Formalization of a Fully-Decoupled Reactive Tuple Space Model for Mobile Middleware

Suddhasil De, Diganta Goswami, Sukumar Nandi, and Suchetana Chakraborty

Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, Assam – 781039, India
{suddhasil,dgoswami,sukumar,suchetana}@iitg.ernet.in

Abstract. This paper suggests an approach for formalizing Tuple Space based Mobile Middleware (TSMM) that contains a fully-decoupled reactive tuple space model as coordination medium. Formalization of TSMM is carried out using Mobile UNITY.

Keywords: mobile middleware, coordination, tuple space, Mobile UNITY.

1 Introduction

Mobile middleware [1], an emergent area of middleware research, originates to support execution of a variety of distributed applications in presence of mobility and dynamics in underlying infrastructure. Like other existing middleware, mobile middleware incorporates a suitable *coordination medium* for managing asynchronous interactions between different active components of an application, called *agents*, whose execution is supported by computing environments called *hosts*. Tuple space model [2], a popular coordination model, supports multiple inherent decoupling qualities [3], and as such is a potential coordination medium for mobile middleware [4], called Tuple Space based Mobile Middleware (TSMM). In TSMM, *tuple* is considered as basic unit of information exchanged during interaction of agents via a shared repository (called *tuple space*), while *antituple* is considered as basic unit of search key to identify some specific tuples residing in tuple space. Tuple space model subsequently includes reactivity, a powerful programming construct, to accomplish *synchronization decoupling*, another decoupling quality for agent interaction [3]. Recently, further decoupling ability is added to reactivity itself to achieve complete coordination decoupling in agent interaction [5]. TSMM, with this fully-decoupled tuple space model, facilitates application designers in developing robust and flexible applications.

Like other software/hardware design, formalization of TSMM is essential for performing an appropriate analysis of robustness and flexibility in its design. This paper suggests an approach for formally specifying and developing a TSMM, which incorporates a fully-decoupled reactive tuple space model, to define its precise semantics and lay the foundation for its implementation. A general-purpose formal reasoning tool, Mobile UNITY [6], which is an extension of well-known UNITY model [7], is used for formalizing this TSMM. After specifying and stepwise refining behaviors of TSMM in terms of Mobile UNITY, if the specifications

C. Borcea et al. (Eds.): MobilWare 2012, LNICST 65, pp. 77–91, 2013.

satisfy desired safety and progress properties, that TSMM is considered suitable for supporting robust and flexible applications. Authors believe that exhaustive formalization of any TSMM has not yet presented, though preliminary specifications of some functionalities exist in literature [8,9]. In both these works, basic tuple space operations and agent mobility are respectively formalized using Mobile UNITY, with reference to LIME. Moreover, in [9], formalization of agent mobility of other coordination models are also depicted. These works differ from this paper in several ways. First, this paper focuses on formalizing different aspects of a particular TSMM exclusively. Second, this TSMM has achieved full decoupling while coordinating agent interactions, which is widely dissimilar from LIME. Third, all functionalities of this TSMM, including fully-decoupled coordination, reactivity as well as associated communication and discovery mechanisms, are formalized in this paper. Agent mobility is only abstracted in this formalization as one macro to simplify its representation. Fourth, this paper also shows the construction of formal representation of an entire TSMM by combining individual specifications of its different functionalities using notations of a standard formal tool. Rest of the paper is organized as follows. Section 2 gives a brief overview of TSMM having a fully-decoupled reactive tuple space model, which is next formalized using Mobile UNITY in Section 3. Finally, Section 4 concludes the paper.

2 Overview of TSMM Having Fully-Decoupled Reactive Tuple Space Model

TSMM is the coordination tool to support agent interaction in mobile distributed applications, and it intends to provide ubiquity to the wide variety of activities a user performs. It assumes that connectivity of underlying network infrastructure can be *dynamic* and *unreliable*, whereas coordination between its two interacting agents is *asymmetric*. Former assumptions are essential to deal with host mobility and wireless connectivity of underlying infrastructure, while latter assumption brings more control on interacting agent, as it can accept/deny interactions with other available agents based on context, like users' choice, link capacity etc.

Architecture. TSMM comprise of several components, each of which are specific to agent or host. Each agent contains a local tuple space, called *agent tuple space* (`ATS`), and interfaces of `ATS`. Besides these two components, another component handles invoke of local primitives, while a pair of components handle invoke of remote primitives. Also, asymmetric interaction in each agent is enforced by acquaintance list. Each instance of host, running in each device, supports execution of multiple agents. In each host, different components manage functionalities of communication, discovery, host's core functionalities, a common tuple space called *host tuple space* (`HTS`), interfaces of `HTS`, agent management and mobility etc. Architecture of TSMM with all its components is shown in figure 1.

Tuple Space Model. In TSMM, tuples and antituples are considered as *unordered* sequence of heterogeneously typed fields, as presented in [10]. During

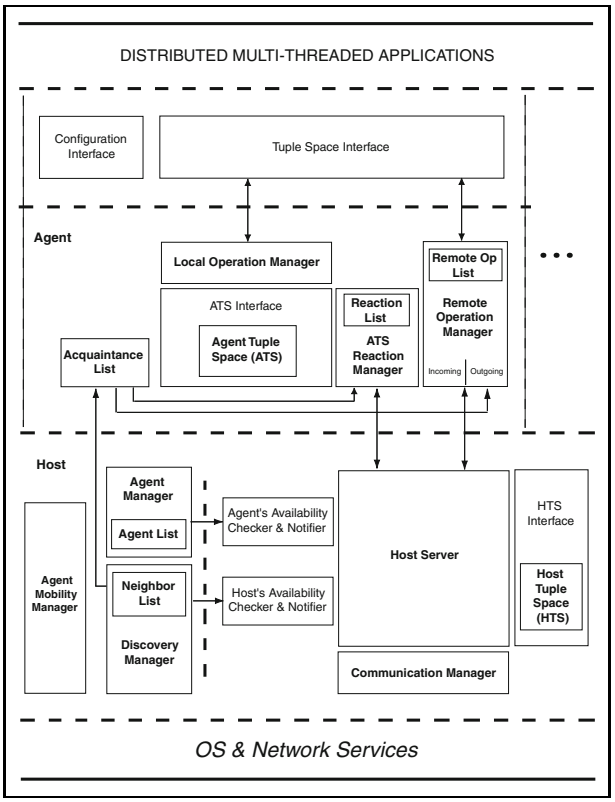

Fig. 1. Architecture of TSMM showing its significant components

interaction between any pair of agents (initiator of interaction is *reference agent* and destination becomes *target agent*), reference agent is interested in some tuples of tuple space, termed *sought tuples* [11], which are related to its interaction. It uses *antituple* to identify these sought tuples. While searching for sought tuples, antituple fields are compared with tuple fields following 'type-value', 'exact value' and 'polymorphic' matching conditions. Only fields of sought tuples match positively with fields of given antituple. Before reading/withdrawing sought tuples, they are first identified from tuple space by following tuple-antituple matching using given antituple. Different primitives are defined to carry out writing, reading and withdrawing tuples from tuple space. Tuple space is partitioned into *preamble* and *tuple store* for identifying *apposite* tuples. Apposite tuples refer to those tuples present in tuple space, whose fields are suitable for matching with all constituent fields of an antituple according to matching conditions. In other words, sought tuples are selected from the set of apposite tuples. Preamble of tuple space holds all index tables corresponding to different constituent fields of all tuples present in tuple space, while tuple store is the actual storehouse of those tuples. Each index table is a list holding a set of indices of tuples in tuple store, which contains at least one constituent field having name or type

identical or polymorphically-related to table name. Any tuple-reading or -consuming primitive first identifies index table, whose content indicates locations of different apposite tuples in tuple store for given antituple. Moreover, tuple-consuming primitives, after withdrawing one/more sought tuples, update all relevant index tables in preamble. On the other hand, tuple-producing primitives first write given tuple(s) in tuple store, and update indices of written tuple(s) in all required index tables. Both `ATS` and `HTS` follow this structure of tuple space.

Both local and remote tuple-producing, tuple-reading and tuple-consuming primitives are present for handling tuple space operations in `ATS`. Tuple-producing primitives cover `out` and `outg`, while tuple-reading primitives include `rd`, `rdp`, `rdg` and `rdgp`, and tuple-consuming primitives are `in`, `inp`, `ing` and `ingp`. Remote primitives are both blocking as well as nonblocking, whereas local primitives are solely nonblocking. Each agent carries out invoked local primitives in its `ATS`. Local primitives include `out`, `outg`, `rdp`, `rdgp`, `inp` and `ingp`, whereas, remote primitives supported are `out`, `outg`, `rd`, `rdp`, `rdg`, `rdgp`, `in`, `inp`, `ing` and `ingp`, details of which are given in [5]. For executing remote operations, parameters of invoked primitives are shipped by reference agent to specified target agent(s), executed in `ATS` of each target agent and results of execution, if any, are sent back to reference agent. On the other hand, only two special primitives, viz. `inject` and `eject`, which are tuple-producing and -consuming respectively in nature, are provided for managing operations locally on `HTS`. However, only primitives corresponding to `ATS` are provided as application programming interfaces (APIs) for application programmers.

Reactivity Model. For achieving synchronization decoupling (i.e. decoupling reference agent from its invoked remote primitives), TSMM incorporates *reactivity* in `ATS`, which is the ability of `ATS` to monitor and respond to different circumstances (called *events*) during execution [12]. Reactivity is implemented by generating and registering *reaction* in `ATS` for monitoring and responding to events (like, presence of a particular sought tuple in tuple space etc.). For recognizing relevant event, reaction expects some condition to be specified by means of antituple. If condition gets satisfied, desired event is said to happen and corresponding registered reaction fires. Firing of reaction signifies that some application-defined actions (called reactive codes) will be executed subsequently, like notifying presence of tuples, withdrawing tuples from `ATS` etc, and responses are sent back to reference agent. Mode of a reaction indicates its active period, and is of two types in TSMM, viz. `ONCE` and `ONCE/TUPLE`. With `ONCE` modality, reactions fire once irrespective of the number of matching tuples and immediately get deregistered, while reactions with `ONCE/TUPLE` mode continue firing for each positively-matched tuple of `ATS`. Typically, a reaction comprises of antituple, name of invoked primitive, reactive code, identity of `ATS`, mode, user identity etc., of which antituple, invoked primitive name, reactive code and `ATS` identity are mandatory components.

Fully-Decoupled Coordination Model. In TSMM, interactions among different agents are completely decoupled by using decoupled reactivity model [5].

In this reactivity model, `HTS` is the additional layer of decoupling medium that accomplishes complete decoupling of agent interaction. `HTS` is used for storing two special tuples (viz. reaction tuple and response tuple). Reaction tuples are created from different parameters of invoked remote primitives, while response tuples are created from the result of execution of different remote primitives as well as while maintaining consistency in agent interaction. Reaction tuples and response tuples are both unordered tuples [10], and so their arity and nature of constituent fields vary with nature of invoked remote primitives. Reaction tuple is first inserted into `HTS` of reference host using `inject` primitive. On availability of target host (different from reference host), it is withdrawn from reference host's `HTS` using `eject`, passed over communication links to reach target host, and subsequently inserted into its `HTS`. Eventually, reaction tuple is withdrawn from target host's `HTS`, once desired target agent becomes available. It is processed next to extract parameters of invoked primitive, and execution of invoked remote primitive starts at `ATS` of target agent. In case of remote tuple-reading and -consuming primitives, target agent packs results of execution (viz. sought tuple(s) from `ATS` of target agent) and other necessary parameters into response tuple. Following previous approach, that response tuple eventually reaches reference agent, and sought tuple(s) are extracted from it. For achieving consistency in this asynchronous form of coordination, reference agent responds back with ACK tuple and NACK tuple when it has invoked any tuple-consuming primitives. ACK tuple positively acknowledges target agent about selection of its responded tuple as sought tuple, whereas NACK tuple returns non-selected responded tuple back to target agent. These special tuples are converted into response tuples before dispatch to target agents.

Additional Supporting Concepts. For execution over unreliable and dynamic underlying infrastructure, TSMM includes its own communication and discovery mechanisms that interfaces with transport service of corresponding device to achieve data transmission. Among the underlying infrastructure, this paper considers that Infrastructure Basic Service Set (iBSS) is deployed under TSMM. When deployed over iBSS, three categories of hosts are earmarked for TSMM, viz. *stationary host*, *mobile host* and *access point*. Stationary hosts are provided with only wired network connectivity, whereas mobile hosts are only having wireless network connectivity. Access point acts as a "mediator" either between a pair of mobile hosts, or between a mobile host and a stationary host, as it contains both wired and wireless network interfaces. Discovery mechanism furnishes an updated knowledge of available agents (along with their hosts) that are reachable from (i.e. neighbors of) reference host. This knowledge, utilized by other components of TSMM, is attained by sending and receiving beacons and is preserved in `NeighborList`. However, communication mechanism emphasizes on reliably transferring reaction/response tuples from one host to another. It uses additional acknowledgement mechanism to achieve this reliability. However, acknowledgement mechanism is only required when mobile hosts and their associated access point are communicating via wireless network interfaces.

System *TSMM*

Program $host(i)$ **at** λ

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

Program $agent(k)$ **at** λ

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

Components

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

Interactions

{Attach $\mathcal{T}_{\mathrm{W}}$ of all hosts with wired network interfaces as transiently-shared variable}

$shared_{\mathrm{W}}$ ::

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

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

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

$\rangle$

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

$[\!] \; shared_{\mathrm{WL}}$ ::

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

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

$\wedge \, (host(i)\Gamma' host(j))$

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

$\rangle$

{Prepare to register active agents in respective hosts}

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

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

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

end

Fig. 2. Mobile UNITY system of TSMM

3 Proposed Approach of Formalization of TSMM

This section proposes an approach of formalization of TSMM as a Mobile UNITY system, comprising of a set of formal programs representing different agents and hosts. Favoring Mobile UNITY over other formal tools is due to its suitability for formalizing inherently non-terminating programs (like mobile middleware), reasoning about agents temporal behavior using its proof rules, and following stepwise specification and refining. In **System** *TSMM*, as shown in Figure 2, several instances of two Mobile UNITY programs are components of whole system, and their interaction are specified in **Interactions** section. 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. Different conditions for two hosts or a host and an agent to interact in **Interactions** section are enforced through **when** clauses. **engage** and **disengage** clauses, and **current** construct are used for effecting transient sharing between different hosts. Also, first two statements in **Interactions** section, labeled as $shared_{\mathrm{W}}$ and $shared_{\mathrm{WL}}$, are reactive statements as they have used "$\approx$" notation [13].

Program $agent(k)$ **at** λ

declare

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

always

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

initially

$\lambda = \texttt{Location}(k)$ [] $TAs = 0$ [] $rform = 0$ [] $\mathbf{T} = \perp$ [] $ROL = \perp$ [] $RL = \perp$ [] $\mathbb{T} = \emptyset$
[] $t = \varepsilon$ [] $tuple = \varepsilon$ [] $\mathbf{t} = \emptyset$ [] $tuples = \emptyset$ [] $\mathbf{a} = \varepsilon$ [] $atuple = \varepsilon$ [] $\mathcal{Q}_{T^S_{a_k}} = \perp$ [] $\mathcal{Q}_{T^R_{a_k}} = \perp$
[] $UsrRdy4Evt = false$

assign

{Migrate to different location}
[] $\lambda := \texttt{Location}(\texttt{Move}())$

{Capture different parameters when user application is ready}
[] $\langle$ $prType, prName, UsrRdy4Evt := \texttt{getPrimType}(), \texttt{getPrimName}(), false$
$\|$ $prRdIn, prBulk := \texttt{getPrimRDorIN}(), \texttt{getPrimBulk}()$
$\|$ $tuple := \texttt{getTuple}()$ if $\big((prRdIn = false) \wedge (prBulk = false)\big)$
$\|$ $tuples := \texttt{getTuples}()$ if $\big((prRdIn = false) \wedge (prBulk = true)\big)$
$\|$ $atuple := \texttt{getAntiTuple}()$ if $(prRdIn = true)$
$\|$ $TAs := \texttt{getTargetAgentCount}()$ if $(prType = remote)$
$\|$ $\langle\| a : 1 \le a \le TAs :: taids[a] := \texttt{getTargetAgentID}(a)\rangle$ if $(prType = remote)$
$\|$ $mode := \texttt{getMode}(\texttt{ONCE}, \texttt{ONCE/TUPLE})$ if $\big((prType = remote) \wedge (prRdIn = true)\big)$
$\rangle$ if $(UsrRdy4Evt = true)$

{- - - - - - - - - - Start of Local Operation Manager - - - - - - - - - -}

{Perform different local tuple space primitives}
[] $\langle\ t, tuple, prType := tuple, \varepsilon, \varepsilon \parallel \texttt{out}(t, \mathbf{T})\ \rangle$ if $\big((prType = local) \wedge (prName = \mathsf{OUT}) \wedge \neg(tuple = \varepsilon)\big)$
[] $\langle\ \mathbf{t}, tuples, prType := tuples, \emptyset, \varepsilon \parallel \texttt{outg}(\mathbf{t}, \mathbf{T})$
$\rangle$ if $\big((prType = local) \wedge (prName = \mathsf{OUTG}) \wedge \neg(tuples = \emptyset)\big)$
[] $\langle$ $\mathbf{a}, atuple, prType := atuple, \varepsilon, \varepsilon$
$\|\ \langle\ t := \texttt{rdp}(\mathbf{a}, \mathbf{T}) \parallel \texttt{retTuple2Usr}(t)\ \rangle$ if $(prName = \mathsf{RDP})$
$\|\ \langle\ \mathbf{t} := \texttt{rdgp}(\mathbf{a}, \mathbf{T}) \parallel \texttt{retTuples2Usr}(\mathbf{t})\ \rangle$ if $(prName = \mathsf{RDGP})$
$\|\ \langle\ t := \texttt{inp}(\mathbf{a}, \mathbf{T}) \parallel \texttt{retTuple2Usr}(t)\ \rangle$ if $(prName = \mathsf{INP})$
$\|\ \langle\ \mathbf{t} := \texttt{ingp}(\mathbf{a}, \mathbf{T}) \parallel \texttt{retTuples2Usr}(\mathbf{t})\ \rangle$ if $(prName = \mathsf{INGP})$
$\rangle$ if $\big((prType = local) \wedge \neg(atuple = \varepsilon)\big)$

{- - - - - - - - - - End of Local Operation Manager - - - - - - - - - -}

Fig. 3. Mobile UNITY **Program** $agent(k)$: part 1

Fig. 4. Mobile UNITY **Program** $agent(k)$: part 2

Different agent behavior, including functionalities of ATS, Local Operation Manager, Remote Operation Manager, ATS Reaction Manager etc. are contained in $agent(k)$ as shown in Figure 3, Figure 4, and Figure 5. Similarly, functionalities of different components of host, including Transport Interface, Discovery Manager, Communication Manager, Host Server, Agent Manager etc., are contained in $host(i)$ as shown in Figure 6, Figure 7, Figure 8, Figure 9, and Figure 10. However, in above formal system, many aspects of TSMM are not directly formalized, to keep this formal system simple. Among these aspects, formalizing the mechanism to handle agent mobility (i.e. migration of agents from one host to another) is already shown in literature [14,9]. Also, correctness of above formal system (i.e. proving its safety/progress properties, and safety/progress properties of its individual components and of statements specified in **Interactions** section) is omitted here.

Fig. 5. Mobile UNITY **Program** $agent(k)$: part 3

Different variables pertaining to behavior of hosts and agents in TSMM are used in this formal system. For instance, $\mathcal{Q}$ is used to express any queue used to define different activities of TSMM; its subscripts represent purpose of using it. In this specification, $\texttt{head}(\mathcal{Q})$ returns front element of $\mathcal{Q}$, while $\texttt{tail}(\mathcal{Q})$ returns all elements of $\mathcal{Q}$ except front element. Also, $\mathcal{Q} \bullet M$ inserts M in the rear end of $\mathcal{Q}$ and returns updated $\mathcal{Q}$. Each message M comprises of message identity mid, source host's identity src, destination host's identity $dest$, type of message $kind$, data encapsulated within the message $data$, and network interface, ni, through which the message will be transmitted. M is generated by calling $\texttt{newMsg}(src, dest, kind, data, ni)$, which inserts its mid to return the complete message. Possible types of messages included in the specification are BCON, RT, ACK, Locate, and Found messages.

Program $host(i)$ **at** λ

declare

$type$: $\in\{stationary, mobile, accesspoint\}$
[] hid : hostid
[] $nwdeploy$: $\in\{iBSS, IBSS\}$
[] $status$: $\in\{standalone, associated\}$
[] $\mathbf{T}'$: tuple space
[] $\mathcal{Q}_{T^S_{a_k}}, \mathcal{Q}_{T^R_{a_k}}$: queue of RT_{tuple}
[] a : agentid
[] $\mathcal{A}$: 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)
[] $\mathcal{CS}$: message
[] $\mathcal{L}RT$: 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)
[] $\mathcal{Q}_{S_B}, \mathcal{Q}_{R_B}$: queue of message
[] $\mathcal{Q}_{S_{RT}}, \mathcal{Q}_{R_{RT}}$: queue of message
[] $\mathcal{Q}_{RT_S}, \mathcal{Q}_{RT_R}$: queue of RT_{tuple}
[] r : RT_{tuple}
[] $\mathcal{T}_{\text{W}}, \mathcal{T}_{\text{WL}}$: message
[] $\mathcal{Q}_{S_{\text{W}}}, \mathcal{Q}_{S_{\text{WL}}}$: queue of message
[] $\mathcal{Q}_S, \mathcal{Q}_R$: queue of message
[] M, m : message
[] $clock, lastHTSchk, lastRTsent, lastBsent, newRTGap, rtAtmpt$: natural

Fig. 6. Mobile UNITY **Program** $host(i)$: part 1

3.1 Formalization of *agent*(*k*)

Each agent is represented by program $agent(k)$, which comprises of ***declare***, ***always***, ***initially*** and ***assign*** sections. Agent behavior is specified by different variables that are declared in ***declare*** section. In particular, aid and $type$ are declared as agent identity and nature (viz. stationary agent/mobile agent) of any $agent(k)$. $\mathbf{T}$ is declared as `ATS` of $agent(k)$. Also, $prid$ is declared as identity of invoked primitive of $agent(k)$. ROL is declared as remote operation list of $agent(k)$, and RL is declared as reactive list of $agent(k)$. $\mathcal{Q}_{T^S_{a_k}}$ and $\mathcal{Q}_{T^R_{a_k}}$ are declared as queues to interface between agents and their supported hosts. These queues are defined to transfer request/response tuples from agents to hosts and vice versa. When user application is generating an event for any tuple space operation, corresponding agent must capture different parameters required to complete that operation. In the specification, readiness of user application is abstracted by $UsrRdy4Evt$, a boolean variable. Once user application is ready, capturing values of different parameters are specified by using different functions.

always

$B_{iBSS_{\mathrm{W}}} = \textbf{IBSSBroadcastAddress}_{\mathbf{DS}}$ [] $B_{iBSS_{\mathrm{WL}}} = \textbf{IBSSBroadcastAddress}_{\mathbf{BSA}}$
[] $B_{IBSS_{\mathrm{WL}}} = \textbf{IBSSBroadcastAddress}$
[] $\lambda := \texttt{Location}(i)$
[] $hid := \texttt{getMyHostID}(i)$
[] $type := \texttt{getHostType}(stationary, mobile, accesspoint)$
[] $nwdeploy := \texttt{getUnderlyingInfra}(iBSS, IBSS)$
[] $mhGap = \textsc{SystemMHValidityInterval}$ [] $HTSaccessGap = \textsc{SystemHTSAccessInterval}$
[] $locateGap = \textsc{SystemLocateMsgInterval}$ [] $bconGap = \textsc{SystemBeaconInterval}$
[] $mhGap = \textsc{SystemMHValidityInterval}$ [] $bLife = \textsc{SystemBeaconLifetime}$
[] $\texttt{isPresent}_{\mathcal{H}}(mhid) \equiv \langle\ \exists e : (e \in \mathcal{H}) \wedge (e \uparrow 1 = mhid)\ \rangle$
[] $\texttt{isPresent}_{\mathcal{L}}(mhid) \equiv \langle\ \exists e : (e \in \mathcal{L}) \wedge (e \uparrow 1 = mhid)\ \rangle$
[] $\texttt{isPresent}_{\mathcal{N}}(hostid) \equiv \langle\ \exists e : (e \in \mathcal{N}) \wedge (e \uparrow 1 = hostid)\ \rangle$
[] $\texttt{isPresent}_{\mathcal{LRT}}(hostid) \equiv \langle\ \exists e : (e \in \mathcal{LRT}) \wedge (e \uparrow 1 = hostid)\ \rangle$
[] $\texttt{isRepeat}_{\mathcal{LRT}}(hostid, msgid) \equiv \langle\ \exists e : (e \in \mathcal{LRT}) \wedge (e \uparrow 1 = hostid) \wedge (e \uparrow 2 = msgid)\ \rangle$
[] $\texttt{isValid}_{\mathcal{H}}(e, now) \equiv \big((e \in \mathcal{H}) \wedge ((now - e \uparrow 3) \leq mhGap)\big)$
[] $\texttt{isValid}_{\mathcal{L}}(e, now) \equiv \big((e \in \mathcal{L}) \wedge ((now - e \uparrow 3) \leq locateGap)\big)$
[] $\texttt{isValid}_{\mathcal{N}}(e, now) \equiv \big((e \in \mathcal{N}) \wedge ((now - e \uparrow 3) \leq e \uparrow 4)\big)$
[] $\texttt{isMsgBcon}(msg) \equiv (msg{\cdot}\, kind = Beacon)$
[] $\texttt{isMsgRT}(msg) \equiv (msg{\cdot}\, kind = RT)$
[] $\texttt{isMsgACK}(msg) \equiv (msg{\cdot}\, kind = ACK)$
[] $\texttt{isMsgLocate}(msg) \equiv (msg{\cdot}\, kind = Locate)$
[] $\texttt{isMsgFound}(msg) \equiv (msg{\cdot}\, kind = Found)$
[] $\texttt{isNotOwnMsg}(msg) \equiv \neg(msg{\cdot}\, src = hid)$
[] $\texttt{isSH}(host) \equiv (host{\cdot}\, type = stationary)$
[] $\texttt{isMH}(host) \equiv (host{\cdot}\, type = mobile)$
[] $\texttt{isAP}(host) \equiv (host{\cdot}\, type = accesspoint)$

initially

$clock = 0$ [] $lastHTSchk = 0$ [] $lastRTsent = 0$ [] $lastBsent = 0$
[] $status = standalone$ [] $assoc = \emptyset$ [] $\mathcal{H} = \emptyset$ [] $\mathcal{L} = \emptyset$ [] $\mathcal{LRT} = \emptyset$ [] $\mathcal{A} = \emptyset$ [] $\mathcal{N} = \emptyset$
[] $\mathbf{T}' = \perp$ [] $\mathcal{T}_{\mathrm{W}} = \perp$ [] $\mathcal{T}_{\mathrm{WL}} = \perp$ [] $\mathcal{CS} = \perp$
[] $\mathcal{Q}_{T^S_{a_k}} = \perp$ [] $\mathcal{Q}_{T^R_{a_k}} = \perp$ [] $\mathcal{Q}_{in} = \perp$ [] $\mathcal{Q}_{out} = \perp$ [] $\mathcal{Q}_{RT_S} = \perp$ [] $\mathcal{Q}_{RT_R} = \perp$
[] $\mathcal{Q}_{S_B} = \perp$ [] $\mathcal{Q}_{R_B} = \perp$ [] $\mathcal{Q}_{S_{RT}} = \perp$ [] $\mathcal{Q}_{R_{RT}} = \perp$ [] $\mathcal{Q}_{S_{\mathrm{W}}} = \perp$ [] $\mathcal{Q}_{S_{\mathrm{WL}}} = \perp$ [] $\mathcal{Q}_S = \perp$ [] $\mathcal{Q}_R = \perp$

Fig. 7. Mobile UNITY **Program** $host(i)$: part 2

3.2 Formalization of *host*(*i*)

Like $agent(k)$, $host(i)$ is also composed of ***declare***, ***always***, ***initially*** and ***assign*** sections. Different variables related to host behavior is declared in ***declare*** section. In particular, *hid* is declared as host identity of any $host(i)$, whereas *type* specifies nature of $host(i)$ (viz. stationary host/mobile host/access point). $\mathbf{T}'$ is declared as its `HTS`. $\mathcal{H}$ and $\mathcal{L}$ are declared for `History` (that records path of successful data transfer to different mobile hosts) and location list (that keeps mobile hosts with ongoing location search) respectively for $host(i)$ of stationary hosts and access points. Moreover, $\mathcal{LRT}$ and $\mathcal{CS}$ are declared for `LastRT` (that records message identity of last data messages received from different hosts) and `CommStash` (that buffers data messages) respectively of $host(i)$ of mobile hosts

assign

{Increment the clock}
[] $clock := clock + 1$

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

{Organize a message for onward transmission}
[] ⟨ $M, \mathcal{Q}_S := \texttt{head}(\mathcal{Q}_S), \texttt{tail}(\mathcal{Q}_S)$
‖ ⟨ $\mathcal{Q}_{S_{\mathrm{W}}} := \mathcal{Q}_{S_{\mathrm{W}}} \bullet M$ if $(M \cdot ni = \mathrm{W})$ ‖ $\mathcal{Q}_{S_{\mathrm{WL}}} := \mathcal{Q}_{S_{\mathrm{WL}}} \bullet M$ if $(M \cdot ni = \mathrm{WL})$ ⟩
⟩ if $\neg(\mathcal{Q}_S = \perp)$

{Transfer a message from $\mathcal{Q}_{S_{\mathrm{W}}}$ to $\mathcal{T}_{\mathrm{W}}$; make $\mathcal{T}_{\mathrm{W}}$ empty after some time}
[] $transmit\&reset_{\mathrm{W}}$ :: ⟨ $\mathcal{T}_{\mathrm{W}}, \mathcal{Q}_{S_{\mathrm{W}}} := \texttt{head}(\mathcal{Q}_{S_{\mathrm{W}}}), \texttt{tail}(\mathcal{Q}_{S_{\mathrm{W}}})$ if $\neg(\mathcal{Q}_{S_{\mathrm{W}}} = \perp) \wedge (\mathcal{T}_{\mathrm{W}} = \perp)$;
$\mathcal{T}_{\mathrm{W}} := \perp$ ⟩

{Transfer a message from $\mathcal{Q}_{S_{\mathrm{WL}}}$ to $\mathcal{T}_{\mathrm{WL}}$; make $\mathcal{T}_{\mathrm{WL}}$ empty after some time}
[] $transmit\&reset_{\mathrm{WL}}$ :: ⟨ $\mathcal{T}_{\mathrm{WL}}, \mathcal{Q}_{S_{\mathrm{WL}}} := \texttt{head}(\mathcal{Q}_{S_{\mathrm{WL}}}), \texttt{tail}(\mathcal{Q}_{S_{\mathrm{WL}}})$ if $\neg(\mathcal{Q}_{S_{\mathrm{WL}}} = \perp) \wedge (\mathcal{T}_{\mathrm{WL}} = \perp)$;
$\mathcal{T}_{\mathrm{WL}} := \perp$ ⟩

{Transfer a message from $\mathcal{T}_{\mathrm{W}}$ to $\mathcal{Q}_R$}
[] ⟨ $\mathcal{Q}_R := \mathcal{Q}_R \bullet \mathcal{T}_{\mathrm{W}}$ if $\texttt{isNotOwnMsg}(\mathcal{T}_{\mathrm{W}})$ ⟩ reacts-to $\neg(\mathcal{T}_{\mathrm{W}} = \perp)$

{Transfer a message from $\mathcal{T}_{\mathrm{WL}}$ to $\mathcal{Q}_R$}
[] ⟨ $\mathcal{Q}_R := \mathcal{Q}_R \bullet \mathcal{T}_{\mathrm{WL}}$ if $\texttt{isNotOwnMsg}(\mathcal{T}_{\mathrm{WL}})$ ⟩ reacts-to $\neg(\mathcal{T}_{\mathrm{WL}} = \perp)$

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

{Organize a received Beacon/RT/ACK/Locate/Found message for further processing}
[] ⟨ $M, \mathcal{Q}_R := \texttt{head}(\mathcal{Q}_R), \texttt{tail}(\mathcal{Q}_R)$
‖ ⟨ $\mathcal{Q}_{R_B} := \mathcal{Q}_{R_B} \bullet M$ if $\texttt{isMsgBcon}(M)$
‖ $\mathcal{Q}_{R_{RT}} := \mathcal{Q}_{R_{RT}} \bullet M$ if $\texttt{isMsgRT}(M) \vee \texttt{isMsgACK}(M) \vee \texttt{isMsgLocate}(M) \vee \texttt{isMsgFound}(M)$
⟩
⟩ if $\neg(\mathcal{Q}_R = \perp)$

Fig. 8. Mobile UNITY **Program** $host(i)$: part 3

and access points. Also, $\mathcal{N}$ and $\mathcal{A}$ are declared to represent `NeighborList` and `AgentList` respectively of any host. Different macros related to various aspects of discovery and communication mechanisms, used in this specification, are skipped in this paper for space limitations.

At the lowest level, TSMM interacts with transport service of supporting device, which is formalized as Transport Interface by a set of assignment statements. Discovery Manager and Communication Manager interchange messages with Transport Interface through two different queues, viz. $\mathcal{Q}_S$ and $\mathcal{Q}_R$. Behavior of Discovery Manager and Communication Manager are abstracted according to the nature of host, which is subscripted in corresponding macro. These macros are, in turn, used in different assignment statements to complete various functionalities of Discovery Manager and Communication Manager. Host Server interchanges request/response tuples (represented as $\mathrm{RT}_{\mathrm{tuple}}$) with Communication Manager through $\mathcal{Q}_{RT_S}$ and $\mathcal{Q}_{RT_R}$, which is formalized via a set of assignment statements. Similarly, in this specification, a pair of assignment statements formalizes registration/deregistration functionalities of Agent Manager.

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

{Prepare to send Beacon message to destination}

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

$\|\ \ \mathcal{Q}_{S_B}, lastBsent := \mathcal{Q}_{S_B} \bullet \texttt{discSend}_{\texttt{WL}_{iBSS}}(), clock$ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = iBSS)\big)$

$\|\ \ \mathcal{Q}_{S_B}, lastBsent := \big(\mathcal{Q}_{S_B} \bullet \texttt{discSend}_{\texttt{W}_{iBSS}}()\big) \bullet \texttt{discSend}_{\texttt{WL}_{iBSS}}(), clock$ if $\big(\texttt{isAP}(hid) \wedge (nwdeploy = iBSS)\big)$

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

{Process received Beacon message}

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

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

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

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

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

$[]\ \texttt{discValid}_{\mathcal{N}_{iBSS}}()$ if $\big((\texttt{isSH}(hid) \vee \texttt{isMH}(hid) \vee \texttt{isAP}(hid)) \wedge (nwdeploy = iBSS)\big)$

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

$[]\ \langle\ \texttt{discUpdt}_{\texttt{MH}_{iBSS}}()$ if $\big(\texttt{isMH}(hid) \wedge (nwdeploy = iBSS)\big)\ \rangle$

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

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

{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 $\neg(\mathcal{Q}_{S_B} = \perp)$

{- - - - - - - - - - Start of Host Server - - - - - - - - - -}

{Process received RT from different agents}

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

{Process received RT from COMMUNICATION module}

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

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

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

$\|\ \langle \| e : (e \in \mathcal{N}) \wedge (\mathrm{A} = e \uparrow 2) :: \langle \| a : a \in \mathrm{A} :: r := \texttt{eject}(a, \mathsf{T}') \parallel \langle \mathcal{Q}_{RT_S} := \mathcal{Q}_{RT_S} \bullet r$ if $\neg(r = \varepsilon)\rangle\ \rangle\ \rangle$

$\|\ lastHTSchk := clock$

$\rangle$ if $(clock - lastHTSchk > HTSaccessGap)$

{- - - - - - - - - - End of Host Server - - - - - - - - - -}

Fig. 9. Mobile UNITY **Program** $host(i)$: part 4

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

{Prepare to send RT/Locate message to destination}

$[\!]\ \langle\ \texttt{commSend}_{\text{SH}_{iBSS}}(\mathcal{Q}_{RT_S})$ if $(\texttt{isSH}(hid) \wedge (nwdeploy = iBSS))$
$\|\ \texttt{commSend}_{\text{MH}_{iBSS}}(\mathcal{Q}_{RT_S})$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = iBSS))$
$\|\ \texttt{commSend}_{\text{AP}_{iBSS}}(\mathcal{Q}_{RT_S})$ if $(\texttt{isAP}(hid) \wedge (nwdeploy = iBSS))$
$\rangle$ if $\neg(\mathcal{Q}_{RT_S} = \perp)$

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

$[\!]\ \langle\ \texttt{commRcv}_{\text{SH}_{iBSS}}(\mathcal{Q}_{R_{RT}})$ if $(\texttt{isSH}(hid) \wedge (nwdeploy = iBSS))$
$\|\ \texttt{commRcv}_{\text{MH}_{iBSS}}(\mathcal{Q}_{R_{RT}})$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = iBSS))$
$\|\ \texttt{commRcv}_{\text{AP}_{iBSS}}(\mathcal{Q}_{R_{RT}})$ if $(\texttt{isAP}(hid) \wedge (nwdeploy = iBSS))$
$\rangle$ if $\neg(\mathcal{Q}_{R_{RT}} = \perp)$

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

$[\!]\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{commReSend}_{\text{RT}_{iBSS}}()$ if $((\texttt{isMH}(hid) \vee \texttt{isAP}(hid)) \wedge (nwdeploy = iBSS))$
$\rangle$ if $((clock - lastRTsent) > newRTGap)$

{Process RT message whose destination is presently not available}

$[\!]\ \langle\ \langle\ \mathcal{Q}_{RT_R} := \mathcal{Q}_{RT_R} \bullet \mathcal{CS}.data\ \|\ \mathcal{CS} := \perp\ \rangle$ if $(\texttt{isMH}(hid) \wedge (nwdeploy = iBSS))$
$\|\ \langle\ \mathcal{Q}_{S_{RT}} := \mathcal{Q}_{S_{RT}} \bullet \texttt{newMsg}(hid, B_{iBSS_{\text{W}}}, Locate, \mathcal{CS}.dest, \text{W})$
$\|\ \mathcal{L} := \mathcal{L} \cup \{(\mathcal{CS}.dest, \mathcal{CS}.data, clock)\}\ \rangle$ if $(\texttt{isAP}(hid) \wedge (nwdeploy = iBSS))$
$\rangle$ if $((\neg(\mathcal{CS} = \perp) \wedge (rtAtmpt > 3))$

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

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

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

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

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

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

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

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

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

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

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

end

Fig. 10. Mobile UNITY **Program** $host(i)$: part 5

4 Conclusion

This paper has proposed an approach of formalization of a TSMM, which incorporates a fully-decoupled reactive tuple space model, using Mobile UNITY. It has been formally specified as a Mobile UNITY system, which is comprised of components representing different behaviors of agents and hosts of TSMM.

References

1. Bruneo, D., Puliafito, A., Scarpa, M.: Mobile Middleware: Definition and Motivations. In: Bellavista, P., Corradi, A. (eds.) The Handbook of Mobile Middleware, pp. 145–167. Auerbach Pub. (2007)
2. Gelernter, D.: Generative Communication in Linda. Transactions on Programming Languages and Systems 7(1), 80–112 (1985)

3. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of Publish/Subscribe. Computing Surveys 35(2), 114–131 (2003)
4. Cabri, G., Ferrari, L., Leonardi, L., Mamei, M., Zambonelli, F.: Uncoupling Coordination: Tuple-Based Models for Mobility. In: Bellavista, P., Corradi, A. (eds.) The Handbook of Mobile Middleware, pp. 229–255. Auerbach Pub. (2007)
5. De, S., Nandi, S., Goswami, D.: Modeling an Enhanced Tuple Space based Mobile Middleware in UNITY. In: Proc. 11th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2012 (June 2012)
6. Roman, G.C., McCann, P.J., Plun, J.Y.: Mobile UNITY: Reasoning and Specification in Mobile Computing. Transactions on Software Engineering and Methodology 6(3), 250–282 (1997)
7. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading (1988)
8. Murphy, A.L., Picco, G.P., Roman, G.C.: Lime: A Coordination Model and Middleware supporting Mobility of Hosts and Agents. Transactions on Software Engineering and Methodology 15(3), 279–328 (2006)
9. Roman, G.-C., Payton, J.: Mobile UNITY Schemas for Agent Coordination. In: Börger, E., Gargantini, A., Riccobene, E. (eds.) ASM 2003. LNCS, vol. 2589, pp. 126–150. Springer, Heidelberg (2003)
10. De, S., Nandi, S., Goswami, D.: On Performance Improvement Issues in Unordered Tuple Space based Mobile Middleware. In: Proc. 2010 Annual IEEE India Conference, INDICON 2010 (December 2010)
11. Gelernter, D., Bernstein, A.J.: Distributed Communication via Global Buffer. In: Proc. 1st Symp. on Principles of Distributed Computing (PODC 1982), pp. 10–18 (August 1982)
12. Denti, E., Natali, A., Omicini, A.: On the Expressive Power of Language for Programming Coordination Media. In: Proc. Symposium on Applied Computing (SAC 1998), pp. 169–177 (August 1998)
13. McCann, P.J., Roman, G.C.: Compositional Programming Abstractions for Mobile Computing. Transactions on Software Engineering 24(2), 97–110 (1998)
14. Picco, G.P., Roman, G.C., McCann, P.J.: Reasoning about Code Mobility with Mobile UNITY. Transactions on Software Engineering and Methodology 10(3), 338–395 (2001)