exeley INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS

Article | DOI: 10.21307/ijssis-2020-003 Issue 1 | Vol. 13 (2020)

Intelligent agent for formal modelling of temporal multi-agent systems

Awais Qasim[1,2]*, Zeeshan Aziz[2], Syed Asad Raza Kazmi[1], Adnan Khalid[1], Ilyas Fakhir[1] and Jawad Hassan[3]

[1]Department of Computer Science, GC University, Lahore, Pakistan.

[2]School of Science, Engineering and Environment, University of Salford, Salford, UK.

[3]Lahore Garrison University, Lahore, Pakistan.

*E-mail: awais@gcu.edu.pk

This paper was edited by Ivan Laktionov.

Received for publication December 10, 2019.

Abstract

Software systems are becoming complex and dynamic with the passage of time, and to provide better fault tolerance and resource management they need to have the ability of self-adaptation. Multi-agent systems paradigm is an active area of research for modeling real-time systems. In this research, we have proposed a new agent named SA-ARTIS-agent, which is designed to work in hard real-time temporal constraints with the ability of self-adaptation. This agent can be used for the formal modeling of any self-adaptive real-time multi-agent system. Our agent integrates the MAPE-K feedback loop with ARTIS agent for the provision of self-adaptation. For an unambiguous description, we formally specify our SA-ARTIS-agent using Time-Communicating Object-Z (TCOZ) language. The objective of this research is to provide an intelligent agent with self-adaptive abilities for the execution of tasks with temporal constraints. Previous works in this domain have used Z language which is not expressive to model the distributed communication process of agents. The novelty of our work is that we specified the non-terminating behavior of agents using active class concept of TCOZ and expressed the distributed communication among agents. For communication between active entities, channel communication mechanism of TCOZ is utilized. We demonstrate the effectiveness of the proposed agent using a real-time case study of traffic monitoring system.

Keywords

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

Multi-agent system has been an active area of research for specifying complex and adaptive systems. These complex and adaptive systems when deployed in a real-time domain have to work with hard temporal constraints. An agent is defined as a computer software system which works autonomously in an environment to achieve its objectives (Jennings et al., 1998). Such an agent with restrictive timing constraints is called a real-time agent (RTA). The correct functioning of these RTA agents does not solely depend on whether they complete the task rather than it depends on whether they complete the task within the deadline or not. Previously, these RTA agents have been classified as hard real-time agents and soft real-time agents in the study of Julian and Botti (2004). In soft real-time agents, there is a slight marginal period for the fulfillment of their temporal restrictions. A multi-agent system with at least one real-time agent is called a real-time multi-agent system (RTMAS). This dynamism of real-time software systems has led to a new category of software systems called self-adaptive software system. These self-adaptive systems possess the necessary knowledge to adapt their behavior in response to environmental context. In the studies of 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. Formal modeling corresponds to constructing a mathematical representation of a software or a hardware system using some level of abstraction. Formal specification provides an unambiguous and precise meaning of the different entities of the system leading to its enhanced understanding. Moreover, with formal semantics a system's domain functionality can be validated using different formal methods techniques like model checking. It has been argued in the study of Filieri et al. (2014) that formal methods should be used for the automated verification of safety critical and real-time systems to ensure their correct functioning.

Multi-agent systems have been formally specified and verified by many in the past but not self-adaptive real-time multi-agent systems, according to our knowledge. Reynisson et al. (2014) have formally modeled real-time systems using an extension of the Rebeca language. They used structural operational semantics for modeling distributed systems with temporal constraints. In the study of Chen (2012), a new language named STeC (an extension of process algebra) has been proposed for the formal specification of location-trigger real-time systems. In the study of Logenthiran et al. (2012), a multi-agent system approach has been presented for the real-time operation of scheduling and demand management in microgrids. Multi-agent systems have been formally specified and verified using modal mu-calculus and Timed-Arc Petri-nets in the study of Qasim et al. (2015a, b, 2016). Lomuscio et al. (2015) has presented a new model checker named MCMAS for the formal verification of multi-agent systems. Their model checker can be used to verify the epistemic, strategic, and temporal properties of interest for these multi-agent systems. Konur et al. (2013) have presented a new combined model checking approach for eliminating the problem of introducing new logics for the verification of different aspects of multi-agent systems like knowledge and time, knowledge and probability, real-time and knowledge, etc. This will help to reduce the problem of having different model checking tools targeting different aspects of multi-agent systems. In the study of Sun et al. (2013), hierarchical real-time systems have been formally modeled and verified using an extension of Timed CSP called Stateful Timed CSP. Majorly, they solved the problem of verification with non-zeroness assumption. In the study of Weyns et al. (2012), a framework for formal modeling of distributed self-adaptive systems has been proposed called FORMS, which provides different modeling elements and a set of relationships guiding the design of self-adaptive software systems. 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 the study of 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. In the study of Varzaneh et al. (2018), a recommender system based on association rules has been presented that detects the similarities among the users through association rules among voted items. Ettefagh et al. (2017) extended the Kautz parametrization of the model predictive control (MPC) method for linear time-varying systems. They showed how Kautz network can be used to maintain a satisfactory performance, while the number of decision variables is reduced considerably. Dammalage (2018) evaluated the effects of site-dependent errors on C/A code differential GPS correction accuracies by providing special emphasis on the multi-path error. 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.

However, up to our knowledge no real-time agent with self-adaptive abilities has been proposed in the past. For the specification of self-adaptive real-time multi-agent systems, there is a dire need of such an agent. In this paper, we have proposed a formal real-time agent having self-adaptive ability which can be used for the formal modeling of any real-time multi-agent system. Our self-adaptive real-time agent makes use of the ARTIS agent architecture proposed in the study of Botti et al. (1999) and MAPE-K feedback loop proposed in the study of Kephart and Chess (2003). For complex systems, formal specifications are devised at conceptual design before the systems are implemented in many areas of software engineering. Such specifications describe the semantics of the system being implemented without the concern for implementation details and can be used as a basis for the verification and validation of the functionality of the system. Hence, we provide a complete formal specification of our self-adaptive real-time agent

using Timed Communicating Object-Z (TCOZ). One of the major reason for choosing TCOZ as a formal specification language is that we can utilize the active class concept of TCOZ to express the non-terminating behavior of autonomous agents.

The rest of this paper is divided as follows. In the "Preliminaries" section, some preliminaries for the entities description of ARTIS agent architecture are explained. The "Proposed SA-Artis-agent" section describes the proposed SA-ARTIS-agent and its formal specification using TCOZ. In the "Discussion and future work" section, we provide future directions of our proposed work. The "Conclusion" section concludes the paper.

Preliminaries

Artis agent architecture

ARTIS agent architecture was proposed in the study of Botti et al. (1999) and it is an extension of the blackboard model that has been modified to work in environments with hard temporal constraints. This agent guarantees that it will meet its temporal constraints by the use of an off-line schedulability analysis. Agents' perception occurs through a set of sensors and the systems response is exhibited using a set of effectors. These perception and action processes are real-time in nature. The agent has two different categorization of processes, namely, reflex process and a deliberative process. Every ARTIS agent has a number of internal agents (In-agent) that provides the domain functionality. Every In-agent is designed to solve a particular problem. Every In-agent is characterized as critical or acritical. A critical In-agent has a period and a deadline and the agent must perform its operations within those deadlines. In other words, it provides the minimum system functionality. On the contrary, acritical In-agent can utilize artificial intelligence techniques to better achieve the system goal. Every In-agent has two layers, namely, reflex layer and real-time deliberative layer. When a task arrives for execution, the In-agent checks the deadline if it can provide a response via a real-time deliberative layer. The real-time deliberative layer provides an improved response as compared to reflex layer, hence it needs more time. The reflex layer only provides a minimal quality 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 In-agents make up the optional phase of an ARTIS agent. A reflex layer is absent in a non-critical In-agents and only the real-time deliberative layer is present. For real-time environments, most of the In-agents are critical in nature. Each In-agent has a set of beliefs comprising the domain knowledge relevant to it. Each ARTIS agent has a control module which controls the execution of all the In-agents that belongs to it. It is divided into two submodules, namely, the reflex server (RS) and the deliberative server (DS). Reflex server controls the execution of tasks with critical temporal restrictions. Deliberative server controls the execution of deliberative tasks.

Mape-K feedback loop

A self-adaptive system typically consists of a feedback loop that deals with the architectural adaptation of the system and a managed system, which provides the domain functionality. Adaptation based on architecture always requires a system to interact with the environment, reason about its models based on the stimulus received and then adapt itself. The feedback loop is known as MAPE-K and it was proposed in the study of Kephart and Chess (2003). The MAPE represents the monitor, analyze, plan, and execute phase, whereas the K represents the knowledge, which consists of the models of the system and the adaptation goals. MAPE-K feedback loop-based self-adaptation ensures that the overall system's functionality is not affected by making a clear distinction between the managed system and the managing system. We refer the reader Kephart and Chess's (2003) study for details concerning the MAPE-K feedback loop.

Proposed SA-Artis-agent

A self-adaptive system typically consists of a managed system which provides the domain functionality and a feedback loop which deals with architectural adaptations of the system. Architecture-based adaptation requires a system to interact with the environment, reason about its models based on the stimulus received and then adapt itself. The feedback loop is known as MAPE-K and it was proposed in the study of Kephart and Chess (2003). The MAPE represents monitor, analyze, plan, and execute phase, whereas the K represents the models of the system, its environment, and adaptation goals. We propose a modification of the ARTIS-agent named self-adaptive-ARTIS-agent (SA-ARTIS-agent) which will have the ability of self-adaptation. Figure 1 shows the architecture of SA-ARTIS-agent. Each ARTIS agent has a MAPE-K loop to continuously monitor all the In-agents. In monitor phase, the system

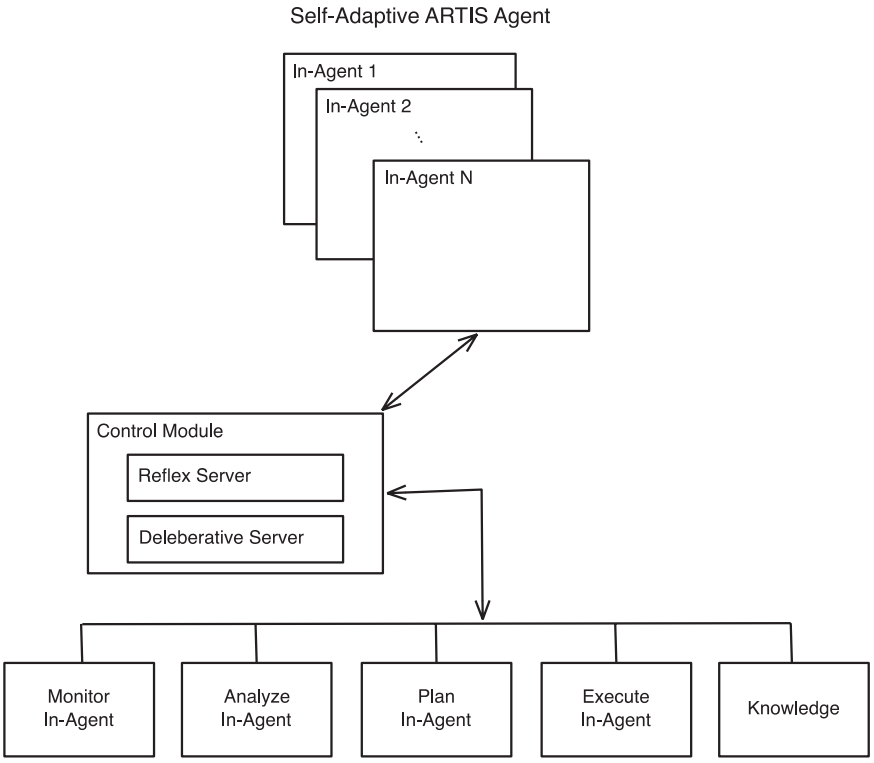

Figure 1: Proposed SA-ARTIS-agent architecture.

continuously perceives the environment and after any pre-processing of data it updates its models and trigger the next phase, i.e. analyze. In analyze phase, decision regarding whether the adaptations are needed or not is made. In case an adaptation is needed, it triggers the plan phase. In plan phase, a set of tasks/actions are generated that are required for the adaptation and then the execute phase is triggered. In execute phase, all the planned tasks are executed to perform self-adaptation. Knowledge corresponds to models representing aspects of the environment, system, and adaptation goals. It should be mentioned that all the activities of the system are considered as event triggered. We will give a complete formal specification of the SA-ARTIS-agent in the next section. This is because it has been advocated in the past that the use of precise and unambiguous notation of formal methods is beneficial for self-adaptive systems specification (Iglesia and Weyns, 2015). Although the basic SA-ARTIS agent will guarantee that it meets its deadlines for the tasks it has been designed to execute but if the designer decides to include feature like communicating with agents of other types then this may prevent this real-time behavior.

Our SA-ARTIS-agent will consist of the following entities summarized as follows. Task represents any task that will be executed by the agent. The task can be executed to provide the domain functionality or to adapt the agent. In-Agent will perform a specific task for which it has been designed. A single SA-ARTIS-agent may contain multiple In-agents, each providing different functionality. SA-ARTIS-Agent will provide the systems domain functionality and will possess the necessary knowledge required for adapting itself according to the goals. Monitor-In-Agent will continuously monitor the environment and communicate with all the other In-agents of the system. In case an event of interest occurs it will trigger the Analyze-In-agent and update the Knowledge accordingly. Analyze-In-Agent will be responsible for making the decisions if the adaptation decisions are required. In case the agent does need to adapt, it will trigger the Plan-In-agent. Plan-In-Agent is responsible for planning the necessary actions in case of adaptation and triggering the Execute-In-Agent. Execute-In-Agent will execute the adaptation actions of the generated plans. Knowledge entity will serve as model which the system can use to make the adaptation

decisions. It will be represented by domain models. Control Module is responsible for the real-time execution of the all the In-agents in the system. Reflex Server controls the execution of processes with critical temporal restrictions. Deliberative Server controls the execution of deliberative processes.

TCOZ specification of SA-ARTIS-agent

In this section, we will formally specify all the entities of SA-ARTIS-agent as was described in the previous section. We define 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. Here, 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. It is important to mention here that a task can be executed both to provide the domain functionality and for adaptation. For the categorization of tasks and agents, we define two types as:

TaskType :: = REFLEX | DELIBERATIVE

AgentType :: = INAGENT | ARTISAGENT

Task

$resource : Resource$

$length : T$

$margin : \mathbb{N}$

INIT

$length > 0$

ReflexExecute

$\Delta length$

$\delta = length$

$length' = 0$

DeliberativeExecute

$\Delta length$

$\delta = length + margin$

$length' = 0$

Every InAgent is configured to solve a particular type of problem. Here *id* represents the unique identifier of the InAgent. *tasks* represent the set of tasks, which this agent has to execute. The attribute *alloc* contains information about which resources have been allocated to this agent. *c* represents the single communication channel that this agent will use to communicate with the other entities. Here *margin* represents the time that will be used to decide if a task should be executed on reflex server or deliberative server. We define a new type *State* to represent the current status of any agent so here state represents the current state of this In-agent. *type* represents the agent type, i.e. In-agent or ARTIS agent. *rModel* represents the set of all representations required for the adaptation.

InAgent

$id : \mathbb{N}$

$tasks : \mathbb{P}\ task$

$alloc : (Resource \times T) \rightarrow InAgent$

$c : \textbf{chan}$

$margin : \mathbb{N}$

$state : \mathbb{P}\ State$

$type : AgentType$

$\forall\ r1, r2 : Resource; t : T \mid r1 \neq r2 \bullet alloc(self, r1, t) \neq alloc(self, r2, t)$

INIT

$type = INAGENT$

$request = \varnothing$

CanExecuteDeliberately

$task? : Task$

$task? \in tasks \land$

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

$ExecuteTask \triangleq [task? : Task] \bullet ((CanExecuteDeliberately \land task?.DeliberativeExecute) \Box (!CanExecuteDeliberately \land task?.ReflexExecute))$

$MAIN \triangleq \mu T \bullet [tasks \neq \varnothing] \bullet ExecuteTask; T$

Each SA-ARTIS-agent will manage multiple In-agents providing the domain functionality. There should be at least one In-agent for every ARTIS-agent. Here *id* represents the unique identifier of this agent. *agents* represent the set of In-agent that this agent manages. *models* represent the set of domain models. *tasks* represent the set of tasks which will then be delegated to the different In-agents. *c* represents the single communication

channel that this agent will use to communicate with the other entities. *type* represents the agent type, i.e. In-agent or ARTIS-agent. *state* represents the current state of this SA-ARTIS-agent. *cm* represents the control module. The attributes *mAgent*, *aAgent*, *pAgent*, *eAgent* correspond to the Monitor-In-Agent, Analyze-In-Agent, Plan-In-Agent, and Execute-In-Agent, respectively, which will handle the adaptation. We use a function *SUM*, which will return the sum of all the tasks of In-agents that any ARTIS Agent has.

ARTISAgent

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

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

INIT

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

AddTask'

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

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

RemoveTask'

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

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

AddTask ≙ [t? : Task; agent? : InAgent;
dm? : ℙ DomainModel] • c?(t, agent, dm) → cm.AnalyzeTask → AddTask′ → c!(dm) → SKIP
RemoveTask ≙ [t? : Task; agent? :
InAgent; dm? : ℙ DomainModel] • c?(t, agent, dm) → RemoveTask′ → c!(dm) → SKIP
MAIN ≙ μ T • (AddTask □ RemoveTask); T

The *Monitor_In_Agent* will continuously perceive the environment and after any pre-processing of data it will update the models and trigger the next, i.e. *Analyze_In_Agent*. *aInAgent* represents the *Analyze_In_Agent* to whom this agent will notify in case an event of interest occurs requiring adaptation.

Monitor_In_Agent

InAgent
$aInAgent : Analyze\_In\_Agent$

INIT

$type = INAGENT$

Sense'

$senseAction : \mathbb{P}\ Subsystem\ X\ \mathbb{P}\ State \rightarrow \mathbb{P}\ State$
$subsystem? : Subsystem$

$state' = senseAction(subsystem?, state)$

GetData'

$readAction : \mathbb{P}\ ReflectionModel\ X\ \mathbb{P}\ State \rightarrow \mathbb{P}\ State$

$state' = readAction(rModel, state)$

UpdateKnowledge'

$writeAction : \mathbb{P}\ State\ X\ \mathbb{P}\ ReflectionModel \rightarrow \mathbb{P}\ ReflectionModel$

$rModel' = writeAction(state, aInAgent)$

Trigger'

$triggerAction : \mathbb{P}\ State\ X\ Analyze\_In\_Agent \rightarrow Analyze\_In\_Agent$

$aInAgent' = triggerAction(state, aInAgent)$

Sense ≙ [subsystem? : Subsystem] • Sense′ → SKIP
GetData ≙ GetData′ → SKIP
UpdateKnowledge ≙ UpdateKnowledge′ → SKIP
Trigger ≙ Trigger′ → SKIP
MAIN ≙ μ T • Sense; GetData; UpdateKnowledge; Trigger ; T

The *Analyze_In_Agent* will make decisions regarding whether the adaptations are needed or not. In case an adaptation is needed it will triggers the *Plan_In_Agent*. *pInAgent* represents the *Plan_In_Agent* to whom this agent will notify to plan for the necessary adaptations. *requiredResources* represent the resources that this agent needs to complete its assigned tasks. *availableResources* are the resources that has been assigned to this agent. At any time the attribute *rRequirement* represents the situation of resources for this agent. Resource requirement can be divided into four classes, one in which the system does not require additional resources, second in which the system has more resources than it needs, third in which the system needs more resources, and fourth in which it is not possible to get a predictable total of the system resources.

ResourcesRequirement :: = SATISFIED | OVERSATISFIED | UNSATISFIED | UNDETERMINED

Analyze_In_Agent

InAgent
pInAgent : Plan_In_Agent
requiredResources : ℙ *Resource*
availableResources : ℙ *Resource*
rRequirement : ResourceRequirement

INIT
type = INAGENT
rRequirement = UNDETERMINED

GetData'
readAction : ℙ ReflectionModel X ℙ State→ ℙ State
state' = readAction(rModel, state)

AnalyzeResources'
ΔRequiredResources
ΔAvailableResources

UpdateKnowledge'
writeAction : ℙ State X ℙ ReflectionModel → ℙ ReflectionModel
rModel' = writeAction(state, aInAgent)

Trigger'
triggerAction : ℙ State X Plan_In_Agent → Plan_In_Agent
aInAgent' = triggerAction(state, pInAgent)

GetData ≙ GetData' → SKIP
AnalyzeResources ≙ [requiredResources?, availableResources : Resource] •
((AnalyzeResources' ^ rRequirement = SATISFIED)
□ (AnalyzeResources' ^ rRequirement = UNDERSATISFIED))
□ (AnalyzeResources' ^ rRequirement = OVERSATISFIED)) → SKIP
UpdateKnowledge ≙ UpdateKnowledge' → SKIP
Trigger ≙ Trigger' → SKIP
MAIN ≙ μ T • GetData; AnalyzeResources; UpdateKnowledge; Trigger ; T

The *Plan_In_Agent* will prepare a set of tasks/ actions that are required for the adaptation and then it will trigger the *Execute_In_Agent*. It basically plans two types of actions. In case the system resources are unsatisfied it creates plans to add resources to the system. On the contrary, if the system resources are oversatisfied it creates plans to release the extra resources. *eInAgent* represents the *Execute_In_ Agent* to whom this agent will notify to execute the actions required for the necessary adaptations. In case the resources are under-satisfied, a set of plan actions are devised in order to add resources to the managed system. Similarly, in case of oversatisfied, a set of plan actions are devised in order to release extra resources of the managed system. We define a new type *Plan* which is a collection of tasks.

Plan ::= *PTask*

Plan_In_Agent

InAgent
eInAgent : Execute_In_Agent
rRequirement : ResourceRequirement

INIT
type = INAGENT
rRequirement = UNDETERMINED

DevisePlanForUnsatidfied'
rRequirement? : UNSATISFIED
plans? : Plan
c!plans

DevisePlanForOversatidfied'
rRequirement? : OVERSATISFIED
plans? : Plan
c!plans

Trigger'
triggerAction : ℙ State X Execute_In_Agent → Execute_In_Agent
eInAgent' = triggerAction(state, eInAgent)

GetData ≙ GetData' → SKIP
DevisePlanForUnsatisfied ≙ [rRequirement : ResourceRequirement ; plans? : Plan] •
(DevisePlanForUnsatisfied' ^ requirement = UNSATISFIED) • c?(plans) → SKIP
DevisePlanForOversatisfied ≙ [rRequirement : ResourceRequirement ; plans? : Plan] •
(DevisePlanForOversatisfied' ^ requirement = OVERSATISFIED) • c?(plans) → SKIP
Trigger ≙ Trigger' → SKIP
MAIN ≙ μ T • (DevisePlanForUnsatisfied □ DevisePlanForOversatisfied); Trigger ; T

The *Execute_In_Agent* is responsible for executing the adaptation actions of the generated plans. There are three phases in an execute behavior: *PreProcess, ExecutePlan*, and *PostProcess*. In *PreProcess*, the agent acquires all the resources that are required for the adaptation goals. Once all the pre-processing has been completed, the agent performs *ExecutePlans* to perform the necessary adaptations. After all the plans have been executed the agent performs *PostProcess* to release all the acquired resources.

Execute_In_Agent

InAgent
requiredResources : ℙ Resource
availableResources : ℙ Resource
rRequirement : ResourceRequirement

INIT
type = INAGENT
rRequirement = UNDETERMINED

AddResouces'
newResources? ℙ Resource
ΔRequiredResources
ΔAvailableResources
Δalloc
availableResources' = availableResources ∪ {newResources}
requiredResources' = requiredResourcesn \ {newResources}

ExecutePlans'
plans? : Plan
execute : Plan X ℙ State X ℙ RelectionModel → ℙ RelectionModel
plans ≠∅
| ∀ *(plan ∈plans* ^ execute (plan, state, rModel))

ReleaseResources'
extraResources ? ℙ Resource
Δ AvailableResources
Δ allocesources
availableResources' = availableResources \ {extraResources}

PreProcess ≙ [newResources? : Resource] • (AddResources' ^ rRequirement = UNSATISFIED) • → SKIP
ExecutePlans ≙ [plans? : Plan] • ExecutePlans' → SKIP
PostProcess ≙ [extraResources? : Resource] • (ReleaseResources' ^ rRequirement = OVERSATISFIED) → SKIP
Trigger ≙ Trigger' → SKIP
MAIN ≙ μ T • PreProcess ; ExecutePlans ; PostProcess ; T

Application of the proposed SA-Artis-agent to real-time traffic monitoring

In this section we will demonstrate how the proposed SA-ARTIS-agent can be used to monitor and analyze the real-time traffic and ensure its correct functioning. An overview of the single monitoring station is shown in Figure 2. Monitor ARTIS agent will be the main agent responsible for monitoring and controlling the traffic. It is assumed that each monitoring station will work independently without any human intervention. This includes not only turning the signals but also managing the duration of each signal. This agent will analyze the real-time traffic using the Image Sensor In-agent and Video Sensor In-agent. Image Sensor In-agent will analyze the traffic based on imagery data, therefore it can process its data faster. Video Sensor In-agent will analyze the traffic based on video data. Each monitoring station will communicate with the rest of the stations using SIMBA communicator agent. This agent will also provide the yellow-pages and white-pages services to the other agents using DF In-agent and AMS In-agent, respectively. These monitoring stations need to have decentralized control so that we do not have a single point of failure. Each monitoring station will operate the signals they are controlling in two ways. First, under normal traffic the signals will operate based on fixed timing. This will give equal opportunity to each signal and the adaptation decisions will not be required. Second, in case congestion is detected then the adaptive decision taking module will be active and real-time timings for the signals will be derived.

Traffic Monitoring System attributes == {TrafficSignal, TrafficSensorInAgent, Monitor-Agent, Analyze-Agent, Plan-Agent, Execute-Agent}

Traffic Monitoring System processes == {Change TrafficStatus, ChangeTrafficType, UpdateNeighbors, AnalyzeImageData, AnalyzeVideoData, GetTrafficData, UpdateKnowledge, Trigger, AnalyzeTraffic-Data, Update Knowledge, DevisePlanForFullCongestion, DevisePlan ForNearCongestion, DevisePlanForFarCongestion, Acquire Resources, ExecutePlans, ReleaseResources}

We will integrate the bounded delay response requirements of agents as part of the QoS services. For verification that agent's actions conform to bound delay response, its requirements will be formally represented

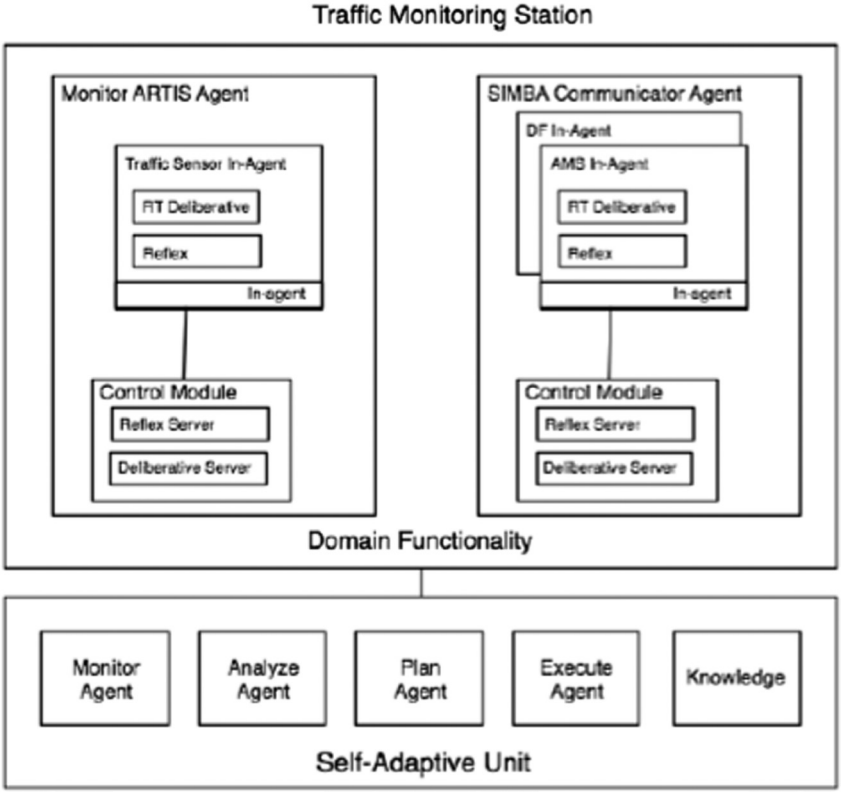

Figure 2: Overview of single traffic monitoring station.

as a validity problem. The validity problem will then be solved with the help of TAPAAL model checker. The TAPAAL is a verification tool for extended timed-arc Petri-nets with its own verification engine. We can create models of the system under consideration and perform automated verification using fragments of TCTL via transformation to timed automata.

The timed Petri-net model of the Muslim Town Mor signal is shown in Figure 3. The five stations of Muslim Town Mor signal have been represented by the acronym MTM. The simulation will start with the Start state with a single token. The transition Initialize will forward the token to all the Red states of the five signals. The transition MTM RedToYellow depicts the change in signal state from red to yellow. Initially, all the signals will be in red state. Since there can be only one signal in green state at any time, we have defined five constants for this purpose, namely, MTM1 Delay, MTM2 Delay, MTM3 Delay, MTM4 Delay, and MTM5 Delay. A single unit of delay corresponds to 20 sec. This means that each signal will remain green for 20 sec and it will for other signals in red or yellow state for 80 sec. The transition MTM YellowToGreen depicts the change in signal state from yellow to green. Since there can be only one green signal at a time we have a converging state named Converge. We have two transitions for each signal to show the status of green light. For this purpose, Turn MTM Green depicts that the signal is about to be green, whereas MTM Turned Green depicts that the signal has been green. The properties of interest we want to verify on our model are specified in Table 1.

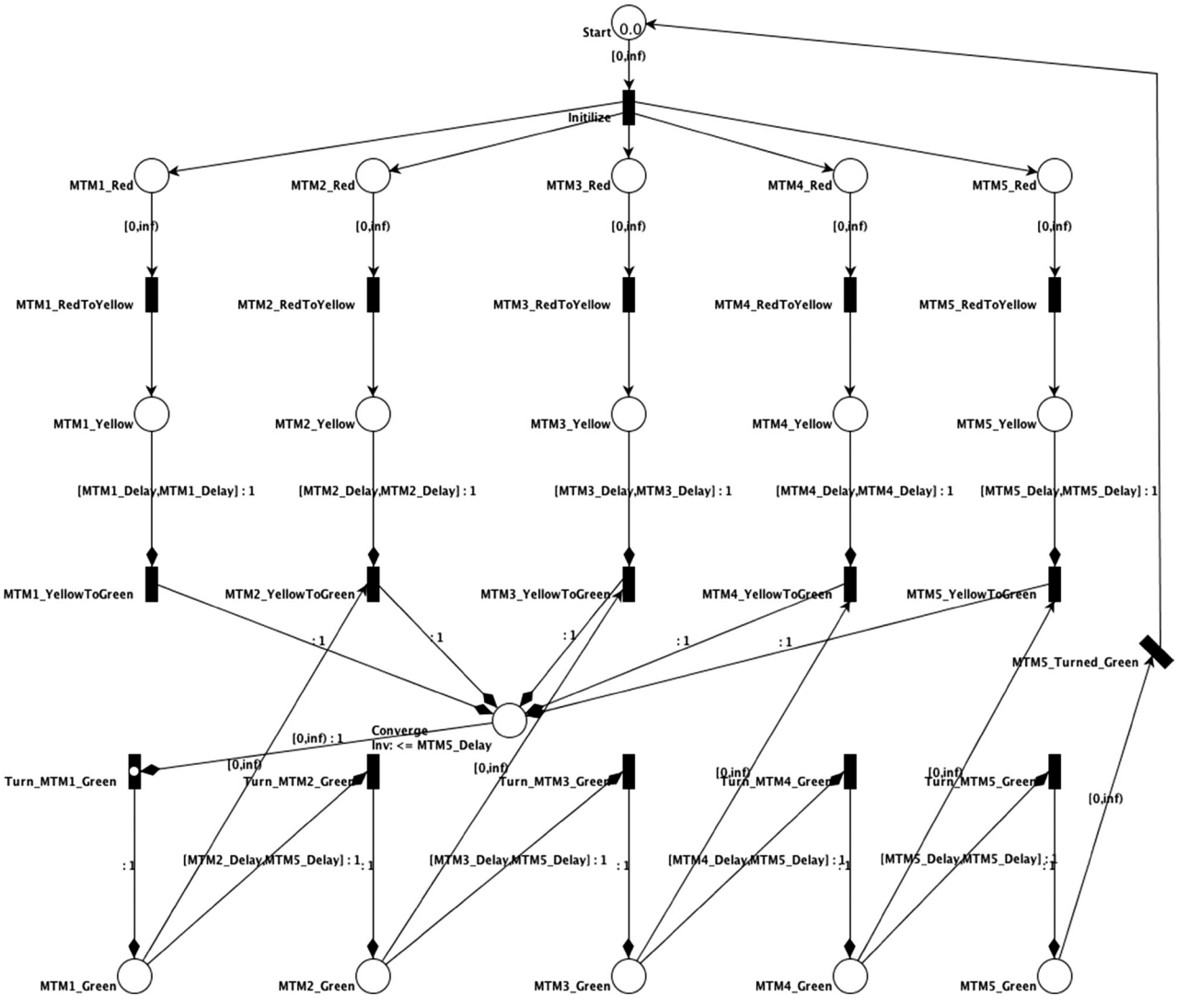

Figure 3: Timed Petri-Net model of the Muslim Town Mor signal.

Table 1. Formal verification of the traffic monitoring system.

Query	Formula	Result
Is MTM3 signal's green light working?	EF (MTM Signal.MTM3 Red=0 and MTM Signal.MTM3 Yellow=0 and MTM Signal.MTM3 Green=1)	Satisfied
Is MTM3 signal's yellow light working?	EF (MTM Signal.MTM3 Red=0 and MTM Signal.MTM3 Yellow=1 and MTM Signal.MTM3 Green=0)	Satisfied
Is MTM3 signal's red light working?	EF (MTM Signal.MTM3 Red=1 and MTM Signal.MTM3 Yellow=0 and MTM Signal.MTM3 Green=0)	Satisfied
Is MTM4 signal working?	EF ((MTM Signal.MTM4 Red=1and MTM Signal.MTM4 Yellow=0 and MTM Signal.MTM4 Green=0) or (MTM Signal.MTM4 Red=0 and MTM Signal.MTM4 Yellow=1 and MTM Signal.MTM4 Green=0) or (MTM Signal.MTM4 Red=0 and MTM Signal.MTM4 Yellow=0 and MTM Signal.MTM4 Green=1))	Satisfied
Is MTM4 signal's green light working?	EF (MTM Signal.MTM4 Red=0 and MTM Signal.MTM4 Yellow=0 and MTM Signal.MTM4 Green=1)	Satisfied
Is MTM4 signal's yellow light working?	EF (MTM Signal.MTM4 Red=0 and MTM Signal.MTM4 Yellow=1 and MTM Signal.MTM4 Green=0)	Satisfied
Is MTM4 signal's red light working?	EF (MTM Signal.MTM4 Red=1 and MTM Signal.MTM4 Yellow=0 and MTM Signal.MTM4 Green=0)	Satisfied

Discussion and future work

The proposed SA-ARTIS-agent can used to design any multi-agent system with ability of self-adaptation. One example of such multi-agent system is shown in Figure 4. The proposed Self-Adaptive SIMBA (SA-SIMBA) agent architecture makes use of the SIMBA agent architecture as proposed in the study of Julian et al. (2002) and FORMS reference model for the adaptation as proposed in the study of Weyns

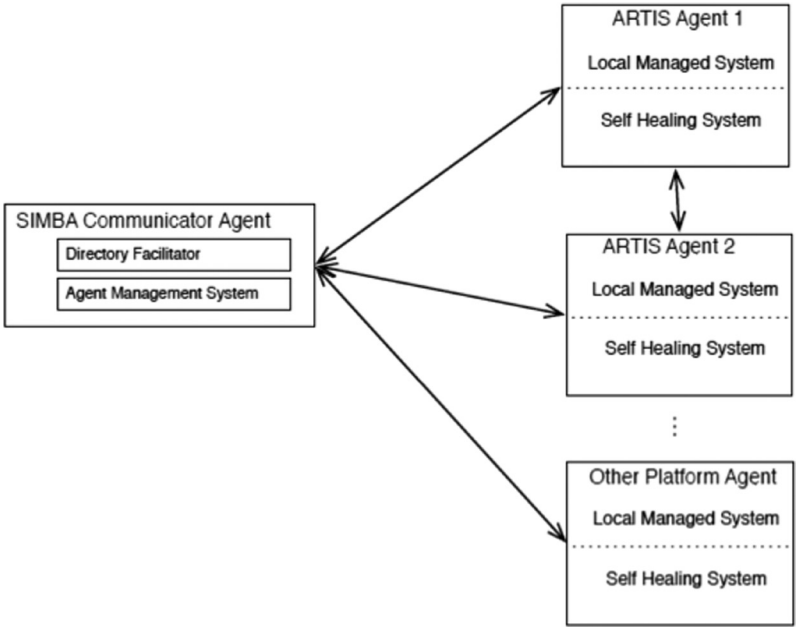

Figure 4: Self-adaptive SIMBA-Agent architecture.

et al. (2012). Basic agents of SA-SIMBA are SA-ARTIS-agents that will provide the systems domain functionality by using In-agents and adaptation using the MAPE-K feedback loops. In Figure 4, the local managed system corresponds to all those In-Agents that will perform some specific task. The self-healing subsystem, however, corresponds to the Monitor_In_Agent, Analyze_In_Agent, Plan_In_Agent, and Execute_In_Agent providing the adaptation logic. SA-ARTIS-agents have been designed to work in

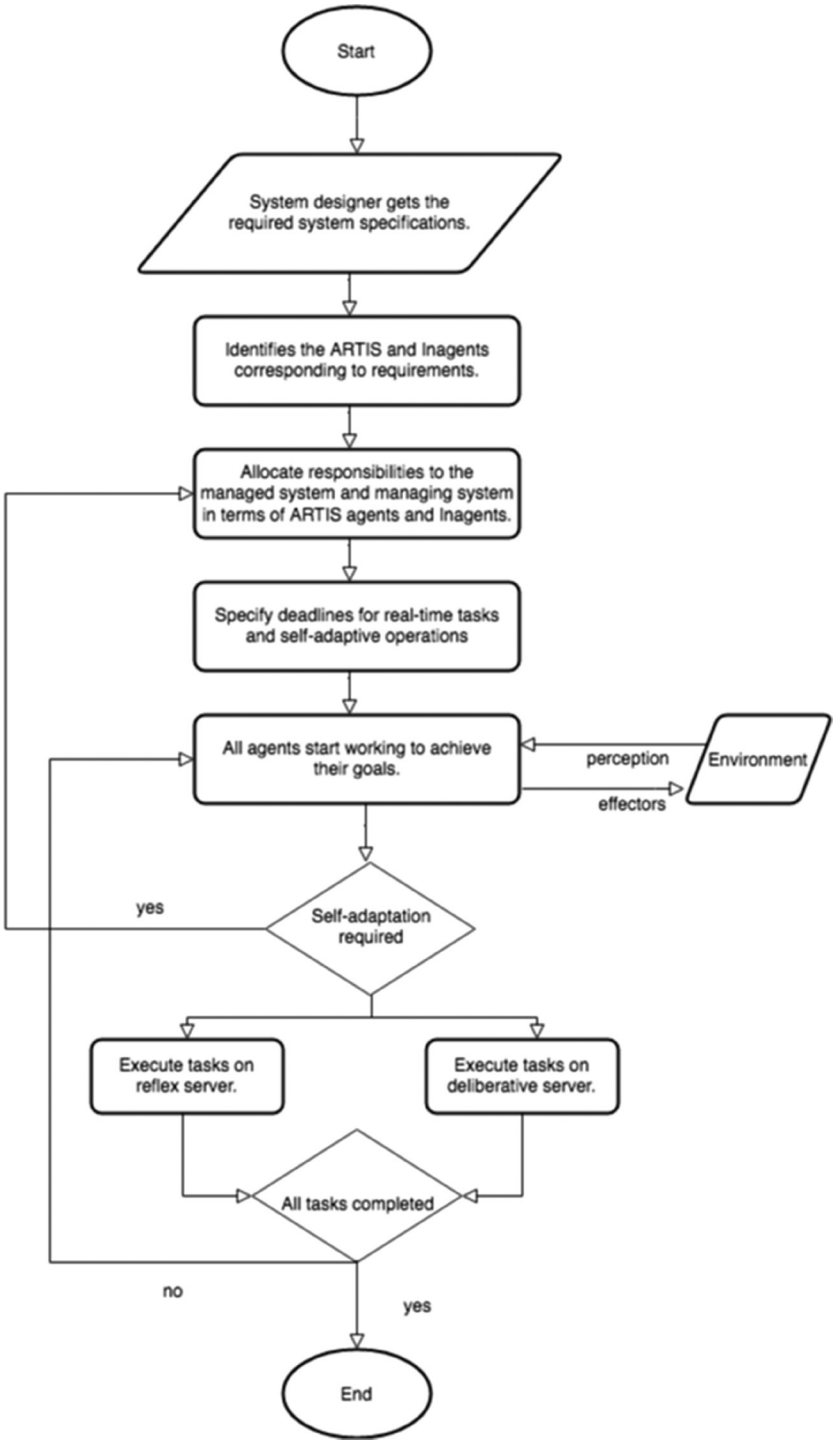

Figure 5: Flow chart describing the agent's initiation and working process.

dynamic environments with temporal constraints. The proposed SA-ARTIS can be used for the architectural specification of any self-adaptive real-time multi-agent system. We also intend to work on the issues of communication between multiple self-adaptive systems using diverse agent platforms. Specifically, issues related to agent communication languages for diverse agents with self-adaptation ability. Figure 5 provides a complete flow chart on the usage of the proposed agent. As compared to previous work where either only Z language or Petri-nets have been used for the controlling real-time traffic, our work provided expressiveness to model the state and communication ability at each traffic station. This provided the ability to check any traffic signal that it is deadlock free always and provides the maximum efficiency by controlling congestion.

Conclusion

In this paper, we proposed a new self-adaptive real-time agent named SA-ARTIS-agent that can be used for the formal modeling of self-adaptive real-time multi-agent systems. Self-adaptation was provided by incorporating MAPE-K feedback loops in each SA-ARTIS-agent. For a precise and unambiguous notation, we formally specified the SA-ARTIS-agent using TCOZ language. There are two major benefits of using TCOZ as a specification language. First, we can utilize the active class concept of TCOZ to express the non-terminating behavior of autonomous agents. Second, the provision of communication channels in TCOZ greatly simplifies the reference class definitions, enhancing their modularity. The multi-agent system paradigm has been in use for the modeling of systems in ubiquitous and pervasive environments. The dynamism in the execution of such systems has led to the development of self-adaptive systems. According to our knowledge, no work has been done for the proposition of real-time agent with self-adaptive ability. The runtime schedulability analysis ensures that the agent meet their deadline when deployed, hence ensuring the reliability of such systems. Our approach provides future directions for integrating TCOZ, timed petri-nets, and agent communication as a flexible and powerful tool for formal modeling of intelligent real-time systems. Formal verification will help to prove the correctness of the system being modeled which in turn will increase the confidence in the correctness of these systems. Our formal vocabulary is generic enough to express a real-time multi-agent system of any domain. It is also fine grained enough to test the properties of the system for the provision of domain functionality. The research conducted will help to formally model self-adaptive real-time multi-agent systems at the design time.

Literature Cited

Botti, V., Carrascosa, C., Julián, V. and Soler, J. 1999. Modelling agents in hard real-time environments. in Garijo F. J., Boman M. (Eds), *Multi-Agent System Engineering* Springer, Berlin, Heidelberg, pp. 63–76, available at: https://doi.org/10.1007/3-540-48437-X_6

Chen, Y. 2012. STeC: a location-triggered specification language for real-time systems. 2012 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), IEEE, pp. 1–6, available at: https://doi.org/10.1109/ISORCW.2012.11

Dammalage, T. L. 2018. Effects of site-dependent errors on the accuracy of C/A code DGPS positioning. *Civil Engineering Journal* 4(10): 2296–2304, available at: https://doi.org/10.28991/cej-03091159

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K. M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D. B., Sousa, J. P., Tahvildari, L., Wong, K. and Wuttke, J. 2013. Software engineering for self-adaptive systems: a second research roadmap. in de Lemos R., Giese H., Müller H. A., Shaw M. (Eds), *Software Engineering for Self-Adaptive Systems II* Springer, Berlin, Heidelberg, pp. 1–32, available at: https://doi.org/10.1007/978-3-642-35813-5_1

El Kholy, W., El Menshawy, M., Laarej, A., Bentahar, J., Al-Saqqar, F. and Dssouli, R. 2015. Real-time conditional commitment logic. in Chen, Q., Torroni, P., Villata, S., Hsu J., Omicini, A. (Eds), *PRIMA 2015: Principles and Practice of Multi-Agent Systems* Springer, Cham, pp. 547–556, available at: https://doi.org/10.1007/978-3-319-25524-8_37

Ettefagh, M. H., De Doná, J., Naraghi, M. and Towhidkhah, F. 2017. Control of constrained linear-time varying systems via Kautz parametrization of model predictive control scheme. *Emerging Science Journal* 1(2): 65–74, available at: https://doi.org/10.28991/esj-2017-01117

Filieri, A., Hoffmann, H. and Maggio, M. 2014. Automated design of self-adaptive software with control-theoretical formal guarantees. Proceedings of the 36th International Conference on Software Engineering, ACM, pp. 299–310, available at: https://doi.org/10.1145/2568225.2568272

Guo, M. and Dimarogonas, D. V. 2015. Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research* 34(2): 218–235, available at: https://doi.org/10.1177/0278364914546174

Herrero, Navarro, M., Corchado, E. and Julián, V. 2013. RT-MOVICAB-IDS: addressing real-time intrusion detection. *Future Generation Computer Systems* 29(1): 250–261, available at: https://doi.org/10.1016/j.future.2010.12.017

Iglesia, D. G. D. L. and Weyns, D. 2015. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 10(3): 1–31, available at: https://doi.org/10.1145/2724719

Jennings, N. R., Sycara, K. and Wooldridge, M. 1998. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems* 1(1): 7–38, available at: https://doi.org/10.1023/A:1010090405266

Julian, V. and Botti, V. 2004. Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering* 11(2): 135–149, available at: https://doi.org/10.3233/ICA-2004-11204

Julian V., Carrascosa C., Rebollo M., Soler J. and Botti V. 2002. SIMBA: an approach for real-time multi-agent systems. in Escrig M. T., Toledo F., Golobardes E. (Eds), *Topics in Artificial Intelligence* Springer, Berlin, Heidelberg, pp. 282–293, available at: https://doi.org/10.1007/3-540-36079-4_25

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

Konur, S., Fisher, M. and Schewe, S. 2013. Combined model checking for temporal, probabilistic, and real-time logics. *Theoretical Computer Science* 503: 61–88, available at: https://doi.org/10.1109/MC.2003.1160055

Logenthiran, T., Srinivasan, D., Khambadkone, A. M. and Aung, H. N. 2012. Multiagent system for real-time operation of a microgrid in real-time digital simulator. *IEEE Transactions on Smart Grid* 3(2): 925–933, available at: https://doi.org/10.1109/TSG.2012.2189028

Lomuscio, A., Qu, H. and Raimondi, F. 2015. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 19(1): 1–22, available at: https://doi.org/10.1007/s10009-015-0378-x

Nair, R. R., Behera, L., Kumar, V. and 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, available at: https://doi.org/10.1109/JSYST.2014.2335442

Qasim, A., Kazmi, S. A. R. and Fakhir, I. 2015a. Executable semantics for the formal specification and verification of e-agents. *Indian Journal of Science and Technology* 8(16): 1–8, available at: https://doi.org/10.17485/ijst/2015/v8i16/55160

Qasim, A., Kazmi, S. A. R. and Fakhir, I. 2015b. Formal specification and verification of real-time multi-agent systems using timed-arc petri nets. *Advances in Electrical and Computer Engineering* 15(3): 73–78, available at: https://doi.org/10.4316/AECE.2015.03010

Qasim, A. and Kazmi, S. A. R. 2016. MAPE-K interfaces for formal modeling of real-time self-adaptive multi-agent systems. *IEEE Access* 4: 4946–4958, available at: https://doi.org/10.1109/ACCESS.2016.2592381

Reynisson, A. H., Sirjani, M., Aceto, L., Cimini, M., Jafari, A., Ingolfsdottir, A. and Sigurdarson, S. H. 2014. Modelling and simulation of asynchronous real-time systems using Timed Rebeca. *Science of Computer Programming* 89: 41–68, available at: https://doi.org/10.1016/j.scico.2014.01.008

Sun, J., Liu, Y., Dong, J. S., Liu, Y., Shi, L. and André E. 2013. Modeling and verifying hierarchical real-time systems using Stateful Timed CSP. *ACM Transactions on Software Engineering and Methodology* 22(1): 1–29, available at: https://doi.org/10.1145/2430536.2430537

Tesar, D. (2016), Next wave of technology. *Intelligent Automation & Soft Computing*, 22(2): 211–225, available at: https://doi.org/10.1080/10798587.2015.1118202

Varzaneh, H. H., Neysiani, B. S., Ziafat, H. and Soltani, N. 2018. Recommendation systems based on association rule mining for a target object by evolutionary algorithms. *Emerging Science Journal* 2(2): 100–107, available at: https://doi.org/10.28991/esj-2018-01133

Weyns, D., Malek, S. and Andersson, J. 2012. Forms: unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems* 7(1): 1–61, available at: https://doi.org/10.1145/2168260.2168268