Intelligent Automation And Soft Computing, 2019

Vol.25,no.1,49–63

Formal Modelling of Real-Time Self-Adaptive Multi-Agent Systems

Awais Qasim[a], Syed Asad Raza Kazmi[b]

[a,b]Department of Computer Science, Government College University Lahore, Pakistan

ABSTRACT

The paradigm of multi-agent systems is very expressive to model distributed real-time systems. These real-time multi-agent systems by their working nature have temporal constraints as they need to operate in pervasive, dynamic and unpredictable environments. To achieve better fault-tolerance, they need to have the ability of self-adaptivity making them adaptable to the failures. Presently there is a lack of vocabulary for the formal modelling of real-time multi-agent systems with self-adaptive ability. In this research we proposed a framework named SMARTS for the formal modelling of self-adaptive real-time multi-agent systems. Our framework integrates MAPE-K interfaces, reflection perspective and unification with distribution perspective into the SIMBA agent architecture. For a precise semantic description of the constructs of our framework, we have used Timed Communicating Object-Z language.

KEY WORDS: Autonomic computing, Formal methods, Multi-agent systems, Real-time systems, Self-adaptation, TCOZ.

1 INTRODUCTION

FORMAL modelling is the process of constructing mathematical model of a software and hardware system at some level of abstraction. Its basic purpose is to utilize the unambiguous notation of formal methods to precisely specify and analyze the system, leading to its enhanced understanding. In this work we concentrate on the formal modelling of multi-agent systems with critical temporal constraints and self-adaptive functionality. These real-time multi-agent systems by their working nature have high degree of unpredictability in their execution context and require extensive analysis at the design time before their implementation to ensure their correct functioning. This dynamism of real-time multi-agent systems has led to a new category of software systems called real-time self-adaptive multi-agent systems. These systems possess the necessary knowledge to adapt their behavior in real-time in response to the environmental context. A computer software system that works autonomously in an environment to achieve its objectives can be categorized as an agent according to Jennings, et. al. (1998). These agents are expected to face unpredictable situations with their confined knowledge. In real-time environments these agents need to work collaboratively to achieve the common objective with minimum communication between them for timely response. A Real-Time Agent (RTA) is an agent working with hard temporal constraints. The correct functioning of these RTAs does not solely depend on whether they complete the task, rather it depends on whether they complete the task within the deadline or not. Previously these RTAs have been classified as hard real-time agents and soft real-time agents in Julian and Botti (2004). In soft real-time agents there is a slight marginal period for the fulfillment of their temporal restrictions. A Real-Time Multi-Agent system (RTMAS) is basically a multi-agent system where at least one of the agent is real-time in nature. Presently real-time systems like mobile robots, online auction systems, intrusion detection systems, control processes, etc are effectively modelled by multi-agent systems paradigm. With the ability of self-adaptation a multi-agent system possesses the necessary knowledge to re-organize itself according to the changes in environment and user requirements leading to better fault tolerance in case of node failure. In (Tesar (2016); Nair, et. al. (2015); De Lemos, et. al. (2013)) it has been argued that the development of autonomous physical systems with real-time constraints is a challenging task. With self-adaptive real-time multi-agent systems, the problem is more challenging as these systems have

CONTACT Awais Qasim awais@gcu.edu.pk

tight temporal constraints. Self-Adaptation should ensure that the overall system's functionality is not effected. Formal modelling of these real-time multi-agent systems will increase the confidence in the correctness of such systems.

In the past formal specification and verification of these multi-agent systems have been done extensively but not for self-adaptive real-time multi-agent systems. Multi-agent systems have been formally specified in Li and Miao (2015) using Object-Z with trace semantics. They focused on the safety and liveness properties of the system. In Wooldridge (2000) verifiable semantics based on computational models have been proposed for e-agents communication. In Guerin (2002) a framework for communication among e-agents was presented. In particular, they used different languages for the specification of communication among e-agents. Herrero, et. al. (2013) have proposed a real-time multi-agent architecture for Intrusion Detection System called RT-MOVICAB-IDS. Their architecture ensures that the agent's response (reflex or deliberative) conforms to temporal constraints of the system in case of an intrusion. In Guo and Dimarogonas (2015) a cooperative motion and task planning scheme for multi-agent systems has been proposed. According to their scheme the agent's tasks, categorized with hard or soft deadlines are specified as Linear Temporal Logic formulas. The tasks with hard temporal constraints are always executed within the deadline and the agent tries to improve the result for soft deadline. El Kholy, et. al. (2015) presented an extension of Computation Tree Logic called RTCTLcc for the specification of real-time properties of multi-agent systems. They argued that RTCTLcc can be used to formally model the interaction among agents with temporal constraints. Weyns and Calinescu (2015) have proposed a Tele Assistance System (TAS) to compare the effectiveness of different solutions for self-adaptive systems. Its another purpose is to enhance understanding among researchers in the domain of self-adaptive systems. In Johnson, et. al. (2015) a framework for component-based multi-agent systems has been proposed for their formal verification. Major entity of their approach is Agent Verification Engine (AVE). AVE handles the Belief-Desire-Intention (BDI) agents to verify the complete system after a component has been added/removed from the system. Kl.s, et. al. (2015) proposed an extended MAPE-K feedback loop to deal with uncertainty in self-adaptive systems with predefined set of rules. Their approach requires a structured knowledge base comprising a global goal model, environment model, abstract system and current adaptation rules. With their approach new rules are generated at run-time and the adaptation logic itself becomes dynamic. Gascue, et. al. (2012) have integrated Model-driven engineering and agent-oriented software development for the development of multi-agent systems. Ntika, et. al. (2014) have formally modelled autonomous multi-agent systems for the targeted drug delivery. They simulated the nanorobots using agent technology to demonstrate the future possibilities of drug delivery system using multi-agent systems. Graja, et. al. (2014) presented a Event-B based formal modeling technique for multi-agent systems with self-organizing ability. Their technique uses a step-wise refinement for individual agent's behavior, which in turn enable the verification of properties for the complete system. In Webster, et. al. (2014) a model checking technique for formally verifying the scheduling activities of robotic assistants for humans has been proposed by translating the requirements to the model checker's language i.e PROMELA. A multi-agent system's approach with self-adaptive ability for collaborative mobile learning has been proposed in de la Iglesia, et. al. (2015). They discussed the issues of node failure and resource availability due to system dynamism and environment and how to tackle it with self-adaptation. Mao, et. al. (2014) presented a two-layer approach for the development of self-adaptive multi-agent system in open environment. The two layers corresponds to self-adaptation at the agent's behavior level and agent' organization level. Bonnet, et. al. (2015) proposed a self-adaptive multi-agent system for solving the satellite mission planning. They focused collective planning among agents for effective load balancing. Guo, et. al. (2013) have used self-adaptive multi-agent systems for solving the car pooling problem. Their approach is based on dynamic heuristics for the systems learning process. Self-adaptive multi-agent system's and their interaction has been discussed in (Sanderson, et, al. (2013); Shan, et. al. (2015)). (Mac.as-Escriv, et. al. (2013); Abbas, et. al. (2015); Krupitzer, et. al. (2015); Baresi (2014); Puviani, et. al. (2015); Weyns and Andersson, (2013)) has presented an in depth analysis of the challenges, application and approaches of self-adaptive systems. Qasim, et. al. (2015a, b) have used mu-calculus and Timed-Arc Petri-nets for the formal specification and verification of multi-agent systems. In Weyns, et. al. (2012) a framework for the formal modelling of distributed self-adaptive systems has been proposed called FORMS, which provides different modelling elements and a set of relationships guiding the design of self-adaptive software systems. (Qureshi, et. al. (2015); Alrashed, et. al. (2016)) have discussed a technique for efficient scheduling of real-time systems. In Iglesia and Weyns (2015) MAPE-K feedback loop based formal templates are specified in timed automata and targets the behavior and property aspects of self-adaptive systems. Weyns, et. al. (2014) have comprehensively analyzed the self-adaptive software systems for their run-time assurances regarding the domain functionality under unpredictable situations.

Abbas, et. al. (2016) proposed an extension of the Architectural Reasoning Framework (eARF) to ensure that the design complies to the requirements. It has been argued in (Schaefer and Hahnle (2011); Gil de La Iglesia (2014)) that the formal methods should be used for the automated verification of safety critical and real-time systems to ensure their correct functioning.

However very limited work has been done for the formal modelling of self-adaptive real-time multi-agent systems. For complex systems, their formal specification is devised at the conceptual design phase before the system is implemented. Such specifications describe the semantics of the system being implemented without the concern for the implementation details and can be used as a basis for the verification and validation of the system's functionality. Presently there is a lack of vocabulary for the formal modelling of real-time multi-agent systems with self-adaptive ability that will be expressive enough to capture their key architectural characteristics. In this paper we have proposed a framework for the formal modelling of real-time multi-agent systems named SMARTS (Self-adaptive Multi-Agent Real-Time Systems). Our framework makes use of the SIMBA agent architecture as proposed in Julian, et. al. (2002) and FORMS reference model for adaptation as proposed in Weyns, et. al. (2012). Basic agents of SMARTS are ARTIS agents, which will provide the system's domain functionality. ARTIS agents have been designed to work in dynamic environments with temporal constraints. The proposed framework can be used for the architectural specification of any self-adaptive real-time multi-agent system. Our framework is FIPA (Foundation for Intelligent Physical Agents)-compliant and supports communication with agents of other platform. FIPA works for the standardization of inter-agent and intra-agent communication for the multi-agent systems paradigm. A system's designer should thus decide which features the ARTIS agents are going to have because communication with other agent platforms may prevent this real-time behavior. For readability we elaborate SMARTS framework using UML notations in this paper. For a precise semantic description of the constructs we use Timed Communicating Object-Z (TCOZ) language.

The rest of this paper is divided as follows. In section 2 some preliminaries for entity descriptions of SIMBA agent architecture are explained. Section 3 describes the proposed SMARTS framework. In section 4 we formally specify the SMARTS framework using TCOZ. Section 5 concludes the paper.

2 SIMBA AGENT ARCHITECTURE

SIMBA agent architecture was proposed for RTMAS in Julian, et. al. (2002). It is an extension of the ARTIS agent architecture proposed in Botti, et. al. (1999). Basically this architecture is formed by the collaboration of multiple ARTIS agents deployed in an environment with strict temporal restrictions. Major advantage of SIMBA system is that we can have different types of agents in our system to handle the non-critical activities. This architecture guarantees that all the agents will meet their temporal constraints by the use of an off-line schedulability analysis. Each ARTIS agent has a number of In-agents (internal agents) which actually executes the tasks to achieve its goals. An In-agent is another agent possessing the necessary knowledge to solve a specific problem. These In-agents periodically performs a particular task. In-agents are classified as critical or non-critical depending on their temporal restrictions. Every In-agent has two layers namely reflex layer and real-time deliberative layer. The reflex layer only provides a minimal quality response and the deliberative layer provides an improved response. The mandatory phase of an ARTIS agent consists of reflex layers of all the In-agents it has. Similarly, the real-time deliberative layers of all the In-agents makes up the optional phase of an ARTIS agent. A reflex layer is absent in a non-critical In-agent and only the real-time deliberative layer is present. There is a Control Module in every ARTIS agent that controls the execution of all the In-agents that belongs to it. Major advantage of SIMBA architecture is that it is FIPA-compliant and allows communication between diverse agent platforms. FIPA-compliance requires an agent architecture to at least implement the Agent Communication Language specification and Agent Management specification. The Agent Management specification requires an agent architecture to implement a Directory Facilitator (DF) and an Agent Management System (AMS). DF will provide yellow-pages service to the agents involved. AMS will maintain the addresses of all the agents registered in the platform (white-pages service). The agent communication language specification enforces a standard message format to be used in the agent communication. Within every SIMBA architecture there is a single SIMBA communicator agent, which serves as a mediator agent for inter and intra-platform agent communication.

3 PROPOSED SMARTS FRAMEWORK

AN overview of the SMARTS's primitives is provided in Figure 1. A TCOZ specification of the framework is provided in the next section. The shaded entities are part of the SMARTS framework whereas the un-shaded entities are taken from the FORMS reference model Weyns, et. al. (2012). As shown in the figure a distributed self-adaptive system is a self-

adaptive system comprising one or more local self-adaptive multi-agent systems. A local self-adaptive multi-agent system comprises multiple local managed systems, self-adaptive units and a single SIMBA communicator agent. A local managed system provides domain functionality of the system in the form of ARTIS agents. An ARTIS agent corresponds to single local managed system. A local managed system is a subsystem comprising multiple In-agents, a set of domain models, a single dynamic agent organization and a single control module. The entities Monitor Agent, Analyze Agent, Plan Agent and Execute Agent corresponds to the MAPE-K interfaces that we have proposed in Qasim and Kazmi (2016).

A model is a representation describing entities of interest in the physical or conceptual world. A domain model contains representations of entities necessary for the provision of required features. An environment comprises attributes and processes and corresponds to both conceptual and physical entities. An attribute is any observable characteristic of the environment. A process represents any activity that can modify the attributes of the environment. As described previously an In-agent is an agent configured to solve a particular problem. It periodically performs a specific task. An ARTIS agent will autonomously work in an environment and execute the required tasks with the help of In-agents by reading from and writing to the domain model. A self-adaptive unit is a subsystem responsible for implementing the adaptation process in a computing system. It can manage multiple local managed systems and self-adaptive units. It will manage another self-adaptive unit in case of more than one reflective levels in the system. It comprises multiple reflection models and reflective computations. A reflection model is similar to the meta-data and provides concrete instances of entities needed for the adaptation. It corresponds to the architectural models of the system. A reflective computation is analogous to an In-agent but it acts and reasons about the reflection models. It is responsible for environment monitoring to determine the required adaptations. A reflective computation is not capable of changing the environment directly and needs support from the other agents for doing so. A local reflective computation is basically a reflective computation with coordination mechanism. The coordination mechanism provides an ability so that the agents providing domain functionality and the local reflective computations can coordinate with other agents in the same layer.

SMARTS's coordination mechanism consists of a coordination protocol and a coordination model. A coordination model contains information like coordination partners, their roles, ongoing interaction's information, entity representations needed by the local reflective computation to communicate with other self-adaptive unit's reflective computations. The coordination protocol is a set of rules for governing the communication among the entities. For communication within the system, TCOZ channel communication will be utilized. The channel will be an abstraction for communication (message exchange or shared tuple spaces) between entities. The dynamic agent organization manages groups of the agents in the form of master/slave relationship. An agent in the role of master will manage the dynamics of its organization by communicating with all the slave/master agents of neighboring organizations. By default, an ARTIS agent is member of a single organization. Multiple ARTIS agents can however merge in one organization depending upon circumstances.

4 TCOZ SPECIFICATION OF SMARTS FRAMEWORK

AN environment will keep on updating itself after certain time units. With each update event the new attributes are available at the output channel. We formally specify the environment entity below. Concrete models can have different types of representations. We specify the models as generic constant. In domain model the EnvironmentRepresentation specifies the attributes of the environment.

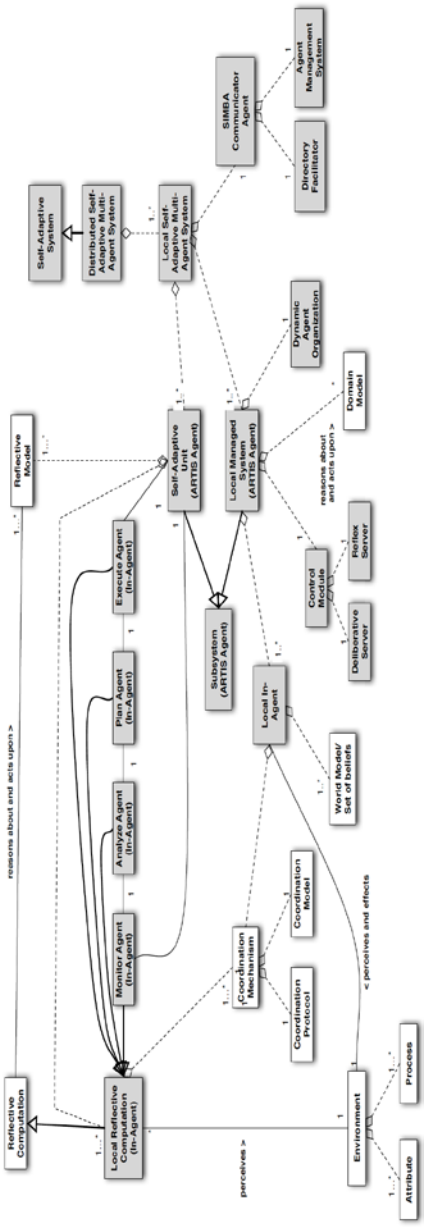

Figure 1. **Proposed SMARTS Framework.**

Environment

attributes : $\mathbb{P}$ *Attribute*

processes : $\mathbb{P}$ *Process*

c : ***chan***

INIT

$attributes \neq \varnothing$

$processes \neq \varnothing$

Update

$\Delta attributes$

$\Delta processes$

$c! = attributes$

$MAIN \triangleq \mu T \bullet (Update \bullet WaitUntil\ 1s); T$

Model [*Representation*]]

rep : $\mathbb{P}$ *Representaion*

$rep \neq \varnothing$

Domain model is defined as a passive class. The channel c is used for communication with any active class.

DomainModel

Model[EnvironmentRepresentation]

map : $\mathbb{P}$ *Attribute* $\leftrightarrow$ *EnvironmentRepresentation*

envr : *chan*

$dom\ map \subseteq \{ attrs : \mathbb{P}\ Attribute \mid attrs \subseteq envr?.attributes\}$

$ran\ map = \{ r : EnvironmentRepresentation \mid r \in rep\}$

We specify a passive class named *Task* to represent any task in the system. Each task requires a single resource for certain duration without which it cannot execute. For brevity we have only handled the case of one resource per task but the approach can be extended for multiple resources per task. The

ReflexExecute operation models the execution of a task when the agent executing it does not have extra time to improve the result. The *DeliberativeExecute* models the execution of a task when the task has soft deadline. The variable *length* represents the expected amount of time, which the task will take to execute. In SMARTS *length* is considered as the deadline before which the task should have been executed. *margin* represents additional time for soft deadline. In case a margin is available for a task then the *DeliberativeExecute* process will be executed.

TaskType ::= REFLEX | DELIBERATIVE
AgentType ::= INAGENT | ARTISAGENT

Task

resource : Resource
length : T
margin : ℕ

INIT
length > 0

ReflexExecute
Δlength
δ = length
length′ = 0

DeliberativeExecute
Δlength
δ = length + margin
length′ = 0

Each agent can be uniquely identified by an *id*. We define the type id as
ID == N

No In-agent can occupy more than one resource at given anytime. An execution of a task necessitates that the In-agent executing it has the required resource for the needed duration. This condition is specified as a predicate in task execution. An In-agent might get multiple requests to solve the same type of tasks. To ensure their autonomous working every In-agent is specified as an active object. The set *tasks* represent those tasks for which the In-agent has been designed. The set *agents* represent those In-agents which come under the same ARTIS agent. The *alloc* maps resources to the In-agents. Since an active class can only communicate with other active classes via channels so an In-agent has a channel *c* to communicate with the ARTIS agent. The *id* represents unique identifier of the agent.

InAgent

id : ℕ
tasks : ℙ *task*
alloc : (*Resource* × *T*) → *InAgent*
c : ***chan***
margin : ℕ
state : ℙ *State*
type : *AgentType*

∀ *r*1 , *r*2 : *Resource*; *t* : *T r*1 ≠ *r*2 •
alloc (*self* , *r*1 , *t*) ≠ *alloc* (*self* , *r*2, *t*)

INIT
type = INAGENT
request = ∅

CanExecuteDeliberately
task? : *Task*
task? ∈ *tasks* ∧
∃ *currentTime == clock.time; l ==*
task?.length; r == task?.resource •
currentTime..currentTime + l + margin
⊆ {*t* : *T* | *alloc*(*r* , *t*) = *self* }

ExecuteTask ≙ [*task? : Task*] •
((CanExecuteDeliberately ∧
task?.DeliberativeExecute)
□*(!CanExecuteDeliberately* ∧
task?.ReflexExecute))
MAIN ≙ *μT* • [*tasks* ≠∅] •
ExecuteTask; T

The DF provides yellow-pages service to the platform agents. Agents of the platform register with the DF. This allows one agent to trace other agents that provide the services it requires. The method *RegisterARTISAgent* will register an ARTIS agent with the DF. There is only one DF in each platform. We define a *service* as an entity having name and type only.

ServiceDescription

INIT

$type \neq \varnothing$

$name \neq \varnothing$

$type : \mathbb{N}$

$name : \mathbb{N}$

DirectoryFacilitator

$inagents : \mathbb{P}\ InAgent$

$artisagents : \mathbb{P}\ ARTISAgent$

$services : \mathbb{P}\ ServiceDescription$

$c : chan$

Init

$requests = \varnothing$

RegisterInAgent'

$agent? : InAgent$

$services? : \mathbb{P}\ ServiceDescription$

$inagents' = inagents \cup \{agent\}$

$services' = services \cup \{agent.services\}$

RegisterARTISAgent'

$agent? : ARTISAgent$

$services? : \mathbb{P}\ ServiceDescription$

$artisagents' = artisagents \cup \{agent\}$

$services' = services \cup \{agent.services\}$

$PublishServices \triangleq c!services \rightarrow SKIP$

$SearchService \triangleq [name : N; type : \mathbb{N}\ ] \bullet$

$(\exists\ s : services \mid s.name = name \wedge$

$s.type = type) \bullet c!s \rightarrow SKIP$

$RegisterInAgent \triangleq [agent? : InAgent;$

$services? : \mathbb{P}\ ServiceDescription] \bullet$

$RegisterInAgent' \rightarrow SKIP$

$RegisterARTISAgent \triangleq [agent? :$

$ARTISAgent; services? : \mathbb{P}$

$ServiceDescription] \bullet$

$RegisterARTISAgent' \rightarrow SKIP$

$MAIN \triangleq \mu\ T \bullet (PublishServices \Box$

$SearchService \Box RegisterInAgent \Box$

$RegisterARTISAgent); T$

AMS is responsible for providing the white-pages service and maintains the identifiers of all the agents in the platform. It ensures that each agent in the platform has a unique identifier. It performs several management tasks as well, such as creating and destroying agents. There is only one AMS in each platform and all the agents must be registered in the AMS. The operation *AddARTISAgent* and *AddInAgent* will create an instance of a ARTIS agent and In-agent respectively with unique identifiers. The *lsamas* represents a channel with which the AMS will communicate with *LocalSelfAdaptiveMultiAgentSystem*. The method *CreateNewAgent* takes as input the agent type and creates an instance with unique identifier. The method *RemoveAgentByIdentifier* removes an agent from the platform by taking the agent identifier and type.

AgentManagementSystem

inagents : InAgent
artisagents : ARTISAgent
lsamas : ***chan***

AddARTISAgent′
agent : ARTISAgent
agent : CreateNewAgent[type = ARTIS]
artisagents′ = artisagents ∪ {agent}
#artisagents′ = (# artisagents + 1)

AddInAgent′
agent : InAgent
agent : CreateNewAgent[type = INAGENT]
inagents′ = inagents ∪{agent}
#inagents′ = (# inagents + 1)

RemoveARTISAgent′
id? : ℕ
RemoveAgentByIdentifier[type = ARTIS, id]
#artisagents′ = (#artisagents - 1)

RemoveInAgent′
id? : ℕ
RemoveAgentByIdentifier[type = INAGENT, id]
#inagents′ = (#inagents - 1)

RemoveARTISAgent ≙ [id? : N, result : BOOL] □ RemoveARTISAgent′ → lsamas!(result) → SKIP
RemoveInAgent ≙ [id? : N, result : BOOL] □ RemoveInAgent′ → lsamas!(result) → SKIP
AddARTISAgent ≙ [agent! : ARTISAgent] □ AddARTISAgent′ → lsamas!(agent) → SKIP
AddInAgent ≙ [agent! : AddInAgent] □ AddInAgent′ → lsamas!(agent) → SKIP
MAIN ≙ μ T □ (RemoveARTISAgent □ RemoveInAgent □ AddARTISAgent □ AddInAgent); T

SIMBA communicator agent is a mediator agent and makes the communication possible with agents of other platform. It is basically equivalent to *Message Transport Service (MTS)* supporting the communication of FIPA ACL (Agent Communication Language) messages between agents of inter-agent platform and intra-agent platform. The channels *df*, *ams* are used to communicate with DF and AMS of the agent platform respectively. The *mts* represents mediator agents of other agent platforms with which SIMBA communicator agent is interacting. The SIMBA communicator agent can send message to any agent within the agent platform by using the method *SendMessageWithinAgentPlatform*. It can also communicate with any agent of other platform by using the method SendMessageAcrossAgentPlatform.

SIMBACommunicatorAgent

df , ams : ***chan***
mts : ℙ MessageTransportService
inagents : ℙ InAgent
artisagents : ℙ ARTISAgent

INIT
inagents ≠ ∅ ∧ artisagents ≠ ∅

SendMessageWithinAgentPlatform′
message? : Message
sender? : AgentType
receiver? : AgentType
(sender ∈inagents ∨sender ∈ artisagents)
^(receiver ∈inagents ∨receiver ∈artisagents)
SendMessage(sender, receiver , message);

SendMessageAcrossAgentPlatform′
message? : Message
sender? : AgentType
receiver? : AgentType
mts? : MessageTransportService

$\{ \exists agent \in inagents \mid agent = (sender \vee receiver)\}$
$\vee \{ \exists agent \in artisagents \mid agent = (sender \vee receiver)\}$
$SendMessage(sender, receiver, mts, message);$

$SendMessageWithinAgentPlatform \triangleq [$
$message? : Message; sender? : Agent;$
$receiver? : Agent]$
$\Box\ SendMessageWithinAgentPlatform'$
$\rightarrow SKIP$
$SendMessageAcrossAgentPlatform \triangleq$
$[message? : Message; sender? : Agent;$
$receiver? : Agent;$
$mts? : MessageTransportService] \Box$
$SendMessageAcrossAgentPlatform' \rightarrow$
$SKIP$
$MAIN \triangleq \mu T \Box$
$(SendMessageWithinAgentPlatform$
$\square\ SendMessageAcrossAgentPlatform); T$

Each ARTIS agent manages multiple In-agents providing the domain functionality. It includes a set of domain models. There should be at least one In-agent for every ARTIS agent. We use a function *SUM* which will return the sum of all the tasks of In-agents that any ARTIS agent has.

ARTISAgent

$id : \mathbb{N}$
$agents : \mathbb{P}\ InAgent$
$models : \mathbb{P}\ DomainModel$
$tasks : \mathbb{P}\ Task$
$c : \textbf{chan}$
$type : AgentType$
$state : \mathbb{P}\ State$
$cm : ControlModule$
$readAction : \mathbb{P}\ DomainModel \times \mathbb{P}\ State \rightarrow \mathbb{P}\ State$

$writeAction : \mathbb{P}\ State \times \mathbb{P}\ DomainModel \rightarrow \mathbb{P}\ DomainModel$
$perceiveAction : \mathbb{P}\ State \times Context \rightarrow \mathbb{P}\ State$
$effectAction : \mathbb{P}\ State \times Context \rightarrow Context$

$dom\ tasks \subseteq SUM(\forall agent : agents \Box agent.tasks)$

INIT

$agents \neq \emptyset, models \neq \emptyset, tasks \neq \emptyset$
$type = ARTISAGENT$

AddTask'

$\Delta models, \Delta tasks$
$t? : Task$
$dm? : \mathbb{P}\ DomainModel$
$dm! : \mathbb{P}\ DomainModel$
$agent? : InAgent$

$dm? \subseteq models \wedge tasks' = tasks \cup \{t\} \wedge$
$dm! = writeAction(state, dm?) \wedge$
$models' = models \mid dm? \cup dm!$
$agent.tasks' = agent.tasks' \cup \{t\}$

RemoveTask'

$\Delta models, \Delta tasks$
$t? : Task$
$dm? : \mathbb{P}\ DomainModel$
$dm! : \mathbb{P}\ DomainModel$
$agent? : InAgent$

$tasks \neq \emptyset \wedge dm? \subseteq models \wedge$
$tasks' = tasks \mid \{t\} \wedge$
$dm! = writeAction(state, dm?) \wedge$
$models' = models \mid dm? \cup dm!$
$agent.tasks' = agent:tasks' \mid \{t\}$

AddTask $\triangleq$ [t? : Task; agent? : InAgent;
dm? : $\mathbb{P}$ DomainModel] $\Box$ c?(t, agent, dm)
$\rightarrow$ cm.AnalyzeTask $\rightarrow$ AddTask′
$\rightarrow$ c!(dm) $\rightarrow$ SKIP
RemoveTask $\triangleq$ [t? : Task; agent? :
InAgent; dm? : $\mathbb{P}$ DomainModel] $\Box$
c?(t, agent, dm) $\rightarrow$ RemoveTask′
$\rightarrow$ c!(dm) $\rightarrow$ SKIP
MAIN $\triangleq \mu$ T $\Box$ (AddTask $\square$
RemoveTask); T

ARTIS agent has a single control module and controls the execution of tasks by the In-agents that belongs to it. It is divided into two sub-modules namely *Reflex server (RS)* and the *Deliberative Server (DS)*. RS controls the execution of components with critical temporal restrictions. DS controls the execution of deliberative components. Control

Module ensures that only tasks that can be executed are added to the system.

ControlModule

$result : BOOL$
$artis : \textbf{chan}$
$rs : ReflexServer$
$ds : DeliberativeServer$

$AnalyzeTask \triangleq [task? : Task] \Box artis?task \rightarrow (ds.CanDeliberativeExecute \wedge task.type = DELIBERATIVE \wedge artis!:result) \Box (rs:CanReflexExecute \wedge task.type = REFLEX \wedge artis!.result)$

ReflexServer

$c : \textbf{chan}$

CanReflexExecute

$task? : Task$

$\exists currentTime == clock.time; l == task?.length; r == task?.resource \Box currentTime..currentTime + l \subseteq \{t : T \mid alloc(r , t) = self \}$

DeliberativeServer

$c : \textbf{chan}$
$margin : \mathbb{N}$

CanDeliberativeExecute

$task? : Task$

$\exists currentTime == clock.time; l == task?.length; r == task?.resource \Box currentTime..currentTime + l + margin \subseteq \{t : T \mid alloc(r , t) = self \}$

A *computation* is an activity that manages its own state. Its current status is represented by attribute *state*.

Computation

$state : \mathbb{P} State$
$compute : \mathbb{P} State \leftrightarrow \mathbb{P} State$
$dom\ compute = \{s : \mathbb{P} State \mid s \subseteq state\}$

INIT

$state \neq \varnothing$

ComputationOp

$\Delta state$
$\Delta compute$
$s?, s! : \mathbb{P} State$
$s! = compute(s?) \wedge state' = state \mid s? \cup s!$

A *reflection model* is similar to meta-data and provides concrete instances of entities needed for adaptation. It corresponds to architectural models of the system.

ReflectionModel

$Model$ [$ReflectionModelRepresentation$]

A *reflective computation* is analogous to an In-agent but it acts and reasons about the reflection models. It is responsible for environment monitoring to determine the required adaptations. A reflective computation is not capable of changing the environment directly and needs support from the agents for doing so.

[*ReflectiveComputation[LMS]*]

$Computation$
$read : \mathbb{P} ReflectionModel \times \mathbb{P} State \rightarrow \mathbb{P} State$
$writeAction : \mathbb{P} State \times \mathbb{P} ReflectionModel \rightarrow \mathbb{P} ReflectionModel$
$perceiveAction : Context \times \mathbb{P} State \rightarrow \mathbb{P} State$
$senseAction : \mathbb{P} LocalManagedSystem \times \mathbb{P} State \rightarrow \mathbb{P} State$
$adaptAction : \mathbb{P} LocalManagedSystem \times \mathbb{P} State \rightarrow \mathbb{P} LocalManagedSystem$
$triggerAction : \mathbb{P} State \times \mathbb{P} ReflectiveComputation[LocalManagedSystem] \rightarrow \mathbb{P} ReflectiveComputation[LocalManagedSystem]$

A *coordination mechanism* contains coordination protocol and a coordination model.

[*CoordinationMechanism[Protocol,Model]*]

protocol : Protocol
model : Model

A *local managed system* contains multiple domain models and an ARTIS agent. Each local managed system corresponds to a single ARTIS agent.

[*LocalManagedSystem[Protocol;Model]*]

agent : ARTISAgent
models : $\mathbb{P}$ *DomainModel*

A *local reflective computation* is basically a reflective computation with coordination mechanisms.

[*LRC[LMS, Protocol,Model]*]

Computation
coordinationMechanism :
CoordinationMechanism[Protocol,Model]
readAction : $\mathbb{P}$*ReflectionModel*×
$\mathbb{P}$ *State* → $\mathbb{P}$ *State*
writeAction : $\mathbb{P}$ *State* × $\mathbb{P}$
ReflectionModel → $\mathbb{P}$ *ReflectionModel*
triggerAction : $\mathbb{P}$ *State* × $\mathbb{P}$
LocalReflectiveComputation
[LocalManagedSystem, Protocol,Model] →
$\mathbb{P}$ *LocalReflectiveComputation*
[LocalManagedSystem, Protocol, Model]
senseAction : $\mathbb{P}$ *LocalManagedSystem* ×
$\mathbb{P}$ *State* → $\mathbb{P}$ *State*
adaptAction : $\mathbb{P}$ *LocalManagedSystem* ×
$\mathbb{P}$ *State* → $\mathbb{P}$ *LocalManagedSystem*
perceiveAction : Context × $\mathbb{P}$ *State* →
$\mathbb{P}$ *State*

A *self-adaptive unit* is a reflective subsystem comprising reflection models and local reflective computations. It is defined as an active class continuously monitoring the system.

SelfAdaptiveUnit

state : $\mathbb{P}$ *State*
models : $\mathbb{P}$ *ReflectionModel*
computations : $\mathbb{P}$
LRC[LMS, Protocol, Model]
c : ***chan***
readAction : $\mathbb{P}$ *ReflectionModel* ×
$\mathbb{P}$ *State* → $\mathbb{P}$ *State*
writeAction : $\mathbb{P}$ *State* ×
$\mathbb{P}$ *ReflectionModel* → $\mathbb{P}$ *ReflectionModel*
triggerAction : $\mathbb{P}$ *State* ×
$\mathbb{P}$ *LRC[LMS, Protocol, Model]* →
$\mathbb{P}$ *LRC[LMS, Protocol, Model]*

$\forall$*c : computations* □
dom c.readAction = {m : $\mathbb{P}$
ReflectionModel | m ⊆*models* □
(m, c.state)} ∧
dom c.writeAction = {m : $\mathbb{P}$
ReflectionModel | m ⊆*models* □
(c.state, m)} ∧
dom c.triggerAction = {ct : $\mathbb{P}$
LRC[LMS, Protocol, Model]
| ct ⊆*computations | {c}* □ *(c.state, ct)}*
Read ≙ *[m? :* $\mathbb{P}$ *ReflectionModel; state? :*
$\mathbb{P}$ *State]* □ *c?(m, state)* → *readAction* →
c!(state) → *SKIP*
Write ≙ *[state? :* $\mathbb{P}$ *State; m? :* $\mathbb{P}$
ReflectionModel] □ *c?(state,m)* →
writeAction → *c!(m)* → *SKIP*
Trigger ≙ *[state? :* $\mathbb{P}$ *State; ct :* $\mathbb{P}$ *LRC*
[LMS, Protocol,
Model]] □ *c?(state, ct)* → *triggerAction*
→ *c!(ct)* → *SKIP*
MAIN ≙ *μ T* □ *(Read* □ *Write* □ *Trigger);*
T

A *local self-adaptive multi-agent system* contains multiple local managed systems and self-adaptive units. Local managed systems can be sensed and adapted by the self-adaptive units. It will contain a list of all the ARTIS agents and In-agents. It also contains SIMBA communicator agent for communication with any agent within the platform or outside the platform. An agent is added to the multi-agent system via AMS which assigns a unique identifier to every agent. Similarly, an agent is removed from the multi-agent system via AMS.

LocalSelfAdaptiveMultiAgentSystem

$localManagedSystems : \mathbb{P}\, LMS\ [Protocol,Model]$
$selfAdaptiveUnits : \mathbb{P}\, SelfAdaptiveUnit\ [LMS, Protocol, Model]$
$sca : SIMBACommunicatorAgent$

$\forall sau : selfAdaptiveUnits; lrcSense; lrcAdapt : LocalReflectiveComputation \bullet$
$lrcSense \in sau : computations \land lrca \in sau : computations \land$
$dom\ lrcSense.sense = \{lms : \mathbb{P}\, LMS \mid lms \subseteq localManagedSystems \bullet (lms, lrcSense.state)\} \land$
$dom\ lrcAdapt.adapt = \{lms : \mathbb{P}\, LMS \mid lms \subseteq localManagedSystems \bullet (lms, lrcAdapt.state)\}$

Finally, a distributed self-adaptive system contains multiple local self-adaptive multi-agent systems. We use external choice operator between all the active classes to allow a choice of behavior according to the environmental events.

DistributedSelfAdaptiveSystem

$localSelfAdaptiveSystems : \mathbb{P}\ LocalSelfAdaptiveMultiAgentSystem\ [Protocol, Model]$
$MAIN \triangleq \mu\, T \bullet (Environment \Box InAgent \Box DirectoryFacilitator \Box AgentManagementSystem \Box SIMBA\ Communicator\ Agent \Box ARTIS\ Agent \Box SelfAdaptiveUnit); T$

5 DISCUSSION AND FUTURE WORK

THE proposed SMARTS framework can be used for the formal architectural specification of any real-time multi-agent systems and it is an extension of the FORMS reference model as proposed in Weyns, et. al. (2012). The major limitation of the FORMS reference model is that it is not directly applicable to the multi-agent systems. Our framework uses MAPE-K interfaces Qasim and Kazmi (2016), reflection perspective and unification with distribution perspective of the FORMS reference model into the SIMBA agent architecture and overcomes this limitation. It is to be noted that in our framework the agents themselves are not adaptable, but the multi-agent system as a whole is adaptable. The framework provides a clear distinction between those computations providing the domain functionality and the reflective computations (computations needed for adaptation). We intend to modify the ARTIS agents in future to make them adaptable by integrating the MAPE-K loops as proposed in Kephart and Chess (2003) into them. State of the art in the self-adaptive software systems recommend the use of formal methods. We used TCOZ for the formal specification of our framework and utilized the expressiveness of Z-language and Process Algebra. The active class and passive class concept of TCOZ makes it easier for the designer to differentiate the entities responsible for capturing the agents state and entities responsible for dynamic interaction. The provision of communication channels in TCOZ greatly simplify the class definitions, class referencing, enhancing their modularity. For future work we intend to work on the issues of communication between multiple self-adaptive systems using diverse agent platforms. Also TCOZ is a specification language so no system properties like Liveness, Safety, etc. can be verified at the design time. We intend to overcome this problem by using some other formal modelling technique like Petri-nets.

6 CONCLUSION

In this paper we have shown how self-adaptive real-time multi-agent systems can be formally modelled. We proposed a formal framework named SMARTS for the formal modelling of such systems. The multi-agent systems paradigm has been in use for the ubiquitous and pervasive environments and they are seen as a key enabling technology to model critical components of a large number of distributed applications. To ensure a high degree of reliability of these systems their formal modelling is necessary as state of the art in the self-adaptive software systems recommend the use of formal methods. A comprehensive review of the related works reveals that there is a lack of research on consolidating design knowledge for real-time self-adaptive multi-agent systems. Hence there is a dire need of formal vocabulary that can be used for the conceptual design of any real-time multi-agent system with self-adaptation. SMARTS incorporates various aspects of self-adaptation and the domain functionality is handled by the SIMBA real-time agent architecture. The framework as a whole is FIPA-compliant and the run-time schedulability ensures that the tasks meet their deadline when deployed. The research conducted will help to formally model self-adaptive real-time multi-agent systems at the design time.

DISCLOSURE STATEMENT

No potential conflict of interest was reported by the authors.

REFERENCES

Abbas, H. A., Shaheen, S. I., & Amin, M. H. (2015). Organization of multi-agent systems: an overview. *arXiv preprint arXiv:1506.09032*.

Abbas, N., Andersson, J., Iftikhar, M. U., & Weyns, D. (2016, April). Rigorous architectural reasoning for self-adaptive software systems. In *Software Architectures (QRASA), 2016 Qualitative Reasoning about* (pp. 11-18). IEEE.

Alrashed, S., Alhiyafi, J., Shafi, A., & Min-Allah, N. (2016). An efficient schedulability condition for non-preemptive real-time systems at common scheduling points. *The Journal of Supercomputing*, *72*(12), 4651-4661.

Baresi, L. (2014, September). Self-adaptive systems, services, and product lines. In *Proceedings of the 18th International Software Product Line Conference-Volume 1* (pp. 2-4). ACM.

Bonnet, J., Gleizes, M. P., Kaddoum, E., Rainjonneau, S., & Flandin, G. (2015, September). Multi-satellite mission planning using a self-adaptive multi-agent system. In *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on* (pp. 11-20). IEEE.

Botti, V., Carrascosa, C., Julián, V., & Soler, J. (1999). Modelling agents in hard real-time environments. *Multi-Agent System Engineering*, 63-76.

de la Iglesia, D. G., Calderón, J. F., Weyns, D., Milrad, M., & Nussbaum, M. (2015). A Self-adaptive multi-agent system approach for collaborative mobile learning. *IEEE Transactions on Learning Technologies*, *8*(2), 158-172.

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., and Weyns, D. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II* (pp. 1-32). Springer Berlin Heidelberg.

El Kholy, W., El Menshawy, M., Laarej, A., Bentahar, J., Al-Saqqar, F., & Dssouli, R. (2015, October). Real-Time Conditional Commitment Logic. In *International Conference on Principles and Practice of Multi-Agent Systems* (pp. 547-556). Springer, Cham.

Gascueña, J. M., Navarro, E., & Fernández-Caballero, A. (2012). Model-driven engineering techniques for the development of multi-agent systems. *Engineering Applications of Artificial Intelligence*, *25*(1), 159-173.

Gil de la Iglesia, D. (2014). *A formal approach for designing distributed self-adaptive systems* (Doctoral dissertation, Linnaeus University Press).

Graja, Z., Migeon, F., Maurel, C., Gleizes, M. P., & Kacem, A. H. (2016). A stepwise refinement-based development of self-organising multi-agent systems: application to the foraging ants. *International Journal of Agent-Oriented Software Engineering*, *5*(2-3), 134-166.

Guerin, F. (2002). *Specifying agent communication languages* (Doctoral dissertation, University of London).

Guo, M., & Dimarogonas, D. V. (2015). Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, *34*(2), 218-235.

Guo, Y., Goncalves, G., & Hsu, T. (2013). A multi-agent based self-adaptive genetic algorithm for the long-term car pooling problem. *Journal of Mathematical Modelling and Algorithms*, 1-22.

Herrero, Á., Navarro, M., Corchado, E., & Julián, V. (2013). RT-MOVICAB-IDS: Addressing real-time intrusion detection. *Future Generation Computer Systems*, *29*(1), 250-261.

Iglesia, D. G. D. L., & Weyns, D. (2015). MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *10*(3), 15.

Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development.

Autonomous agents and multi-agent systems, *1*(1), 7-38.

Johnson, K., Sinha, R., Calinescu, R., & Ruan, J. (2015, August). A multi-agent framework for dependable adaptation of evolving system architectures. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on* (pp. 159-166). IEEE.

Julian, V., & Botti, V. (2004). Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering*, *11*(2), 135-149.

Julian, V., & Botti, V. (2004). Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering*, *11*(2), 135-149.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41-50.

Klös, V., Göthel, T., & Glesner, S. (2015, August). Adaptive knowledge bases in self-adaptive system design. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on* (pp. 472-478). IEEE.

Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G., & Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, *17*, 184-206.

Li, Z., & Miao, H. (2015, June). Formal specification and reasoning for situated multi-agent system. In *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on* (pp. 455-460). IEEE.

Macías-Escrivá, F. D., Haber, R., Del Toro, R., & Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, *40*(18), 7267-7279.

Mao, X., Dong, M., & Zhu, H. (2014). A two-layer approach to developing self-adaptive multi-agent systems in open environment. *International Journal of Agent Technologies and Systems (IJATS)*, *6*(1), 65-85.

Nair, R. R., Behera, L., Kumar, V., & Jamshidi, M. (2015). Multisatellite formation control for remote sensing applications using artificial potential field and adaptive fuzzy sliding mode control. *IEEE Systems Journal*, *9*(2), 508-518.

Ntika, M., Kefalas, P., & Stamatopoulou, I. (2014). Formal modelling and simulation of a multi-agent nano-robotic drug delivery system. *Scalable Computing: Practice and Experience*, *15*(3), 217-230.

Puviani, M., Cabri, G., Capodieci, N., & Leonardi, L. (2015, January). Building self-adaptive systems by adaptation patterns integrated into agent methodologies. In *International Conference on Agents and Artificial Intelligence* (pp. 58-75). Springer, Cham.

Qasim, A., & Kazmi, S. A. R. (2016). MAPE-K interfaces for formal modeling of real-time self-adaptive multi-agent systems. *IEEE Access*, *4*, 4946-4958.

Qasim, A., Kazmi, S. A. R., & Fakhir, I. (2015). Executable semantics for the formal specification and verification of E-agents. *Indian Journal of Science and Technology*, *8*(16), 1.

Qasim, A., Kazmi, A. R., & Fakhir, I. (2015). Formal specification and verification of real-time multi-agent system using timed arc Petri nets. *Advances in Electrical and Computer Engineering*, *15*(3), 73-8.

Qureshi, M. B., Alrashed, S., Min-Allah, N., Kołodziej, J., & Arabas, P. (2015). Maintaining the Feasibility of Hard Real–Time Systems with a Reduced Number of Priority Levels. *International Journal of Applied Mathematics and Computer Science*, *25*(4), 709-722.

Sanderson, D., Pitt, J., & Busquets, D. (2013, November). Interactions of Multiple Self-Adaptive Mechanisms in Multi-agent Systems. In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02* (pp. 301-308). IEEE Computer Society.

Schaefer, I., & Hahnle, R. (2011). Formal methods in software product line engineering. *Computer*, *44*(2), 82-85.

Shan, L., Du, C., & Zhu, H. (2015, July). Modeling and Simulating Adaptive Multi-Agent Systems with CAMLE. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual* (Vol. 2, pp. 147-152). IEEE.

Tesar, D. (2016). Next Wave of Technology. *Intelligent Automation & Soft Computing*, *22*(2), 211-225.

Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K. L., & Dautenhahn, K. (2014). Formal verification of an autonomous personal robotic assistant. *Proc. AAAI FVHMS*, 74-79.

Weyns, D., & Andersson, J. (2013, July). On the challenges of self-adaptation in systems of systems. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems* (pp. 47-51). ACM.

Weyns, D., Bencomo, N., Calinescu, R., Camara, J., Ghezzi, C., Grassi, V., and Mirandola, R. (2017). Perpetual assurances in self-adaptive systems.

Weyns, D., & Calinescu, R. (2015, May). Tele assistance: A self-adaptive service-based system examplar. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (pp. 88-92). IEEE Press.

Weyns, D., Malek, S., & Andersson, J. (2012). FORMS: Unifying reference model for formal specification of distributed self-adaptive systems.

ACM Transactions on Autonomous and Adaptive Systems (TAAS), *7*(1), 8.

Wooldridge, M. (2000). Semantic issues in the verification of agent communication languages. *Autonomous agents and multi-agent systems*, *3*(1), 9-31.

7 NOTES ON CONTRIBUTORS

Awais Qasim received the B.S. degree in Computer Science from the Punjab University College of Information Technology (PUCIT), Lahore, Pakistan, in 2009, M.S. degree in Computer Science from Lahore University of Management Sciences (LUMS), Lahore, Pakistan, in 2011 and the Ph.D degree in Computer Science from Government College University, Lahore, Pakistan, in 2017. He has worked as a Software Engineer in Industry and developed a number of iPhone and Android applications. He joined the Computer Science Department, Government College University in 2012 as a Lecturer. He has published 9 research papers in peer-reviewed ISI indexed journals. His research interests include model checking, multi-agent systems, real-time systems, self-adaptive systems.

Syed Asad Raza Kazmi received the M.S. degree in Computer Science from Sir Syed Engineering University, Karachi, Pakistan. He received the Ph.D. degree in Computer Software and Theory from State Key Laboratory of Computer Science Institute of Software Chinese, Academy of Sciences, Beijing, China, in 2008. He is currently working as an Assistant Professor/In-charge at the Computer Science Department, Government College University, Lahore. He conducts research in the area of Formal Methods specifically using techniques like LTL, CTL, modal mu-calculus to specify the properties of reactive and concurrent systems for the verification of properties like Safety, Liveness, Deadlock, etc. He has also worked in the area of formal modeling of real-time multi-agent systems, model checking, compositional reasoning, fixed point theory and has written over 10 peer-reviewed ISI indexed journals.