \perp$ [] $\mathcal{T}_{\text{W}} = \perp$ [] $\mathcal{T}_{\text{WL}} = \perp$ [] $\mathcal{CS} = \perp$

[] $\mathcal{Q}_{T^S_{a_k}} = \perp$ [] $\mathcal{Q}_{T^R_{a_k}} = \perp$ [] $\mathcal{Q}_{in} = \perp$ [] $\mathcal{Q}_{out} = \perp$ [] $\mathcal{Q}_{RT_S} = \perp$ [] $\mathcal{Q}_{RT_R} = \perp$

[] $\mathcal{Q}_{S_B} = \perp$ [] $\mathcal{Q}_{R_B} = \perp$ [] $\mathcal{Q}_{S_{RT}} = \perp$ [] $\mathcal{Q}_{R_{RT}} = \perp$ [] $\mathcal{Q}_{S_{\text{W}}} = \perp$ [] $\mathcal{Q}_{S_{\text{WL}}} = \perp$ [] $\mathcal{Q}_S = \perp$ [] $\mathcal{Q}_R = \perp$

Figure 5.6: Mobile UNITY '**Program** $host(i)$': part 1.

assign

{Increment the clock}

$[]\ clock := clock + 1$

{- - - - - - - - - - Start of Transport Interface - - - - - - - - - -}

{Organize a message for onward transmission}

$[]\ \langle\ M, \mathcal{Q}_S := \texttt{head}(\mathcal{Q}_S), \texttt{tail}(\mathcal{Q}_S)$

$[]\ \langle\ \mathcal{Q}_{S_{\text{W}}} := \mathcal{Q}_{S_{\text{W}}} \bullet M \quad \text{if } (M \cdot ni = \text{W})\ \|\ \mathcal{Q}_{S_{\text{WL}}} := \mathcal{Q}_{S_{\text{WL}}} \bullet M \quad \text{if } (M \cdot ni = \text{WL})\ \rangle\ \rangle \quad \text{if } \neg(\mathcal{Q}_S = \perp)$

{Transfer a message from $\mathcal{Q}_{S_{\text{W}}}$ to $\mathcal{T}_{\text{W}}$; make $\mathcal{T}_{\text{W}}$ empty after some time}

$[]\ transmit\&reset_{\text{W}} :: \langle \mathcal{T}_{\text{W}}, \mathcal{Q}_{S_{\text{W}}} := \texttt{head}(\mathcal{Q}_{S_{\text{W}}}), \texttt{tail}(\mathcal{Q}_{S_{\text{W}}}) \quad \text{if } \big(\neg(\mathcal{Q}_{S_{\text{W}}} = \perp) \wedge (\mathcal{T}_{\text{W}} = \perp)\big);\ \mathcal{T}_{\text{W}} := \perp\ \rangle$

{Transfer a message from $\mathcal{Q}_{S_{\text{WL}}}$ to $\mathcal{T}_{\text{WL}}$; make $\mathcal{T}_{\text{WL}}$ empty after some time}

$[]\ transmit\&reset_{\text{WL}} :: \langle \mathcal{T}_{\text{WL}}, \mathcal{Q}_{S_{\text{WL}}} := \texttt{head}(\mathcal{Q}_{S_{\text{WL}}}), \texttt{tail}(\mathcal{Q}_{S_{\text{WL}}}) \quad \text{if } \big(\neg(\mathcal{Q}_{S_{\text{WL}}} = \perp) \wedge (\mathcal{T}_{\text{WL}} = \perp)\big);\ \mathcal{T}_{\text{WL}} := \perp \rangle$

{Transfer a message from $\mathcal{T}_{\text{W}}$ to $\mathcal{Q}_R$}

$[]\ \langle\ \mathcal{Q}_R := \mathcal{Q}_R \bullet \mathcal{T}_{\text{W}} \quad \text{if } \texttt{isNotOwnMsg}(\mathcal{T}_{\text{W}})\ \rangle \quad \text{reacts-to } \neg(\mathcal{T}_{\text{W}} = \perp)$

{Transfer a message from $\mathcal{T}_{\text{WL}}$ to $\mathcal{Q}_R$}

$[]\ \langle\ \mathcal{Q}_R := \mathcal{Q}_R \bullet \mathcal{T}_{\text{WL}} \quad \text{if } \texttt{isNotOwnMsg}(\mathcal{T}_{\text{WL}})\ \rangle \quad \text{reacts-to } \neg(\mathcal{T}_{\text{WL}} = \perp)$

{Organize a received Beacon/RT/ACK/Locate/Found message for further processing}

$[]\ \langle\ M, \mathcal{Q}_R := \texttt{head}(\mathcal{Q}_R), \texttt{tail}(\mathcal{Q}_R)$

$[]\langle\ \mathcal{Q}_{R_B} := \mathcal{Q}_{R_B} \bullet M \quad \text{if } \texttt{isMsgBcon}(M)$

$\|\ \mathcal{Q}_{R_{RT}} := \mathcal{Q}_{R_{RT}} \bullet M \quad \text{if } \texttt{isMsgRT}(M) \vee \texttt{isMsgACK}(M) \vee \texttt{isMsgLocate}(M) \vee \texttt{isMsgFound}(M)\rangle$

$\rangle \quad \text{if } \neg(\mathcal{Q}_R = \perp)$

{- - - - - - - - - - End of Transport Interface - - - - - - - - - -}

{- - - - - - - - - - Start of Host Tuple Space Handler - - - - - - - - - -}

{Process received RT from different agents}

$[]\ \langle\ []\ k :: \ \langle\ r, \mathcal{Q}_{T^S_{a_k}} := \texttt{head}(\mathcal{Q}_{T^S_{a_k}}), \texttt{tail}(\mathcal{Q}_{T^S_{a_k}})\ \ []\ \texttt{inject}(r, \mathbf{T}')\ \rangle \quad \text{if } \neg(\mathcal{Q}_{T^S_{a_k}} = \perp)\ \rangle$

{Process received RT from Communication Manager}

$[]\ \langle\ r, \mathcal{Q}_{RT_R} := \texttt{head}(\mathcal{Q}_{RT_R}), \texttt{tail}(\mathcal{Q}_{RT_R})\ \ []\ \texttt{inject}(r, \mathbf{T}')\ \rangle \quad \text{if } \neg(\mathcal{Q}_{RT_R} = \perp)$

{Periodically extract RT from HTS for onward transfer to target agents in same/different hosts}

$[]\ \langle\ \langle []\ a : (a \in \mathcal{A}) :: \ r := \texttt{eject}(a, \mathbf{T}')\ \ []\ \langle \mathcal{Q}_{T^R_a} := \mathcal{Q}_{T^R_a} \bullet r \quad \text{if } \neg(r = \varepsilon)\rangle\ \rangle$

$[]\ \langle []\ e : (e \in \mathcal{N}) :: \text{A} := e \uparrow 2\ \ []\ \langle []\ a : (a \in \text{A}) :: r := \texttt{eject}(a, \mathbf{T}')\ []\ \langle \mathcal{Q}_{RT_S} := \mathcal{Q}_{RT_S} \bullet r \quad \text{if } \neg(r = \varepsilon)\rangle\rangle\rangle$

$[]\ lastHTSchk := clock$

$\rangle \quad \text{if } \big((clock - lastHTSchk) > HTSaccessGap\big)$

{- - - - - - - - - - End of Host Tuple Space Handler - - - - - - - - - -}

{- - - - - - - - - - Start of Agent Manager - - - - - - - - - -}

{Register active agents in $\mathcal{A}$}

$[]\ \mathcal{A}, \mathcal{Q}_{in} := \mathcal{A} \cup \texttt{head}(\mathcal{Q}_{in}), \texttt{tail}(\mathcal{Q}_{in}) \quad \text{if } \neg(\mathcal{Q}_{in} = \perp)$

{Deregister terminated/migrated agents from $\mathcal{A}$}

$[]\ \mathcal{A}, \mathcal{Q}_{out} := \mathcal{A} \setminus \texttt{head}(\mathcal{Q}_{out}), \texttt{tail}(\mathcal{Q}_{out}) \quad \text{if } \big(\neg(\mathcal{Q}_{out} = \perp) \wedge (\texttt{head}(\mathcal{Q}_{out}) \in \mathcal{A})\big)$

{- - - - - - - - - - End of Agent Manager - - - - - - - - - -}

Figure 5.7: Mobile UNITY '**Program** $host(i)$': part 2.

{- - - - - - - - - - Start of Discovery Manager - - - - - - - - - -}

{Prepare to send Beacon message to destination}

$[]\ \langle\ \mathcal{Q}_{S_B}, lastBsent := \mathcal{Q}_{S_B} \bullet \texttt{discSendW}_{iBSS}(), clock$ if $(\texttt{isSH}(hid) \wedge (nwdeploy = \textsf{iBSS}))$

$[]\ \mathcal{Q}_{S_B}, lastBsent := \mathcal{Q}_{S_B} \bullet \texttt{discSendWL}_{iBSS}(), clock$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = \textsf{iBSS}))$

$[]\ \mathcal{Q}_{S_B}, lastBsent := (\mathcal{Q}_{S_B} \bullet \texttt{discSendW}_{iBSS}()) \bullet \texttt{discSendWL}_{iBSS}(), clock$
if $(\texttt{isAP}(hid) \wedge (nwdeploy = \textsf{iBSS}))$

$[]\ \mathcal{Q}_{S_B}, lastBsent := \mathcal{Q}_{S_B} \bullet \texttt{discSendWL}_{IBSS}(), clock$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = \textsf{IBSS}))$

$[]\ \mathcal{Q}_{S_B}, lastBsent := \mathcal{Q}_{S_B} \bullet \texttt{discSendWL}_{HMANET}(), clock$
if $((\texttt{isMH}(hid) \vee \texttt{isGH}(hid)) \wedge (nwdeploy = \textsf{HMANET}))$

$\rangle$ if $((clock - lastBsent) > bconGap)$

{Process received Beacon message}

$[]\ \langle\ \texttt{discRcv}_{\texttt{SH}_{iBSS}}(\mathcal{Q}_{R_B})$ if $(\texttt{isSH}(hid) \wedge (nwdeploy = \textsf{iBSS}))$

$[]\ \texttt{discRcv}_{\texttt{MH}_{iBSS}}(\mathcal{Q}_{R_B})$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = \textsf{iBSS}))$

$[]\ \texttt{discRcv}_{\texttt{AP}_{iBSS}}(\mathcal{Q}_{R_B})$ if $(\texttt{isAP}(hid) \wedge (nwdeploy = \textsf{iBSS}))$

$[]\ \texttt{discRcv}_{\texttt{MH}_{IBSS}}(\mathcal{Q}_{R_B})$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = \textsf{IBSS}))$

$[]\ \texttt{discRcv}_{\texttt{MH}_{HMANET}}(\mathcal{Q}_{R_B})$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = \textsf{HMANET}))$

$[]\ \texttt{discRcv}_{\texttt{GH}_{HMANET}}(\mathcal{Q}_{R_B})$ if $(\texttt{isGH}(hid) \wedge (nwdeploy = \textsf{HMANET}))$

$\rangle$ if $\neg(\mathcal{Q}_{R_B} = \perp)$

{Remove expired entries from $\mathcal{N}$}

$[]\ \texttt{discValid}_{\mathcal{N}}()$ if $\Big(((\texttt{isSH}(hid) \vee \texttt{isMH}(hid) \vee \texttt{isAP}(hid)) \wedge (nwdeploy = \textsf{iBSS}))$
$\vee\ (\texttt{isMH}(hid) \wedge (nwdeploy = \textsf{IBSS}))$
$\vee\ ((\texttt{isMH}(hid) \vee \texttt{isGH}(hid)) \wedge (nwdeploy = \textsf{HMANET}))\Big)$

{Update *assoc* on account of change in associated AP of MH}

$[]\ \langle\ \texttt{discUpdt}_{\texttt{MH}_{iBSS}}()$ if $(nwdeploy = \textsf{iBSS})$

$\rangle$ if $(\texttt{isMH}(hid) \wedge (\neg\texttt{isPresent}_{\mathcal{N}}(assoc[0]) \vee \neg\texttt{isValid}_{\mathcal{N}}(\langle \exists e : e \uparrow 1 = assoc[0] :: e \rangle, clock)))$

{Update *status* on account of change in connectivity of MH}

$[]\ \texttt{discUpdt}_{\texttt{MH}}()$ if $(\texttt{isMH}(hid) \wedge ((nwdeploy = \textsf{IBSS}) \vee (nwdeploy = \textsf{HMANET})))$

{Organize a Beacon message for onward transmission}

$[]\ \langle\ \mathcal{Q}_S, \mathcal{Q}_{S_B} := \mathcal{Q}_S \bullet \texttt{head}(\mathcal{Q}_{S_B}), \texttt{tail}(\mathcal{Q}_{S_B})\ \rangle$ if $\neg(\mathcal{Q}_{S_B} = \perp)$

{- - - - - - - - - - End of Discovery Manager - - - - - - - - - -}

Figure 5.8: Mobile UNITY '**Program** $host(i)$': part 3.

{- - - - - - - - - - Start of Communication Manager - - - - - - - - - -}

{Prepare to send RT/Locate message to destination}

[] ⟨ $\texttt{commSend}_{\text{SH}_{iBSS}}(\mathcal{Q}_{RT_S})$ if $\big(\texttt{isSH}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] $\texttt{commSend}_{\text{MH}_{iBSS}}(\mathcal{Q}_{RT_S})$ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] $\texttt{commSend}_{\text{AP}_{iBSS}}(\mathcal{Q}_{RT_S})$ if $\big(\texttt{isAP}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] $\texttt{commSend}_{\text{MH}_{IBSS}}(\mathcal{Q}_{RT_S})$ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = \text{IBSS})\big)$

[] $\texttt{commSend}_{\text{MH-GH}_{HMANET}}(\mathcal{Q}_{RT_S})$ if $\big(\big(\texttt{isMH}(hid) \vee \texttt{isGH}(hid)\big) \wedge (nwdeploy = \text{HMANET})\big)$

⟩ if $\neg(\mathcal{Q}_{RT_S} = \perp)$

{Process received RT/Locate/Found message, and prepare to send RT/ACK/Found message}

[] ⟨ $\texttt{commRcv}_{\text{SH}_{iBSS}}(\mathcal{Q}_{R_{RT}})$ if $\big(\texttt{isSH}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] $\texttt{commRcv}_{\text{MH}_{iBSS}}(\mathcal{Q}_{R_{RT}})$ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] $\texttt{commRcv}_{\text{AP}_{iBSS}}(\mathcal{Q}_{R_{RT}})$ if $\big(\texttt{isAP}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] $\texttt{commRcv}_{\text{MH}_{IBSS}}(\mathcal{Q}_{R_{RT}})$ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = \text{IBSS})\big)$

[] $\texttt{commRcv}_{\text{MH-GH}_{HMANET}}(\mathcal{Q}_{R_{RT}})$ if $\big(\big(\texttt{isMH}(hid) \vee \texttt{isGH}(hid)\big) \wedge (nwdeploy = \text{HMANET})\big)$

⟩ if $\neg(\mathcal{Q}_{R_{RT}} = \perp)$

{Resend RT message whose ACK fails to reach before timeout}

[] ⟨ $\mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{commReSend}_{\text{RT}}()$ if $\Big(\big((\texttt{isMH}(hid) \vee \texttt{isAP}(hid)) \wedge (nwdeploy = \text{iBSS})\big)$
$\vee \big(\texttt{isMH}(hid) \wedge (nwdeploy = \text{IBSS})\big)$
$\vee \big((\texttt{isMH}(hid) \vee \texttt{isGH}(hid)) \wedge (nwdeploy = \text{HMANET})\big)\Big)$

⟩ if $\big((clock - lastRTsent) > newRTGap\big)$

{Process RT message whose destination is presently not available}

[] ⟨ ⟨ $\mathcal{Q}_{RT_R}, \mathcal{CS} := \mathcal{Q}_{RT_R} \bullet \mathcal{CS}.data, \perp$ ⟩ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] ⟨ $\mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, B_{iBSS_{\text{W}}}, Locate, \mathcal{CS}.dest, \text{W}\big)$
$\| \ \mathcal{L} := \mathcal{L} \cup \{(\mathcal{CS}.dest, \mathcal{CS}.data, clock)\}$ ⟩ if $\big(\texttt{isAP}(hid) \wedge (nwdeploy = \text{iBSS})\big)$

[] ⟨ $\mathcal{Q}_{RT_R}, \mathcal{CS} := \mathcal{Q}_{RT_R} \bullet \mathcal{CS}.data, \perp$
⟩ if $\big((\texttt{isMH}(hid) \wedge (nwdeploy = \text{IBSS})) \vee ((\texttt{isMH}(hid) \vee \texttt{isGH}(hid)) \wedge (nwdeploy = \text{HMANET}))\big)$

⟩ if $\big(\neg(\mathcal{CS} = \perp) \wedge (rtAtmpt > 3)\big)$

{Remove expired entries from $\mathcal{H}$ and $\mathcal{L}$, and preserve unsent RT}

[] $\texttt{commValid}_{\mathcal{HL}_{iBSS}}()$ if $\big((\texttt{isSH}(hid) \vee \texttt{isAP}(hid)) \wedge (nwdeploy = \text{iBSS})\big)$

{Organize RT/ACK/Locate/Found message for onward transmission}

[] ⟨ $\mathcal{Q}_S, \mathcal{Q}_{S_{RT}} := \mathcal{Q}_S \bullet \texttt{head}(\mathcal{Q}_{S_{RT}}), \texttt{tail}(\mathcal{Q}_{S_{RT}})$ ⟩ if $\neg(\mathcal{Q}_{S_{RT}} = \perp)$

{- - - - - - - - - - End of Communication Manager - - - - - - - - - -}

end

Figure 5.9: Mobile UNITY '**Program** $host(i)$': part 4.

declared in the ***declare*** section, and their assignment statements are stated in the ***assign*** section. In particular, *hid* is declared as the identity of $host(i)$, whereas *type* specifies the nature of $host(i)$ (viz. stationary host or mobile host or access point or gateway host). Tuple space $\mathbf{T}'$ is declared as its HTS. The underlying deployed network is captured by *nwdeploy*, which considers the value of iBSS, IBSS and HMANET for iBSS, IBSS and H-MANET respectively. The variables $\mathcal{H}$ and $\mathcal{L}$ are declared to represent *History* (that records the path of successful data transfer to different mobile hosts) and *Loc Stash* (that maintains the list of mobile hosts with their ongoing location searches) of $host(i)$ respectively for stationary hosts and access points in iBSS. Moreover, the variables $\mathcal{L}RT$ and $\mathcal{CS}$ are declared for *Last RT* (that records the list of identities of last RT-messages received from different hosts) and *Comm Stash* (that buffers the transmitted RT-message) of $host(i)$ respectively for mobile hosts and access points in both iBSS and IBSS. Also, $\mathcal{N}$ and $\mathcal{A}$ are declared to represent the *Neighbor List* (that stores the discovery information of reachable hosts) and *Agent List* (that lists the presently colocated mobile applications) of any host respectively. Each entry of $\mathcal{N}$, denoted as e, is a 4-tuple, comprising of "host identity", "associate", "timestamp" and "extant". The macro $\texttt{isValid}_{\mathcal{N}}$ checks for any entry in $\mathcal{N}$ matching the given host identity, whereas macro $\texttt{isPresent}_{\mathcal{N}}$ determines the presence of any entry of given host.

At the bottom level, $\mathcal{CUISMuHN}$ platform interacts with the transport services of hosting device, which is modeled as Transport Interface, by a set of assignment statements. The insights of transmission of messages from one host to another, are modeled by assigning them to the transiently-shared variable at one end and subsequently reading them at the other end. Discovery Manager and Communication Manager interchange messages with Transport Interface through two different queues, viz. $\mathcal{Q}_S$ and $\mathcal{Q}_R$. The behaviors of Discovery Manager and Communication Manager are abstracted according to the nature of host, which is subscripted in the corresponding macro. These macros are, in turn, used in different assignment statements to complete the functionalities of Discovery Manager and Communication Manager. Host Server interchanges reaction/response-tuples (represented here as $\mathrm{RT}_{\mathrm{tuple}}$) with Communication Manager through $\mathcal{Q}_{RT_S}$ and $\mathcal{Q}_{RT_R}$, which is modeled using a set of assignment statements. Similarly, a pair of assignment statements models

the registration/deregistration functionalities of Agent Manager.

The macros, used in Figure 5.8 and Figure 5.9, define specific discovery and communication tasks. These macros are defined as follows.

Macros related to Discovery Manager.

$$M := \texttt{discSend}_{\mathrm{W}_{iBSS}}() \triangleq \langle\ M := \texttt{newMsg}(hid, B_{iBSS_{\mathrm{W}}}, \mathsf{Beacon}, \texttt{buildBcon}(\mathcal{A}, bLife), \mathrm{W})\ \rangle$$

$$M := \texttt{discSend}_{\mathrm{WL}_{iBSS}}() \triangleq \langle\ M := \texttt{newMsg}(hid, B_{iBSS_{\mathrm{WL}}}, \mathsf{Beacon}, \texttt{buildBcon}(\mathcal{A}, bLife), \mathrm{WL})\ \rangle$$

$$M := \texttt{discSend}_{\mathrm{WL}_{IBSS}}() \triangleq \langle\ M := \texttt{newMsg}(hid, B_{IBSS_{\mathrm{WL}}}, \mathsf{Beacon}, \texttt{buildBcon}(\mathcal{A}, bLife), \mathrm{WL})\ \rangle$$

$$M := \texttt{discSend}_{\mathrm{WL}_{HMANET}}() \triangleq \langle\ M := \texttt{newMsg}(hid, B_{HMANET}, \mathsf{Beacon}, \texttt{buildBcon}(\mathcal{A}, bLife), \mathrm{WL})\ \rangle$$

$$\begin{aligned}
&\texttt{discRcv}_{\mathrm{SH}_{iBSS}}(\mathcal{Q}_{R_B}) \triangleq \\
&\quad \langle\ m, \mathcal{Q}_{R_B} := \texttt{head}(\mathcal{Q}_{R_B}), \texttt{tail}(\mathcal{Q}_{R_B}) \\
&\quad\ [\!]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\} \quad \text{if } \neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src) \\
&\qquad \|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant} \\
&\qquad\quad \rangle \quad \text{if } \texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isSH}(m{\cdot}\,src) \vee \texttt{isAP}(m{\cdot}\,src)\big) \\
&\quad \rangle
\end{aligned}$$

$$\begin{aligned}
&\texttt{discRcv}_{\mathrm{MH}_{iBSS}}(\mathcal{Q}_{R_B}) \triangleq \\
&\quad \langle\ m, \mathcal{Q}_{R_B} := \texttt{head}(\mathcal{Q}_{R_B}), \texttt{tail}(\mathcal{Q}_{R_B}) \\
&\quad\ [\!]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\} \quad \text{if } \neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src) \\
&\qquad \|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant} \\
&\qquad\quad \rangle \quad \text{if } \texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src) \\
&\qquad \|\ status, assoc[0] := associated, m{\cdot}\,src \\
&\qquad \rangle \quad \text{if } \big(\texttt{isAP}(m{\cdot}\,src) \wedge (status = standalone)\big) \\
&\quad\ [\!]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\} \quad \text{if } \neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src) \\
&\qquad \|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant} \\
&\qquad\quad \rangle \quad \text{if } \texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src) \\
&\qquad \|\ assoc[0] := m{\cdot}\,src \\
&\qquad \rangle \quad \text{if } \big(\texttt{isAP}(m{\cdot}\,src) \wedge (status = associated)\big) \\
&\quad \rangle
\end{aligned}$$

$\texttt{discRcv}_{\text{AP}_{iBSS}}(\mathcal{Q}_{R_B}) \triangleq$

$\langle\ m, \mathcal{Q}_{R_B} := \texttt{head}(\mathcal{Q}_{R_B}), \texttt{tail}(\mathcal{Q}_{R_B})$

$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\}$ if $\neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$

$\rangle$ if $\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\rangle$ if $\big(\texttt{isSH}(m{\cdot}\,src) \vee \texttt{isAP}(m{\cdot}\,src)\big)$

$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\}$ if $\neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$

$\rangle$ if $\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ assoc := assoc \cup \{m{\cdot}\,src\}$

$\rangle$ if $\texttt{isMH}(m{\cdot}\,src)$

$\rangle$

$\texttt{discRcv}_{\text{MH}_{IBSS}}(\mathcal{Q}_{R_B}) \triangleq$

$\langle\ m, \mathcal{Q}_{R_B} := \texttt{head}(\mathcal{Q}_{R_B}), \texttt{tail}(\mathcal{Q}_{R_B})$

$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\}$ if $\neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$

$\rangle$ if $\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ status := connected$

$\rangle$ if $\big(\texttt{isMH}(m{\cdot}\,src) \wedge (status = standalone)\big)$

$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\}$ if $\neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$

$\rangle$ if $\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\rangle$ if $\big(\texttt{isMH}(m{\cdot}\,src) \wedge (status = connected)\big)$

$\rangle$

$\texttt{discRcv}_{\text{MH}_{HMANET}}(\mathcal{Q}_{R_B}) \triangleq$

$\langle\ m, \mathcal{Q}_{R_B} := \texttt{head}(\mathcal{Q}_{R_B}), \texttt{tail}(\mathcal{Q}_{R_B})$

$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\}$ if $\neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$

$\rangle$ if $\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ status := connected$

$\rangle$ if $\big(\big(\texttt{isMH}(m{\cdot}\,src) \vee \texttt{isGH}(m{\cdot}\,src)\big) \wedge (status = standalone)\big)$

$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\}$ if $\neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$

$\rangle$ if $\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$

$\rangle$ if $\big(\big(\texttt{isMH}(m{\cdot}\,src) \vee \texttt{isGH}(m{\cdot}\,src)\big) \wedge (status = connected)\big)$

$\rangle$

$$\texttt{discRcv}_{\text{GH}_{HMANET}}(\mathcal{Q}_{R_B}) \triangleq$$
$$\langle\ m, \mathcal{Q}_{R_B} := \texttt{head}(\mathcal{Q}_{R_B}), \texttt{tail}(\mathcal{Q}_{R_B})$$
$$[]\ \langle\ \mathcal{N} := \mathcal{N} \cup \{m{\cdot}\,src, m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}\} \quad \text{if } \neg\texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$$
$$\|\ \langle \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2, e \uparrow 3, e \uparrow 4 := m{\cdot}\,data \uparrow \text{agids}, clock, m{\cdot}\,data \uparrow \text{extant}$$
$$\rangle \quad \text{if } \texttt{isPresent}_{\mathcal{N}}(m{\cdot}\,src)$$
$$\rangle \quad \text{if } \big(\texttt{isMH}(m{\cdot}\,src) \vee \texttt{isGH}(m{\cdot}\,src)\big)$$
$$\rangle$$

$$\texttt{discValid}_{\mathcal{N}}() \triangleq \langle\ \|\ e : e \in \mathcal{N} :: \mathcal{N} := \mathcal{N} \setminus \{e\} \quad \text{if } \neg\texttt{isValid}_{\mathcal{N}}(e, clock)\ \rangle$$

$$\texttt{discUpdt}_{\text{MH}_{iBSS}}() \triangleq$$
$$\langle\ assoc[0] := \perp \quad \text{if } (\mathcal{N} = \emptyset) \qquad \sim \qquad a.(a \in \mathcal{N}) \quad \text{if } \neg(\mathcal{N} = \emptyset)$$
$$[]\ status := standalone \quad \text{if } (assoc[0] = \perp) \qquad \sim \qquad associated \quad \text{if } \neg(assoc[0] = \perp)$$
$$\rangle$$

$$\texttt{discUpdt}_{\text{MH}}() \triangleq \langle\ status := standalone \quad \text{if } (\mathcal{N} = \emptyset) \qquad \sim \qquad connected \quad \text{if } \neg(\mathcal{N} = \emptyset)\ \rangle$$

Macros related to Communication Manager.

$$\texttt{commSend}_{\text{SH}_{iBSS}}(\mathcal{Q}_{RT_S}) \triangleq$$
$$\langle\ r, \mathcal{Q}_{RT_S} := \texttt{head}(\mathcal{Q}_{RT_S}), \texttt{tail}(\mathcal{Q}_{RT_S})\ []\ dstid := r \uparrow \text{dstHost}$$
$$[]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, dstid, \mathsf{RT}, r, \mathrm{W}\big) \quad \text{if } \big(\texttt{isSH}(dstid) \wedge (host_{hid}\Gamma' host_{dstid})\big)$$
$$[]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \big(\texttt{isSH}(dstid) \wedge \neg(host_{hid}\Gamma' host_{dstid})\big)$$
$$[]\ \langle\ apid := \langle\ \exists e : \big((e \in \mathcal{H}) \wedge (e \uparrow 1 = dstid)\big) :: e \uparrow 2\ \rangle$$
$$[]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, apid, \mathsf{RT}, r, \mathrm{W}\big) \quad \text{if } (host_{hid}\Gamma' host_{apid})$$
$$[]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \neg(host_{hid}\Gamma' host_{apid})$$
$$\rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge \texttt{isPresent}_{\mathcal{H}}(dstid)\big)$$
$$[]\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, B_{iBSS_{\mathrm{W}}}, \mathsf{Locate}, dstid, \mathrm{W}\big)\ \|\ \mathcal{L} := \mathcal{L} \cup \{(dstid, r, clock)\}$$
$$\rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge \neg\texttt{isPresent}_{\mathcal{H}}(dstid)\big)$$
$$\rangle$$

$$\texttt{commSend}_{\text{MH}_{iBSS}}(\mathcal{Q}_{RT_S}) \triangleq$$
$$\langle\ r, \mathcal{Q}_{RT_S} := \texttt{head}(\mathcal{Q}_{RT_S}), \texttt{tail}(\mathcal{Q}_{RT_S})\ []\ m := \texttt{newMsg}\big(hid, assoc[0], \mathsf{RT}, r, \mathrm{WL}\big)$$
$$[]\ \mathcal{Q}_{S_{RT}}, \mathcal{CS}, lastRTsent, newRTGap, rtAtmpt := \mathcal{Q}_{S_{RT}} \bullet m, m, clock, rtGap, 0$$
$$\text{if } (host_{hid}\Gamma' host_{assoc[0]})$$
$$[]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \neg(host_{hid}\Gamma' host_{assoc[0]})$$
$$\rangle \quad \text{if } \big((status = associated) \wedge (\mathcal{CS} = \perp)\big)$$

$$
\begin{aligned}
&\texttt{commSend}_{\text{AP}_{iBSS}}(\mathcal{Q}_{RT_S}) \triangleq \\
&\quad \langle\ r, \mathcal{Q}_{RT_S} := \texttt{head}(\mathcal{Q}_{RT_S}), \texttt{tail}(\mathcal{Q}_{RT_S})\ [\!]\ dstid := r \uparrow \text{dstHost} \\
&\quad [\!]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, dstid, \mathsf{RT}, r, \text{W}\big) \quad \text{if } \big(\texttt{isSH}(dstid) \wedge (host_{hid}\Gamma' host_{dstid})\big) \\
&\quad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \big(\texttt{isSH}(dstid) \wedge \neg(host_{hid}\Gamma' host_{dstid})\big) \\
&\quad [\!]\ \langle\ m := \texttt{newMsg}\big(hid, dstid, \mathsf{RT}, r, \text{WL}\big) \\
&\qquad [\!]\ \mathcal{Q}_{S_{RT}}, \mathcal{CS}, lastRTsent, newRTGap, rtAtmpt := \mathcal{Q}_{S_{RT}} \bullet m, m, clock, rtGap, 0 \\
&\qquad\qquad \text{if } (host_{hid}\Gamma' host_{dstid}) \\
&\qquad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \neg(host_{hid}\Gamma' host_{dstid}) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge (dstid \in assoc) \wedge (\mathcal{CS} = \perp)\big) \\
&\quad [\!]\ \langle\ apid := \langle\ \exists e : \big((e \in \mathcal{H}) \wedge (e \uparrow 1 = dstid)\big) :: e \uparrow 2\ \rangle \\
&\qquad [\!]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, apid, \mathsf{RT}, r, \text{W}\big) \quad \text{if } (host_{hid}\Gamma' host_{apid}) \\
&\qquad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \neg(host_{hid}\Gamma' host_{apid}) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge \neg(dstid \in assoc) \wedge \texttt{isPresent}_{\mathcal{H}}(dstid)\big) \\
&\quad [\!]\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, B_{iBSS_{\text{W}}}, \mathsf{Locate}, dstid, \text{W}\big) \parallel \mathcal{L} := \mathcal{L} \cup \{(dstid, r, clock)\} \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge \neg(dstid \in assoc) \wedge \neg\texttt{isPresent}_{\mathcal{H}}(dstid)\big) \\
&\quad \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{commSend}_{\text{MH}_{IBSS}}(\mathcal{Q}_{RT_S}) \triangleq \\
&\quad \langle\ r, \mathcal{Q}_{RT_S} := \texttt{head}(\mathcal{Q}_{RT_S}), \texttt{tail}(\mathcal{Q}_{RT_S})\ [\!]\ dstid := r \uparrow \text{dstHost}\ [\!]\ m := \texttt{newMsg}\big(hid, dstid, \mathsf{RT}, r, \text{WL}\big) \\
&\quad [\!]\ \mathcal{Q}_{S_{RT}}, \mathcal{CS}, lastRTsent, newRTGap, rtAtmpt := \mathcal{Q}_{S_{RT}} \bullet m, m, clock, rtGap, 0 \\
&\qquad\qquad \text{if } \big(\texttt{isMH}(dstid) \wedge (host_{hid}\Gamma' host_{dstid})\big) \\
&\quad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \big(\neg\texttt{isMH}(dstid) \vee \neg(host_{hid}\Gamma' host_{dstid})\big) \\
&\quad \rangle \quad \text{if } \big((status = connected) \wedge (\mathcal{CS} = \perp)\big)
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{commSend}_{\text{MH-GH}_{HMANET}}(\mathcal{Q}_{RT_S}) \triangleq \\
&\quad \langle\ r, \mathcal{Q}_{RT_S} := \texttt{head}(\mathcal{Q}_{RT_S}), \texttt{tail}(\mathcal{Q}_{RT_S})\ [\!]\ dstid := r \uparrow \text{dstHost} \\
&\quad [\!]\ \langle\ m := \texttt{newMsg}\big(hid, dstid, \mathsf{RT}, r, \text{WL}\big) \\
&\qquad [\!]\ \mathcal{Q}_{S_{RT}}, \mathcal{CS}, lastRTsent, newRTGap, rtAtmpt := \mathcal{Q}_{S_{RT}} \bullet m, m, clock, rtGap, 0 \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge \texttt{isPresent}_{\mathcal{N}}(dstid) \wedge (host_{hid}\Gamma' host_{dstid})\big) \\
&\quad [\!]\ \langle\ dstid' := \langle \exists e : (e \in \mathcal{N}') \wedge (e \uparrow 1 = dstid) :: e \uparrow 3 \rangle\ [\!]\ m := \texttt{newMsg}\big(hid, dstid', \mathsf{RT}, r, \text{WL}\big) \\
&\qquad [\!]\ \mathcal{Q}_{S_{RT}}, \mathcal{CS}, lastRTsent, newRTGap, rtAtmpt := \mathcal{Q}_{S_{RT}} \bullet m, m, clock, rtGap, 0 \\
&\qquad\qquad \text{if } \big(\big(\texttt{isMH}(dstid') \vee \texttt{isGH}(dstid')\big) \wedge \texttt{isPresent}_{\mathcal{N}}(dstid') \wedge (host_{hid}\Gamma' host_{dstid'})\big) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMH}(dstid) \wedge \texttt{isRchbl}_{\mathcal{N}'}(dstid) \wedge \neg(host_{hid}\Gamma' host_{dstid})\big) \\
&\quad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet r \quad \text{if } \neg\texttt{isMH}(dstid) \\
&\quad \rangle \quad \text{if } \big((status = connected) \wedge (\mathcal{CS} = \perp)\big)
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{commRcv}_{\mathrm{SH}_{iBSS}}(\mathcal{Q}_{R_{RT}}) \triangleq \\
&\quad \langle\ m, \mathcal{Q}_{R_{RT}} := \texttt{head}(\mathcal{Q}_{R_{RT}}), \texttt{tail}(\mathcal{Q}_{R_{RT}}) \\
&\quad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m.data \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \big(\texttt{isSH}(m.src) \vee \texttt{isAP}(m.src)\big) \wedge (m.dest = hid)\big) \\
&\quad [\!]\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m.src, \mathsf{RT}, \langle \exists\, e : ((e \in \mathcal{L}) \wedge (e \uparrow 1 = m.data)) :: e \uparrow 2\rangle, \mathrm{W}\big) \\
&\qquad [\!]\ \mathcal{H} := \mathcal{H} \cup \{(m.data, m.src, clock)\} \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMsgFound}(m) \wedge \texttt{isAP}(m.src) \wedge \texttt{isPresent}_{\mathcal{L}}(m.data)\big) \\
&\quad \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{commRcv}_{\mathrm{MH}_{iBSS}}(\mathcal{Q}_{R_{RT}}) \triangleq \\
&\quad \langle\ m, \mathcal{Q}_{R_{RT}} := \texttt{head}(\mathcal{Q}_{R_{RT}}), \texttt{tail}(\mathcal{Q}_{R_{RT}}) \\
&\quad [\!]\ \langle\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m.data \ \|\ \langle \exists e : (e \in \mathcal{L}RT) \wedge (e \uparrow 1 = m.src) :: e \uparrow 2 := m.mid\rangle \\
&\qquad \rangle \quad \text{if } \big(\texttt{isPresent}_{\mathcal{L}RT}(m.src) \wedge \neg\texttt{isRepeat}_{\mathcal{L}RT}(m.src, m.mid)\big) \\
&\qquad [\!]\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m.data \ \|\ \mathcal{L}RT := \mathcal{L}RT \cup \{(m.src, m.mid)\}\ \rangle \quad \text{if } \neg\texttt{isPresent}_{\mathcal{L}RT}(m.src) \\
&\qquad [\!]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m.src, \mathsf{ACK}, m.mid, \mathrm{WL}\big) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \texttt{isAP}(m.src) \wedge (m.src = assoc[0]) \wedge (m.dest = hid)\big) \\
&\quad [\!]\ \mathcal{CS} := \perp \quad \text{if } \big(\texttt{isMsgACK}(m) \wedge \texttt{isAP}(m.src) \wedge (m.src = assoc[0]) \wedge (m.dest = hid) \wedge \neg(\mathcal{CS} = \perp) \\
&\qquad\qquad \wedge ((clock - lastRTsent) < newRTGap) \wedge (rtAtmpt < 3) \wedge (m.mid = \mathcal{CS}.mid)\big) \\
&\quad \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{commRcv}_{\mathrm{AP}_{iBSS}}(\mathcal{Q}_{R_{RT}}) \triangleq \\
&\quad \langle\ m, \mathcal{Q}_{R_{RT}} := \texttt{head}(\mathcal{Q}_{R_{RT}}), \texttt{tail}(\mathcal{Q}_{R_{RT}}) \\
&\quad [\!]\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m.data \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \big(\texttt{isSH}(m.src) \vee \texttt{isAP}(m.src)\big) \wedge (m.dest = hid)\big) \\
&\quad [\!]\ \langle\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m.data \ \|\ \langle \exists e : (e \in \mathcal{L}RT) \wedge (e \uparrow 1 = m.src) :: e \uparrow 2 := m.mid\rangle \\
&\qquad \rangle \quad \text{if } \big(\texttt{isPresent}_{\mathcal{L}RT}(m.src) \wedge \neg\texttt{isRepeat}_{\mathcal{L}RT}(m.src, m.mid)\big) \\
&\qquad [\!]\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m.data \ \|\ \mathcal{L}RT := \mathcal{L}RT \cup \{(m.src, m.mid)\}\ \rangle \quad \text{if } \neg\texttt{isPresent}_{\mathcal{L}RT}(m.src) \\
&\qquad [\!]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m.src, \mathsf{ACK}, m.mid, \mathrm{WL}\big) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \texttt{isMH}(m.src) \wedge (m.src \in assoc) \wedge (m.dest = hid)\big) \\
&\quad [\!]\ \mathcal{CS} := \perp \quad \text{if } \big(\texttt{isMsgACK}(m) \wedge \texttt{isMH}(m.src) \wedge (m.src \in assoc) \wedge (m.dest = hid) \wedge \neg(\mathcal{CS} = \perp) \\
&\qquad\qquad \wedge ((clock - lastRTsent) < newRTGap) \wedge (rtAtmpt < 3) \wedge (m.mid = \mathcal{CS}.mid)\big) \\
&\quad [\!]\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m.src, \mathsf{RT}, \langle \exists\, e : ((e \in \mathcal{L}) \wedge (e \uparrow 1 = m.data)) :: e \uparrow 2\rangle, \mathrm{W}\big) \\
&\qquad [\!]\ \mathcal{H} := \mathcal{H} \cup \{(m.data, m.src, clock)\} \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMsgFound}(m) \wedge \texttt{isAP}(m.src) \wedge \texttt{isPresent}_{\mathcal{L}}(m.data)\big) \\
&\quad [\!]\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m.src, \mathsf{Found}, m.data, \mathrm{W}\big) \quad \text{if } (m.data \in assoc) \\
&\qquad \rangle \quad \text{if } \big(\texttt{isMsgLocate}(m) \wedge \big(\texttt{isSH}(m.src) \vee \texttt{isAP}(m.src)\big) \wedge \texttt{isPresent}_{\mathcal{L}}(m.data)\big) \\
&\quad \rangle
\end{aligned}
$$

$$\texttt{commRcv}_{\text{MH}_{IBSS}}(\mathcal{Q}_{R_{RT}}) \triangleq$$
$$\langle\ m, \mathcal{Q}_{R_{RT}} := \texttt{head}(\mathcal{Q}_{R_{RT}}), \texttt{tail}(\mathcal{Q}_{R_{RT}})$$
$$[]\ \langle\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m{\cdot}\,data \ \|\ \langle \exists e : (e \in \mathcal{LRT}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2 := m{\cdot}\,mid \rangle$$
$$\rangle \quad \text{if } \big(\texttt{isPresent}_{\mathcal{LRT}}(m{\cdot}\,src) \wedge \neg\texttt{isRepeat}_{\mathcal{LRT}}(m{\cdot}\,src, m{\cdot}\,mid)\big)$$
$$[]\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m{\cdot}\,data \ \|\ \mathcal{LRT} := \mathcal{LRT} \cup \{(m{\cdot}\,src, m{\cdot}\,mid)\}\ \rangle \quad \text{if } \neg\texttt{isPresent}_{\mathcal{LRT}}(m{\cdot}\,src)$$
$$[]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m{\cdot}\,src, \mathsf{ACK}, m{\cdot}\,mid, \text{WL}\big)$$
$$\rangle \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \texttt{isMH}(m{\cdot}\,src) \wedge (m{\cdot}\,dest = hid)\big)$$
$$[]\ \mathcal{CS} := \perp \quad \text{if } \big(\texttt{isMsgACK}(m) \wedge \texttt{isMH}(m{\cdot}\,src) \wedge (m{\cdot}\,dest = hid) \wedge \neg(\mathcal{CS} = \perp)$$
$$\wedge ((clock - lastRTsent) < newRTGap) \wedge (rtAtmpt < 3) \wedge (m{\cdot}\,mid = \mathcal{CS}{\cdot}\,mid)\big)$$
$$\rangle$$

$$\texttt{commRcv}_{\text{MH-GH}_{HMANET}}(\mathcal{Q}_{R_{RT}}) \triangleq$$
$$\langle\ m, \mathcal{Q}_{R_{RT}} := \texttt{head}(\mathcal{Q}_{R_{RT}}), \texttt{tail}(\mathcal{Q}_{R_{RT}})$$
$$[]\ \langle\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m{\cdot}\,data \ \|\ \langle \exists e : (e \in \mathcal{LRT}) \wedge (e \uparrow 1 = m{\cdot}\,src) :: e \uparrow 2 := m{\cdot}\,mid \rangle$$
$$\rangle \quad \text{if } \big(\texttt{isPresent}_{\mathcal{LRT}}(m{\cdot}\,src) \wedge \neg\texttt{isRepeat}_{\mathcal{LRT}}(m{\cdot}\,src, m{\cdot}\,mid)\big)$$
$$[]\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet m{\cdot}\,data \ \|\ \mathcal{LRT} := \mathcal{LRT} \cup \{(m{\cdot}\,src, m{\cdot}\,mid)\}\ \rangle \quad \text{if } \neg\texttt{isPresent}_{\mathcal{LRT}}(m{\cdot}\,src)$$
$$[]\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m{\cdot}\,src, \mathsf{ACK}, m{\cdot}\,mid, \text{WL}\big)$$
$$\rangle \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \big(\texttt{isMH}(m{\cdot}\,src) \vee \texttt{isGH}(m{\cdot}\,src)\big) \wedge (m{\cdot}\,dest = hid)\big)$$
$$[]\ \langle\ \mathcal{Q}_{RT_S} := \mathcal{Q}_{RT_S} \bullet m{\cdot}\,data \ []\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}\big(hid, m{\cdot}\,src, \mathsf{ACK}, m{\cdot}\,mid, \text{WL}\big)$$
$$\rangle \quad \text{if } \big(\texttt{isMsgRT}(m) \wedge \big(\texttt{isMH}(m{\cdot}\,src) \vee \texttt{isGH}(m{\cdot}\,src)\big) \wedge \neg(m{\cdot}\,dest = hid)\big)$$
$$[]\ \mathcal{CS} := \perp \quad \text{if } \big(\texttt{isMsgACK}(m) \wedge \big(\texttt{isMH}(m{\cdot}\,src) \vee \texttt{isGH}(m{\cdot}\,src)\big) \wedge (m{\cdot}\,dest = hid) \wedge \neg(\mathcal{CS} = \perp)$$
$$\wedge ((clock - lastRTsent) < newRTGap) \wedge (rtAtmpt < 3) \wedge (m{\cdot}\,mid = \mathcal{CS}{\cdot}\,mid)\big)$$
$$\rangle$$

$$M := \texttt{commReSend}_{\mathsf{RT}}() \triangleq$$
$$\langle\ M, lastRTsent, newRTGap, rtAtmpt := \mathcal{CS}, clock, (2 * newRTGap), (rtAtmpt + 1)$$
$$\text{if } \big(\neg(\mathcal{CS} = \perp) \wedge (rtAtmpt < 3)\big)$$
$$\rangle$$

$$\texttt{commValid}_{\mathcal{HL}_{iBSS}}() \triangleq$$
$$\langle\ \langle \| e : e \in \mathcal{H} :: \mathcal{H} := \mathcal{H} \setminus \{e\} \quad \text{if } \neg\texttt{isValid}_{\mathcal{H}}(e, clock)\ \rangle$$
$$[]\ \langle \| e : e \in \mathcal{L} :: (\mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet e \uparrow 2) \ \|\ \mathcal{L} := \mathcal{L} \setminus \{e\} \quad \text{if } \neg\texttt{isValid}_{\mathcal{L}}(e, clock)\ \rangle$$
$$\rangle$$

In the proposed modeling approach of $\mathcal{CUTSMuHN}$ platform, an invoke of a tuple space primitive by reference agent is modeled either as local or remote invoke. The local invoke of any tuple space primitive is expressed by allowing the operation on Agent Tuple Space

of reference agent. On the other hand, the remote invoke is modeled as propagation of invoke parameters through Host Tuple Spaces of reference host, target host as well as intermediate hosts, if any, and allowing the operation on Agent Tuple Space of target agent. Similarly, the result of such invoke is also modeled to have propagated through tuple spaces of 'en route' hosts. The intermediate host tuple spaces decouple the remote invoke of reference agent from its process of forwarding, as well as from its execution. These tuple spaces also decouple response delivery of target agent from its process of forwarding. Moreover, these tuple spaces also separate the unavailability concerns arising due to mobile, dynamic and unreliable environments of underlying wireless networks.

5.3 Comparative Analysis

The Mobile UNITY model has been used in literature for modeling of tuple space coordination in mobile middleware systems [87, 98, 99]. In these works, the formal representations of either tuple space operations, or other aspects of the tuple space model are expressed. For instance, in [87], the semantics of tuple space operations (in the context of LIME platform) are obscurely specified, whereas in [98], such semantics are tenuously generalized for any platform of agent coordination. Closely related to the proposed modeling approach is the work in [99], where formal representation of the tuple space model (as well as other coordination models) are framed and a common strategy of their verification is hinted. The proposed modeling approach differs from [99] in both the approaches of formal modeling as well as verification of the same for deriving correctness. In particular, the proposed modeling approach repeatedly uses the refinement policy of the Mobile UNITY model to present a vivid and comprehensive formal description of the functionalities of TSMM platform. Such comprehensiveness is lacking in the formal representation of [99].

The formalization and verification of other systems, like Mobile IP [83], as well as functionalities within a system, viz. code mobility [95], context awareness [63] etc., using the Mobile UNITY model also exist in literature. In [83] as well as in [95] and [63],

formal representations of the respective systems are thoroughly described. Other extensions of the UNITY model for formal representations of different systems also exist in literature. For instance, [104] shows an extension of the UNITY model to formal describe a tightly-coupled system, which is radically different from very loosely-coupled $\mathcal{CUISMuHN}$ platform.

The approaches of modeling of different TSMM platforms also exist in literature [27, 36]. The formal analysis of other mobile computing systems also exist [30]. These formalization approaches mostly follow the process algebraic formal models for their specifications. The process algebraic models treat mobility in a manner, which is radically different from the Mobile UNITY model. In these models, mobility and movement-space of mobile entities cannot be directly represented, in contrast to the Mobile UNITY model, where movement-space can be arbitrarily defined, location is included in state of each component, and mobility is reduced to value assignments to location variables.

5.4 Conclusion

The existing approaches of formal modeling of TSMM platforms using the Mobile UNITY model have the shortcomings of not being vivid and comprehensive. This chapter has proposed a new modeling approach for thoroughly describing the behaviors of a TSMM platform, and applies the proposed modeling approach on expressing $\mathcal{CUISMuHN}$ platform. The proposed approach formally specifies $\mathcal{CUISMuHN}$ platform as a Mobile UNITY system, comprising of its components representing different behaviors of agents and hosts. This initiative of modeling not only has presented the insightful abstractions in expressing the behaviors of a coordination platform in presence of multiple underlying heterogeneous networks, but also has paved the way for presenting a strategy of verifying the correctness of $\mathcal{CUISMuHN}$ platform using it.

Chapter 6

Formal Verification of *CUTSMuHN*

6.1 Introduction

The modeling of *CUTSMuHN* platform, presented in the preceding chapter, is proposed with the important objective to formally verify its correctness. This chapter proposes a strategy of verification of correctness of *CUTSMuHN* platform using proof logic of the Mobile UNITY model. For formally verifying its correctness, a set of formal properties are defined from its proposed modeling. Defining formal properties to demonstrate correctness is agreeable due to the fact that there is no accepted definition of correctness of a mobile middleware platform, because of underlying mobility, dynamics and unreliability. In fact, a mobile node, depending on its movement pattern, may never receive any communicating information for its agents during coordination. However, employing a verification strategy for *CUTSMuHN* platform is still beneficial, as it elucidates an informal notion of correctness of the expected behavior of TSMM platform under common conditions. This section first defines a set of formal properties based on the modeling of *CUTSMuHN* platform to state its correctness, and next proposes a new verification strategy to prove these properties to show correctness of **System** *CUTSMuHN*.

6.2 Proposed Verification Strategy

This section proposes a new strategy of formal verification of the captured abstractions of *CUTSMuHN* platform in its modeling using proof logic of the Mobile UNITY model.

The outcome of this verification can be further used to justify the improvements in *CUISMuHN* platform. The free variables in formal properties (viz. i, j, k and l) are assumed to be quantified over appropriate ranges.

6.2.1 Verification of Tuple Space model of *CUISMuHN*

For verifying correctness of the tuple space model of *CUISMuHN* platform, it is obligatory to prove its safety properties. For instance, the safety property of the set of apposite tuples **A** states that all tuples of that set must satisfy *APPOSITE* property, where *APPOSITE* property is defined as:

$$\begin{aligned} &\mathit{APPOSITE} \triangleq \\ &\textbf{inv}.\ \Big\langle \forall t : (t \in \mathbf{A}) :: (\mathtt{mod}(a) = \mathbf{0}) \vee \Big((\mathtt{mod}(a) > \mathbf{0}) \wedge (\mathtt{mod}(a) \leq \mathtt{mod}(t)) \wedge \\ &\qquad \big\langle \forall j : 1 \leq j \leq \mathtt{mod}(a) :: \big\langle \exists i : 1 \leq i \leq \mathtt{mod}(t) :: (\mathsf{f}_j^a \cdot \mathsf{name} = \mathsf{f}_i^t \cdot \mathsf{name}) \big\rangle \ \big\rangle \Big) \Big\rangle \end{aligned} \tag{6.1}$$

where, '**inv**.' denotes an invariant property of the Mobile UNITY model. In the above expression, *APPOSITE* property states that for every tuple t in **A** corresponding to given antituple a, which is non-NULL, the names of constituent fields of t matches the names of all constituent fields of a. Similarly, the safety property of the set of sought tuples **S** states that for given a, all tuples in that set are sought tuples, i.e. it must satisfy the *SOUGHT* property, where *SOUGHT* property is defined as:

$$\mathit{SOUGHT} \triangleq\ \textbf{inv}.\ \big\langle \forall t : (t \in \mathbf{S}) :: \mathfrak{M}(t, a) \big\rangle \tag{6.2}$$

These properties are shown to hold by the following lemmas.

Lemma 6.1 *Set of apposite tuples* **A** *satisfies APPOSITE property.*

Proof: A predicate p is defined to specify that a tuple τ (where, $\tau \in \mathbf{A}$) has at least one constituent field, whose name matches the name of one constituent field of a, if a is not empty:

$$p \equiv (p_1 \wedge p_2^\tau) \vee \big(p_3 \wedge p_2^\tau \wedge p_4^\tau \wedge \langle \exists i, j :: p_5 \wedge p_6^\tau \rangle\big)$$

where $p_1 \equiv (\texttt{mod}(a) = 0)$, $p_2^\tau \equiv (\tau \in \mathbf{A})$, $p_3 \equiv (\texttt{mod}(a) > 0)$, $p_4^\tau \equiv (1 \leq i \leq \texttt{mod}(\tau))$, $p_5 \equiv (1 \leq j \leq \texttt{mod}(a))$ and $p_6^\tau \equiv (\mathsf{f}_j^a \cdot \mathsf{name} = \mathsf{f}_i^\tau \cdot \mathsf{name})$. For proving Lemma 6.1, it is sufficient to show that p is invariant, i.e. p holds for all possible memberships of $\mathbf{A}$ for given a. First, p is shown to be stable, i.e. once p holds, it continues to hold for every new added member of $\mathbf{A}$. This is shown by presenting a Hoare triple representation of Equation (5.4), from which the corresponding proof logic of Hoare triple is derived and subsequently simplified to show that p is stable. Next, p is also shown to hold at the initial state of $\mathbf{A}$. Combining these two properties together proves that p is invariant.

$$
\begin{aligned}
&\| \; ind : (ind \in \mathcal{I}) :: \{p\}\ \mathbf{A} := \mathbf{A} \cup \{t\} \quad \text{if } (t = \mathbf{T}^{\mathsf{s}}_{ind})\ \{p\} && \text{, } \textit{Hoare triple} \text{ representation} \\
&\equiv \{(ind \in \mathcal{I}) \wedge (t = \mathbf{T}^{\mathsf{s}}_{ind}) \wedge p\}\ \mathbf{A} := \mathbf{A} \cup \{t\}\ \{p\} && \text{, proof logic of } \textit{Hoare triple} \\
&\equiv \big((ind \in \mathcal{I}) \wedge (t = \mathbf{T}^{\mathsf{s}}_{ind}) \wedge p\big) \\
&\quad \Rightarrow \big(p_1 \wedge (ind \in I.(I \in \mathbf{T}^{\mathsf{p}}))\big) \wedge \big(p_3 \wedge \langle \exists\ I, j :: p_5 \wedge (ind \in I) \wedge (I \in \mathbf{T}^{\mathsf{p}}) \\
&\qquad \wedge (I \cdot \mathsf{name} = \mathsf{f}_j^a \cdot \mathsf{name})\rangle\big) \wedge (t = \mathbf{T}^{\mathsf{s}}_{ind}) \wedge p && \text{, replacing } \mathcal{I} \text{ using Equation (5.3)} \\
&\quad \Rightarrow \big(\big(p_1 \wedge (ind \in I.(I \in \mathbf{T}^{\mathsf{p}})) \wedge (t = \mathbf{T}^{\mathsf{s}}_{ind})\big) \wedge \big(p_3 \wedge \langle \exists\ I, j :: p_5 \wedge (ind \in I) \wedge (I \in \mathbf{T}^{\mathsf{p}}) \\
&\qquad \wedge (I \cdot \mathsf{name} = \mathsf{f}_j^a \cdot \mathsf{name})\rangle \wedge (t = \mathbf{T}^{\mathsf{s}}_{ind})\big)\big) \wedge p \\
&\quad \Rightarrow \Big(p_2^t \wedge \big(p_3 \wedge \langle \exists\ I, i, j :: p_4^t \wedge p_5 \wedge (I \in \mathbf{T}^{\mathsf{p}}) \wedge (I \cdot \mathsf{name} = \mathsf{f}_j^a \cdot \mathsf{name}) \wedge (I \cdot \mathsf{name} = \\
&\qquad \mathsf{f}_i^t \cdot \mathsf{name})\rangle\big)\Big) \wedge p && \text{, replacing } (t = \mathbf{T}^{\mathsf{s}}_{ind}) \text{ using Equation (5.2), and simplifying} \\
&\quad \Rightarrow \big(p_2^t \wedge \big(p_3 \wedge \langle \exists i, j :: p_4^t \wedge p_5 \wedge p_6^t\rangle\big)\big) \wedge p && \text{, by eliminating } I \cdot \mathsf{name} \text{ from above} \\
&\quad \Rightarrow p_2^t \wedge p && \text{, as } \big(p_3 \wedge \langle \exists i, j :: p_4^t \wedge p_5 \wedge p_6^t\rangle\big) \equiv p_2^t \\
&\quad \Rightarrow \big(p_1 \wedge (\tau \in \mathbf{A} \cup \{t\})\big) \vee \big(p_3 \wedge (\tau \in \mathbf{A} \cup \{t\}) \wedge \langle \exists i, j :: p_4^\tau \wedge p_5 \wedge p_6^\tau\rangle\big) \\
&\qquad && \text{, as } p_2^t \equiv (t \in \mathbf{A}) \text{ from definition of } p_2 \text{ and, } p_2^t \wedge p \text{ means } (\tau \in \mathbf{A} \cup \{t\}) \\
&\quad \Rightarrow p^{\mathbf{A}}_{\mathbf{A} \cup \{t\}}
\end{aligned}
$$

where, $p^{\mathbf{A}}_{\mathbf{A}\cup\{t\}}$ is predicate p having $\mathbf{A}\cup\{t\}$ substitutes $\mathbf{A}$. Also, initially ($\mathbf{A} = \emptyset$), and so both disjuncts of p holds vacuously. □

Lemma 6.2 *Set of sought tuples* $\mathbf{S}$ *satisfies the SOUGHT property.*

Proof: A predicate p' is defined to denote that tuple τ' (where, $\tau' \in \mathbf{S}$) matches positively with a:

$$p' \equiv \big((\tau' \in \mathbf{S}) \wedge \mathfrak{M}(\tau', a)\big)$$

Like Lemma (6.1), to prove Lemma (6.2), it is sufficient to show that p' is invariant. First, it is shown that p' is stable:

$$\begin{aligned}
&\| \quad t : (t \in \mathbf{A}) \wedge \mathfrak{M}(t, a) :: \{p'\}\ \mathbf{S} := \mathbf{S} \cup \{t\}\ \{p'\} && \text{, } \textit{Hoare triple} \text{ representation}\\
&\equiv \{(t \in \mathbf{A}) \wedge \mathfrak{M}(t, a) \wedge p'\}\ \mathbf{S} := \mathbf{S} \cup \{t\}\ \{p'\} && \text{, proof logic of } \textit{Hoare triple}\\
&\equiv \big((t \in \mathbf{A}) \wedge \mathfrak{M}(t, a) \wedge p'\ \big) \Rightarrow (t \in \mathbf{S}) \wedge \big((\tau' \in \mathbf{S}) \wedge \mathfrak{M}(\tau', a)\big) && \text{, simplifying}\\
&\qquad\qquad \Rightarrow \big(\tau' \in \mathbf{S} \cup \{t\}\big) \wedge \mathfrak{M}(\tau', a) \Rightarrow p'^{\mathbf{S}}_{\mathbf{S}\cup\{t\}}
\end{aligned}$$

Furthermore, initially ($\mathbf{S} = \emptyset$), and so p' holds vacuously. □

Proving the safety properties of the tuple space model of $\mathcal{CUTSMuHN}$ platform establishes its correctness. Based on this verification approach, correctness of the entire $\mathcal{CUTSMuHN}$ platform has been verified in the subsequent sections.

6.2.2 Formal Properties of Correctness

The formal properties of **System** *CUTSMuHN* are defined successively in a bottom-up fashion, starting with the lowest layer of *host*.

Formal Properties of Transport Interface

The purpose of Transport Interface of *host* is to transfer different messages from one host (say, $host(i)$) to another host (say, $host(j)$) via some transport service. This behavior is represented formally as *TRANSPORT* property, which is defined as:

$$\begin{aligned}
TRANSPORT(M) \triangleq\ & \big(\mathtt{head}(host(i).\mathcal{Q}_S) = M\big)\\
& \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee\\
& \qquad \neg(host(i)\Gamma' host(j)) \vee \big(\mathtt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \quad (6.3)
\end{aligned}$$

Here $TRANSPORT(M)$ asserts that if a message M is ready for transmission to other remote hosts, then it will eventually be enqueued in the receipt queues of Transport Interface of available and connected hosts in the same networks. This assertion is true, irrespective of the nature of underlying networks. That is, *TRANSPORT* property holds for iBSS, IBSS and HMANET networks in this thesis, and can easily extends its support to other underlying networks.

An extension of *TRANSPORT* property is *TRANSMIT* property, defined as:

$$\begin{aligned} TRANSMIT(M) \triangleq\ & \big(\mathtt{last}(host(i).\mathcal{Q}_S) = M\big) \\ & \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \\ & \neg(host(i)\Gamma' host(j)) \vee \big(\mathtt{head}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \end{aligned} \quad (6.4)$$

In $TRANSMIT(M)$ property, $TRANSPORT(M)$ property is extended by including the progress of $\mathcal{Q}_S$ and $\mathcal{Q}_R$ themselves. $TRANSMIT(M)$ asserts that, if M is handed over by Communication Manager or Discovery Manager of reference host, then it will be eventually ready to be handed over to Communication Manager or Discovery Manager of designated target host(s).

Formal Properties of Discovery Manager

Discovery Manager fulfils two objectives — (i) advertising up-to-date knowledge of active agents present in a host, and (ii) registering up-to-date knowledge of active agents present in other hosts that are reachable from (i.e. neighbors of) it. These two behaviors are represented formally as *ADVERTISE* property and *REGISTER* property. *ADVERTISE* property is defined as:

$$\begin{aligned} ADVERTISE(\mathcal{A},\mathcal{N}) \triangleq\ & \big((host(i).clock - host(i).lastBsent) > \textsc{SystemBeaconInterval}\big) \\ & \wedge \big(M \cdot data \uparrow \text{agids} \subseteq (host(i).\mathcal{A}_{host(i).clock} \cup host(i).\mathcal{N}_{host(i).clock}) \uparrow \text{agids}\big) \\ & \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg(host(i)\Gamma' host(j)) \vee \big(\mathtt{head}(host(j).\mathcal{Q}_{R_B}) = M\big) \vee \\ & \neg\big(host(i).nwdeploy = host(j).nwdeploy\big)\big)\big\rangle \end{aligned} \quad (6.5)$$

In this definition, $ADVERTISE(\mathcal{A}, \mathcal{N})$ asserts that if a Beacon message M is generated with updated list of active agents from $\mathcal{A}$ of any host for broadcasting, and the interval between two consecutive Beacon messages has elapsed, then M will eventually be enqueued in the receipt queues of Discovery Manager of other available hosts. Again, *REGISTER* property is defined as:

$$REGISTER(M) \triangleq \big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big) \mapsto \big(\{M \cdot data \uparrow \text{agids}, M \cdot data \uparrow \text{extant}, M \cdot data \uparrow \text{hopcount}, M \cdot data \uparrow \text{ihids}\} \in host(j).\mathcal{N}\big) \quad (6.6)$$

Here, $REGISTER(M)$ asserts that if Discovery Manager of any available host receives a Beacon message M, all the relevant details present inside M (viz. availability information about agents of its sending host) are stored in *Neighbor List* $\mathcal{N}$ of receiving host.

Formal Property of Communication Manager

The responsibility of Communication Manager is to reliably deliver RT from one host to another. This behavior is represented formally as *DELIVER* property:

$$DELIVER(r) \triangleq \big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r \uparrow \text{dstHost} = host(j').hid\big) \mapsto \big\langle \exists j' : \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\big\rangle \vee \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \quad (6.7)$$

$DELIVER(r)$ asserts that if Host Server of $host(i)$ enqueues RT r, which is destined to some target host (say, $host(j')$), in its $\mathcal{Q}_{RT_S}$, then r is either delivered to Host Server of $host(j')$ (by enqueuing in $\mathcal{Q}_{RT_R}$ of $host(j')$) or returned back to Host Server of $host(i)$ (by enqueuing back in $\mathcal{Q}_{RT_R}$ of $host(i)$).

Formal Properties of Host Server

Host Server is responsible for decoupling of the transfer of RT from reference agent to target agent (in same or different host) and vice versa. This decoupling behavior can be represented formally as $DECOUPLE_{IntraHost}$ and $DECOUPLE_{InterHost}$ properties.

$DECOUPLE_{IntraHost}$ property is defined as:

$$DECOUPLE_{IntraHost}(r) \triangleq \big(r \in host(i).\mathbf{T}'\big) \wedge \big(r \uparrow \mathrm{dstAg} = host(i).agent(l).aid\big) \\ \mapsto \big(\mathtt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big(r \in host(i).\mathbf{T}'\big) \quad (6.8)$$

Here, $DECOUPLE_{IntraHost}(r)$ asserts that if RT r is waiting in HTS $\mathbf{T}'$ of $host(i)$ and it is destined to target agent (say, $agent(l)$) residing in the same host, then it will eventually be enqueued in $\mathcal{Q}_{T^R_{a_l}}$ of $agent(l)$ or continues to wait in $\mathbf{T}'$ of $host(i)$ till $agent(l)$ becomes available for data transfer.

Similarly, $DECOUPLE_{InterHost}$ property is defined as:

$$DECOUPLE_{InterHost}(r) \triangleq \big(r \in host(i).\mathbf{T}'\big) \wedge \big(r \uparrow \mathrm{dstAg} = host(j').agent(l).aid\big) \\ \mapsto \big\langle \exists j' : \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big) \quad (6.9)$$

In this definition, $DECOUPLE_{InterHost}(r)$ asserts that, if r is waiting in $\mathbf{T}'$ of $host(i)$ and it is destined to $agent(l)$ residing in different host (say, $host(j')$), then it will eventually be stored in $\mathbf{T}'$ of $host(j')$ or continues to wait in $\mathbf{T}'$ of $host(i)$ till $host(j')$ becomes available for data transfer.

A combination of $DECOUPLE_{IntraHost}$ and $DECOUPLE_{InterHost}$ properties results in $TRANSFER$ property, which is defined as:

$$TRANSFER(r) \triangleq \big(\mathtt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r \uparrow \mathrm{dstAg}) \uparrow \mathrm{Ag} = agent(l).aid\big) \\ \mapsto \neg\Gamma_{agent(l)} \vee \big(\mathtt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' : \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \quad (6.10)$$

$TRANSFER(r)$ asserts that r will be eventually delivered to $agent(l)$, whether available in same host (viz. $host(i)$) or different host (viz. $host(j')$).

Formal Properties of Remote Operation Manager

Remote Operation Manager is responsible for handling the invokes of remote primitives of reference agent destined to be executed at target agent (in same/different host). This is

defined as *INVOKE* property:

$$INVOKE \triangleq \big(UsrRdy4Evt = \mathsf{TRUE}\big) \wedge \neg\big(prName = \epsilon\big) \wedge \big\langle \exists\, agent(l) : \big(agent(l).aid \in taids\big)\big\rangle$$
$$\wedge\, \Gamma_{agent(l)} \wedge \big(prType = \mathsf{remote}\big) \mapsto \big(\mathtt{exec}(r^{(1)}\uparrow \mathrm{prName}, r^{(1)}\uparrow \mathrm{data}) = \mathsf{TRUE}\big) \vee \neg\Gamma_{agent(l)} \quad (6.11)$$

In the above definition, *INVOKE* asserts that if reference agent $agent(k)$ is ready to invoke a remote tuple space primitive destined to some available target agent $agent(l)$, then the invoked primitive is eventually executed at the destination tuple space of $agent(l)$ if it remains available. The predicate $\big(\mathtt{exec}(r^{(1)}\uparrow \mathrm{prName},\, r^{(1)}\uparrow \mathrm{data}) = \mathsf{TRUE}\big)$ is used to abstract unmarshalling at target agent, i.e. the parameters of marshalled primitive $prName$ are unmarshalled for successful execution. Moreover, $r^{(1)}$ specifies a reaction-tuple packing the invoke parameters, and hence $r^{(1)}$ is one of the four kinds of RT possible in $\mathcal{CUTSMuHN}$ platform, i.e. $r^{(1)}$ is a form of r.

Remote Operation Manager is also responsible for positively or negatively acknowledging the receipt of responses from target hosts for invokes it has initiated. This is defined as *ACKNOWLEDGE* property:

$$ACKNOWLEDGE \triangleq$$
$$\big(\mathtt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \wedge \big(r^{(2)}\uparrow \mathrm{rAid} = agent(k).aid\big) \wedge \big(r^{(2)}\uparrow \mathrm{tAid} = agent(l).aid\big)$$
$$\wedge\, \big\langle \exists e : (e \in ROL) \wedge \big(r^{(2)}\uparrow \mathrm{prid} = e\uparrow 1\big) \wedge \big(agent(l).aid \in e\uparrow 3\big)\big\rangle$$
$$\wedge\, \big(\big(r^{(2)}\uparrow \mathrm{prName} = \mathsf{ING}\big) \vee \big(r^{(2)}\uparrow \mathrm{prName} = \mathsf{INGP}\big)\big) \wedge \neg\big(r^{(2)}\uparrow \mathrm{data} = \epsilon\big)$$
$$\mapsto \big(r^{(3)}\uparrow \mathrm{data} = \epsilon\big) \vee \big(\neg\big(r^{(4)}\uparrow \mathrm{data} = \epsilon\big) \wedge \big(r^{(4)}\uparrow \mathrm{data} \in agent(l).\mathbf{T}\big)\big) \vee \neg\Gamma_{agent(l)} \quad (6.12)$$

ACKNOWLEDGE asserts that if Remote Operation Manager of reference agent $agent(k)$ receives a response-tuple $r^{(2)}$ from one/more target agents, it acknowledges the acceptance of their responses by sending back ack-tuple $r^{(3)}$, or the rejection of same by sending back nack-tuple $r^{(4)}$. It is to be noted that $r^{(2)}$, $r^{(3)}$ and $r^{(4)}$ are other remaining forms of RT.

Formal Properties of ATS Reaction Manager

ATS Reaction Manager of target agent is responsible for handling the execution of invoked

remote primitives of reference agent, destined to be executed in same/different host, and sending any generated response back to Remote Operation Manager of reference agent. This is defined as *RESPONSE* property:

$$
\begin{aligned}
&RESPONSE \triangleq \\
&\quad \big(\texttt{exec}(r^{(1)}\uparrow \text{prName}, r^{(1)}\uparrow \text{data}) = \mathsf{TRUE}\big) \wedge \big(r^{(1)}\uparrow \text{rAid} = agent(k).aid\big) \wedge \Gamma_{agent(l)} \\
&\qquad \wedge \neg\big((r^{(1)}\uparrow \text{prName} = \mathsf{OUT}) \vee (r^{(1)}\uparrow \text{prName} = \mathsf{OUTG})\big) \wedge \big(r^{(1)}\uparrow \text{tAid} = agent(l).aid\big) \\
&\quad \mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \vee \neg\Gamma_{agent(l)}
\end{aligned}
\tag{6.13}
$$

Here, *RESPONSE* asserts that if target agent $agent(l)$ has received and executed a remote tuple space primitive (which is not tuple-producing in nature) and is ready to send the results of that execution back to reference agent $agent(k)$, then response-tuple $r^{(2)}$, packing the results of invoke of that primitive, eventually reaches $agent(k)$ (i.e. enqueued in $\mathcal{Q}_{T^R_{a_k}}$) to be handed over to its invoker method.

Definition (6.11) covers both safety and progress properties of correctness in the invokes of remote tuple space primitives of $\mathcal{CUTSMuHN}$ platform. On the other hand, definition (6.13) includes the safety and progress properties of correctness in the corresponding response-delivery. In the above definitions, the time bound is not predicated, as $\mathcal{CUTSMuHN}$ platform temporally decouples each agent interaction. An important objective of this verification strategy is also to demonstrate the ability of decoupling of $\mathcal{CUTSMuHN}$ platform, for which a pair of formal properties are specified in definition (6.8) and definition (6.9).

Based on these definitions, the formal correctness property of **System** *CUTSMuHN* is expressed as a UNITY-style conditional property (in hypothesis-conclusion form):

$$
\frac{TRANSPORT(M) \wedge ADVERTISE(\mathcal{A}, \mathcal{N}) \wedge REGISTER(M)}{INVOKE \wedge ACKNOWLEDGE \wedge RESPONSE} \tag{6.14}
$$

The above correctness specification is interpreted as: "given that the knowledge of available agents (of a host and its neighbor hosts) are advertised and subsequently registered by other hosts as well as the messages containing reaction/response-tuples (which,

in turn, carry invokes of tuple space primitives and responses thereof) and availability information are transmitted from one host to another in **System** *CUTSMuHN*, *INVOKE*, *ACKNOWLEDGE* and *RESPONSE* properties can be proven from the text of **System** *CUTSMuHN*".

6.2.3 Proof Techniques

This section shows that the formal properties defined in the preceding section are satisfied in **System** *CUTSMuHN*. Since its formal specifications contains no **inhibit** clauses, so the following assertion holds here.

$$\langle \forall S : (S \in CUTSMuHN) :: \iota(S) := \mathsf{FALSE} \rangle \tag{6.15}$$

Consequently, the inference rule in Equation (A.6) becomes:

$$\frac{\{p\}s\{H\},\ (H \mapsto FP(CUTSMuHN^{\Re}) \wedge q) \text{ in } CUTSMuHN^{\Re}}{\{p\}s^{*}\{q\}} \tag{6.16}$$

Also in **System** *CUTSMuHN*, there are no nonreactive statements that are not reactively-augmented. As such, the inference rule in Equation (A.4) is not applicable in these proofs.

In the proposed verification strategy, $FP(CUTSMuHN^{\Re})$ is first defined from the formal description of **System** *CUTSMuHN*, and is later used while proving its formal properties using the inference rule in Equation (6.16). $FP(CUTSMuHN^{\Re})$ is the fixed point predicate of the reactive program $\Re$ in *CUTSMuHN*, which is obligated to be shown for each assertion of the proofs. The approach of defining $FP(CUTSMuHN^{\Re})$ from reactive statements of **System** *CUTSMuHN*, and the technique of showing it to hold for individual assertions are elucidated in Appendix B.

The following lemmas prove that the formal properties hold in **System** *CUTSMuHN*, thereby satisfying the correctness specification of Equation (6.14). These lemmas are also arranged in a bottom-up manner, so that a proven formal property is used to prove the next higher-level formal property.

Lemma 6.3 *TRANSPORT*(M) *property is satisfied in* **System** *CUTSMuHN, i.e.*

$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee$

$\neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle$ *is provable in* **System** *CUTSMuHN.*

Proof: For proving that *TRANSPORT*(M) property holds in **System** *CUTSMuHN*, the transitive property of '$\mapsto$' (i.e. **leads-to**) relation are applied to the following set of assertions.

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \veebar \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big) \tag{6.17}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \tag{6.18}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \mapsto \big(host(i).\mathcal{T}_{\mathrm{W}} = M\big) \tag{6.19}$$

$$\big(host(i).\mathcal{T}_{\mathrm{W}} = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.20}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big) \tag{6.21}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big) \mapsto \big(host(i).\mathcal{T}_{\mathrm{WL}} = M\big) \tag{6.22}$$

$$\big(host(i).\mathcal{T}_{\mathrm{WL}} = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.23}$$

The above assertions describe the safety and progress properties, when **Program** $host(i)$ communicates with **Program** $host(j)$ through multiple assign statements and several reactive statements in the **Interactions** section. The assertion (6.17) asserts that if a message M is at the front of queue $\mathcal{Q}_S$ of $host(i)$, then it is eventually assigned to either of its queues $\mathcal{Q}_{S_{\mathrm{W}}}$ and $\mathcal{Q}_{S_{\mathrm{WL}}}$ (depending on the value of ni of M). The assertions (6.18) and (6.21) assert that if M is enqueued at the rear end of queue $\mathcal{Q}_{S_{\mathrm{W}}}$ or $\mathcal{Q}_{S_{\mathrm{WL}}}$ of $host(i)$, then it eventually reaches their respective front ends. The assertions (6.19) and (6.22) assert that if message M is at the front of either of the queues $\mathcal{Q}_{S_{\mathrm{W}}}$ and $\mathcal{Q}_{S_{\mathrm{WL}}}$ of $host(i)$, then it is eventually assigned to the transiently-shared variables $\mathcal{T}_{\mathrm{W}}$ and $\mathcal{T}_{\mathrm{WL}}$ respectively of $host(i)$. The assertions (6.20) and (6.23) assert that if $\mathcal{T}_{\mathrm{W}}$ or $\mathcal{T}_{\mathrm{WL}}$ of $host(i)$ holds M, then either $host(i)$ and $host(j)$ are deployed in different networks, or they are not connected (as their association processes are not successfully over or they have different

locations), or M is eventually enqueued to $\mathcal{Q}_R$ of $host(j)$ from either of $\mathcal{T}_{\text{W}}$ and $\mathcal{T}_{\text{WL}}$ of $host(i)$ by virtue of transient sharing.

Each of these assertions is obligated to be proven individually. The assertion (6.17) is actually proven in details in Section B.1.1 of Appendix B to illustrate the proof mechanisms of individual assertions. Other assertions can be proven by following same mechanisms. These assertions can be combined by different properties of '$\mapsto$' for wired and wireless NICs separately. For wired NIC, the above assertions are combined as follows:

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\text{W}}}) = M\big) \tag{6.24}$$

, from assertion (6.17), as in case of wired NIC, $\big(\texttt{last}(host(i).\mathcal{Q}_{S_{\text{W}}}) = M\big) = \mathsf{FALSE}$

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.25}$$

, after applying transitivity rule (A.11) on assertions (6.24), (6.18), (6.19) and (6.20) in succession.

For wireless NIC, the proven assertions are again combined by the transitive property of '$\mapsto$' as follows:

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\text{WL}}}) = M\big) \tag{6.26}$$

, from assertion (6.17), as in case of wireless NIC, $\big(\texttt{last}(host(i).\mathcal{Q}_{S_{\text{WL}}}) = M\big) = \mathsf{FALSE}$

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.27}$$

, after applying transitivity rule (A.11) on assertions (6.26), (6.21), (6.22) and (6.23) in succession.

Combining the assertions (6.25) and (6.27) for wired and wireless NICs, following holds:

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle$$

, after applying finite disjunction rule (A.17) on assertions (6.25) and (6.27) for both

, wired and wireless NICs; this holds as quantified ranges for j in their cases are complementary

This completes the proof of $TRANSPORT(M)$ property in **System** $CUTSMuHN$. □

Lemma 6.4 *$TRANSMIT(M)$ property is satisfied in* **System** *$CUTSMuHN$, i.e.*

$\big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee$

$\neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle$ *is provable in* **System** *$CUTSMuHN$.*

Proof: Like the proof of $TRANSPORT(M)$ property, $TRANSMIT(M)$ property is also proven to satisfy in **System** $CUTSMuHN$ using the different properties of '$\mapsto$' relation. This proof, which also involves the use of $TRANSPORT(M)$ property itself, is concerned with the following set of assertions:

$$\big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \tag{6.28}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.29}$$

$$\big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big) \mapsto \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big) \tag{6.30}$$

Among the above assertions, the assertion (6.28) asserts that if message M is enqueued at the rear end of queue $\mathcal{Q}_S$ of $host(i)$, then it eventually reaches its front end. The assertion (6.29) is actually $TRANSPORT(M)$ property, proven in Lemma 6.3. Lastly, the assertion (6.30) asserts that M, enqueued at the rear end of queue $\mathcal{Q}_R$ of $host(j)$, eventually reaches its front end. These assertion can be proven by the proof mechanisms explained in Section B.1.1 of Appendix B. They are combined by different properties of '$\mapsto$' as follows.

$$\big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.31}$$

, after applying transitivity rule (A.11) on assertions (6.28) and (6.29)

$$\big\langle \wedge j : \neg(j = i) \wedge \big(\texttt{last}(host(j).\mathcal{Q}_R) = M\big)\big\rangle \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big\rangle \tag{6.32}$$

, after quantifying both sides of assertion (6.30) with appropriate ranges of j

$$\big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle$$

, after applying cancellation rule (A.14) on assertions (6.31) and (6.32)

This completes the proof of *TRANSMIT*(M) property in **System** *CUTSMuHN*. □

Lemma 6.5 *ADVERTISE*$(\mathcal{A}, \mathcal{N})$ *property is satisfied in* **System** *CUTSMuHN, i.e.*

$\big((host(i).clock - host(i).lastBsent) > \textsc{SystemBeaconInterval}\big)$

$\wedge\big(M{\cdot}\,data\uparrow \text{agids} \subseteq (host(i).\mathcal{A}_{host(i).clock} \cup host(i).\mathcal{N}_{host(i).clock})\uparrow \text{agids}\big)$

$\mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big) \vee$

$\neg\big(host(i).nwdeploy = host(j).nwdeploy\big)\big)\big\rangle$ *is provable in* **System** *CUTSMuHN*.

Proof: *ADVERTISE*$(\mathcal{A}, \mathcal{N})$ property is proven with the help of *TRANSMIT*(M) property of Transport Interface, proven in Lemma 6.4, and using the different properties of '$\mapsto$' relation. This proof is involved with the following set of assertions:

$$\big((host(i).clock - host(i).lastBsent) > \textsc{SystemBeaconInterval}\big) \wedge \big(M{\cdot}\,data\uparrow \text{agids} \subseteq (host(i).\mathcal{A}_{host(i).clock} \cup host(i).\mathcal{N}_{host(i).clock})\uparrow \text{agids}\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S_B}) = M\big) \tag{6.33}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{S_B}) = M\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{S_B}) = M\big) \tag{6.34}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{S_B}) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \tag{6.35}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \tag{6.36}$$

$$\big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big) \mapsto \big(\texttt{last}(host(j).\mathcal{Q}_{R_B}) = M\big) \tag{6.37}$$

$$\big(\texttt{last}(host(j).\mathcal{Q}_{R_B}) = M\big) \mapsto \big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big) \tag{6.38}$$

Among the above assertions, the assertion (6.33) asserts that after every beacon interval, a (beacon) message M is prepared at $host(i)$, comprising of its supported agents and neighbor hosts/agents, which is eventually enqueued in its queue $\mathcal{Q}_{S_B}$. Again, the assertion (6.34) asserts that message M eventually reaches the front from rear end of $\mathcal{Q}_{S_B}$. The assertion (6.35) asserts that if M is enqueued at the front of $\mathcal{Q}_{S_B}$ of $host(i)$, then it is eventually enqueued at the rear end of its queue $\mathcal{Q}_S$. The assertion (6.36) is already

proven as *TRANSMIT*(M) property in Lemma 6.4. The assertion (6.37) asserts that if M is present at the front end of queue $\mathcal{Q}_R$ of $host(j)$, then it is eventually enqueued to the queue $\mathcal{Q}_{R_B}$ of $host(j)$. Lastly, the assertion (6.38) asserts that if M is enqueued at the rear end of queue $\mathcal{Q}_{R_B}$ of $host(j)$, it eventually reaches its front end. The above assertions can be proven by proof mechanisms shown in Section B.1.1 of Appendix B. These assertions are combined as follows using different properties of '$\mapsto$'.

$$\big(\texttt{last}(host(i).\mathcal{Q}_{S_B}) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_S) = M\big) \tag{6.39}$$

, after applying transitivity rule (A.11) on assertions (6.34) and (6.35)

$$\begin{aligned}&\big((host(i).clock - host(i).lastBsent) > \textsc{SystemBeaconInterval}\big)\\ &\qquad \wedge \big(M{\cdot}\,data{\uparrow}\,\text{agids} \subseteq (host(i).\mathcal{A}_{host(i).clock} \cup host(i).\mathcal{N}_{host(i).clock}){\uparrow}\,\text{agids}\big)\\ &\mapsto \big\langle \wedge\, j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee\\ &\qquad \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle\end{aligned} \tag{6.40}$$

, after applying transitivity rule (A.11) on assertions (6.33), (6.39) and (6.36) successively

$$\big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big) \mapsto \big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big) \tag{6.41}$$

, after applying transitivity rule (A.11) on assertions (6.37) and (6.38)

$$\big\langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big\rangle \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big)\big\rangle \tag{6.42}$$

, after quantifying both sides of assertion (6.41) with appropriate ranges of j

$$\begin{aligned}&\big((host(i).clock - host(i).lastBsent) > \textsc{SystemBeaconInterval}\big)\\ &\qquad \wedge \big(M{\cdot}\,data{\uparrow}\,\text{agids} \subseteq (host(i).\mathcal{A}_{host(i).clock} \cup host(i).\mathcal{N}_{host(i).clock}){\uparrow}\,\text{agids}\big)\\ &\mapsto \big\langle \wedge\, j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee\\ &\qquad \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big)\big)\big\rangle\end{aligned}$$

, after applying cancellation rule (A.14) on assertions (6.40) and (6.42)

This completes the proof of *ADVERTISE*($\mathcal{A}, \mathcal{N}$) property in **System** *CUTSMuHN*. □

Lemma 6.6 *REGISTER*(M) *property is satisfied in* **System** *CUTSMuHN, i.e.*

$\big(\texttt{head}(host(j).\mathcal{Q}_{R_B}) = M\big)$

$\mapsto \big(\{M{\cdot}\,data{\uparrow}\,\text{agids}, M{\cdot}\,data{\uparrow}\,\text{extant}, M{\cdot}\,data{\uparrow}\,\text{hopcount}, M{\cdot}\,data{\uparrow}\,\text{ihids}\} \in host(j).\mathcal{N}\big)$ *is provable*

in **System** *CUTSMuHN*.

Proof: *REGISTER*(M) property is shown to hold in **System** *CUTSMuHN*, as its assertion in its definition can be proven directly by the proof mechanisms illustrated in Section B.1.1 of Appendix B. □

Lemma 6.7 *DELIVER*(r) *property is satisfied in* **System** *CUTSMuHN, i.e.*

$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big)$

$\mapsto \big\langle \exists j' : \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\big\rangle \vee \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big)$ *is provable in* **System** *CUTSMuHN*.

Proof: *DELIVER*(r) property is also proven with the help of *TRANSMIT*(M) property of Transport Interface and different properties of '$\mapsto$' relation. This proof involves the following set of assertions:

$$\textbf{stable}\ \big(r\uparrow \text{dstHost} = host(j').hid\big) \tag{6.43}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{RT_S}) = r\big) \tag{6.44}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{RT_S}) = r\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S_{RT}}) = M\big) \tag{6.45}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{S_{RT}}) = M\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{S_{RT}}) = M\big) \tag{6.46}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{S_{RT}}) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S}) = M\big) \tag{6.47}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{S}) = M\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_{R}) = M\big)\big)\big\rangle \tag{6.48}$$

$$\big(\texttt{head}(host(j).\mathcal{Q}_{R}) = M\big) \mapsto \big(\texttt{last}(host(j).\mathcal{Q}_{R_{RT}}) = M\big) \tag{6.49}$$

$$\big(\texttt{last}(host(j).\mathcal{Q}_{R_{RT}}) = M\big) \mapsto \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big) \tag{6.50}$$

$$\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \big(\neg\big(host(i).nwdeploy = host(j').nwdeploy\big) \vee \neg(host(i)\Gamma' host(j'))\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{RT_R}) = r\big) \tag{6.51}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_R}) = r\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \tag{6.52}$$

$$\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \big\langle \wedge j :: \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\big\rangle \mapsto \big\langle \wedge j :: \neg(j = j') \vee \big(\texttt{last}(host(j).\mathcal{Q}_{RT_R}) = r\big)\big\rangle \tag{6.53}$$

$$\big(\texttt{last}(host(j').\mathcal{Q}_{RT_R}) = r\big) \mapsto \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big) \tag{6.54}$$

Among the above assertions, the assertion (6.43) asserts that after RT r is generated, with its target host field indicating $host(j')$ as destination, it cannot be modified, i.e. it holding a particular value becomes the **stable** property till it is consumed. The assertion (6.44) asserts that r eventually reaches the front end of queue $\mathcal{Q}_{RT_S}$ of $host(i)$ from its rear end. The assertion (6.45) asserts that if r is at the front end of $\mathcal{Q}_{RT_S}$ of $host(i)$, then a message M is generated with r as its data and is eventually enqueued at the rear end of queue $\mathcal{Q}_{S_{RT}}$ of $host(i)$. The assertion (6.46) asserts that M eventually reaches the front end of queue $\mathcal{Q}_{S_{RT}}$ of $host(i)$ from its rear end. The assertion (6.47) asserts that if M is at the front end of $\mathcal{Q}_{S_{RT}}$ of $host(i)$, then it is eventually enqueued at the rear end of queue $\mathcal{Q}_S$ of $host(i)$. The assertion (6.48) is $TRANSMIT(M)$ property itself and is proven in Lemma 6.4. The assertion (6.49) asserts that if M reaches the front end of $\mathcal{Q}_R$ of $host(j)$, then it is eventually enqueued at the rear end of queue $\mathcal{Q}_{R_{RT}}$ of $host(j)$. The assertion (6.50) asserts that M eventually reaches the front end of queue $\mathcal{Q}_{R_{RT}}$ of $host(j)$ from its rear end. The assertion (6.51) asserts that if the destination of r to be delivered is $host(j')$ and it is neither available in the network nor connected to $host(i)$, then r is extracted from M and is enqueued back to the queue $\mathcal{Q}_{RT_R}$ of $host(i)$. The assertion (6.52) asserts that if r is at the rear end of $\mathcal{Q}_{RT_R}$ of $host(i)$, then it eventually reaches the front end of that queue. The assertion (6.53) asserts that if M with r as its data (and $host(j')$ as its destination) reaches the queue $\mathcal{Q}_{R_{RT}}$ of each available $host(j)$, then r is extracted from M only at $host(j')$ and is eventually enqueued to its queue $\mathcal{Q}_{RT_R}$. The assertion (6.54) asserts that r eventually reaches the front end of queue $\mathcal{Q}_{RT_R}$ of $host(j')$ from its rear end. The above assertions can again be proven by the proof mechanisms presented in Section B.1.1 of Appendix B. These assertions are next combined following the different properties of '$\mapsto$' relation.

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \mapsto \big\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\big)\big\rangle \quad (6.55)$$

, after applying transitivity rule (A.11) on assertions (6.44), (6.45), (6.46), (6.47) and (6.48)

$$\big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big) \mapsto \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big) \quad (6.56)$$

, after applying transitivity rule (A.11) on assertions (6.49) and (6.50)

$$\langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_R) = M\big)\rangle \mapsto \langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\rangle \quad (6.57)$$

, after quantifying both sides of assertion (6.56) with appropriate ranges of j

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \mapsto \langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\big)\rangle \quad (6.58)$$

, after applying cancellation rule (A.14) on assertions (6.55) and (6.57)

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big) \mapsto \langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j)) \vee \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\big)\rangle \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big) \quad (6.59)$$

, after applying PSP rule (A.16) on assertions (6.43) and (6.58)

$$\frac{\langle \wedge j : \neg(j = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j).nwdeploy\big) \vee \neg(host(i)\Gamma' host(j))\big)\rangle}{\langle \exists j' : \neg(j' = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j').nwdeploy\big) \vee \neg(host(i)\Gamma' host(j'))\big)\rangle} \quad (6.60)$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big) \mapsto \Big(\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \langle \exists j' : \neg(j' = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j').nwdeploy\big) \vee \neg(host(i)\Gamma' host(j'))\big)\rangle\Big) \vee \Big(\big(r\uparrow \text{dstHost} = host(j).hid\big) \wedge \langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\rangle\Big) \quad (6.61)$$

, after applying conclusion of assertion (6.60) on assertion (6.59), and then simplifying

$$\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \big(\neg\big(host(i).nwdeploy = host(j').nwdeploy\big) \vee \neg(host(i)\Gamma' host(j'))\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \quad (6.62)$$

, after applying transitivity rule (A.11) on assertions (6.51) and (6.52)

$$\Big(\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \langle \exists j' : \neg(j' = i) \wedge \big(\neg\big(host(i).nwdeploy = host(j').nwdeploy\big) \vee \neg(host(i)\Gamma' host(j'))\big)\rangle\Big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \quad (6.63)$$

, after quantifying both sides of assertion (6.62) with appropriate ranges of j

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big) \mapsto \Big(\big(r\uparrow \text{dstHost} = host(j).hid\big) \wedge \langle \wedge j : \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\rangle\Big) \vee \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \quad (6.64)$$

, after applying cancellation rule (A.14) on assertions (6.61) and (6.63)

$$\Big\langle \wedge j :: \neg(j = j') \vee \big(\texttt{last}(host(j).\mathcal{Q}_{RT_R}) = r\big)\Big\rangle$$

$$\mapsto \Big\langle \wedge j :: \neg(j = j') \vee \big(\texttt{head}(host(j).\mathcal{Q}_{RT_R}) = r\big)\Big\rangle \tag{6.65}$$

, after quantifying both sides of assertion (6.54) with appropriate ranges of j

$$\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \Big\langle \wedge j :: \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\Big\rangle$$

$$\mapsto \Big\langle \wedge j :: \neg(j = j') \vee \big(\texttt{head}(host(j).\mathcal{Q}_{RT_R}) = r\big)\Big\rangle \tag{6.66}$$

, after applying transitivity rule (A.11) on assertions (6.53) and (6.65)

$$\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \Big\langle \wedge j :: \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\Big\rangle$$

$$\mapsto \Big\langle \wedge j :: \neg(j = i) \wedge \big(\neg(j = j') \vee \big(\texttt{head}(host(j).\mathcal{Q}_{RT_R}) = r\big)\big)\Big\rangle \tag{6.67}$$

, after applying conjunction of $\neg(j = i)$ on both sides of assertion (6.66)

$$\frac{\Big\langle \wedge j : \neg(j = i) \wedge \big(\neg(j = j') \vee \big(\texttt{head}(host(j).\mathcal{Q}_{RT_R}) = r\big)\big)\Big\rangle}{\Big\langle \exists j' : \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\Big\rangle} \text{ , after simplifying} \tag{6.68}$$

$$\big(r\uparrow \text{dstHost} = host(j').hid\big) \wedge \Big\langle \wedge j :: \neg(j = i) \wedge \big(\texttt{head}(host(j).\mathcal{Q}_{R_{RT}}) = M\big)\Big\rangle$$

$$\mapsto \Big\langle \exists j' : \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\Big\rangle \tag{6.69}$$

, after applying conclusion of assertion (6.68) on assertion (6.67), and then simplifying

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big)$$

$$\mapsto \Big\langle \exists j' : \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\Big\rangle \vee \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big)$$

, after applying cancellation rule (A.14) on assertions (6.64) and (6.69)

This completes the proof of $DELIVER(r)$ property in **System** *CUTSMuHN*. □

Lemma 6.8 *$DECOUPLE_{IntraHost}(r)$ property is satisfied in* **System** *CUTSMuHN, i.e.* $\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(i).agent(l).aid\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big(r \in host(i).\mathbf{T}'\big)$ *is provable in* **System** *CUTSMuHN.*

Proof: $DECOUPLE_{IntraHost}(r)$ property is proven with the help of different properties of '$\mapsto$' relation. This proof centers around the following assertions:

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(i).agent(l).aid\big)$$

$$\mapsto \big(r = \texttt{eject}(host(i).agent(l), host(i).\mathbf{T}')\big) \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.70}$$

$$\big(r = \texttt{eject}(host(i).agent(l), host(i).\mathbf{T}')\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.71}$$

Among these assertions, the assertion (6.70) asserts that if RT r, with $agent(l)$ of the same host $host(i)$ as its destination agent, is residing in HTS $\mathbf{T}'$ of $host(i)$, then either r is being ejected for delivery to its destination or it continues to reside in $\mathbf{T}'$ of $host(i)$. The assertion (6.71) asserts that if r is ejected from $\mathbf{T}'$ of $host(i)$ for delivery to its destination $agent(l)$ of $host(i)$, then it is eventually enqueued in the queue $\mathcal{Q}_{T^R_{a_l}}$ of $host(i)$ (which interfaces with $agent(l)$). These two assertions can be proven by the proof mechanisms illustrated in Section B.1.1 of Appendix B. They are combined by the cancellation property of '$\mapsto$' relation as follows.

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(i).agent(l).aid\big) \mapsto \big(\mathtt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big(r \in host(i).\mathbf{T}'\big)$$

, after applying cancellation rule (A.14) on assertions (6.70) and (6.71)

This shows that $DECOUPLE_{IntraHost}(r)$ property holds in **System** *CUTSMuHN*. □

Lemma 6.9 *$DECOUPLE_{InterHost}(r)$ property is satisfied in* **System** *CUTSMuHN, i.e.*

$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big)$

$\mapsto \big\langle \exists j' : \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big)$ *is provable in* **System** *CUTSMuHN.*

Proof: $DECOUPLE_{InterHost}(r)$ property is proven with the help of $DELIVER(r)$ property of Communication Manager, along with the different properties of '$\mapsto$' relation. This proof involves the following assertions:

$$\textbf{stable}\ \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big) \tag{6.72}$$

$$\big(r \in host(i).\mathbf{T}'\big) \mapsto \big(r = \mathtt{eject}(host(j').agent(l), host(i).\mathbf{T}')\big) \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.73}$$

$$\big(r = \mathtt{eject}(host(j').agent(l), host(i).\mathbf{T}')\big) \mapsto \big(\mathtt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \tag{6.74}$$

$$\big(\mathtt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \mathrm{dstHost} = host(j').hid\big)$$
$$\mapsto \big\langle \exists j' : \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\big\rangle \vee \big(\mathtt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \tag{6.75}$$

$$\big(\mathtt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big) \mapsto \big(r \in host(j').\mathbf{T}'\big) \tag{6.76}$$

Among these assertions, the assertion (6.72) asserts that RT r, with its target agent field indicating $agent(l)$ of $host(j')$ as destination, becomes a **stable** property. The assertion (6.73) asserts that if r is residing in HTS $\mathbf{T}'$ of $host(i)$, then either r is being ejected

from $\mathbf{T}'$ for delivery to its destination or it continues to reside in $\mathbf{T}'$ of $host(i)$. The assertion (6.74) asserts that if r is ejected from $\mathbf{T}'$ of $host(i)$ for delivery to $agent(l)$ of $host(j')$, then it is eventually enqueued in the queue $\mathcal{Q}_{RT_S}$ of $host(i)$. The assertion (6.75) is actually $DELIVER(r)$ property, which is proven in Lemma 6.7. The assertion (6.76) asserts that if r reaches the front end of queue $\mathcal{Q}_{RT_R}$ of $host(j')$, then its is eventually inserted into HTS $\mathbf{T}'$ of $host(j')$. The above assertions can be proven by the proof mechanisms demonstrated in Section B.1.1 of Appendix B. These assertions are combined by the different properties of '$\mapsto$' relation as follows.

$$\big(r \in host(i).\mathbf{T}'\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.77}$$

, after applying cancellation rule (A.14) on assertions (6.73) and (6.74)

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big)$$
$$\mapsto \Big(\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \vee \big(r \in host(i).\mathbf{T}'\big)\Big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big) \tag{6.78}$$

, after applying PSP rule (A.16) on assertions (6.72) and (6.77)

$$\big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{RT_R}) = r\big)\big\rangle \mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \tag{6.79}$$

, after quantifying both sides of assertion (6.76) with appropriate range of j'

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big)$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \tag{6.80}$$

, after applying cancellation rule (A.14) on assertions (6.75) and (6.79)

$$\big(\texttt{head}(host(i).\mathcal{Q}_{RT_R}) = r\big) \mapsto \big(r \in host(i).\mathbf{T}'\big) \tag{6.81}$$

, after appropriately replacing j' by i in both sides of assertion (6.76), as it holds

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstHost} = host(j').hid\big)$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.82}$$

, after applying cancellation rule (A.14) on assertions (6.80) and (6.81)

$$\big(\texttt{last}(host(i).\mathcal{Q}_{RT_S}) = r\big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big)$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.83}$$

, after strengthening left-hand side of assertion (6.82)

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big)$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big)$$
$$\vee \Big(\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big)\Big) \tag{6.84}$$

, after applying cancellation rule (A.14) on assertions (6.78) and (6.83)

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big)$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big)$$, after simplifying assertion (6.84)

This shows that $DECOUPLE_{InterHost}(r)$ property holds in **System** *CUTSMuHN*. □

Lemma 6.10 *TRANSFER*(r) *property is satisfied in* **System** *CUTSMuHN, i.e.*

$\big(\mathtt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r\uparrow \mathrm{dstAg})\uparrow \mathrm{Ag} = agent(l).aid\big) \mapsto \big(\mathtt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee$
$\neg\Gamma_{agent(l)} \vee \big\langle \exists j' : \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle$ *is provable in* **System** *CUTSMuHN*.

Proof: *TRANSFER*(r) property is proven with the help of $DECOUPLE_{IntraHost}(r)$ as well as $DECOUPLE_{InterHost}(r)$ properties of Host Server, along with the different properties of '$\mapsto$' relation. This proof is carried out on the following assertions:

$$\big(\mathtt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(\mathtt{head}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \tag{6.85}$$
$$\big(\mathtt{head}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(r \in host(i).\mathbf{T}'\big) \tag{6.86}$$
$$\Big(\big(r\uparrow \mathrm{dstAg} = host(i).agent(l).aid\big) \vee \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big)\Big)$$
$$\Rightarrow \big((r\uparrow \mathrm{dstAg})\uparrow \mathrm{Ag} = agent(l).aid\big) \tag{6.87}$$
$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(i).agent(l).aid\big)$$
$$\mapsto \big(\mathtt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.88}$$
$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big)$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle \vee \big(r \in host(i).\mathbf{T}'\big) \tag{6.89}$$
$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(i).agent(l).aid\big) \wedge \big(agent(l).\lambda = host(i).\lambda\big)$$
$$\mapsto \neg\big(r \in host(i).\mathbf{T}'\big) \tag{6.90}$$
$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \mathrm{dstAg} = host(j').agent(l).aid\big) \wedge \big(agent(l).\lambda = host(j').\lambda\big)$$
$$\wedge \big(\big(host(i).nwdeploy = host(j).nwdeploy\big) \wedge (host(i)\Gamma' host(j))\big) \mapsto \neg\big(r \in host(i).\mathbf{T}'\big) \tag{6.91}$$

$$\big(r \in host(j').\mathbf{T}'\big) \mapsto \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \neg\big(agent(l).\lambda = host(j').\lambda\big) \tag{6.92}$$

$$\big(agent(l).\lambda = host(i).\lambda\big) \Rightarrow \Gamma_{agent(l)} \tag{6.93}$$

$$\big(agent(l).\lambda = host(j').\lambda\big) \wedge \big(\big(host(i).nwdeploy = host(j).nwdeploy\big) \wedge (host(i)\Gamma' host(j))\big) \Rightarrow \Gamma_{agent(l)} \tag{6.94}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.95}$$

$$\big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.96}$$

Among these assertions, the assertion (6.85) asserts that if RT r is at the rear end of queue $\mathcal{Q}_{T^S_{a_k}}$ of $host(i)$, then r eventually reaches its front end. The assertion (6.86) asserts that r from the front end of queue $\mathcal{Q}_{T^S_{a_k}}$ of $host(i)$ is eventually inserted into its HTS $\mathbf{T}'$. The assertion (6.87) asserts that target agent of r, as specified while generating it, is $agent(l)$, irrespective of being hosted in same host (i.e. $host(i)$) or different host (i.e. $host(j')$). The assertion (6.88) is actually $DECOUPLE_{IntraHost}(r)$ property, proven in Lemma 6.8, whereas assertion (6.89) is $DECOUPLE_{InterHost}(r)$ property, proven in Lemma 6.9. The assertion (6.90) and assertion (6.91) assert that once $agent(l)$ colocates with $host(i)$ or $host(j')$ respectively, r is eventually withdrawn from $\mathbf{T}'$ of $host(i)$, provided there is connectivity. The assertion (6.92) asserts that r from HTS $\mathbf{T}'$ of $host(j')$ are either enqueued to its queue $\mathcal{Q}_{T^R_{a_l}}$ or $agent(l)$ is not colocated with $host(j')$. The assertion (6.93) and assertion (6.94) assert that colocation of $agent(l)$ with $host(i)$ or $host(j')$ respectively makes it available for interaction, provided there is a connectivity from $host(i)$ to $host(j')$ in the latter case. The assertion (6.95) and assertion (6.96) assert that r eventually reaches the front end of queues $\mathcal{Q}_{T^R_{a_l}}$ of $host(i)$ or $\mathcal{Q}_{T^R_{a_l}}$ of $host(j')$ respectively from their rear end. The above assertions can be proven by the proof mechanisms explained in Section B.1.1 of Appendix B. These assertions are combined as:

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(r \in host(i).\mathbf{T}'\big) \tag{6.97}$$

, after applying transitivity rule (A.11) on assertions (6.85) and (6.86)

$$\textbf{stable}\ \big((r \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big) \tag{6.98}$$

, as in assertion (6.87), left-hand side disjuncts **stable** predicates from assertion (6.72)

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(r \in host(i).\mathbf{T}'\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big) \quad (6.99)$$

, after applying PSP rule (A.16) on assertions (6.97) and (6.98)

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(i).agent(l).aid\big)$$
$$\mapsto \neg\big(r \in host(i).\mathbf{T}'\big) \vee \neg\big(agent(l).\lambda = host(i).\lambda\big) \quad (6.100)$$

, after applying $\frac{p\wedge q \;\mapsto\; r}{p \;\mapsto\; \neg q\vee r}$ rule on assertion (6.90)

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(i).agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \neg\big(agent(l).\lambda = host(i).\lambda\big) \quad (6.101)$$

, after applying disjunction of '$\mapsto$' on assertions (6.88) and (6.100), and then simplifying

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big) \mapsto \neg\big(r \in host(i).\mathbf{T}'\big)$$
$$\vee\neg\big(agent(l).\lambda = host(j').\lambda\big) \vee \neg\big(\big(host(i).nwdeploy = host(j).nwdeploy\big) \wedge (host(i)\Gamma' host(j))\big) \quad (6.102)$$

, after applying $\frac{p\wedge q \;\mapsto\; r}{p \;\mapsto\; \neg q\vee r}$ rule on assertion (6.91)

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big) \mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle$$
$$\vee\neg\big(agent(l).\lambda = host(j').\lambda\big) \vee \neg\big(\big(host(i).nwdeploy = host(j).nwdeploy\big) \wedge (host(i)\Gamma' host(j))\big) \quad (6.103)$$

, after applying disjunction of '$\mapsto$' on assertions (6.89) and (6.102), and then simplifying

$$\big\langle \exists j' :: \neg(j' = i) \wedge \big(r \in host(j').\mathbf{T}'\big)\big\rangle$$
$$\mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \vee \neg\big(agent(l).\lambda = host(j').\lambda\big) \quad (6.104)$$

, after quantifying both sides of assertion (6.92) with appropriate range of j'

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big) \mapsto \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle$$
$$\vee\neg\big(agent(l).\lambda = host(j').\lambda\big) \vee \neg\big(\big(host(i).nwdeploy = host(j).nwdeploy\big) \wedge (host(i)\Gamma' host(j))\big) \quad (6.105)$$

, after applying cancellation rule (A.14) on assertions (6.103) and (6.104), and simplifying

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big(r\uparrow \text{dstAg} = host(i).agent(l).aid\big) \wedge \big(r\uparrow \text{dstAg} = host(j').agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle$$
$$\vee\neg\big(agent(l).\lambda = host(j').\lambda\big) \vee \neg\big(\big(host(i).nwdeploy = host(j).nwdeploy\big) \wedge (host(i)\Gamma' host(j))\big) \quad (6.106)$$

, after applying disjunction of '$\mapsto$' on assertions (6.101) and (6.105), and then simplifying

$$\big(r \in host(i).\mathbf{T}'\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \vee \neg\Gamma_{agent(l)} \quad (6.107)$$

, after simplifying assertion (6.106) using assertions (6.87), (6.93) and (6.94)

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \vee \neg\Gamma_{agent(l)} \quad (6.108)$$

, after applying transitivity rule (A.11) on assertions (6.99) and (6.107)

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{last}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \vee \neg\Gamma_{agent(l)} \quad (6.109)$$

, after applying cancellation rule (A.14) on assertions (6.108) and (6.95)

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \vee \neg\Gamma_{agent(l)} \quad (6.110)$$

, after applying cancellation rule (A.14) on assertions (6.110) and (6.96)

This completes the proof of *TRANSFER*(r) property in **System** *CUTSMuHN*. □

Lemma 6.11 *INVOKE property is satisfied in* **System** *CUTSMuHN, i.e.*

$$\big(UsrRdy4Evt = \mathsf{TRUE}\big) \wedge \neg\big(prName = \epsilon\big) \wedge \big\langle \exists\, agent(l) : \big(agent(l).aid \in taids\big)\big\rangle$$
$$\wedge\Gamma_{agent(l)} \wedge \big(prType = \mathsf{remote}\big) \mapsto \big(\texttt{exec}(r^{(1)}\uparrow \text{prName}, r^{(1)}\uparrow \text{data}) = \mathsf{TRUE}\big) \vee \neg\Gamma_{agent(l)}$$

is provable in **System** *CUTSMuHN.*

Proof: The *INVOKE* property is proven with the help of *TRANSFER*(r) property of Host Server, along with the different properties of '$\mapsto$' relation. This proof is based on the following set of assertions:

$$\big(UsrRdy4Evt = \mathsf{TRUE}\big) \wedge (prType = \mathsf{remote}) \wedge \big\langle \exists\, agent(l) : \big(agent(l).aid \in taids\big)\big\rangle \wedge \Gamma_{agent(l)}$$
$$\wedge \neg\big(prName = \epsilon\big) \mapsto \big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(1)}\big) \wedge \big((r^{(1)}\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big) \quad (6.111)$$

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r\big) \quad (6.112)$$

$$\big(\texttt{head}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \quad (6.113)$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big)\big\rangle \vee \neg\Gamma_{agent(l)} \quad (6.114)$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.115}$$

$$\big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.116}$$

$$\big(\texttt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{head}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.117}$$

$$\big(\texttt{head}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big) \mapsto \big(\texttt{exec}(r^{(1)}\uparrow \text{prName}, r^{(1)}\uparrow \text{data}) = \mathsf{TRUE}\big) \tag{6.118}$$

The above assertions describe the safety and progress properties during the invoke phase of remote tuple space primitives through multiple assignment and reactive statements. In the above assertions, the assertion (6.111) asserts that if a mobile agent $agent(k)$ is ready to invoke a remote tuple space primitive $prName$ destined to another agent $agent(l)$, which is available at present, then RT $r^{(1)}$ is generated, encapsulating the required information, and is enqueued in its queue $\mathcal{Q}_{T^S_{a_k}}$. The assertion (6.112) asserts that any RT r eventually reaches the front end of queue $\mathcal{Q}_{T^S_{a_k}}$ of $agent(k)$ from its rear end. The assertion (6.113) asserts that if r is at the front end of the queue $\mathcal{Q}_{T^S_{a_k}}$ of $agent(k)$, then it is eventually enqueued to the queue $\mathcal{Q}_{T^S_{a_k}}$ of $host(i)$ (which supports $agent(k)$). The assertion (6.114) is $TRANSFER(r)$ property, which is proven in Lemma 6.10. The assertion (6.115) and assertion (6.116) assert that if r is at the front end of queue $\mathcal{Q}_{T^R_{a_l}}$ of $host(i)$ or $host(j')$ respectively, then r is eventually enqueued to the queue $\mathcal{Q}_{T^R_{a_l}}$ of target agent $agent(l)$. The assertion (6.117) asserts that if r is at the rear end of queue $\mathcal{Q}_{T^R_{a_l}}$ of $agent(l)$, then it eventually reaches the front end of that queue. Lastly, the assertion (6.118) asserts that once $r^{(1)}$ reaches the front end of queue $\mathcal{Q}_{T^R_{a_l}}$ of $agent(l)$, then eventually the invoked parameters are extracted from $r^{(1)}$ to initiate the tuple space operation at $agent(l)$. Apart from $TRANSFER(r)$, the remaining assertions can be proven by the proof mechanisms shown in Section B.1.1 of Appendix B. These assertions are combined by different properties of '$\mapsto$' as follows:

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(1)}\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(1)}\big) \tag{6.119}$$

, after applying transitivity rule (A.11) on assertions (6.112) and (6.113), as $r^{(1)} = r$

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(1)}\big) \wedge \big((r^{(1)}\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(1)}\big) \wedge \big((r^{(1)}\uparrow \text{dstAg})\uparrow \text{Ag} = agent(l).aid\big) \tag{6.120}$$

, after applying PSP rule (A.16) on assertions (6.119) and (6.98)

$$\big(UsrRdy4Evt = \mathsf{TRUE}\big) \wedge (prType = \mathsf{remote}) \wedge \big\langle \exists\, agent(l) : \big(agent(l).aid \in taids\big)\big\rangle \wedge \Gamma_{agent(l)}$$
$$\wedge \neg\big(prName = \epsilon\big)$$
$$\mapsto \big(\mathtt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big)\big\rangle \vee \neg\Gamma_{agent(l)} \quad (6.121)$$

, after applying transitivity rule (A.11) on assertions (6.111), (6.120) and (6.114) successively

$$\big\langle \exists j' :: \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big)\big\rangle \mapsto \big(\mathtt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big) \quad (6.122)$$

, after quantifying both sides of assertion (6.116) with appropriate range of j'

$$\big(\mathtt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big)\big\rangle$$
$$\mapsto \big(\mathtt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big) \quad (6.123)$$

, after applying disjunction of '$\mapsto$' on assertions (6.115) and (6.122), and then simplifying

$$\big(\mathtt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\mathtt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(1)}\big)\big\rangle$$
$$\mapsto \big(\mathtt{exec}(r^{(1)} \uparrow \mathrm{prName}, r^{(1)} \uparrow \mathrm{data}) = \mathsf{TRUE}\big) \quad (6.124)$$

, after applying transitivity rule (A.11) on assertions (6.123), (6.117) and (6.118) successively

$$\big(UsrRdy4Evt = \mathsf{TRUE}\big) \wedge (prType = \mathsf{remote}) \wedge \big\langle \exists\, agent(l) : \big(agent(l).aid \in taids\big)\big\rangle \wedge \Gamma_{agent(l)}$$
$$\wedge \neg\big(prName = \epsilon\big) \mapsto \big(\mathtt{exec}(r^{(1)} \uparrow \mathrm{prName}, r^{(1)} \uparrow \mathrm{data}) = \mathsf{TRUE}\big) \vee \neg\Gamma_{agent(l)}$$

, after applying cancellation rule (A.14) on assertions (6.121) and (6.124)

This completes the proof that *INVOKE* property holds in **System** *CUTSMuHN*. □

Lemma 6.12 *ACKNOWLEDGE property is satisfied in* **System** *CUTSMuHN, i.e.*

$$\big(\mathtt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \wedge \big(r^{(2)} \uparrow \mathrm{rAid} = agent(k).aid\big) \wedge \big(r^{(2)} \uparrow \mathrm{tAid} = agent(l).aid\big)$$
$$\wedge \big\langle \exists e : (e \in ROL) \wedge \big(r^{(2)} \uparrow \mathrm{prid} = e \uparrow 1\big) \wedge \big(agent(l).aid \in e \uparrow 3\big)\big\rangle$$
$$\wedge \big(\big(r^{(2)} \uparrow \mathrm{prName} = \mathsf{ING}\big) \vee \big(r^{(2)} \uparrow \mathrm{prName} = \mathsf{INGP}\big)\big) \wedge \neg\big(r^{(2)} \uparrow \mathrm{data} = \epsilon\big)$$
$$\mapsto \big(r^{(3)} \uparrow \mathrm{data} = \epsilon\big) \vee \big(\neg\big(r^{(4)} \uparrow \mathrm{data} = \epsilon\big) \wedge \big(r^{(4)} \uparrow \mathrm{data} \in agent(l).\mathbf{T}\big)\big) \vee \neg\Gamma_{agent(l)}$$ *is provable in* **System** *CUTSMuHN.*

Proof: *ACKNOWLEDGE* property is also proven with the help of *TRANSFER*(r) property of Host Server, along with the different properties of '$\mapsto$' relation. This proof is based on the following set of assertions:

$$\big(\mathtt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \wedge \big(r^{(2)} \uparrow \mathrm{rAid} = agent(k).aid\big) \wedge \big(r^{(2)} \uparrow \mathrm{tAid} = agent(l).aid\big) \wedge \neg\big(r^{(2)} \uparrow \mathrm{data} = \epsilon\big)$$

$$\wedge \big\langle \exists e : (e \in ROL) \wedge \big(r^{(2)} \uparrow \text{prid} = e \uparrow 1\big) \wedge \big(agent(l).aid \in e \uparrow 3\big) \big\rangle \wedge \Big(\big(r^{(2)} \uparrow \text{prName} = \mathsf{ING}\big)$$

$$\vee \big(r^{(2)} \uparrow \text{prName} = \mathsf{INGP}\big) \Big) \mapsto \big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\big)$$

$$\vee \big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\big) \tag{6.125}$$

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r\big) \tag{6.126}$$

$$\big(\texttt{head}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \tag{6.127}$$

$$\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r\big) \wedge \big((r \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)$$

$$\mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big) \big\rangle \vee \neg\Gamma_{agent(l)} \tag{6.128}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.129}$$

$$\big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.130}$$

$$\big(\texttt{last}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \mapsto \big(\texttt{head}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r\big) \tag{6.131}$$

$$\big(\texttt{head}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big) \mapsto \big(r^{(3)} \uparrow \text{data} = \varepsilon\big) \tag{6.132}$$

$$\big(\texttt{head}(agent(l).\mathcal{Q}_{T^R_{a_l}}) = r^{(4)}\big) \mapsto \neg\big(r^{(4)} \uparrow \text{data} = \varepsilon\big) \wedge \big(r^{(4)} \uparrow \text{data} \in agent(l).\mathsf{T}\big) \tag{6.133}$$

The above assertions describe the safety and progress properties during the consistency-handing phase of tuples pace coordination, i.e. when responses of remote tuple-consuming primitives have reached the reference agent and results are selected from these responses to effectuate actual tuple space withdrawal. In the above assertions, the assertion (6.125) asserts that if reference agent $agent(k)$ receives RT $r^{(2)}$ from designated target agent $agent(l)$ containing results of invoked remote tuple-consuming operation, it eventually generates either an ack-tuple $r^{(3)}$ or a nack-tuple $r^{(4)}$ with $agent(l)$ as destination, which are subsequently enqueued in its queue $\mathcal{Q}_{T^S_{a_k}}$. The assertion (6.128) is $TRANSFER(r)$ property, which is proven in Lemma 6.10. The assertion (6.126), assertion (6.127), assertion (6.129), assertion (6.130) and assertion (6.131) are already stated as assertion (6.112), assertion (6.113), assertion (6.115), assertion (6.116) and assertion (6.117) respectively. Lastly, the assertion (6.132) and assertion (6.133) assert that once the RT (viz. $r^{(3)}$ or $r^{(4)}$) reaches the front end of queue $\mathcal{Q}_{T^R_{a_l}}$ of $agent(l)$, then the data tuple within $r^{(4)}$ is inserted back to T of $agent(l)$, whereas $r^{(3)}$ is discarded. Apart from the already proven assertions, other assertions can also be proven by the proof mechanisms illustrated in Section B.1.1 of Appendix B. The different properties of '$\mapsto$' relation are

applied on these assertions as follows:

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \tag{6.134}$$

, after applying transitivity rule (A.11) on assertions (6.126) and (6.127), as $r^{(3)} = r$

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big) \tag{6.135}$$

, after applying PSP rule (A.16) on assertions (6.134) and (6.98)

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \tag{6.136}$$

, after applying transitivity rule (A.11) on assertions (6.126) and (6.127), as $r^{(4)} = r$

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big) \tag{6.137}$$

, after applying PSP rule (A.16) on assertions (6.136) and (6.98)

$$\Big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\vee \Big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\mapsto \Big(\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\vee \Big(\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big) \tag{6.138}$$

, after applying disjunction of '$\mapsto$' on assertions (6.135) and (6.137)

$$\Big(\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\vee \Big(\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\mapsto \Big(\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big)\big\rangle\Big) \vee \neg\Gamma_{agent(l)}$$
$$\vee \Big(\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big)\big\rangle\Big) \tag{6.139}$$

, after applying disjunction of '$\mapsto$' on two values of assertion (6.128), viz. $r^{(3)}$ and $r^{(4)}$

$$\Big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\vee \Big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big)$$
$$\mapsto \Big(\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big)\big\rangle\Big) \vee \neg\Gamma_{agent(l)}$$
$$\vee \Big(\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(4)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(4)}\big)\big\rangle\Big) \tag{6.140}$$

, after applying transitivity rule (A.11) on assertions (6.138) and (6.139)

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big) \mapsto \big(r^{(3)} \uparrow \text{data} = \varepsilon\big) \tag{6.141}$$

, applying transitivity rule (A.11) on assertions (6.129), (6.131) and (6.132) successively for $r^{(3)}$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(4)}\big) \mapsto \neg\big(r^{(4)} \uparrow \text{data} = \varepsilon\big) \wedge \big(r^{(4)} \uparrow \text{data} \in agent(l).\mathbf{T}\big) \tag{6.142}$$

, applying transitivity rule (A.11) on assertions (6.129), (6.131) and (6.133) successively for $r^{(4)}$

$$\big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big)\big\rangle \mapsto \big(r^{(3)} \uparrow \text{data} = \varepsilon\big) \tag{6.143}$$

, applying transitivity rule (A.11) on assertions (6.130), (6.131) and (6.132) for $r^{(3)}$ and quantifying

$$\big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big)\big\rangle \mapsto \neg\big(r^{(4)} \uparrow \text{data} = \varepsilon\big) \wedge \big(r^{(4)} \uparrow \text{data} \in agent(l).\mathbf{T}\big) \tag{6.144}$$

, applying transitivity rule (A.11) on assertions (6.130), (6.131) and (6.133) for $r^{(4)}$ and quantifying

$$\begin{aligned}
&\Big(\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(3)}\big)\big\rangle\Big) \\
&\quad \vee \Big(\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_l}}) = r^{(4)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_l}}) = r^{(4)}\big)\big\rangle\Big) \\
&\mapsto \big(r^{(3)} \uparrow \text{data} = \varepsilon\big) \vee \big(\neg\big(r^{(4)} \uparrow \text{data} = \varepsilon\big) \wedge \big(r^{(4)} \uparrow \text{data} \in agent(l).\mathbf{T}\big)\big)
\end{aligned} \tag{6.145}$$

, applying disjunction of '$\mapsto$' on assertions (6.141), (6.142), (6.143) and (6.144), and simplifying

$$\begin{aligned}
&\Big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(3)}\big) \wedge \big((r^{(3)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big) \\
&\quad \vee \Big(\big(\texttt{last}(agent(k).\mathcal{Q}_{T^S_{a_k}}) = r^{(4)}\big) \wedge \big((r^{(4)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(l).aid\big)\Big) \\
&\mapsto \big(r^{(3)} \uparrow \text{data} = \varepsilon\big) \vee \big(\neg\big(r^{(4)} \uparrow \text{data} = \varepsilon\big) \wedge \big(r^{(4)} \uparrow \text{data} \in agent(l).\mathbf{T}\big)\big) \vee \neg\Gamma_{agent(l)}
\end{aligned} \tag{6.146}$$

, after applying cancellation rule (A.14) on assertions (6.140) and (6.145)

$$\begin{aligned}
&\big(\texttt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \wedge \big(r^{(2)} \uparrow \text{rAid} = agent(k).aid\big) \wedge \big(r^{(2)} \uparrow \text{tAid} = agent(l).aid\big) \wedge \neg\big(r^{(2)} \uparrow \text{data} = \epsilon\big) \\
&\quad \wedge \big\langle \exists e : (e \in ROL) \wedge \big(r^{(2)} \uparrow \text{prid} = e \uparrow 1\big) \wedge \big(agent(l).aid \in e \uparrow 3\big)\big\rangle \\
&\quad \wedge \Big(\big(r^{(2)} \uparrow \text{prName} = \mathsf{ING}\big) \vee \big(r^{(2)} \uparrow \text{prName} = \mathsf{INGP}\big)\Big) \\
&\mapsto \big(r^{(3)} \uparrow \text{data} = \varepsilon\big) \vee \big(\neg\big(r^{(4)} \uparrow \text{data} = \varepsilon\big) \wedge \big(r^{(4)} \uparrow \text{data} \in agent(l).\mathbf{T}\big)\big) \vee \neg\Gamma_{agent(l)}
\end{aligned}$$

, after applying transitivity rule (A.11) on assertions (6.125) and (6.146)

This completes the proof of *ACKNOWLEDGE* property in **System** *CUTSMuHN*. □

Lemma 6.13 *RESPONSE property is satisfied in* **System** *CUTSMuHN, i.e.*

$$\begin{aligned}
&\big(\texttt{exec}(r^{(1)} \uparrow \text{prName}, r^{(1)} \uparrow \text{data}) = \mathsf{TRUE}\big) \wedge \big(r^{(1)} \uparrow \text{rAid} = agent(k).aid\big) \wedge \Gamma_{agent(l)} \\
&\wedge \neg\big((r^{(1)} \uparrow \text{prName} = \mathsf{OUT}) \vee (r^{(1)} \uparrow \text{prName} = \mathsf{OUTG})\big) \wedge \big(r^{(1)} \uparrow \text{tAid} = agent(l).aid\big) \\
&\mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \vee \neg\Gamma_{agent(l)}
\end{aligned}$$

is provable in **System** *CUTSMuHN*.

Proof: *RESPONSE* property is again proven with the help of *TRANSFER*(r) property of Host Server, and different properties of '$\mapsto$' relation. This proof is based on the following set of assertions:

$$\begin{aligned}
&\big(\texttt{exec}(r^{(1)}\uparrow \text{prName}, r^{(1)}\uparrow \text{data}) = \mathsf{TRUE}\big) \wedge \big(r^{(1)}\uparrow \text{rAid} = agent(k).aid\big) \wedge \Gamma_{agent(k)} \\
&\quad \wedge \neg\big((r^{(1)}\uparrow \text{prName} = \mathsf{OUT}) \vee (r^{(1)}\uparrow \text{prName} = \mathsf{OUTG})\big) \wedge \big(r^{(1)}\uparrow \text{tAid} = agent(l).aid\big) \\
&\mapsto \big(\big(\texttt{last}(agent(l).\mathcal{Q}_{T^S_{a_l}}) = r^{(2)}\big) \wedge \big((r^{(2)}\uparrow \text{dstAg})\uparrow \text{Ag} = agent(k).aid\big)\big) \qquad (6.147)
\end{aligned}$$

$$\big(\texttt{last}(agent(l).\mathcal{Q}_{T^S_{a_l}}) = r\big) \mapsto \big(\texttt{head}(agent(l).\mathcal{Q}_{T^S_{a_l}}) = r\big) \qquad (6.148)$$

$$\big(\texttt{head}(agent(l).\mathcal{Q}_{T^S_{a_l}}) = r\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_l}}) = r\big) \qquad (6.149)$$

$$\begin{aligned}
&\big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_l}}) = r\big) \wedge \big((r\uparrow \text{dstAg})\uparrow \text{Ag} = agent(k).aid\big) \\
&\quad \mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_k}}) = r\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_k}}) = r\big)\big\rangle \vee \neg\Gamma_{agent(k)} \qquad (6.150)
\end{aligned}$$

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_k}}) = r\big) \mapsto \big(\texttt{last}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r\big) \qquad (6.151)$$

$$\big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_k}}) = r\big) \mapsto \big(\texttt{last}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r\big) \qquad (6.152)$$

$$\big(\texttt{last}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \qquad (6.153)$$

The above assertions describe the safety and progress properties during the response-delivery phase of tuples pace coordination, i.e. when responses of remote tuple-consuming primitives are dispatched to reference agent to carry out consistency-handling mechanisms. Among the above assertions, the assertion (6.147) asserts that if the execution of any remote tuple space primitive (except tuple-producing primitives) at target agent $agent(l)$ is over and reference agent $agent(k)$ is available at present, then RT $r^{(2)}$ is generated, encapsulating the results of remote invoke as well as destined to $agent(k)$, and is enqueued in its queue $\mathcal{Q}_{T^S_{a_l}}$. The assertion (6.150) is *TRANSFER*(r) property, which is proven in Lemma 6.10. Again, the assertion (6.148), assertion (6.149), assertion (6.151), assertion (6.152) and assertion (6.153) are already stated as assertion (6.112), assertion (6.113), assertion (6.115), assertion (6.116) and assertion (6.117) respectively. Apart from the already proven assertions, the remaining assertions can also be proven by the proof mechanisms explained in Section B.1.1 of Appendix B. The different properties of '$\mapsto$' relation are enforced on these assertions to show that *RESPONSE* property holds

in **System** *CUTSMuHN*.

$$\big(\texttt{last}(agent(l).\mathcal{Q}_{T^S_{a_l}}) = r^{(2)}\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_l}}) = r^{(2)}\big) \tag{6.154}$$

, after applying transitivity rule (A.11) on assertions (6.148) and (6.149), as $r^{(2)} = r$

$$\big(\texttt{last}(agent(l).\mathcal{Q}_{T^S_{a_l}}) = r^{(2)}\big) \wedge \big((r^{(2)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(k).aid\big)$$
$$\mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{T^S_{a_l}}) = r^{(2)}\big) \wedge \big((r^{(2)} \uparrow \text{dstAg}) \uparrow \text{Ag} = agent(k).aid\big) \tag{6.155}$$

, after applying PSP rule (A.16) on assertions (6.154) and (6.98)

$$\big(\texttt{exec}(r^{(1)} \uparrow \text{prName}, r^{(1)} \uparrow \text{data}) = \mathsf{TRUE}\big) \wedge \big(r^{(1)} \uparrow \text{rAid} = agent(k).aid\big) \wedge \Gamma_{agent(k)}$$
$$\wedge \neg\big((r^{(1)} \uparrow \text{prName} = \mathsf{OUT}) \vee (r^{(1)} \uparrow \text{prName} = \mathsf{OUTG})\big) \wedge \big(r^{(1)} \uparrow \text{tAid} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big)\big\rangle \vee \neg\Gamma_{agent(k)} \tag{6.156}$$

, after applying transitivity rule (A.11) on assertions (6.147), (6.155) and (6.150) successively

$$\big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big)\big\rangle \mapsto \big(\texttt{last}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \tag{6.157}$$

, after quantifying both sides of assertion (6.152) with appropriate range of j'

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big)\big\rangle$$
$$\mapsto \big(\texttt{last}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \tag{6.158}$$

, after applying disjunction of '$\mapsto$' on assertions (6.151) and (6.157), and then simplifying

$$\big(\texttt{head}(host(i).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \vee \big\langle \exists j' :: \neg(j' = i) \wedge \big(\texttt{head}(host(j').\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big)\big\rangle$$
$$\mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \tag{6.159}$$

, after applying transitivity rule (A.11) on assertions (6.158) and (6.153)

$$\big(\texttt{exec}(r^{(1)} \uparrow \text{prName}, r^{(1)} \uparrow \text{data}) = \mathsf{TRUE}\big) \wedge \big(r^{(1)} \uparrow \text{rAid} = agent(k).aid\big) \wedge \Gamma_{agent(k)}$$
$$\wedge \neg\big((r^{(1)} \uparrow \text{prName} = \mathsf{OUT}) \vee (r^{(1)} \uparrow \text{prName} = \mathsf{OUTG})\big) \wedge \big(r^{(1)} \uparrow \text{tAid} = agent(l).aid\big)$$
$$\mapsto \big(\texttt{head}(agent(k).\mathcal{Q}_{T^R_{a_k}}) = r^{(2)}\big) \vee \neg\Gamma_{agent(k)}$$

, after applying cancellation rule (A.14) on assertions (6.156) and (6.159)

This completes the proof of *RESPONSE* property holding in **System** *CUTSMuHN*. □

Based on the proofs of these lemmas, it can be concluded that the formal correctness property of **System** *CUTSMuHN*, expressed in Equation (6.14), has been proven.

That is, starting with the proofs of $TRANSPORT(M)$ property, $ADVERTISE(\mathcal{A}, \mathcal{N})$ property and $REGISTER(M)$ satisfying in **System** *CUTSMuHN*, *INVOKE* property, *ACKNOWLEDGE* property and *RESPONSE* property are proven to be satisfied in **System** *CUTSMuHN*. The last 3 properties represents the working of remote tuple space operation in $\mathcal{CUTSMuHN}$ platform and becomes the basis of its tuple space coordination. Moreover, two specific intermediate properties, viz. $DECOUPLE_{IntraHost}$ and $DECOUPLE_{InterHost}$, forms the essence of decoupling in tuple space coordination of $\mathcal{CUTSMuHN}$ platform and has been proven to hold in **System** *CUTSMuHN*.

6.3 Related Work

The proof logic of the Mobile UNITY model has rarely been used in literature for modeling of tuple space coordination [99]. In [99], the formal representations of different coordination models are framed and a common strategy of their verification is hinted. This chapter differs from [99] in the approach of verification for deriving correctness. In this chapter, a detailed approach of verifying the behaviors of an entire middleware platform is presented, which is lacking in [99]. The formal verification of other systems, like Mobile IP [83], as well as functionalities within a system, viz. code mobility [95], using the Mobile UNITY model also exist in literature. Other extensions of the UNITY model for formal verification of different systems also exists. For instance, [104] shows an extension of the UNITY model to verify tightly-coupled system, which is radically different from the verification approach for very loosely-coupled $\mathcal{CUTSMuHN}$ platform. The formal verification of different systems using the UNITY model has also shown in the literature [102, 103, 31].

6.4 Conclusion

This chapter has augmented the proposed formal modeling of $\mathcal{CUTSMuHN}$ platform (presented in the preceding chapter) with the proposed strategy of verifying its correctness. The proposed verification strategy, in itself, is thorough, and has also shown to satisfy the fundamental capability of tuple space coordination, viz. decoupling, in $\mathcal{CUTSMuHN}$

platform. These outcome can be utilized to demonstrate the correctness in formalizing decoupling for any tuple space middleware platform. Moreover, implementation and experimentation of *CUTSMuHN* platform supplements the proposed modeling and verification by validating its correct working and demonstrating its capabilities, including versatility, efficiency etc. Hence, *CUTSMuHN* platform can be reasoned as coordination platform for mobile applications in heterogeneous dynamic networks.

Chapter 7

Implementation and Experimentation of *CUTSMuHN*

7.1 Introduction

The emphasis of this thesis is to propose methodologies on different research directions of a TSMM platform, incorporating them into *CUTSMuHN* platform, and formally modeling as well as verifying correctness of its behaviors. However, the implementation of *CUTSMuHN* platform for experimentation purpose is to back up the claim of its correctness. This chapter describes an implementation approach of a prototype of *CUTSMuHN* platform using Java™ [54], and tests its correct working in JVM 1.6.0_17 [71] with developed mobile applications. This chapter also reports the experiments on performances of tuple space operations of *CUTSMuHN* platform to compare its improvements.

7.2 Implementation of *CUTSMuHN*

The implementation of *CUTSMuHN* platform follows a modular approach. In this section, a step-by-step implementation of its working prototype is described, starting with its basic building blocks, viz. tuple, antituple, tuple space, agent and host. *CUTSMuHN* platform is implemented following the object-oriented paradigm using Java™. As such, the deployment of *CUTSMuHN* platform requires Java-enabled hosts. Its implementation is subdivided into multiple modules, in line with its architecture. Each module, in turn, is further subdivided based on the functionality of that module. The present prototype

of *CUTSMuHN* platform supports fully-functional stationary agents, which is going to be extended to support agent mobility. The implementation of agent includes two major submodules for handling coordination and engagement. On the other hand, host is implemented by including the functionalities of Host Server, Agent Manager, Communication Manager and Discovery Manager as major submodules. The remaining functionalities of *CUTSMuHN* platform are clubbed into other submodules.

Tuple: A tuple is created using the `Tuple` class, which is defined as a public class in this implementation. Some of its significant methods are provided as public interfaces (i.e. APIs) for application programmers. A glimpse of the `Tuple` class is shown below.

```
public class Tuple implements TSConstantsIntf
{
   public Tuple(int noField);
   public Tuple(); // Constructor for Empty Tuple
   public boolean chkEmptyTuple();
   public int getFieldCount();
   public int getHashTabLength();
   public int getRehashCount();
   public int[][] getRandom(); // For use by AntiTuple class
   public TField[][] getTable();
   public TField[] getAllFields();
   public TField getField(Class Typ);
   public boolean updateField(TField TF);
   public String showTuple();
   public String showTupleConcised();
   public Tuple addFields(TField[] fields);
}
```

The `Tuple` class uses another class, viz. `TField` class, to declare each of its constituent fields. The `TField` class has following public interfaces.

```
public class TField implements Serializable
{
   public TField(Object t);
   public TField(); // Constructor for Empty Tuple Field
   public boolean getPresent();
   public Class getType();
   public Object getValue(); // Possible NULL value return
}
```

Antituple: An antituple is created using the `AntiTuple` class, which is also defined as a public class. Like the `Tuple` class, several significant methods of the `AntiTuple` class are also provided as public interfaces, as shown below.

```
public class AntiTuple implements Serializable
{
   public AntiTuple(int NOF);
   public AntiTuple(); // Constructor for empty Antituple
   public boolean chkEmptyAT();
   public int getFieldCount();
   public ATField[] getFields();
   public ATField getFirstField();
   public void addFields(ATField[] fields);
   public boolean match(Tuple t) throws ClassNotFoundException;
   public String showAntiTuple();
}
```

The `AntiTuple` class uses the additional `ATField` class to represent each of its constituent fields. The `ATField` class has the following public interfaces.

```
public class ATField implements Serializable
{
   public ATField(Class type);
   public ATField(Class type, Object value);
   public ATField(); // Constructor for Empty AntiTuple Field
   public boolean getPresent();
   public boolean getFormal();
   public Class getType();
   public Object getValue(); // Possible NULL value return
   public boolean equals(TField tF);
}
```

Agent Tuple Space: A tuple space, used for data coordination, is designed as an ATS, which is implemented using another public class, called `AgentTupleSpace` class. This class defines the local tuple space primitives (viz. `out`, `outg`, `rdp`, `rdgp`, `inp` and `ingp`) as public methods. These primitives operate locally on the actual tuple space that is encapsulated within the `AgentTupleSpace` class. The public interfaces of this class are shown next.

```
public final class AgentTupleSpace implements Serializable
{
    public AgentTupleSpace(String name);
    public String getTSname();
    public boolean isEmpty();
    public void out(Tuple t); // Local single tuple-producing operation
    public void outg(Tuple[] tuples); // Local bulk tuple-producing operation
    public Tuple rdp(AntiTuple at); // Local single tuple-reading operation
    public Tuple[] rdgp(AntiTuple at); // Local bulk tuple-reading operation
    public Tuple inp(AntiTuple at); // Local single tuple-consuming operation
    public Tuple[] ingp(AntiTuple at); // Local bulk tuple-consuming operation
    public void showTupleSpace();
    public void showDetailedTupleSpace();
    public void showTupleSpacePreamble();
}
```

Agent: The `Agent` class is a fundamental class in the implementation of *CUTSMuHN* platform. The two classes, viz. `StaticAgent` and `MobileAgent`, are derived from the `Agent` class. While developing applications of *CUTSMuHN* platform, application programmers have to extend either the `StaticAgent` class or the `MobileAgent` class within their applications to incorporate tuple space coordination. The class diagram of related classes implemented at the agent-level are shown in Figure 7.1. The public methods of some of the classes, which are already presented, are not shown in Figure 7.1 for better readability.

Every instance of the `Agent` class holds a private instance of `AgentTupleSpace` class. Reactivity is also maintained by the `Agent` class using `ReactionList` class. After any tuple-producing operation (viz. `out` or `outg`), all registered reactions from the `ReactionList` class are invoked in the pre-defined order to execute their reactive codes. Moreover, the user-defined engagement policies for asymmetric interactions are imposed by the `EngagementManager` class, and reference agent checks this class before invoking/responding any remote operation. In case of enforcing OUT-consistency, the positive acknowledgement is implemented using the `ACK` class, which is derived from the

`Acknowledgement` class. Whereas, the negative acknowledgement is implemented using the `NACK` class, which is also derived from the `Acknowledgement` class. The public interfaces of the `Agent` class are shown as follows.

```
public class Agent extends Thread implements AgentIntf, Serializable, TSConstantsIntf
{
    public Agent(String applID, String TSname);
    public Agent(AgentID aid);
    public void start();
    public void stopTh();
    public void run();
    public void Shutdown(boolean flag);
    public AgentID getAID();
    public void out(Tuple t);
    public void outg(Tuple[] ts);
    public Tuple rd(AntiTuple at);
    public Tuple rdp(AntiTuple at);
    public Tuple[] rdg(AntiTuple at);
    public Tuple[] rdgp(AntiTuple at);
    public Tuple in(AntiTuple at);
    public Tuple inp(AntiTuple at);
    public Tuple[] ing(AntiTuple at);
    public Tuple[] ingp(AntiTuple at);
    public void showDetailedATS();
    public void showATS();
}
```

An instance of the `StaticAgent` class spawns two additional separate threads for handling remote tuple space operations. One of the threads is an instance of the `RemoteTSOpManager` class, which handles all remote operations invoked by the instance of `StaticAgent` class (as reference agent). The `RemoteTSOpManager` converts the invoke of any remote tuple space primitive (by reference agent) into a reaction-tuple using the `RTuple` class, and subsequently hands it over to the instance of `HostServer` class. Again, when the results of any invoked remote primitive arrives from target agent in the form of response-tuple, which is also implemented using the `RTuple` class, `RemoteTSOpManager` class extracts sought tuple(s) from that response-tuple and returns it to the `StaticAgent` class. The other thread, which is an instance of the `AgentTSReactManager` class, acts as

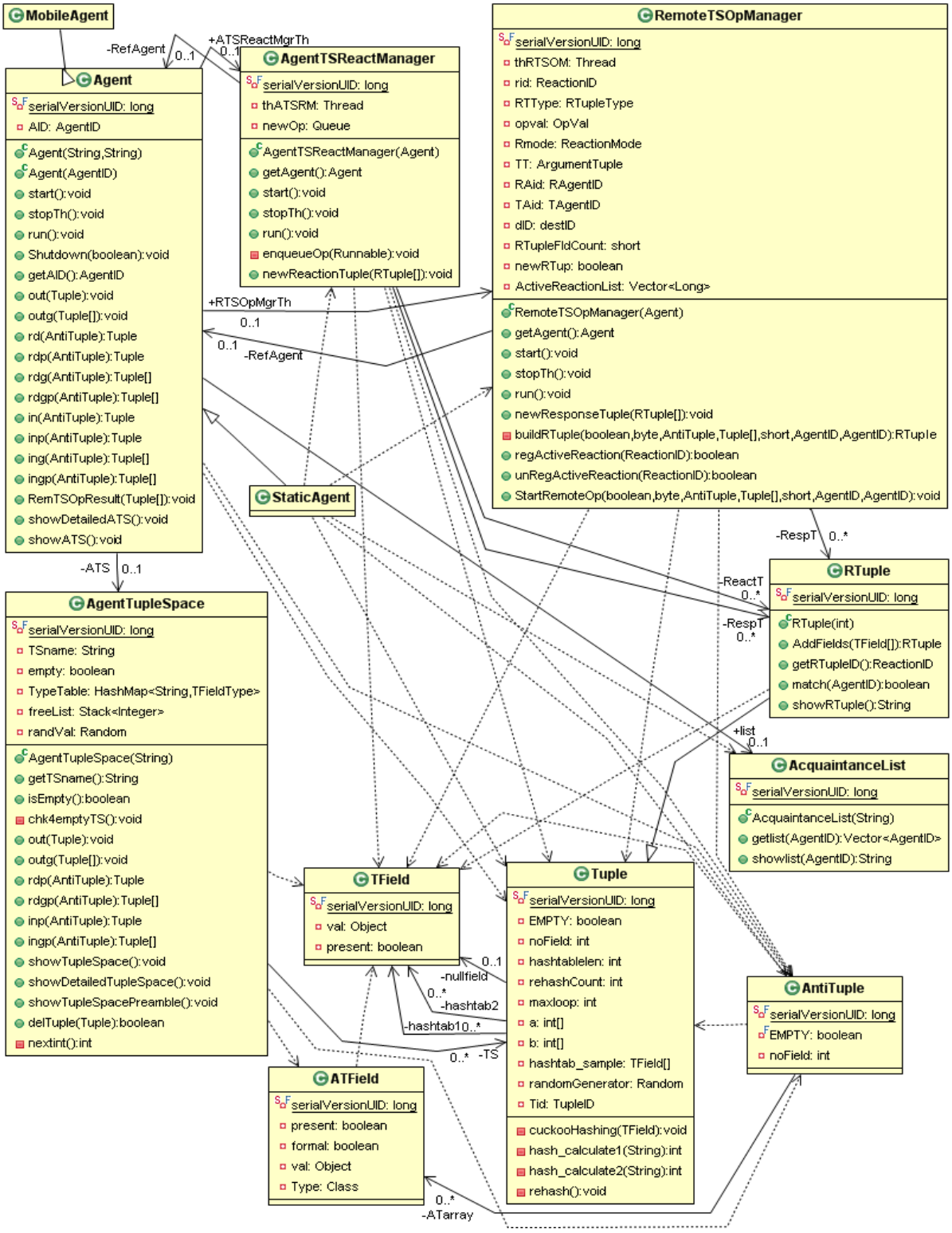


Figure 7.1: Class diagram of the agent-level classes of CUTSMuHN.

a listener for any incoming remote operation request. When any such request comes (via reaction-tuple), it extracts invoke parameters of the operation from that reaction-tuple and hands it to another instance of the `StaticAgent` class, that happens to be target agent for that operation. Later, the `AgentTSReactManager` class converts the results of remote operation into response-tuple, and gives to the `HostServer` class for delivery back to reference agent.

Host: The `HostServer` class, another fundamental component in the implementation of *CUTSMuHN* platform, initiates all instances of required classes, starts all relevant threads, and continues to run itself with its own HTS, which is implemented as the instance of `HostTupleSpace` class. *CUTSMuHN* platform starts its execution in any host, if an instance of the `HostServer` class starts running in that host. However, it is the responsibility of end-user to instantiate the `HostServer` class to run in a host. The class diagram of related classes implemented at host-level are shown in Figure 7.2.

Among the threads that `HostServer` class spawns and instantiates, the instances of `AgentManager` class, `DiscoveryManager` class, and `CommManager` class are significant. Each of these threads, in turn, handles the major functionalities of *CUTSMuHN* platform. Whenever the instances of `RemoteTSOpManager` or `AgentTSReactManager` or `CommManager` hands over some `RTuple` to the instance of `HostServer` class, it stores them in its `HostTupleSpace`. Periodically, it looks for any pending `RTuple` in `HostTupleSpace`, checks the availability of their destination after consulting with `AgentManager` as well as `DiscoveryManager`, and withdraws them to hand over to `RemoteTSOpManager` or `AgentTSReactManager` for local agents, and `CommManager` in case of remote agents. The `HostServer` class, however, does not provide any APIs for application programmers.

The agents can start executing once they have registered to `AgentManager`. Periodically, `AgentManager` intimates `DiscoveryManager` about the agents running in this host. `DiscoveryManager` holds a *Neighbor List*, which is implemented as a private instance of `NeighborList` class, to update the lists of agents from `AgentManager` as well

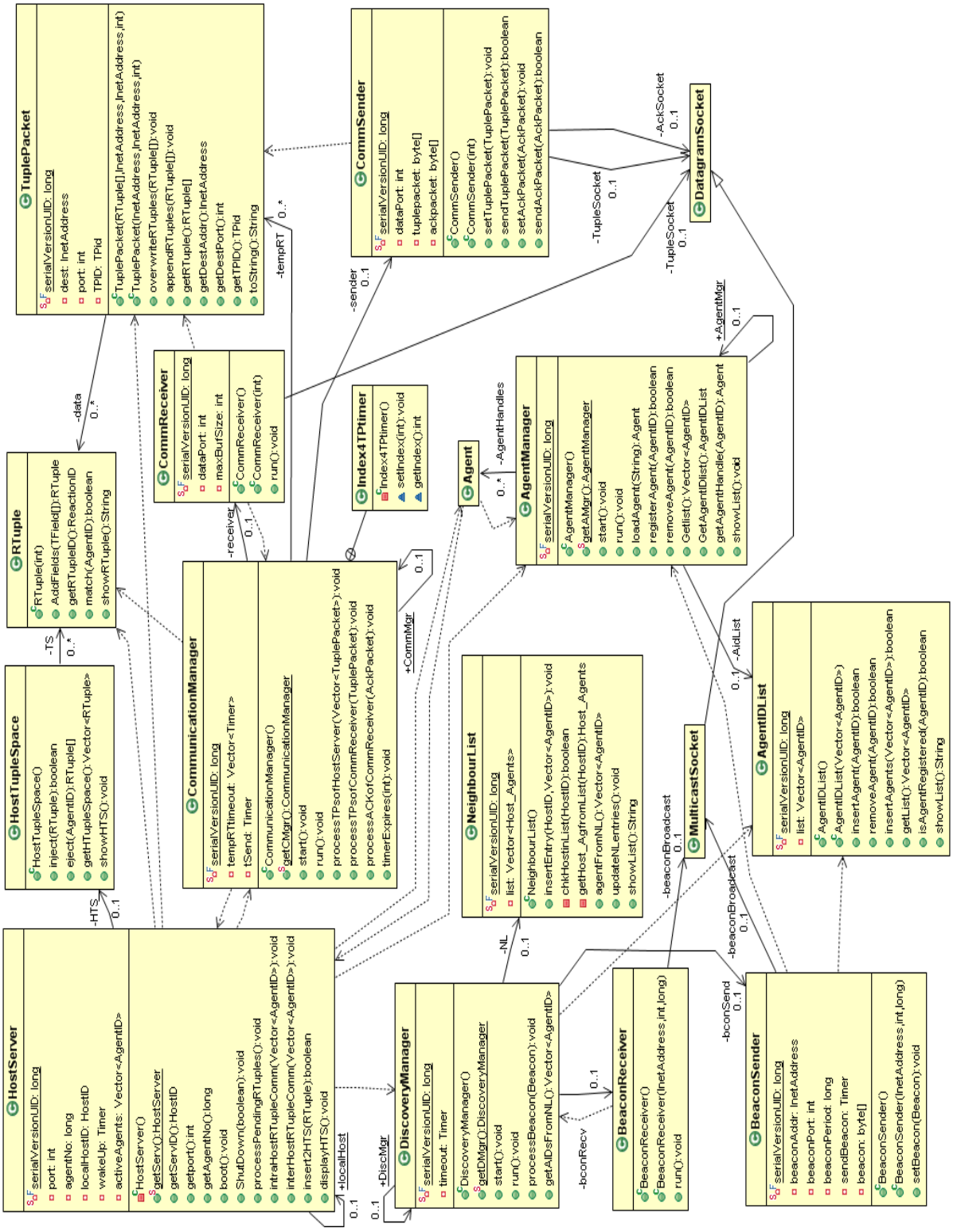


Figure 7.2: Class diagram of the host-level classes of CUISMuHN.

as from other hosts (when they inform by means of beaconing mechanism). The beacons are implemented as instances of the `Beacon` class. `DiscoveryManager` periodically generates a beacon, and hands it over to one of the threads it spawns, viz. an instance of `BeaconSender` class. The other thread it spawned, viz. the instance of `BeaconReceiver` class, listens to beacon broadcasts, and hands over the received `Beacon`s to `DiscoveryManager`. The pair of `BeaconSender` and `BeaconReceiver` threads uses a pre-defined port (for the IP Multicasting protocol) of a particular network interface to carry out discovery operations. If a host contains multiple NICs, one pair of `BeaconSender-BeaconReceiver` threads are spawned for each NIC.

On the other hand, `CommManager` thread uses the datagram protocol to transmit RT-message (containing reaction/response-tuples) from one host to another. It follows a separate pre-defined port for transferring datagrams generated from RT-messages. The `CommManager` thread also spawns a listener, which is an instance of the `CommListener` class, to listen to any incoming datagram at the pre-defined port. RT from the received datagram is subsequently handed over to the `HostServer` thread for further processing. The `CommManager` thread uses the proposed communication mechanisms to ensure reliability in RT-message communication, and to detect their failed transmissions. ACK-message is implemented using the `Ack` class, that is derived from the `DatagramPacket` class. When a host contains multiple NICs, multiple `CommListener` threads are spawned, one for each NIC. However, only one instance of the `CommManager` thread runs for all NICs. While sending datagrams, it spawns multiple temporary threads, one for each NIC, to transmit the generated datagrams through those interfaces.

The aforementioned implementation approach of the prototype of *CUTSMuHN* platform includes the support of its working in wired networks. The support for working in wireless networks (namely, iBSS, IBSS and H-MANET) can be implemented as separate modules and can be easily incorporated in the prototype of *CUTSMuHN* platform, due to its modular implementation approach. For testing about its correct working in wired environment, two mobile applications have been developed using its APIs. The next section deals with the implementation of these supported mobile applications.

7.3 Applicability of $\mathcal{CUTSMuHN}$

The focus of this thesis is to formally verify the correctness of $\mathcal{CUTSMuHN}$ platform. Nonetheless, demonstrating its coordination ability through experimentation is still beneficial. A proficient way of demonstrating its ability to support robust and flexible coordination of mobile applications is by building these applications. Literature shows different sample applications of existing TSMM platforms, including collaborative applications, applications discovering changes in predefined contextual information etc. This section briefs two mobile applications of $\mathcal{CUTSMuHN}$ platform, viz. *MyChat* and *RoamingChess*, both of which are collaborative as well as include automatic discovery of changes in predefined context. These applications not only can demonstrate the multi-dimensional decoupling ability of agent interactions in $\mathcal{CUTSMuHN}$ platform, particularly complete synchronization decoupling, but also its reliability over multiple underlying networks.

7.3.1 *MyChat*

The *MyChat* application is a simple multi-user chat application of $\mathcal{CUTSMuHN}$ platform, where multiple end-users can chat through their own devices. The two screenshots about the working of *MyChat* application using the support of $\mathcal{CUTSMuHN}$ platform is shown in Figure 7.3 and Figure 7.4. These two figures together exhibit the coordination ability of $\mathcal{CUTSMuHN}$ platform for multiple agents, running in different hosts. Figure 7.3 shows that the two agents (corresponding to end-users "carol" and "dave") are instantiated in one host (identified by IP address 172.16.26.241), whereas Figure 7.4 shows the other two agents (corresponding to end-users "alice" and "bob") instantiated in another host (identified by IP address 172.16.26.240) at the same time. The two unassigned ports of Internet Assigned Numbers Authority (IANA), viz. 7735 and 7737, are used for discovery and communication functionalities respectively. Multiple agents of same application in the same host are simply distinguished by assigning different natural numbers.

In *MyChat* application, each chat message is converted to an unordered tuple, with the tuple fields to store the identity of sender (i.e. agent and host), identity of receiver, and the message itself. For example, a tuple corresponding to a chat message sent by "dave"

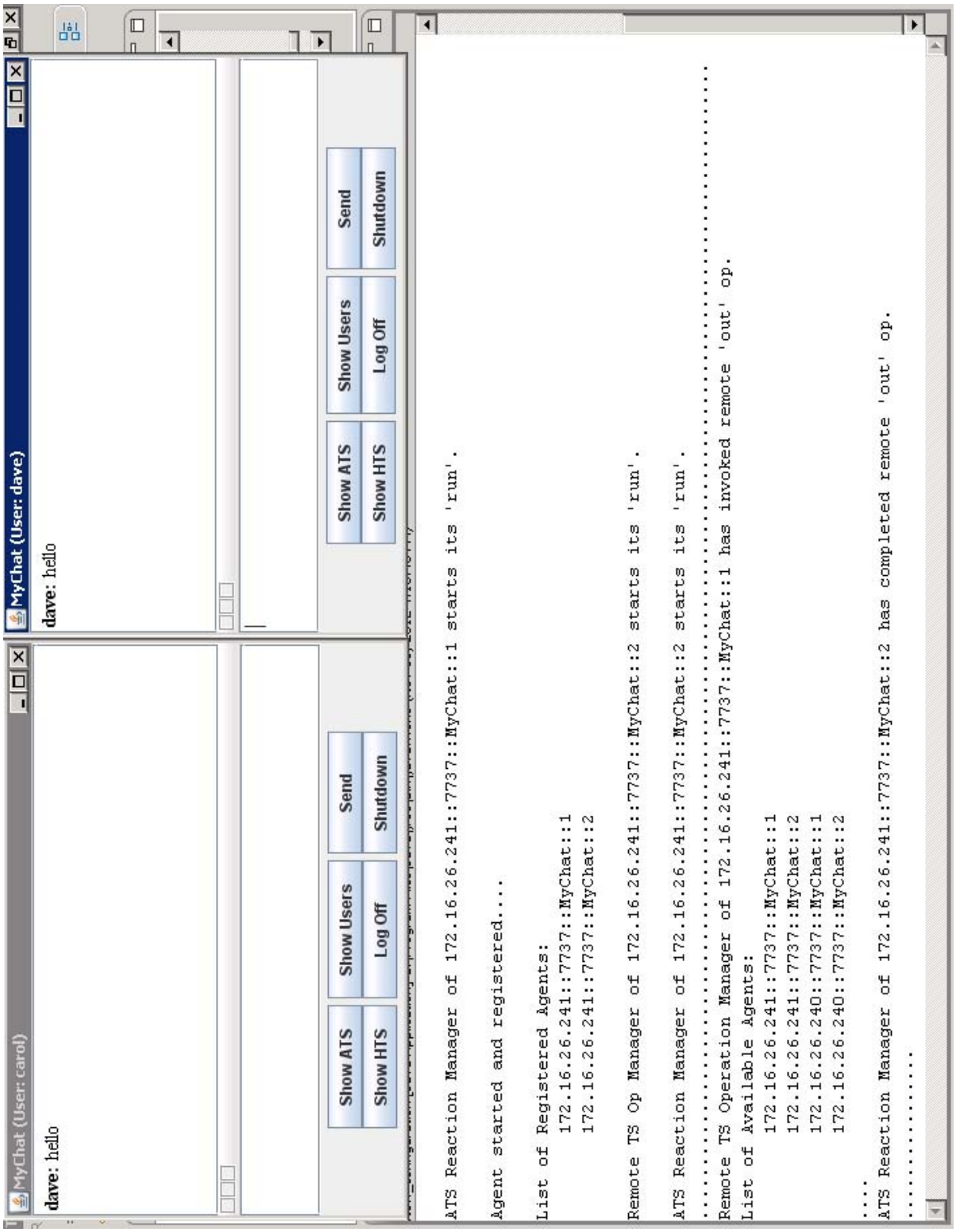


Figure 7.3: Working of *MyChat* application of $\mathcal{CUTSMuHN}$: First Screenshot.

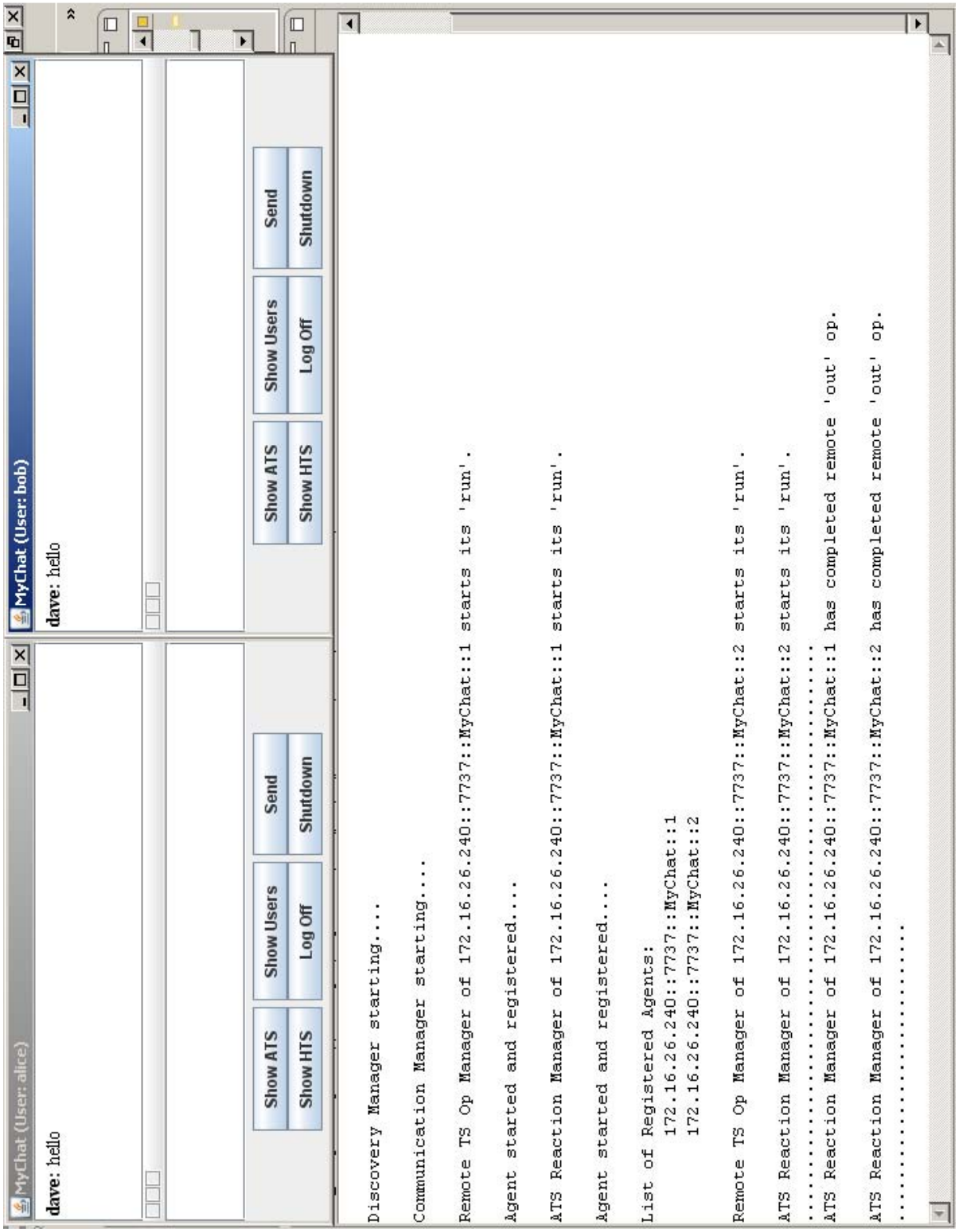


Figure 7.4: Working of *MyChat* application of *CUTSMuHN*: Second Screenshot.

in Figure 7.3 is {"172.16.26.241::7737::MyChat::1", "172.16.26.241::7737::MyChat::2", "hello"}. In fact, *MyChat* application works by broadcasting (at application-level) every chat message to all active end-users. This behavior imitates the chat room concept in *MyChat* application. The process of broadcasting is implemented by writing to every agent's ATS using remote `out` primitive, and withdrawing by respective agents after invoking local `inp` primitive from their own ATS. Consequently, a chat message submitted by "dave" results in the creation of three tuples for three target agents, one of which is shown above, and the three invokes of remote `out` primitive with each of these tuples as argument (the case of multi-destined tuple space operation). Each such invoke generates a reaction-tuple, which is given to *CUTSMuHN* platform to deliver to its respective destination. These processes are carried out by the `RemoteTSOpManager` thread of reference agent. When *CUTSMuHN* platform delivers a reaction-tuple to a target agent, it is received by the `AgentTSReactManager` thread of target agent to complete the operation, viz. inserting tuple (that is extracted from the received reaction-tuple) to ATS of that target agent. The completion of remote `out` primitive is evident from the last entries in screenshots of Figure 7.3 and Figure 7.4. Eventually, all target agents invoke their local `inp` primitive to withdraw that chat message and display in their respective chat windows, as also evident from the screenshots.

7.3.2 *RoamingChess*

The *RoamingChess* application is a multiplayer gaming application of *CUTSMuHN* platform, where two players are playing a chess game on their own devices using virtual chessboards. The multiple chess games can be held simultaneously between different set of players in the same device. The *RoamingChess* application can be thought of as a digitized version of "correspondence chess" [101]. Before starting of each game, both the players have to agree to play in the same game; accordingly, the application sets its engagement policy to show only the opponent player(s) in the respective list of friendly agents. Moreover, the *RoamingChess* application requires several dimensions of decoupling (namely temporal decoupling, spatial decoupling, complete synchronization decoupling etc.), which can be best provided by *CUTSMuHN* platform.

In *RoamingChess* application, each move is represented as an unordered tuple, in which the tuple fields store mandatory information like 'name of player', 'name of piece', 'color of piece', 'source file', 'source rank', 'destination file', 'destination rank' and 'timestamp of move', as well as few optional data like 'capture details', 'castling details', 'en passant details', 'promotion details', 'check information' and 'end of game details'. For instance, one possible tuple generated in a game is {"Player1", "N", "black", "f", 6, "e", 4, "xN", "+Kd2", 20120930T113025}, which states that "a move of a black knight of Player1 from f6 to e4, capture of a white knight standing at e4 and giving check to opponent king standing at d2". This shows that the information encapsulated in a tuple are an annotated and expanded version of standard notation followed in a chess game. Once a player makes the move, the *RoamingChess* application generates the corresponding tuple and stores it in a local tuple space. The opponent player retrieves the tuple corresponding to the latest move from that tuple space by registering a reaction with ONCE/TUPLE mode.

The screenshots of the working of *MyChat* application of *CUTSMuHN* platform is sufficient to establish its correctness, and can be further extended to demonstrate its robustness and flexibility. In order to show the performance improvements of the tuple space model in the working prototype of *CUTSMuHN* platform, it is experimented with other TSMM platforms in the next section.

7.4 Experimentation with *CUTSMuHN*

This section focuses on the performance evaluation of local tuple space operations in the working prototype of *CUTSMuHN* platform. For that purpose, its incorporated tuple space model is executed in JVM 1.6.0_17, along with other unordered tuple space models of existing TSMM platforms, for measuring the time complexities of `out`, `rdgp` and `ingp` in two different scenarios of tuple/antituple field values. These three primitives represent their respective categories and the performances of remaining tuple-producing/reading/consuming primitives can be understood from their measurements.

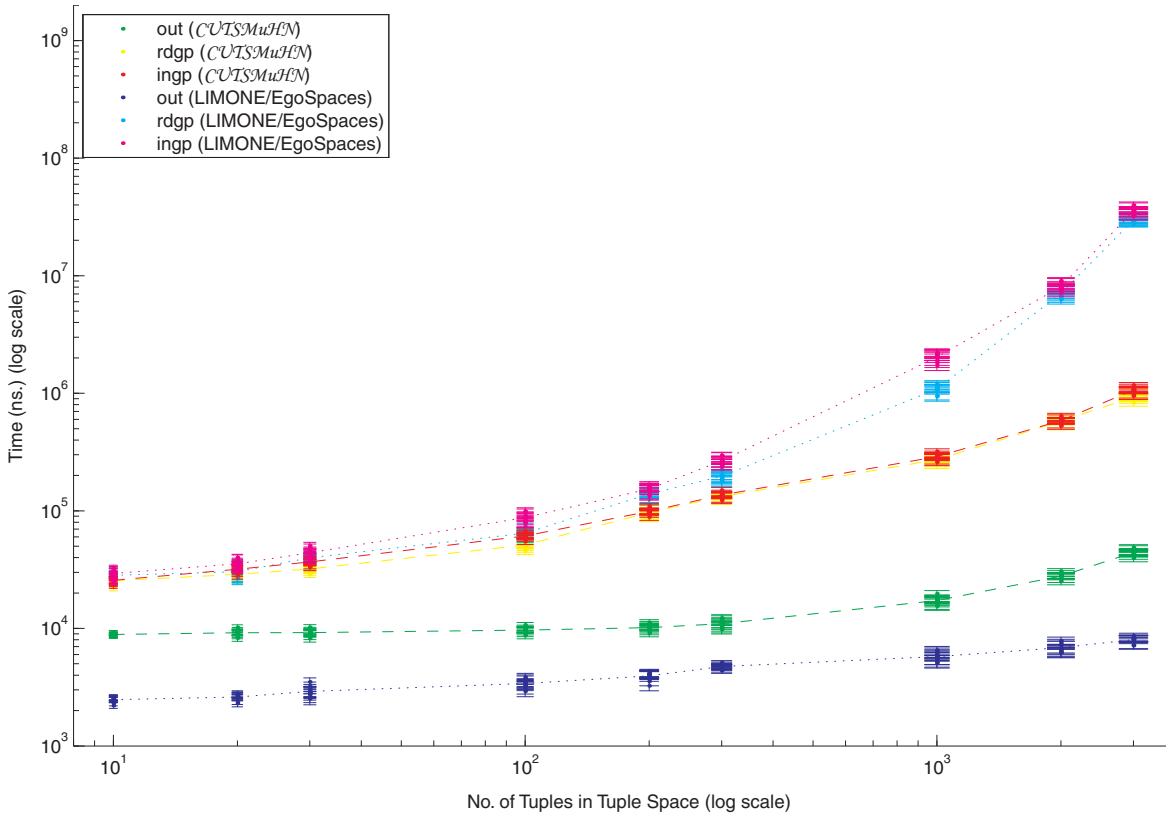


Figure 7.5: Comparative log-log plot of time complexities (along with confidence interval) of `out`, `rdgp` and `ingp` for $\mathcal{CUTSMuHN}$ and existing TSMM platforms: Random tuple scenario.

In the first scenario, the constituent fields of tuples and antituples are generated randomly during the experiment at runtime, i.e. NAME, TYPE, VALUE and SEARCHVALUE are generated randomly at runtime. Once these values are generated, they are used to generate tuples and antituples, and to carry out different tuple space operations. The multiple invokes of `out`, `rdgp` and `ingp` primitives are interleaved to measure their performance overhead in different situations. The performances of the tuple space model of $\mathcal{CUTSMuHN}$ platform and existing TSMM platforms (like LIMONE, EgoSpaces etc.) for this scenario is shown in Figure 7.5. For each case, the average from 10 iterations is taken and plotted against the number of tuples in log-log scale. Figure 7.5 shows two aspects of comparison: (i) the time complexities of `rdgp` and `ingp` primitives of the tuple space model of $\mathcal{CUTSMuHN}$ platform are significantly less compared to the existing TSMM platforms; (ii) the time complexity of `out` primitive of the tuple space model of

$\mathcal{CUTSMuHN}$ platform increases (but, in relatively less order as that of the `rdgp` and `ingp` primitives) compared to the time of `out` primitive in the existing TSMM platforms; this establishes that significant reduction in the time complexities of `rdgp` and `ingp` primitives of $\mathcal{CUTSMuHN}$ platform comes at a price of relatively less order increase in the time complexity of its `out` primitive.

The second scenario of tuple/antituple field values are taken from different tuples and antituples generated while executing *MyChat* application, and other chat applications[1] simultaneously on the same tuple space. Unlike the previous scenario, these field values are factual, and are collected and stored while users are using their respective chat applications. Subsequently, they are used as offline source for generating tuples and antituples in the second experiment scenario. The performance of the tuple space model of $\mathcal{CUTSMuHN}$ platform and existing TSMM platforms for this scenario is shown in Figure 7.6. Like the previous experiment, the average of 10 iterations for each case is taken and plotted against the number of tuples in log-log scale. Figure 7.6 also shows that the time complexities of `rdgp` and `ingp` primitives of $\mathcal{CUTSMuHN}$ platform reduce significantly as compared to the existing TSMM platforms, whereas the time complexity of `out` primitive of $\mathcal{CUTSMuHN}$ platform increases (but with much less order) than that of existing TSMM platforms.

The two preceding scenarios show that overall performance of the tuple space model of $\mathcal{CUTSMuHN}$ platform improves compared to existing TSMM platforms. This is due to the fact that, in a tuple's lifetime, `out` primitive is executed once, while `rdgp` and `ingp` primitives (as well as other tuple-reading/consuming primitives) are executed multiple times till that tuple is withdrawn from tuple space. Consequently, any increase in the time complexity of `out` primitive of $\mathcal{CUTSMuHN}$ platform has very less effect with respect to the manyfold reduction in time complexities of its `rdgp` as well as `ingp` primitives. This improvement is also true for other tuple-reading/consuming primitives. It is noteworthy that the performance of the tuple space model of $\mathcal{CUTSMuHN}$ platform using random tuples has been found to be better than that with applications' tuples, as the cardinality of index tables in the first scenario is found to be less compared to the second scenario.

[1] Other chat applications are the lightweight variants of *MyChat* with different tuple/antituple structure.

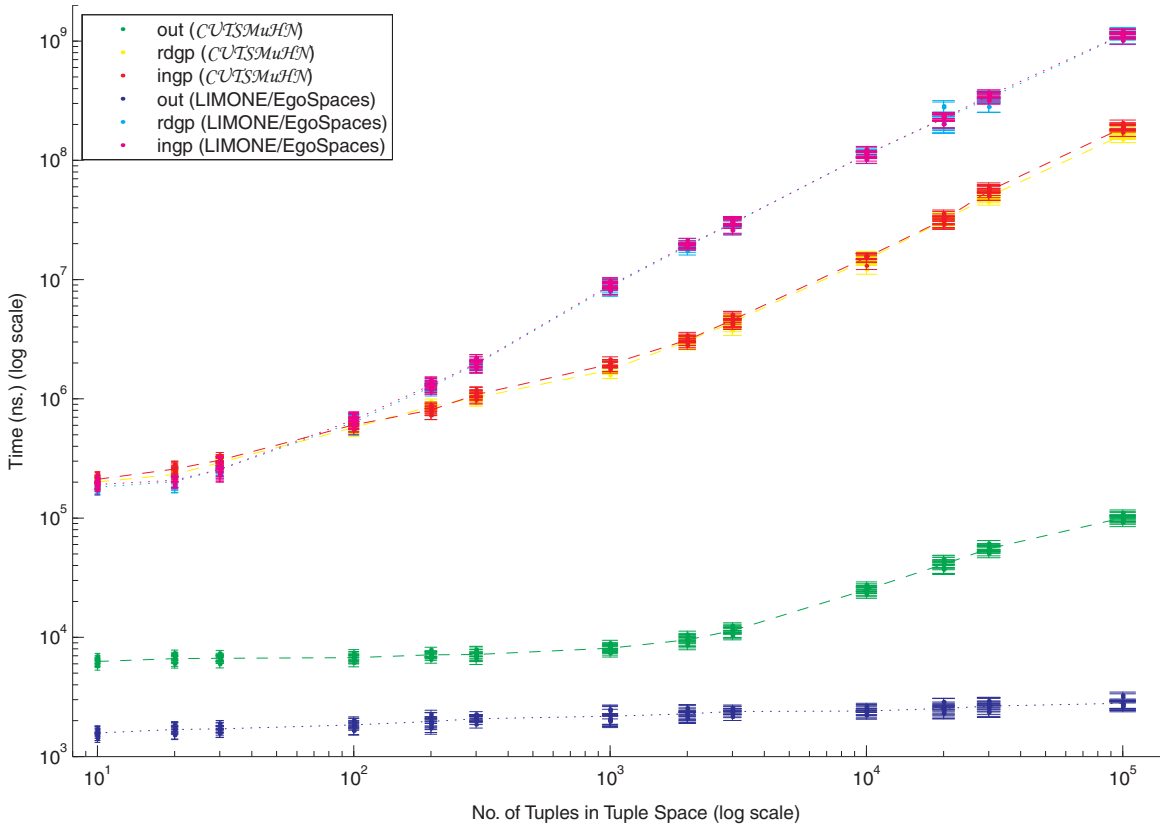


Figure 7.6: Comparative log-log plot of time complexities (along with confidence interval) of `out`, `rdgp` and `ingp` for $\mathcal{CUTSMuHN}$ and existing TSMM platforms: Multiple application tuple scenario.

So, there are less number of tuple-antituple matching operations, which lead to better performance in the former scenario.

Apart from the execution time of the tuple space operations of $\mathcal{CUTSMuHN}$ platform, message overhead and latency during coordination can also be considered as the performance metrics. In Section 3.4.5 of Chapter 3, complexity analysis of message overhead is presented, and a comparison with existing TSMM platforms is tabulated in Table 3.3 in the same chapter. Also, $\mathcal{CUTSMuHN}$ platform is temporally-decoupled and so message latency evaluations are not justifiable. However, message overhead and latency metrics of network layers are not explicitly experimented, as $\mathcal{CUTSMuHN}$ platform is developed at the application layer without modifying network layer behaviors of the underlying networks.

7.5 Conclusion

This chapter has presented a modular approach of implementation of a working prototype of CUTSMuHN platform using Java™, established the relationships between implemented modules through class diagrams, as well as declared the APIs of CUTSMuHN platform for developing mobile applications. The chapter has also shown two mobile applications developed from these APIs, and provided screenshots about the working of CUTSMuHN platform. Moreover, the performances of the tuple space models of CUTSMuHN platform and other existing TSMM platforms have been measured and compared in this chapter. Based on these implementation strategies and experiment results, CUTSMuHN platform can be considered as an efficient TSMM platform for providing ubiquity to the activities of end-users.

Chapter 8

Conclusions and Future Works

8.1 Concluding Remarks

TSMM is a promising genre of mobile middleware platform for supporting host mobility and dynamics amidst heterogeneity in underlying unreliable networks. However, the existing TSMM platforms are not capable enough to support flexible, robust, versatile, simple, reliable, efficient and scalable coordination simultaneously in presence of such environments. This thesis has focused on different research directions of a TSMM platform that directly/indirectly brings such capabilities. In particular, the thesis has identified several limitations of existing TSMM platforms and reported new approaches of handling them in $\mathcal{CUTSMuHN}$ platform.

This thesis has primarily contributed on improving the quality of tuple space coordination of $\mathcal{CUTSMuHN}$ platform. The first contribution of this thesis has proposed a new notion of decoupling for coordination in dynamic environment, which has augmented the existing (partial) synchronization decoupling with an additional dimension of decoupling (viz. time-decoupling of reaction response from reference agent), to achieve 'complete synchronization decoupling' in tuple space coordination. $\mathcal{CUTSMuHN}$ has been designed as a new TSMM platform incorporating the said notion of decoupling in coordination, thereby providing robustness to its mobile applications in dynamic environments. This thesis has also contributed with new consistency handling mechanisms in $\mathcal{CUTSMuHN}$ platform for multi-destined tuple space coordination in dynamic environments. The proposed mechanisms of simultaneously handling both OUT-consistency and IN-consistency

problems have too imparted $\mathcal{CUTSMuHN}$ platform in providing robustness. Moreover, this thesis has also contributed in enhancing the existing unordered tuple/antituple structure (using object-oriented concept and cuckoo hashing scheme), and proposing a new indexed tuple space structure (expediting selection of apposite tuples, thereby reducing look up time in tuple space) for $\mathcal{CUTSMuHN}$ platform, so as to improve its efficiency, without hampering simplicity and versatility of unordered structure. Another contribution of this thesis has proposed new TSMM-specific discovery and communications mechanisms to support tuple space coordination of $\mathcal{CUTSMuHN}$ platform over multiple, heterogeneous networks. The present design of $\mathcal{CUTSMuHN}$ platform has included mechanisms for supporting three distinct underlying networks (viz. iBSS, IBSS and H-MANET). The proposed discovery and communication mechanisms has improved its reliability as well as flexibility of tolerating heterogeneity in underlying networks. The design of $\mathcal{CUTSMuHN}$ platform has also shown its extendibility in supporting other heterogeneous networks. These four thesis contributions have combinedly benefitted tuple space coordination of $\mathcal{CUTSMuHN}$ platform.

The thesis has also contributed in proposing approaches for modeling, verifying and validating the coordination functionality of $\mathcal{CUTSMuHN}$ platform. The vivid and comprehensive approach of modeling an entire TSMM platform using notations of the Mobile UNITY model is the foremost contribution in this regard. The proposed modeling approach has started with modeling of different aspects of tuple space coordination in $\mathcal{CUTSMuHN}$ platform (viz. tuple-antituple matching operation, sought/apposite tuples, tuple space primitives and reactivity), and then has continued to model entire $\mathcal{CUTSMuHN}$ platform as a Mobile UNITY **System**. **System** $CUTSMuHN$, as it has been denoted, has been modeled as a composition of multiple instances of **Program** $host(i)$ and **Program** $agent(k)$, along with their interactions. The proposed modeling approach has presented insightful abstractions about the behaviors of $\mathcal{CUTSMuHN}$ platform, and expressed precise semantics to simplify the tasks of application programmers. The next related contribution of this thesis has proposed the strategy of verifying correctness of $\mathcal{CUTSMuHN}$ platform using its modeling. The proposed verification strategy has defined the formal properties of correctness of $\mathcal{CUTSMuHN}$ platform, and has proven these

properties to verify its correctness, using proof logic of the Mobile UNITY model. The thorough verification strategy has also shown to satisfy decoupling in tuple space coordination of *CUTSMuHN* platform. This thesis has further contributed in presenting a modular approach to implementation of *CUTSMuHN* platform and its mobile applications, which has established its working.

From the contributions of this thesis, it can be concluded that *CUTSMuHN* is a decentralized TSMM platform that single-handedly caters the requirements of end-users over heterogeneous networks.

8.2 Future Works

Even though the coordination functionality, along with multi-dimensional decoupling ability, flexibility, robustness, reliability, efficiency etc., of *CUTSMuHN* platform has been established, its capability can be extended in other aspects also. Such research directions for future developments of *CUTSMuHN* platform are as follows:

1. The present support of *CUTSMuHN* platform spans from iBSS/IBSS to H-MANET. The immediate future work is to extend the support of underlying heterogeneity to other wireless networks, including Low-Rate Wireless Personal Area Network (LR-WPAN) [61] and Wireless Mesh Network (WMN) [4]. LR-WPAN is concerned with "low-cost" and "low-speed" ubiquitous communication of interacting nodes, and allows two distinct nature of nodes as its constituent units, viz. Full-Function Devices (FFD) and Reduced-Function Devices (RFD). On the other hand, WMN combines different wireless technologies (like 802.11, 802.15, 802.16, cellular technology or their combinations), and consequently various types of constituent nodes are available for communication.

2. The next immediate future work is to focus on the lightweight implementation of *CUTSMuHN* platform, where the required methods and library routines are to be loaded dynamically on-the-fly, so that it imposes minimum possible overhead on devices with resource restrictions (in terms of memory and processor capacities).

3. Even though the essence of data coordination is sharing through common medium, such approach is inherently vulnerable to security threats, attacks and other types of malicious activities. *CUTSMuHN* platform, thus, has to be equipped with different mechanisms to handle the security issues, including privacy, access control, and trust policies.

4. The support of context awareness is important in mobile middleware platforms, so that a mobile application can be *aware* of its (execution) context, and can *adapt* accordingly. Context awareness enable the mobile application to render better quality of service to its end-users. Another future research direction for *CUTSMuHN* platform is to incorporate the support of context awareness/adaptation to its mobile applications. In literature, the principle of *reflection* [73] has been used to achieve context awareness/adaptation in different systems, including mobile middleware platforms [19, 23]. The future work in this aspect has to investigate the applicability of reflection in *CUTSMuHN* platform, and adopt a better approach of incorporating context awareness/adaptation in it.

Finally, there are scopes for applying the contributions of this thesis to other paradigms of mobile computing. For instance, the effect of multi-dimensional decoupling in tuple space coordination can also be used to other mobile computing systems, where there is abundance of interactions among the system components, but context is dynamic.

Appendix A

Overview of Mobile UNITY model

A.1 Notations and Proof Logic of UNITY model

The UNITY model [24] is a modular notation system with associated proof logic for expressing the temporal nature of systems or its components. The basic unit of system construction in the UNITY model is a **Program**. Both the concurrent and distributed systems can be specified as a set of **Program**s, composed through their union. Each **Program** consists of ***declare***, ***initially***, ***always*** and ***assign*** sections. The ***declare*** section is used for naming variables and their types. The ***initially*** section is for defining allowed initial conditions of declared variables. The ***always*** section is provided for defining macros and functions of variables used in that **Program**. Lastly, the ***assign*** section is used for specifying assignment statements to define the state transitions of specified system. The notation "[]" acts as a separator for definitions, equations and statements, while the notation "||" is used to coalesce multiple assignments for simultaneous evaluation (i.e. atomic execution). The statements are guarded using constructs like **if**, **when** etc. An expression is denoted by the notation $\langle\, op \;\; var\text{-}list : boolean\text{-}expr :: assertion \rangle$, whose value is obtained by applying *op* to the set of expressions derived by substituting all the instances of variables in *var-list* satisfying *boolean-expr* in the *assertion*. The operators supported for *op* are $\forall$, $\exists$, $\wedge$, $\vee$, $+$, $\times$, **set**, **min**, **max**, **odd**, **even**, **mod** etc.

The quantified single statement in the UNITY model is of form $\langle\| \;\; op \;\; var\text{-}list : boolean\text{-}expr :: s \rangle$, where s represents a single statement. When a **Program** is being executed, the ***initially*** section specifies its initial state, while single or multiple (i.e.

atomic) assignments are executed in each step. The **Program** reaches a fixed point, when its state is not changing any more. The statements in the UNITY model are allowed to get executed in a non-deterministic (but, weakly fair) order. The proof logic of the UNITY model is based on the Hoare triple of form $\{p\}s\{q\}$ [56], where p and q are the predicates and s denotes any statement in the **Program**. The expression $\{p\}s\{q\}$ signifies that execution of s in a state satisfying p results in a state satisfying q, if execution of s terminates. The state of a **Program** at a particular step is the value of all variables of the **Program**. Three fundamental logic relations of the UNITY model, viz. **unless**, **ensures** and **leads-to**, and their special cases, viz. **stable** and **invariant**, are used for verifying the UNITY representations.

A.2 Notations and Proof Logic of Mobile UNITY model

The Mobile UNITY model extends the UNITY model for supporting specifications and reasoning of mobile computing systems. It follows the same basic unit of system construction, viz. **Program**. However, with the Mobile UNITY model, concurrent, distributed and mobile computing systems can be specified as a set of **Program**s under one **System**. The structure of a **Program**, followed by the UNITY model, are preserved in the Mobile UNITY model, i.e. each **Program** of a **System** also consists of ***declare***, ***initially***, ***always*** and ***assign*** sections. The notations of ***declare***, ***initially*** and ***always*** sections are also preserved in the Mobile UNITY model. However, the ***assign*** section is extended with new notations and constructs. Also some new sections, like **Components** and **Interactions**, are added as parts of the **System** for specifying its components and for handling transient behaviour among those components respectively. The construct **reacts-to** is included to express the conditions of reactivity by reactive statements, while **inhibit** construct is added to include a label to that statement which is not required to be performed unless the specified condition is satisfied. Unlike the UNITY model, the Mobile UNITY model follows a different naming convention for program variables, viz. ProgramName.VariableName. The pattern of execution in this model is also different. As the **System** starts executing, the ***initially*** section of each **Program** specifies its initial

state, while single or (atomic) multiple assignment statements inside its ***assign*** section are executed in each step.

The proof logic of the Mobile UNITY model is also based on Hoare triple $\{p\}s\{q\}$ [56], where p and q are predicates and s denotes any statement in any Mobile UNITY **System** (say, *model*). However, the proof rules accommodate its own execution model, where each nonreactive statement of *model* is fairly selected for execution, executed if not inhibited, and then the set of all reactive statements of *model* (denoted as $model^{\Re}$) is allowed to execute until it reaches the fixed point predicate of $model^{\Re}$, denoted as $FP(model^{\Re})$ (which is the set of states that are fixed points for all reactive statements of $model^{\Re}$), after which the next nonreactive statement of *model* is scheduled. Each reactively augmented and possibly inhibited statement is denoted by s^*, for each nonreactive statement s. The set of all nonreactive statements of *model* is denoted by $model^{\aleph}$. The definitions of safety (i.e **co**) and progress (i.e. **transient**) properties are as follows:

$$p \textbf{ co } q \triangleq \langle \forall s : s \in model^{\aleph} :: \{p\}s^*\{q\}\rangle \wedge (p \Rightarrow q) \tag{A.1}$$

$$\textbf{transient } p \triangleq \langle \exists s : s \in model^{\aleph} :: \{p\}s^*\{\neg p\}\rangle \tag{A.2}$$

where, $\{p\}s^*\{q\}$ is defined by the following rule (written in hypothesis-conclusion form) for each s, where s is considered as a nonreactive singleton statement of *model*:

$$\frac{p \wedge \iota(s) \Rightarrow q,\ \{p \wedge \neg\iota(s)\}s\{H\},\ (H \mapsto FP(model^{\Re}) \wedge q) \text{ in } model^{\Re}}{\{p\}s^*\{q\}} \tag{A.3}$$

where, $\iota(s)$ is the disjunction of all **when** predicates of **inhibit** clauses in s and H is considered as the strongest postcondition of $(p \wedge \neg\iota(s))$ with respect to s. The first part of hypothesis states that if s is inhibited in a state satisfying p, then q must be true of that state also. The significance of "in $model^{\Re}$" is that the proof of termination is to be carried out from the text of all reactive statements, ignoring other statements. The inference rule in Equation (A.3) can be simplified to another inference rule (written as

follows in hypothesis-conclusion form), which holds for any nonreactive singleton statement s that is not reactively-augmented:

$$\frac{p \wedge \iota(s) \Rightarrow q,\ \{p \wedge \neg\iota(s)\}s\{q\},\ q \Rightarrow FP(\Re)}{\{p\}s^*\{q\}} \tag{A.4}$$

The inference rule in Equation (A.5) holds for any nonsingleton transaction, defined by sequential composition, where any s_i can be either reactively augmented or not as well as singleton or another transaction.

$$\frac{\{p\}\langle s_1; s_2; \ldots; s_{n-1}\rangle^*\{p'\},\ \{p'\}{s_n}^*\{q\}}{\{p\}\langle s_1; s_2; \ldots; s_{n-1}; s_n\rangle^*\{q\}} \tag{A.5}$$

In case, no **inhibit** clauses are used in *model*, i.e. $\langle \forall s : s \in model :: \iota(s) = \mathsf{FALSE}\rangle$ holds, the inference rules in Equation (A.3) becomes simplified:

$$\frac{\{p\}s\{H\},\ (H \mapsto FP(\Re) \wedge q) \text{ in } \Re}{\{p\}s^*\{q\}} \tag{A.6}$$

The other properties of the Mobile UNITY model are **stable**, **invariant**, **ensures** and **leads-to** (denoted as "$\mapsto$"). The definitions of **stable** and **invariant** properties are:

$$\textbf{stable } p \triangleq p \textbf{ co } p \tag{A.7}$$

$$\textbf{invariant } p \triangleq (\text{initial conditions} \Rightarrow p) \wedge \textbf{ stable } p \tag{A.8}$$

The **ensures** relation is effectively a conjunction of safety and progress properties:

$$p \textbf{ ensures } q \triangleq ((p \wedge \neg q) \textbf{ co } (p \vee q)) \wedge \textbf{transient } (p \wedge \neg q) \tag{A.9}$$

This relation expresses the fact that if a program is in a state satisfying p, it remains in that state unless q is established, and, in addition, it does not remain forever in a state satisfying p but not q. Though **ensures** can express simple progress properties, proofs of complicated progress properties require the use of **leads-to** relation, which is defined inductively as follows, where $\mathcal{P}$ is any set of predicates.

$$\text{(basis)} \qquad \frac{p \textbf{ ensures } q}{p \ \mapsto\ q} \tag{A.10}$$

$$\text{(transitivity)} \qquad \frac{p \;\mapsto\; q,\; q \;\mapsto\; r}{p \;\mapsto\; r} \tag{A.11}$$

$$\text{(disjunction)} \qquad \frac{\langle \forall p : p \in \mathcal{P} :: p \;\mapsto\; q \rangle}{\langle \exists p : p \in \mathcal{P} :: p \rangle \;\mapsto\; q} \tag{A.12}$$

The theorems of **leads-to** that are used in this thesis for formal verification are as follows. Details of these theorems can be found in [24].

$$\text{(Implication)} \qquad \frac{p \;\Rightarrow\; q}{p \;\mapsto\; q} \tag{A.13}$$

$$\text{(Cancellation)} \qquad \frac{p \;\mapsto\; q \vee b,\; b \;\mapsto\; r}{p \;\mapsto\; q \vee r} \tag{A.14}$$

$$\text{(Progress-Safety-Progress (PSP))} \qquad \frac{p \;\mapsto\; q,\; r \textbf{ unless } \mathbf{b}}{p \wedge r \;\mapsto\; (q \wedge r) \vee b} \tag{A.15}$$

$$\text{(Corollary of PSP)} \qquad \frac{p \;\mapsto\; q,\; \textbf{stable } \mathbf{r}}{p \wedge r \;\mapsto\; q \wedge r} \tag{A.16}$$

$$\text{(Finite disjunction)} \qquad \frac{p \;\mapsto\; q,\; p' \;\mapsto\; q'}{p \vee p' \;\mapsto\; q \vee q'} \tag{A.17}$$

The transient sharing concept of the Mobile UNITY model, denoted by "$\approx$" operator, is defined in terms of reactive statements (which includes **reacts-to** clauses) [82, p. 106]:

$$A.x \approx B.y \;\; \textbf{when } p \quad \equiv \quad A.x \rightarrow B.y \;\; \textbf{when } p$$
$$B.y \rightarrow A.x \;\; \textbf{when } p$$

$$A.x \rightarrow B.y \;\; \textbf{when } p \;\equiv B.y, B.y_{A.x}, A.x_{B.y} := A.x, A.x, A.x \quad \textbf{reacts-to } A.x \neq A.x \wedge p$$
$$A.x_{B.y} := A.x \quad \textbf{reacts-to } \neg p$$

$$B.y \rightarrow A.x \;\; \textbf{when } p \;\equiv A.x, A.x_{B.y}, B.y_{A.x} := B.y, B.y, B.y \quad \textbf{reacts-to } B.y \neq B.y \wedge p$$
$$B.y_{A.x} := B.y \quad \textbf{reacts-to } \neg p$$

$$\textbf{engage}(A.x, B.y) \;\; \textbf{when } p \;\; \textbf{value } e \;\equiv$$
$$A.x, B.y,\; status_{A.x,B.y} := e, e, \mathsf{TRUE} \quad \textbf{reacts-to } \neg status_{A.x,B.y} \wedge p$$

$$\textbf{disengage}(A.x, B.y) \;\; \textbf{when } p \;\; \textbf{value } d_1, d_2 \;\equiv$$
$$A.x, B.y,\; status_{A.x,B.y} := d_1, d_2, \mathsf{FALSE} \quad \textbf{reacts-to } status_{A.x,B.y} \wedge \neg p$$

Appendix B

Proof Mechanisms in Mobile UNITY model

The mechanism to prove individual assertions in **System** *CUTSMuHN* follows the proof logic of the Mobile UNITY model mentioned in Section A.2 of Appendix A.

B.1 Fixed Point in System *CUTSMuHN*

The fixed point predicate of reactive program $\Re$ in *CUTSMuHN*, $FP(CUTSMuHN^{\Re})$, is defined by first identifying the reactive statements of **System** *CUTSMuHN* that are part of $CUTSMuHN^{\Re}$. These statements include (i) reactions (that are part of blocking remote primitives having **reacts-to** clauses) of $agent(k)$, (ii) two statements (that are part of Transport Interface having **reacts-to** clauses) of $host(i)$, and (iii) statements in the **Interactions** section (that use "$\approx$" operator, which includes **reacts-to** clauses). These reactive statements, labeled as $shared_{\text{WiBSS}}$, $shared_{\text{WL}_{\text{iBSS}}}$, $shared_{\text{WL}_{\text{IBSS}}}$ and $shared_{\text{WL}_{\text{HMANET}}}$ in Figure 5.2, are elaborated next before defining $FP(CUTSMuHN^{\Re})$. They are redefined as follows, according to the definition given in [82, p. 106]. For instance, the statement labeled $shared_{\text{WiBSS}}$ is redefined as:

$$
\begin{aligned}
shared_{\text{WiBSS}} :: \Big\langle \; [] \; i, j :: host(i).\mathcal{T}_{\text{W}} \approx host(j).\mathcal{T}_{\text{W}} \\
\textbf{when} \; \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
\wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big) \\
\textbf{engage} \;\; host(i).\mathcal{T}_{\text{W}} \qquad \textbf{disengage} \;\; \textbf{current} \parallel \bot \;\; \Big\rangle
\end{aligned}
$$

$$\equiv \Big\langle [\!]\ i,j :: host(j).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}}, host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}} := host(i).\mathcal{T}_{\mathrm{W}}, host(i).\mathcal{T}_{\mathrm{W}}, host(i).\mathcal{T}_{\mathrm{W}}$$
$$\textbf{reacts-to}\ \neg\big(host(i).\mathcal{T}_{\mathrm{W}} = host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big)$$

$$host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}} := host(i).\mathcal{T}_{\mathrm{W}}$$
$$\textbf{reacts-to}\ \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big)$$

$$host(i).\mathcal{T}_{\mathrm{W}}, host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}}, host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}} := host(j).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}}$$
$$\textbf{reacts-to}\ \neg\big(host(j).\mathcal{T}_{\mathrm{W}} = host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big)$$

$$host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}} := host(j).\mathcal{T}_{\mathrm{W}}$$
$$\textbf{reacts-to}\ \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big)$$

$$host(i).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}}, status_{host(i).\mathcal{T}_{\mathrm{W}},host(j).\mathcal{T}_{\mathrm{W}}} := host(i).\mathcal{T}_{\mathrm{W}}, host(i).\mathcal{T}_{\mathrm{W}}, \mathsf{TRUE}$$
$$\textbf{reacts-to}\ \neg status_{host(i).\mathcal{T}_{\mathrm{W}},host(j).\mathcal{T}_{\mathrm{W}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big)$$

$$host(i).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}}, status_{host(i).\mathcal{T}_{\mathrm{W}},host(j).\mathcal{T}_{\mathrm{W}}} := host(i).\mathcal{T}_{\mathrm{W}}, \perp, \mathsf{FALSE}$$
$$\textbf{reacts-to}\ status_{host(i).\mathcal{T}_{\mathrm{W}},host(j).\mathcal{T}_{\mathrm{W}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \big(\mathtt{isSH}(host(i)) \vee \mathtt{isAP}(host(i))\big) \wedge \big(\mathtt{isSH}(host(j)) \vee \mathtt{isAP}(host(j))\big)\Big)$$

$$\Big\rangle \tag{B.1}$$

In the above definition, the statement $shared_{\mathrm{W_{iBSS}}}$ has been split into six reactive sub-statements in Equation (B.1), each of which contributes in defining $FP(CUTSMuHN^{\Re})$. Similarly, the statement labeled $shared_{\mathrm{WL_{iBSS}}}$ is redefined as:

$$shared_{\mathrm{WL_{iBSS}}} :: \Big\langle [\!]\ i,j :: host(i).\mathcal{T}_{\mathrm{WL}} \approx host(j).\mathcal{T}_{\mathrm{WL}}$$
$$\textbf{when}\ \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big)$$
$$\textbf{engage}\ \ host(i).\mathcal{T}_{\mathrm{WL}} \qquad \textbf{disengage}\ \ \textbf{current} \parallel \perp \ \Big\rangle$$

$$
\begin{aligned}
\equiv \Big\langle [\!]\ i,j :: \ & host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}}, host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} := host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}} \\
& \quad \textbf{reacts-to}\ \neg\big(host(i).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} := host(i).\mathcal{T}_{\text{WL}} \\
& \quad \textbf{reacts-to}\ \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}}, host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} := host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}} \\
& \quad \textbf{reacts-to}\ \neg\big(host(j).\mathcal{T}_{\text{WL}} = host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
& host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} := host(j).\mathcal{T}_{\text{WL}} \\
& \quad \textbf{reacts-to}\ \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} := host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}}, \mathsf{TRUE} \\
& \quad \textbf{reacts-to}\ \neg status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} := host(i).\mathcal{T}_{\text{WL}}, \bot, \mathsf{FALSE} \\
& \quad \textbf{reacts-to}\ status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
\Big\rangle &
\end{aligned}
\tag{B.2}
$$

The above definition in Equation (B.2) also results in six reactive substatements, which is going to be used while defining $FP(CUTSMuHN^{\Re})$. In the same way, the statements labeled $shared_{\text{WL}_{\text{IBSS}}}$ and $shared_{\text{WL}_{\text{HMANET}}}$ are redefined as follows:

$$
\begin{aligned}
shared_{\text{WL}_{\text{IBSS}}} :: \Big\langle [\!]\ i,j :: \ & host(i).\mathcal{T}_{\text{WL}} \approx host(j).\mathcal{T}_{\text{WL}} \\
& \textbf{when}\ \Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big) \\
& \textbf{engage}\ \ host(i).\mathcal{T}_{\text{WL}} \qquad \textbf{disengage}\ \ \textbf{current} \parallel \bot \quad \Big\rangle
\end{aligned}
$$

$$
\begin{aligned}
\equiv \Big\langle \; [] \; i,j :: & \; host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}}, host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} := host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}} \\
& \textbf{reacts-to } \neg\big(host(i).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} := host(i).\mathcal{T}_{\text{WL}} \\
& \textbf{reacts-to } \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}}, host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} := host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}} \\
& \textbf{reacts-to } \neg\big(host(j).\mathcal{T}_{\text{WL}} = host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big) \\
& host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} := host(j).\mathcal{T}_{\text{WL}} \\
& \textbf{reacts-to } \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} := host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}}, \mathsf{TRUE} \\
& \textbf{reacts-to } \neg status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big) \\
& host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} := host(i).\mathcal{T}_{\text{WL}}, \bot, \mathsf{FALSE} \\
& \textbf{reacts-to } status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(i)\Gamma' host(j)\big) \\
& \qquad \wedge \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big) \\
\Big\rangle & \qquad \text{(B.3)}
\end{aligned}
$$

$$
\begin{aligned}
shared_{\text{WLHMANET}} :: \Big\langle \; [] \; i,j :: & \; host(i).\mathcal{T}_{\text{WL}} \approx host(j).\mathcal{T}_{\text{WL}} \\
& \textbf{when } \Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \\
& \qquad \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \\
& \qquad\quad \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big)\Big) \\
& \textbf{engage } \; host(i).\mathcal{T}_{\text{WL}} \qquad \textbf{disengage } \; \textbf{current} \parallel \bot \quad \Big\rangle
\end{aligned}
$$

$$
\begin{aligned}
\equiv \Big\langle \; [] \; i,j :: & \; host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}}, host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} := host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}} \\
& \textbf{reacts-to } \neg\big(host(i).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \\
& \qquad \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(i)\Gamma' host(j)\big)
\end{aligned}
$$

$$\wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \Big) \Big)$$

$host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} := host(i).\mathcal{T}_{\text{WL}}$
reacts-to $\neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \Big) \Big)$

$host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}}, host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} := host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}$
reacts-to $\neg\big(host(j).\mathcal{T}_{\text{WL}} = host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}}\big) \wedge \Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \Big) \Big)$

$host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} := host(j).\mathcal{T}_{\text{WL}}$
reacts-to $\neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \Big) \Big)$

$host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} := host(i).\mathcal{T}_{\text{WL}}, host(i).\mathcal{T}_{\text{WL}}, \mathsf{TRUE}$
reacts-to $\neg status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \Big) \Big)$

$host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}, status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} := host(i).\mathcal{T}_{\text{WL}}, \bot, \mathsf{FALSE}$
reacts-to $status_{host(i).\mathcal{T}_{\text{WL}}, host(j).\mathcal{T}_{\text{WL}}} \wedge \Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(i)\Gamma' host(j)\big) \wedge \Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isGH}(host(j))\big) \vee \big(\texttt{isGH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \vee \big(\texttt{isMH}(host(i)) \wedge \texttt{isMH}(host(j))\big) \Big) \Big)$

$\Big\rangle$ (B.4)

Based on the reactive interaction statements redefined in Equation (B.1), Equation (B.2), Equation (B.3) and Equation (B.4) as well as the reactive statements in tuple space primitives, $FP(CUTSMuHN^{\Re})$ predicate is defined as follows:

$FP(CUTSMuHN^{\Re}) \equiv \Big($

$$\Big(\big(host(i).\mathcal{T}_{\mathrm{W}} = \perp\big) \vee \neg host(i).\texttt{isNotOwnMsg}(host(i).\mathcal{T}_{\mathrm{W}}) \vee \big(\texttt{last}(host(i).\mathcal{Q}_R) = host(i).\mathcal{T}_{\mathrm{W}}\big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\mathrm{WL}} = \perp\big) \vee \neg host(i).\texttt{isNotOwnMsg}(host(i).\mathcal{T}_{\mathrm{WL}}) \vee \big(\texttt{last}(host(i).\mathcal{Q}_R) = host(i).\mathcal{T}_{\mathrm{WL}}\big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{W}} = host(i).\mathcal{T}_{\mathrm{W}}\big) \vee \big(host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}} = host(i).\mathcal{T}_{\mathrm{W}}\big) \vee \big(host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}} = host(i).\mathcal{T}_{\mathrm{W}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}} = host(i).\mathcal{T}_{\mathrm{W}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big)$$
$$\vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\Big(\big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\mathrm{W}} = host(j).\mathcal{T}_{\mathrm{W}}\big) \vee \big(host(i).\mathcal{T}_{\mathrm{W}_{host(j).\mathcal{T}_{\mathrm{W}}}} = host(j).\mathcal{T}_{\mathrm{W}}\big) \vee \big(host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}} = host(j).\mathcal{T}_{\mathrm{W}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{W}_{host(i).\mathcal{T}_{\mathrm{W}}}} = host(j).\mathcal{T}_{\mathrm{W}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big)$$
$$\vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\Big(\big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{W}} = host(i).\mathcal{T}_{\mathrm{W}}\big) \vee \big(status_{host(i).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}}} = \mathsf{TRUE}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{W}} = \perp\big) \vee \neg\big(status_{host(i).\mathcal{T}_{\mathrm{W}}, host(j).\mathcal{T}_{\mathrm{W}}} = \mathsf{TRUE}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isSH}(host(i)) \vee \texttt{isAP}(host(i))\big) \wedge \big(\texttt{isSH}(host(j)) \vee \texttt{isAP}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{WL}} = host(i).\mathcal{T}_{\mathrm{WL}}\big) \vee \big(host(j).\mathcal{T}_{\mathrm{WL}_{host(i).\mathcal{T}_{\mathrm{WL}}}} = host(i).\mathcal{T}_{\mathrm{WL}}\big) \vee \big(host(i).\mathcal{T}_{\mathrm{WL}_{host(j).\mathcal{T}_{\mathrm{WL}}}} = host(i).\mathcal{T}_{\mathrm{WL}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isAP}(host(j))\big) \vee \big(\texttt{isAP}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\mathrm{WL}_{host(j).\mathcal{T}_{\mathrm{WL}}}} = host(i).\mathcal{T}_{\mathrm{WL}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big)$$
$$\vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isAP}(host(j))\big) \vee \big(\texttt{isAP}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{WL}} = host(i).\mathcal{T}_{\mathrm{WL}}\big) \vee \big(host(j).\mathcal{T}_{\mathrm{WL}_{host(i).\mathcal{T}_{\mathrm{WL}}}} = host(i).\mathcal{T}_{\mathrm{WL}}\big) \vee \big(host(i).\mathcal{T}_{\mathrm{WL}_{host(j).\mathcal{T}_{\mathrm{WL}}}} = host(i).\mathcal{T}_{\mathrm{WL}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isAP}(host(j))\big) \vee \big(\texttt{isAP}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{WL}_{host(i).\mathcal{T}_{\mathrm{WL}}}} = host(j).\mathcal{T}_{\mathrm{WL}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big)$$
$$\vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isAP}(host(j))\big) \vee \big(\texttt{isAP}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\mathrm{WL}} = host(i).\mathcal{T}_{\mathrm{WL}}\big) \vee \big(status_{host(i).\mathcal{T}_{\mathrm{WL}}, host(j).\mathcal{T}_{\mathrm{WL}}} = \mathsf{TRUE}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\texttt{isMH}(host(i)) \wedge \texttt{isAP}(host(j))\big) \vee \big(\texttt{isAP}(host(i)) \wedge \texttt{isMH}(host(j))\big)\Big)\Big)$$

$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = \bot\big) \vee \neg\big(status_{host(i).\mathcal{T}_{\text{WL}},host(j).\mathcal{T}_{\text{WL}}} = \mathsf{TRUE}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{iBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{iBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isAP}(host(j))\big) \vee \big(\mathtt{isAP}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} = host(i).\mathcal{T}_{\text{WL}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big)\Big)$$
$$\vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} = host(j).\mathcal{T}_{\text{WL}}\big) \vee \big(host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} = host(j).\mathcal{T}_{\text{WL}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} = host(j).\mathcal{T}_{\text{WL}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big)\Big)$$
$$\vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(status_{host(i).\mathcal{T}_{\text{WL}},host(j).\mathcal{T}_{\text{WL}}} = \mathsf{TRUE}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = \bot\big) \vee \neg\big(status_{host(i).\mathcal{T}_{\text{WL}},host(j).\mathcal{T}_{\text{WL}}} = \mathsf{TRUE}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{IBSS}\big) \wedge \big(host(j).nwdeploy = \mathsf{IBSS}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} = host(i).\mathcal{T}_{\text{WL}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isGH}(host(j))\big) \vee \big(\mathtt{isGH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)$$
$$\vee \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big)$$
$$\wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isGH}(host(j))\big)$$
$$\vee \big(\mathtt{isGH}(host(i)) \wedge \mathtt{isMH}(host(j))\big) \vee \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(i).\mathcal{T}_{\text{WL}} = host(j).\mathcal{T}_{\text{WL}}\big) \vee \big(host(i).\mathcal{T}_{\text{WL}_{host(j).\mathcal{T}_{\text{WL}}}} = host(j).\mathcal{T}_{\text{WL}}\big) \vee \big(host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} = host(j).\mathcal{T}_{\text{WL}}\big)$$
$$\vee \neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big)$$
$$\vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isGH}(host(j))\big) \vee \big(\mathtt{isGH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)$$
$$\vee \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big)$$
$$\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}_{host(i).\mathcal{T}_{\text{WL}}}} = host(j).\mathcal{T}_{\text{WL}}\big) \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big)$$
$$\wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big) \vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isGH}(host(j))\big)$$
$$\vee \big(\mathtt{isGH}(host(i)) \wedge \mathtt{isMH}(host(j))\big) \vee \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big)$$

$$
\begin{array}{l}
\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = host(i).\mathcal{T}_{\text{WL}}\big) \vee \big(status_{host(i).\mathcal{T}_{\text{WL}},host(j).\mathcal{T}_{\text{WL}}} = \mathsf{TRUE}\big) \\
\quad \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big) \\
\quad \vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isGH}(host(j))\big) \vee \big(\mathtt{isGH}(host(i)) \wedge \mathtt{isMH}(host(j))\big) \\
\quad \vee \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
\wedge \Big(\big(host(j).\mathcal{T}_{\text{WL}} = \perp\big) \vee \neg\big(status_{host(i).\mathcal{T}_{\text{WL}},host(j).\mathcal{T}_{\text{WL}}} = \mathsf{TRUE}\big) \\
\quad \vee \neg\Big(\big(host(i).nwdeploy = \mathsf{HMANET}\big) \wedge \big(host(j).nwdeploy = \mathsf{HMANET}\big)\Big) \vee \neg\big(host(i)\Gamma' host(j)\big) \\
\quad \vee \neg\Big(\big(\mathtt{isMH}(host(i)) \wedge \mathtt{isGH}(host(j))\big) \vee \big(\mathtt{isGH}(host(i)) \wedge \mathtt{isMH}(host(j))\big) \\
\quad \vee \big(\mathtt{isMH}(host(i)) \wedge \mathtt{isMH}(host(j))\big)\Big)\Big) \\
\wedge \Big\langle \forall t : \neg\big((t \in \mathbf{T}) \wedge \mathfrak{M}(t,a) \wedge \neg(t = \varepsilon)\big) \vee \neg\big(enabled_{\mathsf{R}_{rd}} = \mathsf{TRUE}\big) \vee \big(reacted_{\mathsf{R}_{rd}} = reacted_{\mathsf{R}_{rd}} \cup \{t \cdot \mathsf{id}\}\big) \\
\quad \vee \neg\Big(\big((\mathsf{m}_{\mathsf{R}_{rd}} = \mathtt{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{rd}} = \emptyset)\big) \wedge \big((\mathsf{m}_{\mathsf{R}_{rd}} = \mathtt{ONCE/TUPLE}) \Rightarrow (t \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{rd}})\big)\Big)\Big\rangle \\
\wedge \Big\langle \forall t : \neg\big((t \in \mathbf{T}) \wedge \mathfrak{M}(t,a) \wedge \neg(t = \varepsilon)\big) \vee \neg\big(enabled_{\mathsf{R}_{rdg}} = \mathsf{TRUE}\big) \vee \big(reacted_{\mathsf{R}_{rdg}} = reacted_{\mathsf{R}_{rdg}} \cup \{t \cdot \mathsf{id}\}\big) \\
\quad \vee \neg\Big(\big((\mathsf{m}_{\mathsf{R}_{rdg}} = \mathtt{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{rdg}} = \emptyset)\big) \wedge \big((\mathsf{m}_{\mathsf{R}_{rdg}} = \mathtt{ONCE/TUPLE}) \Rightarrow (t \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{rdg}})\big)\Big)\Big\rangle \\
\wedge \Big\langle \forall t : \neg\big((t \in \mathbf{T}) \wedge \mathfrak{M}(t,a) \wedge \neg(t = \varepsilon)\big) \vee \neg\big(enabled_{\mathsf{R}_{in}} = \mathsf{TRUE}\big) \\
\quad \vee \Big(\big(\mathbf{T} = \mathbf{T} \setminus \{t\}\big) \wedge \big(reacted_{\mathsf{R}_{in}} = reacted_{\mathsf{R}_{in}} \cup \{t \cdot \mathsf{id}\}\big)\Big) \\
\quad \vee \neg\Big(\big((\mathsf{m}_{\mathsf{R}_{in}} = \mathtt{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{in}} = \emptyset)\big) \wedge \big((\mathsf{m}_{\mathsf{R}_{in}} = \mathtt{ONCE/TUPLE}) \Rightarrow (t \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{in}})\big)\Big)\Big\rangle \\
\wedge \Big\langle \forall t : \neg\big((t \in \mathbf{T}) \wedge \mathfrak{M}(t,a) \wedge \neg(t = \varepsilon)\big) \vee \neg\big(enabled_{\mathsf{R}_{ing}} = \mathsf{TRUE}\big) \\
\quad \vee \Big(\big(\mathbf{T} = \mathbf{T} \setminus \{t\}\big) \wedge \big(reacted_{\mathsf{R}_{ing}} = reacted_{\mathsf{R}_{ing}} \cup \{t \cdot \mathsf{id}\}\big)\Big) \\
\quad \vee \neg\Big(\big((\mathsf{m}_{\mathsf{R}_{ing}} = \mathtt{ONCE}) \Rightarrow (reacted_{\mathsf{R}_{ing}} = \emptyset)\big) \wedge \big((\mathsf{m}_{\mathsf{R}_{ing}} = \mathtt{ONCE/TUPLE}) \Rightarrow (t \cdot \mathsf{id} \notin reacted_{\mathsf{R}_{ing}})\big)\Big)\Big\rangle \\
\Big)
\end{array}
\tag{B.5}
$$

The construction of $FP(CUTSMuHN^{\Re})$ predicate in Equation (B.5) follows the approach of defining fixed points in [24, p. 53]. In this definition, conjuncts of $FP(CUTSMuHN^{\Re})$ comes from the reactive statements in $CUTSMuHN$. The predicate $FP(CUTSMuHN^{\Re})$ is used in the proofs of individual assertions; one instance of such proof is shown in the next section.

B.1.1 Proof of assertion (6.17)

Lemma B.1 $\big(\mathit{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big(\mathit{last}(host(i).\mathcal{Q}_{S_{\text{W}}}) = M\big) \veebar \big(\mathit{last}(host(i).\mathcal{Q}_{S_{\text{WL}}}) = M\big)$.

Proof: For proving the above assertion, used in the proof of Lemma 6.3 in Chapter 6, it is required to show that in the source host (say, $host(i)$), a message M from its $\mathcal{Q}_S$ will eventually reach either its queues $\mathcal{Q}_{S_{\text{W}}}$ or $\mathcal{Q}_{S_{\text{WL}}}$, but not both of them at the same

time. That is:

$$
\begin{aligned}
& \big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \textbf{ ensures } \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \trianglerighteq \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big) \\
\equiv\ & \hat{\Delta} \textbf{ ensures } (\alpha \trianglerighteq \alpha') \qquad (B.6) \\
\equiv\ & \big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha') \textbf{ co } \hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big) \wedge \textbf{transient}\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big) \qquad \text{, using Equation (A.9)} \\
\equiv\ & \Big\langle \forall S : \big(S \in \mathit{CUTSMuHN}^{\aleph}\big) :: \big\{\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big\}\ S^*\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\} \Big\rangle \\
& \wedge \Big(\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big) \Rightarrow \big(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big)\Big) \\
& \wedge \Big\langle \exists S' : \big(S' \in \mathit{CUTSMuHN}^{\aleph}\big) :: \big\{\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big\}\ S'^*\ \big\{\neg\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big)\big\} \Big\rangle \\
& \qquad \text{, using Equation (A.1) and Equation (A.2)}
\end{aligned}
$$

where, $\hat{\Delta} \equiv \big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big)$, $\alpha \equiv \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big)$, and $\alpha' \equiv \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big)$. Among the three conjuncts in the last expression above, the conjunct $\Big(\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big) \Rightarrow \big(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big)\Big)$ is first shown to hold as follows:

(1)	$\hat{\Delta}$	, premise
(2)	$\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big)$	, hypothesis
(3)	$\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big) \Rightarrow \hat{\Delta}$	, from program text , also after applying conjunction elimination inference rule on *(2)*
(4)	$\hat{\Delta} \Rightarrow \big(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big)$	, after applying disjunctive addition inference rule on *(1)*
(5)	$\big(\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big) \Rightarrow \big(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big)$	(B.7) , after applying hypothetical syllogism inference rule on *(3)* and *(4)*

Next, the conjunct $\Big\langle \forall S : \big(S \in \mathit{CUTSMuHN}^{\aleph}\big) :: \big\{\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big\}\ S^*\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\} \Big\rangle$ is shown to hold:

(1)	$\{\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\}\ S\ \{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\}$	, premise
(2)	$\hat{\Delta} \Rightarrow \neg(\alpha \trianglerighteq \alpha')$	, from program text
(3)	$\big(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big) \Rightarrow \big(\neg(\alpha \trianglerighteq \alpha') \vee (\alpha \trianglerighteq \alpha')\big) \equiv \mathsf{TRUE}$	, using *(2)*
(4)	$\{\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\}\ S\ \{\mathsf{TRUE}\}$	, replacing postcondition of *(1)* by *(3)*; , *(4)* holds for all S of $\mathit{CUTSMuHN}^{\aleph}$ according to basic axiom; so, *(1)* holds
(5)	$\Big\langle \forall S'' : \big(S'' \in \mathit{CUTSMuHN}^{\Re}\big) :: \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg\big(\mathit{FP}(\mathit{CUTSMuHN}^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big)\big\}$ $S''\ \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \vee \big(\mathit{FP}(\mathit{CUTSMuHN}^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big)\big\} \Big\rangle$	, premise

(6) $\Big\langle \forall S'' : \big(S'' \in CUTSMuHN^{\Re}\big) :: \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\big\}\ S''\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\}\Big\rangle$
, simplifying (5)

(7) $\big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\}\ S''\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\}$
, holds from program text, as no S'' has any effect on $(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))$

(8) $\big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\big\}\ S''\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\}$
, after applying consequence rule on (7) for precondition strengthening

(9) $\Big\langle \forall S'' : \big(S'' \in CUTSMuHN^{\Re}\big) :: \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\big\}\ S''\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\}\Big\rangle$
, quantifying (8); thus (5) holds

(10) $\Big\langle \exists S''' : \big(S''' \in CUTSMuHN^{\Re}\big) :: \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg\big(FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big)\big\}$
$S'''\ \big\{FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big\}\Big\rangle$, premise

(11) $\Big\langle \exists S''' : \big(S''' \in CUTSMuHN^{\Re}\big) :: \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\big\}$
$S'''\ \big\{FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big\}\Big\rangle$, simplifying (10)

(12) $\big\{\neg FP(CUTSMuHN^{\Re})\big\}\ S'''\ \big\{FP(CUTSMuHN^{\Re})\big\}$, holds from program text
, as such an S''' exists in $CUTSMuHN^{\Re}$ to eventually establish $FP(CUTSMuHN^{\Re})$

(13) $\big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\big\}\ S'''\ \big\{FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big\}$
, after applying conjunction of $(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))$ on (12)

(14) $\Big\langle \exists S''' : \big(S''' \in CUTSMuHN^{\Re}\big) :: \big\{(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\big\}$
$S'''\ \big\{FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big\}\Big\rangle$
, quantifying (13); this shows that (10) holds

(15) $\Big((\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))$ **ensures** $\big(FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big)\Big)$ in $CUTSMuHN^{\Re}$
, after conjuncting (5) and (10), and using definition of **ensures** (of UNITY [24, p. 50])

(16) $\Big((\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')) \mapsto \big(FP(CUTSMuHN^{\Re}) \wedge (\hat{\Delta} \vee (\alpha \trianglerighteq \alpha'))\big)\Big)$ in $CUTSMuHN^{\Re}$
, after applying definition of "$\mapsto$" from Equation (A.10) on (15)

(17) $\Big\langle \forall S : \big(S \in CUTSMuHN^{\aleph}\big) :: \big\{\hat{\Delta} \wedge \neg(\alpha \trianglerighteq \alpha')\big\}\ S^*\ \big\{\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big\}\Big\rangle$ (B.8)
, after applying Equation (6.16) on (1) & (16), and then quantifying.

In the above proof, no S'' statement in $CUTSMuHN^{\Re}$ affects $\big(\hat{\Delta} \vee (\alpha \trianglerighteq \alpha')\big)$; these statements only affect the state of $\mathcal{Q}_R$ and $\mathcal{T}_{\mathrm{w}}$ in any *host*. Also, the statement S''' happens to be the last statement in $CUTSMuHN^{\Re}$ that eventually establishes $FP(CUTSMuHN^{\Re})$. It is to be noted here that, as the statements in $CUTSMuHN^{\Re}$ are executed consecutively, the conjuncts of $FP(CUTSMuHN^{\Re})$ corresponding to those statements are being

satisfied successively, until eventually all conjuncts of $FP(CUTSMuHN^{\Re})$ are satisfied.

Lastly, conjunct $\left\langle \exists S' : \left(S' \in CUTSMuHN^{\aleph}\right) :: \left\{\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')\right\} S'^{*} \left\{\neg\left(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')\right)\right\} \right\rangle$ is shown to be satisfied as follows:

(1) $\left\{\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')\right\}\ S'\ \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\}$, premise

(2) $\hat{\Delta} \Rightarrow \neg(\alpha \veebar \alpha')$, from program text

(3) $\left(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')\right) \Rightarrow \left(\neg(\alpha \veebar \alpha') \wedge \neg(\alpha \veebar \alpha')\right) \equiv \neg(\alpha \veebar \alpha')$, using (2)

(4) $\left\{\neg(\alpha \veebar \alpha')\right\}\ S'\ \left\{\alpha \veebar \alpha'\right\}$, replacing precondition and postcondition of (1) by (3)
, (4) holds as there exists such an S' in $CUTSMuHN^{\aleph}$; so, (1) holds

(5) $\left\langle \forall S'''' : \left(S'''' \in CUTSMuHN^{\Re}\right) :: \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg\left(FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right)\right\} \right.$
$\left. S''''\ \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \vee \left(FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right)\right\} \right\rangle$, premise

(6) $\left\langle \forall S'''' : \left(S'''' \in CUTSMuHN^{\Re}\right) :: \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\right\} \right.$
$\left. S''''\ \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\} \right\rangle$, after simplifying (5)

(7) $\left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\}\ S''''\ \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\} \equiv \left\{\alpha \veebar \alpha'\right\}\ S''''\ \left\{\alpha \veebar \alpha'\right\}$
, using (2); holds from program text, as no S'''' of $CUTSMuHN^{\Re}$ affects $(\alpha \veebar \alpha')$

(8) $\left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\right\}\ S''''\ \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\}$
, after applying consequence rule on (7) for precondition strengthening

(9) $\left\langle \forall S'''' : \left(S'''' \in CUTSMuHN^{\Re}\right) :: \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\right\} \right.$
$\left. S''''\ \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\} \right\rangle$, quantifying (8); thus (5) holds

(10) $\left\langle \exists S''''' : \left(S''''' \in CUTSMuHN^{\Re}\right) :: \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg\left(FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right)\right\} \right.$
$\left. S'''''\ \left\{FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\} \right\rangle$, premise

(11) $\left\langle \exists S''''' : \left(S''''' \in CUTSMuHN^{\Re}\right) :: \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\right\} \right.$
$\left. S'''''\ \left\{FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\} \right\rangle$, simplifying (10)

(12) $\left\{\neg FP(CUTSMuHN^{\Re})\right\}\ S'''''\ \left\{FP(CUTSMuHN^{\Re})\right\}$, holds from program text
, as such an S''''' exists in $CUTSMuHN^{\Re}$ to eventually establish $FP(CUTSMuHN^{\Re})$

(13) $\left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\right\}\ S'''''\ \left\{FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\}$
, after applying conjunction of $\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))$ on (12)

(14) $\left\langle \exists S''''' : \left(S''''' \in CUTSMuHN^{\Re}\right) :: \left\{\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \wedge \neg FP(CUTSMuHN^{\Re})\right\} \right.$
$\left. S'''''\ \left\{FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\right\} \right\rangle$
, quantifying (13); this shows that (10) holds

(15) $\Big(\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))$ **ensures** $\big(FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\big)\Big)$ in $CUTSMuHN^{\Re}$

, after conjuncting (5) and (10), and using definition of **ensures** (of UNITY [24, p. 50])

(16) $\Big(\neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')) \mapsto \big(FP(CUTSMuHN^{\Re}) \wedge \neg(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))\big)\Big)$ in $CUTSMuHN^{\Re}$

, after applying definition of "$\mapsto$" from Equation (A.10) on (15)

(17) $\Big\langle \exists S' : \big(S' \in CUTSMuHN^{\Re}\big) :: \big\{\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')\big\}\ S'^{*}\ \big\{\neg\big(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha')\big)\big\}\Big\rangle$ (B.9)

, after applying Equation (6.16) on (1) & (16), and then quantifying.

In the above proof, S' represents the second substatement of following assign statement:

$$\langle\ M, \mathcal{Q}_S := \texttt{head}(\mathcal{Q}_S), \texttt{tail}(\mathcal{Q}_S)$$
$$[]\ \langle\ \mathcal{Q}_{S_{\mathrm{W}}} := \mathcal{Q}_{S_{\mathrm{W}}} \bullet M \quad \text{if } (M.ni = \mathrm{W})\ \|\ \mathcal{Q}_{S_{\mathrm{WL}}} := \mathcal{Q}_{S_{\mathrm{WL}}} \bullet M \quad \text{if } (M.ni = \mathrm{WL})\ \rangle\ \rangle \quad \text{if } \neg(\mathcal{Q}_S = \perp)$$

Before its execution, $\neg(\alpha \veebar \alpha')$ holds, and after its execution $(\alpha \veebar \alpha')$ gets satisfied. Again, no statement S'''' of $CUTSMuHN^{\Re}$ affects $(\hat{\Delta} \wedge \neg(\alpha \veebar \alpha'))$. Moreover, S'''' is the last statement in $CUTSMuHN^{\Re}$ that establishes $FP(CUTSMuHN^{\Re})$.

The conjunction of assertion (B.7), assertion (B.8) and assertion (B.9) gives assertion (B.6), from which assertion (6.17) is derived as follows:

$$\frac{\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big)\ \textbf{ensures}\ \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \veebar \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big)}{\big(\texttt{head}(host(i).\mathcal{Q}_S) = M\big) \mapsto \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{W}}}) = M\big) \veebar \big(\texttt{last}(host(i).\mathcal{Q}_{S_{\mathrm{WL}}}) = M\big)}$$

, after applying Equation (A.10)

This completes the proof of assertion (6.17). □

References

[1] Anurag Acharya, M. Ranganathan, and Joel H. Saltz, *Sumatra: A Language for Resource-Aware Mobile Programs*, Mobile Object Systems Towards the Programmable Internet (Jan Vitek and Christian Tschudin, eds.), Lecture Notes in Computer Science, vol. 1222, Springer-Verlag, Berlin/Heidelberg, Germany, 1997, pp. 111–130.

[2] Sudhir Ahuja, Nicholas Carriero, and David Gelernter, *Linda and Friends*, Computer **19** (1986), no. 8, 26–34.

[3] Bob Aiken, John Strassner, Brian E Carpenter, Ian Foster, Clifford Lynch, Joe Mambretti, Reagan Moore, and Benjamin Teitelbaum, *Network Policy and Services: A Report of a Workshop on Middleware*, RFC 2768 (2000).

[4] Ian F. Akyildiz, Xudong Wang, and Weilin Wang, *Wireless Mesh Networks: A Survey*, Computer Networks **47** (2005), no. 4, 445–487.

[5] Lachlan Aldred, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M ter Hofstede, *On the Notion of Coupling in Communication Middleware*, On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE (Robert Meersman, Zahir Tari, Mohand-Saïd Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoğlu, H.-Arno Jacobsen, Joseph Loyall, Michael Kifer, and Stefano Spaccapietra, eds.), Lecture Notes in Computer Science, vol. 3761, Springer-Verlag, Berlin/Heidelberg, Germany, 2005, pp. 1015–1033.

[6] Özalp Babaoğlu, Hein Meling, and Alberto Montresor, *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems*, Proceedings of the 22nd

International Conference on Distributed Computing Systems (Los Alamitos, CA, USA), ICDCS '02, IEEE Computer Society Press, July 2002, pp. 15–22.

[7] R. J. R. Back and K. Sere, *Stepwise Refinement of Parallel Algorithms*, Science of Computer Programming **13** (1990), no. 2-3, 133–180.

[8] J. Baumann, F. Hohl, K. Rothermel, and M. Straßer, *Mole – Concepts of a Mobile Agent System*, World Wide Web **1** (1998), no. 3, 123–137.

[9] Paolo Bellavista, Antonio Corradi, and Cesare Stefanelli, *Mobile Agent Middleware for Mobile Computing*, Computer **34** (2001), no. 3, 73–81.

[10] Umesh Bellur and Siddharth Bondre, *xSpace - A Tuple Space for XML & Its Application in Orchestration of Web Services*, Proceedings of the 2006 Symposium on Applied Computing (New York, NY, USA), SAC '06, ACM Press, April 2006, pp. 766–772.

[11] Philip A. Bernstein, *Middleware: A Model for Distributed System Services*, Communications of the ACM **39** (1996), no. 2, 86–98.

[12] Burnie Blakeley, Harry Harris, and Rhys Lewis, *Messaging and Queueing Using the MQI*, McGraw-Hill, Inc., New York, NY, USA, 1995.

[13] Jeffrey M. Bradshaw, *Software Agents*, MIT Press, Cambridge, MA, USA, 1997.

[14] Dario Bruneo, Antonio Puliafito, and Marco Scarpa, *Mobile Middleware: Definition and Motivations*, The Handbook of Mobile Middleware (Paolo Bellavista and Antonio Corradi, eds.), Auerbach Publications, Boston, MA, USA, 2006, pp. 145–167.

[15] Nadia Busi, Cristian Manfredini, Alberto Montresor, and Gianluigi Zavattaro, *PeerSpaces: Data-driven Coordination in Peer-to-Peer Networks*, Proceedings of the 2003 Symposium on Applied Computing (New York, NY, USA), SAC '03, ACM Press, March 2003, pp. 380–386.

[16] Giacomo Cabri, Luca Ferrari, Letizia Leonardi, Marco Mamei, and Franco Zambonelli, *Uncoupling Coordination: Tuple-Based Models for Mobility*, The Handbook

of Mobile Middleware (Paolo Bellavista and Antonio Corradi, eds.), Auerbach Publications, Boston, MA, USA, 2006, pp. 229–255.

[17] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli, *MARS: A Programmable Coordination Architecture for Mobile Agents*, Internet Computing **4** (2000), no. 4, 26–35.

[18] ———, *Mobile-Agent Coordination Models for Internet Applications*, Computer **33** (2000), no. 2, 82–89.

[19] Licia Capra, Gordon S. Blair, Cecilia Mascolo, Wolfgang Emmerich, and Paul Grace, *Exploiting Reflection in Mobile Computing Middleware*, SIGMOBILE Mobile Computing and Communications Review **6** (2002), no. 4, 34–44.

[20] Luca Cardelli and Andrew D. Gordon, *Mobile Ambients*, Foundations of Software Science and Computation Structures (Maurice Nivat, ed.), Lecture Notes in Computer Science, vol. 1378, Springer-Verlag, Berlin/Heidelberg, Germany, 1998, pp. 140–155.

[21] Nicholas Carriero and David Gelernter, *Linda in Context*, Communications of the ACM **32** (1989), no. 4, 444–458.

[22] S. Castellani, Paolo Ciancarini, and Davide Rossi, *The ShaPE of ShaDe: A Coordination System*, Tech. Report UBLCS-96-5, University of Bologna, Bologna, Italy, 1996.

[23] Alvin T. S. Chan and Siu-Nam Chuang, *MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing*, Transactions on Software Engineering **29** (2003), no. 12, 1072–1085.

[24] K. Mani Chandy and Jayadev Misra, *Parallel Program Design: A Foundation*, Addison-Wesley Publishing Co., Inc., Boston, MA, USA, 1988.

[25] Shigeru Chiba, Kazuhiko Kato, and Takashi Masuda, *Optimization of Distributed Communication in Multiprotocol Tuple Space*, Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing (Los Alamitos, CA, USA), SPDP '91, IEEE Computer Society Press, December 1991, pp. 282–285.

[26] ———, *Exploiting a Weak Consistency to Implement Distributed Tuple Space*, Proceedings of the 12th International Conference on Distributed Computing Systems (Los Alamitos, CA, USA), ICDCS '92, IEEE Computer Society Press, June 1992, pp. 416–423.

[27] Paolo Ciancarini, F. Franzé, and Cecilia Mascolo, *Using a Coordination Language to Specify and Analyze Systems Containing Mobile Components*, Transactions on Software Engineering and Methodology **9** (2000), no. 2, 167–198.

[28] Paolo Ciancarini, Adreas Knoche, Robert Tolksdorf, and Fabio Vitali, *PageSpace: An Architecture to Coordinate Distributed Applications on the Web*, Computer Networks and ISDN Systems **28** (1996), no. 7-11, 941–952.

[29] Paolo Ciancarini, Robert Tolksdorf, Fabio Vitali, Davide Rossi, and Adreas Knoche, *Redesigning the Web: From Passive Pages to Coordinated Agents in PageSpaces*, Proceedings of the 3rd International Symposium on Autonomous Decentralized Systems (Los Alamitos, CA, USA), ISADS '97, IEEE Computer Society Press, April 1997, pp. 377–384.

[30] Gabriel Ciobanu and Calin Juravle, *Flexible Software Architecture and Language for Mobile Agents*, Concurrency and Computation: Practice and Experience **24** (2012), no. 6, 559–571.

[31] Pierre Collette and Edgar Knapp, *A Foundation for Modular Reasoning About Safety and Progress Properties of State-based Concurrent Programs*, Theoretical Computer Science **183** (1997), no. 2, 253–279.

[32] Nigel Davies, Adrian Friday, Stephen Paul Wade, and Gordon S. Blair, *L^2imbo: A Distributed Systems Platform for Mobile Computing*, Mobile Networks and Applications **3** (1998), no. 2, 143–156.

[33] Nigel Davies, Stephen Paul Wade, Adrian Friday, and Gordon S. Blair, *Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications*, Proceedings of the

International Conference on Open Distributed Processing and Distributed Platforms (London, UK), ICODP/ICDP '97, Chapman & Hall, Ltd., 1997, pp. 291–302.

[34] U. Dayal, *The HiPAC Project: Combining Active Databases and Timining Constraints*, SIGMOD Record **17** (1988), no. 1, 51–70.

[35] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese, *KLAIM: A Kernel Language for Agents Interaction and Mobility*, Transactions on Software Engineering **24** (1998), no. 5, 315–330.

[36] Rocco De Nicola, Diego Latella, and Mieke Massink, *Formal Modeling and Quantitative Analysis of KLAIM-based Mobile Systems*, Proceedings of the 20th Annual Symposium on Applied Computing (New York, NY, USA), SAC '05, ACM Press, March 2005, pp. 428–435.

[37] S. E. Deering, *Host Extensions for IP Multicasting*, RFC 1112 (INTERNET STANDARD), August 1989, URL: "http://www.ietf.org/rfc/rfc1112.txt" [last accessed: 10-Sep-2014].

[38] Enrico Denti, Antonio Natali, and Andrea Omicini, *On the Expressive Power of Language for Programming Coordination Media*, Proceedings of the 1998 Symposium on Applied Computing (New York, NY, USA), SAC '98, ACM Press, August 1998, pp. 169–177.

[39] Enrico Denti, Antonio Natali, Andrea Omicini, and Marco Venuti, *An Extensible Framework for the Development of Coordinated Applications*, Coordination Languages and Models (Paolo Ciancarini and Chris Hankin, eds.), Lecture Notes in Computer Science, vol. 1061, Springer-Verlag, Berlin/Heidelberg, Germany, 1996, pp. 305–320.

[40] Alan Dickman, *Designing Applications with MSMQ: Message Queuing for Developers*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[41] Peter Dömel, Anselm Lingnau, and Oswald Drobnik, *Mobile Agent Interaction in*

Heterogeneous Environments, Mobile Agents (Kurt Rothermel and Radu Popescu-Zeletin, eds.), Lecture Notes in Computer Science, vol. 1219, Springer-Verlag, Berlin/Heidelberg, Germany, 1997, pp. 136–148.

[42] Wolfgang Emmerich, Mikio Aoyama, and Joe Sventek, *The Impact of Research on the Development of Middleware Technology*, Transactions on Software Engineering and Methodology **17** (2008), no. 4, 19:1–19:48.

[43] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, *The Many Faces of Publish/Subscribe*, Computing Surveys **35** (2003), no. 2, 114–131.

[44] Ming-Dong Feng, Y. Q. Gao, and Chung-Kwong Yuen, *Distributed Linda Tuplespace Algorithms and Implementations*, Parallel Processing: CONPAR 94 – VAPP VI (Bruno Buchberger and Jens Volkert, eds.), Lecture Notes in Computer Science, vol. 854, Springer-Verlag, Berlin/Heidelberg, Germany, 1994, pp. 581–592.

[45] Yuhong Feng and Wentong Cai, *Execution Coordination in Mobile Agent-based Distributed Job Workflow Execution*, Journal of Systems Architecture **54** (2008), no. 10, 944–956.

[46] Chien-Liang Fok, Gruia-Catalin Roman, and Gregory Hackmann, *A Lightweight Coordination Middleware for Mobile Computing*, Coordination Models and Languages (Rocco Nicola, Gian-Luigi Ferrari, and Greg Meredith, eds.), Lecture Notes in Computer Science, vol. 2949, Springer-Verlag, Berlin/Heidelberg, Germany, 2004, pp. 135–151.

[47] Eric Freeman, Ken Arnold, and Susanne Hupfer, *JavaSpaces Principles, Patterns, and Practice*, Addison-Wesley Longman Ltd., Reading, MA, USA, 1999.

[48] Abdulbaset Gaddah and Thomas Kunz, *A Survey of Middleware Paradigms for Mobile Computing*, Tech. Report SCE-03-16, Carleton University, Systems and Computing Engineering, Ottawa, Canada, 2003.

[49] Kurt Geihs, *Middleware Challenges Ahead*, Computer **34** (2001), no. 6, 24–31.

[50] David Gelernter, *Generative Communication in Linda*, Transactions on Programming Languages and Systems **7** (1985), no. 1, 80–112.

[51] ———, *Multiple Tuple Spaces in Linda*, PARLE '89 Parallel Architectures and Languages Europe (Eddy Odijk, Martin Rem, and Jean-Claude Syre, eds.), Lecture Notes in Computer Science, vol. 366, Springer-Verlag, Berlin/Heidelberg, Germany, 1989, pp. 20–27.

[52] David Gelernter and Arthur J. Bernstein, *Distributed Communication via Global Buffer*, Proceedings of the 1st SIGACT-SIGOPS Symposium on Principles of Distributed Computing (New York, NY, USA), PODC '82, ACM Press, August 1982, pp. 10–18.

[53] P. B. Gibbons, *A Stub Generator for Multilanguage RPC in Heterogeneous Environments*, Transactions on Software Engineering **SE-13** (1987), no. 1, 77–87.

[54] James Gosling, Bill Joy, and Guy L. Steele, *The Java^{TM} Language Specification*, 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[55] Klaus Herrmann, Gero Mühl, and Michael A. Jaeger, *MESH*Mdl *Event Spaces – A Coordination Middleware for Self-organizing Applications in Ad Hoc Networks*, Pervasive and Mobile Computing **3** (2007), no. 4, 467–487.

[56] C. A. R. Hoare, *An Axiomatic Basis for Computer Programming*, Communications of the ACM **12** (1969), no. 10, 576–583.

[57] Chi-Fu Huang, Hung-Wei Lee, and Yu-Chee Tseng, *A Two-Tier Heterogeneous Mobile Ad Hoc Network Architecture and Its Load-Balance Routing Problem*, Mobile Networks and Applications **9** (2004), no. 4, 379–391.

[58] Yongqiang Huang and Hector Garcia-Molina, *Publish/Subscribe in a Mobile Environment*, Wireless Networks **10** (2004), no. 6, 643–652.

[59] Susanne C. Hupfer, *Melinda: Linda with Multiple Tuple Spaces*, Tech. Report YALEU/DCS/RR-766, Yale University, New Haven, CT, USA, 1990.

[60] IEEE 802.11 WG Std., *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Tech. Report 802.11™-2007, June 2007.

[61] IEEE 802.15 WG Std., *Low-Rate Wireless Personal Area Networks (LR-WPANs)*, Tech. Report 802.15.4™-2011 (Revision of IEEE Std 802.15.4-2006), September 2011.

[62] Brad Johanson and Armando Fox, *The Event Heap: A Coordination Infrastructure for Interactive Workspaces*, Proceedings of the 4th Workshop on Mobile Computing Systems and Applications (Los Alamitos, CA, USA), WMCSA '02, IEEE Computer Society Press, June 2002, pp. 83–93.

[63] Christine Julien, Jamie Payton, and Gruia-Catalin Roman, *Reasoning About Context-Awareness in the Presence of Mobility*, Electronic Notes in Theoretical Computer Science **97** (2004), 259–276.

[64] Christine Julien and Gruia-Catalin Roman, *EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications*, Transactions on Software Engineering **32** (2006), no. 5, 281–298.

[65] Joseph Kiniry and Daniel Zimmerman, *A Hands-On Look at Java Mobile Agents*, Internet Computing **1** (1997), no. 4, 21–30.

[66] Matthias Klusch and Katia Sycara, *Brokering and Matchmaking for Coordination of Agent Societies: A Survey*, Coordination of Internet Agents: Models, Technologies and Applications (Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, eds.), Springer-Verlag, Berlin/Heidelberg, Germany, 2001, pp. 197–224.

[67] Donald Ervin Knuth, *The Art of Computer Progamming, Vol. 3 – Sorting and Searching*, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[68] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko, *Agent TCL: Targeting the Needs of Mobile Computers*, Internet Computing **1** (1997), no. 4, 58–67.

[69] Danny B. Lange and Mitsuru Oshima, *Seven Good Reasons for Mobile Agents*, Communications of the ACM **42** (1999), no. 3, 88–89.

[70] Tobin J. Lehman, Stephen W. McLaughry, and Peter Wyckoff, *T Spaces: The Next Wave*, Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences (Los Alamitos, CA, USA), HICSS '99, vol. 8, IEEE Computer Society Press, January 1999.

[71] Tim Lindholm and Frank Yellin, *The Java™ Virtual Machine Specification*, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[72] Zhengding Lu, Chunlin Li, and Layuan Li, *Coordinating Mobile Agents by the XML-Based Tuple Space*, Journal of Computer Science and Technology **17** (2002), no. 6, 888–900.

[73] Pattie Maes, *Concepts and Experiments in Computational Reflection*, SIGPLAN Notices **22** (1987), no. 12, 147–155.

[74] Thomas W. Malone and Kevin Crowston, *The Interdisciplinary Study of Coordination*, Computing Surveys **26** (1994), no. 1, 87–119.

[75] Marco Mamei and Franco Zambonelli, *Programming Stigmergic Coordination with the TOTA Middleware*, Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (New York, NY, USA), AAMAS '05, ACM Press, July 2005, pp. 415–422.

[76] ———, *Programming Pervasive and Mobile Computing Applications: The TOTA Approach*, Transactions on Software Engineering and Methodology **18** (2009), no. 4, 15:1–15:56.

[77] Marco Mamei, Franco Zambonelli, and Letizia Leonardi, T*uples* O*n* T*he* A*ir: a Middleware for Context-Aware Computing in Dynamic Networks*, Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (Los Alamitos, CA, USA), ICDCSW '03, IEEE Computer Society Press, May 2003, pp. 342–347.

[78] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich, *Mobile Computing Middleware*, Advanced Lectures on Networking (Enrico Gregori, Giuseppe Anastasi, and Stefano Basagni, eds.), Lecture Notes in Computer Science, vol. 2497, Springer-Verlag, Berlin/Heidelberg, Germany, 2002, pp. 20–58.

[79] ———, *Principles of Mobile Computing Middleware*, Middleware for Communications (Qusay H. Mahmoud, ed.), John Wiley & Sons, Ltd, West Sussex, UK, 2005, pp. 261–280.

[80] Cecilia Mascolo, Licia Capra, Stefanos Zachariadis, and Wolfgang Emmerich, *XMIDDLE: A Data-Sharing Middleware for Mobile Computing*, Wireless Personal Communications **21** (2002), no. 1, 77–103.

[81] Satoshi Matsuoka and Satoru Kawai, *Using Tuple Space Communication in Distributed Object-oriented Languages*, SIGPLAN Notices **23** (1988), no. 11, 276–284.

[82] Peter J. McCann and Gruia-Catalin Roman, *Compositional Programming Abstractions for Mobile Computing*, Transactions on Software Engineering **24** (1998), no. 2, 97–110.

[83] ———, *Modeling Mobile IP in Mobile UNITY*, Transactions on Software Engineering and Methodology **8** (1999), no. 2, 115–146.

[84] René Meier, *Communication Paradigms for Mobile Computing*, SIGMOBILE Mobile Computing and Communications Review **6** (2002), no. 4, 56–58.

[85] Ronaldo Menezes and Robert Tolksdorf, *A New Approach to Scalable Linda-systems based on Swarms*, Proceedings of the 2003 Symposium on Applied Computing (New York, NY, USA), SAC '03, ACM Press, March 2003, pp. 375–379.

[86] Stijn Mostinckx, Christophe Scholliers, Eline Philips, Charlotte Herzeel, and Wolfgang De Meuter, *Fact Spaces: Coordination in the Face of Disconnection*, Coordination Models and Languages (Amy L. Murphy and Jan Vitek, eds.), Lecture Notes in Computer Science, vol. 4467, Springer-Verlag, Berlin/Heidelberg, Germany, 2007, pp. 268–285.

[87] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman, *Lime: A Coordination Model and Middleware supporting Mobility of Hosts and Agents*, Transactions on Software Engineering and Methodology **15** (2006), no. 3, 279–328.

[88] Peter Naur and Brian Randell (eds.), *NATO Software Engineering Conference*, Scientific Affairs Division, NATO, October 1968, URL: "`http://home-pages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF`", p. 14 [last accessed: 10-Sep-2014].

[89] Andrea Omicini and Franco Zambonelli, *Coordination for Internet Application Development*, Autonomous Agents and Multi-Agent Systems **2** (1999), no. 3, 251–269.

[90] Rasmus Pagh and Flemming Friche Rodler, *Cuckoo hashing*, Journal of Algorithms **51** (2004), no. 2, 122–144.

[91] George Angelos Papadopoulos and Farhad Arbab, *Coordination Models and Languages*, The Engineering of Large Systems (Marvin V. Zelkowitz, ed.), Advances in Computers, vol. 46, Academic Press, San Diego, CA, USA, 1998, pp. 329–400.

[92] Holger Peine and Torsten Stolpmann, *The Architecture of the Ara Platform for Mobile Agents*, Mobile Agents (Kurt Rothermel and Radu Popescu-Zeletin, eds.), Lecture Notes in Computer Science, vol. 1219, Springer-Verlag, Berlin/Heidelberg, Germany, 1997, pp. 50–61.

[93] Gian Pietro Picco, *μ-Code: A Lightweight and Flexible Mobile Code Toolkit*, Mobile Agents (Kurt Rothermel and Fritz Hohl, eds.), Lecture Notes in Computer Science, vol. 1477, Springer-Verlag, Berlin/Heidelberg, Germany, 1998, pp. 160–171.

[94] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman, *LIME: Linda Meets Mobility*, Proceedings of the 21st International Conference on Software Engineering (New York, NY, USA), ICSE '99, ACM Press, May 1999, pp. 368–377.

[95] Gian Pietro Picco, Gruia-Catalin Roman, and Peter J. McCann, *Reasoning about Code Mobility with Mobile UNITY*, Transactions on Software Engineering and Methodology **10** (2001), no. 3, 338–395.

[96] Gruia-Catalin Roman, Radu Handorean, and Rohan Sen, *Tuple Space Coordination Across Space and Time*, Coordination Models and Languages (Paolo Ciancarini and Herbert Wiklicky, eds.), Lecture Notes in Computer Science, vol. 4038, Springer-Verlag, Berlin/Heidelberg, Germany, 2006, pp. 266–280.

[97] Gruia-Catalin Roman, Peter J. McCann, and Jerome Y. Plun, *Mobile UNITY: Reasoning and Specification in Mobile Computing*, Transactions on Software Engineering and Methodology **6** (1997), no. 3, 250–282.

[98] Gruia-Catalin Roman and Jamie Payton, *Mobile UNITY Schemas for Agent Coordination*, Abstract State Machines 2003 (Egon Börger, Angelo Gargantini, and Elvinia Riccobene, eds.), Lecture Notes in Computer Science, vol. 2589, Springer-Verlag, Berlin/Heidelberg, Germany, 2003, pp. 126–150.

[99] ———, *A Principled Exploration of Coordination Models*, Theoretical Computer Science **336** (2005), no. 2-3, 367–401.

[100] Davide Rossi, Giacomo Cabri, and Enrico Denti, *Tuple-based Technologies for Coordination*, Coordination of Internet Agents (Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, eds.), Springer-Verlag, Berlin/Heidelberg, Germany, 2001, pp. 83–109.

[101] Craig Sadler, *Facts, Games and World Champions of Correspondence Chess*, 2004, URL: "`http://www.schemingmind.com/journalarticle.aspx?article_id=1`" [last accessed: 10-Sep-2014].

[102] Mark G. Staskauskas, *Specification and Verification of Large-Scale Reactive Programs*, Tech. Report CS-TR-92-34, University of Texas at Austin, Austin, TX, USA, 1992.

[103] ———, *Formal Derivation of Concurrent Programs: An Example from Industry*, Transactions on Software Engineering **19** (1993), no. 5, 503–528.

[104] Mark-Oliver Stehr, *Compositionality for Tightly Coupled Systems: A New Application of the Propositions-as-Types Interpretation*, Electronic Notes in Theoretical Computer Science **159** (2006), 299–323.

[105] W. P. Stevens, G. J. Myers, and L. L. Constantine, *Structured Design*, IBM Systems Journal **13** (1974), no. 2, 115–139.

[106] Sun IPR, *JavaSpaces™ Specification—1.0*, Sun Microsystems, Inc., January 1999.

[107] ———, *JavaSpaces™ Service Specification—2.0*, Sun Microsystems, Inc., 2003.

[108] Robert Tolksdorf, *Laura – A Service-based Coordination Language*, Science of Computer Programming **31** (1998), no. 2-3, 359–381.

[109] Mirko Viroli, Danilo Pianini, and Jacob Beal, *Linda in Space-Time: An Adaptive Coordination Model for Mobile Ad-Hoc Environments*, Coordination Models and Languages (Marjan Sirjani, ed.), Lecture Notes in Computer Science, vol. 7274, Springer-Verlag, Berlin/Heidelberg, Germany, 2012, pp. 212–229.

[110] Stephen Paul Wade, *An Investigation into the Use of the Tuple Space Paradigm in Mobile Computing Environments*, Ph.D. thesis, Lancaster University, Lancaster, UK, 1999.

[111] Jim Waldo, *The Jini Specifications*, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[112] James E. White, *Telescript Technology: Mobile Agent*, Mobility: Processes, Computers, and Agents (Dejan Milojičić, Frederick Douglis, and Richard Wheeler, eds.), ACM Press/Addison-Wesley Longman Publishing Co., Inc., New York, NY, USA, 1999, pp. 460–493.

[113] Wikipedia, *Decoupling*, 2013, URL: "`http://en.wikipedia.org/wiki/Decoup-`
`-ling`" [last accessed: 10-Sep-2014].

[114] Peter Wyckoff, Stephen W. McLaughry, Tobin J. Lehman, and Daniel A. Ford, *T Spaces*, IBM Systems Journal **37** (1998), no. 3, 454–474.

[115] George Xylomenos and George C. Polyzos, *IP Multicast for Mobile Hosts*, Communications Magazine **35** (1997), no. 1, 54–58.

Selective Publications from Thesis

1. Suddhasil De, Sukumar Nandi, and Diganta Goswami, *Tuple Space Enhancements for Mobile Middleware*, International Journal of Communication Networks and Distributed Systems, Vol. 12, No. 3, pp. 299-326 (2014), Inderscience Publishers, Geneva, Switzerland.

2. Suddhasil De, Diganta Goswami, and Sukumar Nandi, *Consistent Coordination Decoupling in Tuple Space Based Mobile Middleware: Design and Formal Specifications*, Distributed Computing and Internet Technology: 9th ICDCIT'13 (Chittaranjan Hota, and Pradip K. Srimani, eds.), Lecture Notes in Computer Science, Vol. 7753, pp. 220-231 (2013), Springer-Verlag, Berlin/Heidelberg, Germany.

3. Suddhasil De, Diganta Goswami, and Sukumar Nandi, *Formal Modeling of Mobile Middleware for Tuple Space Coordination over Multiple Heterogeneous Networks*, Quality, Reliability, Security and Robustness in Heterogeneous Networks: 9th QSHINE'13 (Karan Singh, and Amit K. Awasthi, eds.), Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 115, pp. 415-430 (2013), Springer-Verlag, Berlin/Heidelberg, Germany.

4. Suddhasil De, Suchetana Chakraborty, Sukumar Nandi, and Diganta Goswami, *Supporting Tuple Space based Mobile Middleware over Unreliable Mobile Infrastructures: Design and Formal Specifications*, In Proceedings of 6th International Conference on Advanced Networks and Telecommunication Systems (ANTS'12), pp. 123-125, December 2012 (Bangalore, India), IEEE Press, Piscataway, NJ, USA.

5. Suddhasil De, Sukumar Nandi, and Diganta Goswami, *Architectures of Mobile Middleware: A Taxonomic Perspective*, In Proceedings of 2nd International Conference

on Parallel, Distributed and Grid Computing (PDGC'12), pp. 480-484, December 2012 (Solan, Himachal Pradesh, India), IEEE Press, Piscataway, NJ, USA.

6. Suddhasil De, Diganta Goswami, Sukumar Nandi, and Suchetana Chakraborty, *Formalization of a Fully-Decoupled Reactive Tuple Space Model for Mobile Middleware*, Mobile Wireless Middleware, Operating Systems, and Applications: 5th MOBILWARE'12 (Cristian Borcea et al., eds.), Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 65, pp. 77-91 (2013), Springer-Verlag, Berlin/Heidelberg, Germany.

7. Suddhasil De, Sukumar Nandi, and Diganta Goswami, *Modeling an Enhanced Tuple Space based Mobile Middleware in UNITY*, In Proceedings of 11th International Conference on Ubiquitous Computing and Communications (IUCC'12), pp. 1684-1691, June 2012 (Liverpool, United Kingdom), IEEE Computer Society Press, Los Alamitos, CA, USA.

Brief Biography of the Author

I, **Suddhasil De**, was born and brought up in Durgapur, West Bengal, India. My parents, Late Milan Chandra De and Smt. Susreeta De, were both teachers (retd.) not only by profession but also by heart. I completed my schooling in Durgapur from D'Sector Primary School, B'Zone Boys' High School, and Bidhan Chandra Institution. After passing my Higher Secondary (XIIth standard) examination, I graduated in Bachelor of Technology in Computer Engineering from Institute of Engineering and Management, Salt Lake, Kolkata (then under University of Kalyani, now under West Bengal University of Technology) in 2001. I worked as lecturer in I.T. Dept., Jalpaiguri Government Engineering College, West Bengal from December 2001 for 2.5 years. Then I joined National Institute of Technology Silchar as lecturer in C.S.E. Dept. from August 2004 for 3 years till I felt the need for further academic upliftment. So, I left the job and joined the Doctoral programme in C.S.E. Dept. of Indian Institute of Technology Guwahati (in August 2007), where I carry out my research in mobile middleware. I successfully defended my thesis in 9th May 2015. My areas of research interests include distributed systems and middleware.