Open Access
Full Length Article

ORIGINAL ARTICLE

# Formal Modeling of a Mail Transport System based on Multi-Agent System-of-Systems

**Nadeem Akhtar[1], Shafiq Hussain[2]**

[1]Assistant Professor, Department of Computer Science and IT, The Islamia University of Bahawalpur, Pakistan
[2]Assistant Professor/Head, Department of Computer Science and IT, University of Sahiwal, Pakistan

## ABSTRACT

The role of multi-agent System-of-Systems (SoS) has become important in modern complex systems. SoS is a composition of systems having constituent elements as independent functional autonomous systems. It is a specialized system with greater complexity having an emergent behavior. We have proposed a methodology centered on formal modeling, model checking, and formal verification of a Mail Transport System based on multi-agent SoS. The proposed multi-agent SoS is a safety-critical system that must be correct, safe, and reliable. It must ensure the safety and liveness properties of correctness. Our objective in this work is to propose a formal methodology that ensures correctness properties of safety and liveness of the Mail Transport System. Our contribution consists of specifying Gaia based formal multi-agent requirement and design specifications, the liveness properties are specified using regular expression and the safety property is specified in first-order predicate calculus. The verification of Gaia safety and liveness properties and Gaia organizational abstractions by Finite State Processes (FSP) and Labelled Transition System (LTS). Then these FSP specifications are modeled in Event-B for creating exhaustive proofs.

**Keywords:** Safety-Critical System, Multi-agent System, System-of-System (SoS); Labelled Transition System (LTS); Correctness; Safety; Liveness; Finite System Processing (FSP)

Author`s Contribution
*[1,2]Manuscript writing, Data analysis, interpretation, Conception, synthesis, planning of research, Interpretation and discussion, Data Collection*

Address of Correspondence
*Nadeem Akhtar*
*nadeem.akhtar@iub.edu.pk*

Article info.
*Received: May 26, 2018*
*Accepted: June 21, 2019*
*Published: June 30, 2019*

***Cite this article:*** *Akhtar N, Hussain S. Formal Modelling of a Mail Transport System based on Multi-Agent System of Systems. J. Inf. commun. technol. robot. appl.2019; 10(1):68-79*

***Funding Source:*** *Nil*
***Conflict of Interest:*** *Nil*

## INTRODUCTION

From the early days of computing to the present day the complexity of software and the size of the systems reliant on software have grown at a rapid rate from simple to complicated systems and then from complicated to complex systems. Presently, the systems are moving towards the inherently complex system-of-systems. The software has become complex and they are an integral part of the complex systems. With the advent of the Internet of Things (IoT), Internet of Services (IoS), Ubiquitous Computing, Pervasive Computing, and Cyber-Physical Systems the software systems have become more and more complex. In a software-intensive system, the software is essential to enable the behavior of the system. Software influences the design, construction, deployment, and evolution of the system in itself. System-of-Systems (SoS) can be perceived as a composition of systems in which its constituents elements are themselves systems [1].

[1]. the constituent elements of the SoS are independent functional systems that are autonomous, separately built or chosen. These systems are composed during construction or during run-time to form a more

complex system to accomplish a mission. The SoS forms a larger system that creates emergent behavior i.e. it performs a mission not performable by one of the constituent systems alone. Constituent systems fulfill their valid purposes and continue to operate to fulfill those purposes if disassembled from the encompassing SoS.

Intrinsic characteristics of SoS are Geographical distribution of systems; Managerial independence of systems; Operational independence of systems; Evolutionary development of the SoS; Emergent behavior of SoS. Besides, to these characteristics, the Open-World SoS has unpredictable constituents and unpredictable environment. There is a type of SoS called Collaborative SoS in which there is no central management and constituent systems work together to accomplish the role of central management.

[2] have defined an agent as "An agent as a computer system situated in some environment, capable of autonomous actions in this environment to meet its design objectives". According to [3] "multiple agents are necessary to solve a problem, especially when the problem involves distributed data, knowledge, or control. A multi-agent system is a collection of several interacting agents in which each agent has incomplete information or capabilities for solving the problem"

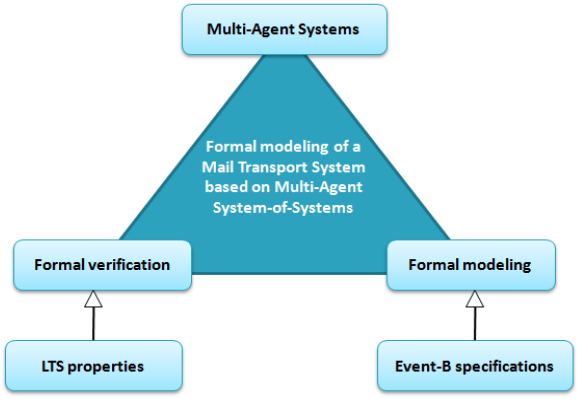



**Figure 1. The three major axis of Research**

Multi-agent SoS are complex systems and multi-agent SoS specifications have multiple levels of abstractions and refinement. One important challenge in multi-agent SoS is to ensure continuous correctness, to support their verification-centered design by a number of refinement layers starting from abstract specifications and ending with concrete specifications, then to full implementation code generation.

Gaia multi-agent methodology [4] [5] are used for the requirement analysis and design of the courier multi-agent system-of-system. It is based on interacting roles and the computation organization of these roles. Gaia design specifications are formally verified by constructing a LTS (i.e. finite automates) by LTS Analyzer [6]; theorem proving is done by using Event-B [7] [8] in RODIN [8] [9] platform. As shown in fig. 1, the proposed approach has three axis of research work that touch the domains of Multi-Agent Systems, Formal verification, and Formal modeling.

- **Problem Definition and Research Objectives**

Multi-agent SoS are specialized systems with greater complexity, autonomy, abstractions, and concurrency. Specialized formal methods, languages, and platforms are required for their exhaustive formal verification and modeling.

The formal foundation for mail management multi-agent SoS provides rigorous analysis of the correctness properties; provides error-free and property-preserving transformations and implementation, and improves the quality of the whole development process.

Two major multi-agent methodologies are Gaia [4] [5] and TROPOS [10]. Gaia recognizes organizational structure and abstractions as a primary focus for the development of agent SoS. Organizational abstractions are needed for both the functional and non-functional requirements. Presently an approach is proposed based on Gaia for requirement definition, FSP based LTS formal verification. There is a need for formal verification of the Gaia specifications of multi-agent SoS. Here we start from Gaia multi-agent method for requirement specification, design specification, and detailed design. FSP is selected for the formal verification by using LTSA and then property proving using Event-B [7] [8] in RODIN [8] [9] platform.

The proposed multi-agent SoS is a safety-critical system that must be correct, safe, and reliable. It must ensure the safety and liveness properties of correctness. Our research objective is to propose a formal methodology that ensures correctness properties of safety and liveness of the Mail Transport System. Our contribution consists of specifying Gaia based formal multi-agent requirement and design specifications, the

liveness properties are specified using regular expression and the safety property is specified in first-order predicate calculus. The verification of Gaia safety and liveness properties and Gaia organizational abstractions by model checking i.e. Finite State Processes (FSP) and Labelled Transition System (LTS). The future work is to take these LTS specifications as input and using Event-B for creating exhaustive proofs.

## LITERATURE REVIEW

### 2.2. Multi-Agent System

The methodology proposed for the Mail transport system is formal and is based on multi-agent SoS. It proposes a SoS consisting of a number of multi-agent systems. Russell and Norvig [11] defined an agent as "a flexible autonomous entity capable of perceiving the environment through the sensors connected to it.". This definition has been supported by [12]. A different perspective was presented in [13], where the authors defined an agent as "an encapsulated computational system that is situated in some environment and this is capable of flexible, autonomous action in that environment in order to meet its design objective.""

According to [14] an agent is "an entity which is placed in an environment and senses different parameters that are used to make a decision based on the goal of the entity. The entity performs the necessary action on the environment based on this decision". The above definition consists of four keywords which are described below as:

**1. Entity:** Entity defines the type of the agent.

**2. Environment:** This defines the place where the agent is located. The environment can be a software, network, or a hardware. An agent senses the changes in environment and based on these changes makes decisions [15]:

a) **Accessibility:** It defines the degree of accuracy with which an agent can sense data from the environment.

b) **Determinism:** This defines the predictability of the results of an action. Results are predictable in a deterministic environment.

c) **Dynamism:** This defines the changes that occur in the environment that are not dependent on the actions taken by the agents. In a static environment, the changes occur as a consequence of the action of the agents. In a dynamic environment changes can occur without the consequence of agent actions.

d) **Continuity:** This refers to the discreteness or continuity of the agent's environment.

**3. Parameters:** The data that an agent can sense from the environment are defined as parameters.

**4. Action**: Each agent can bring changes into the environment by an action. The following features enable agents to have broad applicability and solve complex tasks [16] [17]:

**Sociability:** Agents can interact with other agents, share knowledge, request information from other agents in order to reach their goals.

**Autonomy:** Each agent can independently execute the decision making process and take appropriate action.

**Proactivity:** Each agent predicts by using history, sensed parameters, and information of other agents.

According to [18] "multiple agents that collaborate to solve a complex task are known as multi-agent systems.

According to [19] "the salient features of multi-agent system, including efficiency, low cost, flexibility, and reliability, make it an effective solution to solve complex tasks. In these multi-agent systems a complex a complex task is divided into multiple smaller tasks, each of which is assigned to a distinct agent. Each agent can solve the assigned task with pre-defined knowledge with high degree of autonomy and flexibility".

This distributed nature of problem solving in multi-agent system results in high reliability.

In order to study multi-agent systems graphs are used to model agents and their relations Graphs have been used in artificial intelligence and software engineering for modeling complex systems and analyzing them mathematically e.g. social media [20]. In a graph each vertex is an agent and an edge between two vertices indicates agent communication. [21], [22] are complete resources to study, analyze, and design multi-agent systems as graphs.

### 2.2. Gaia Multi-Agent Methodology

Gaia [4] [5] methodology identifies organizational abstractions in a multi-agent mail transport SoS and provides detailed models for the analysis and design of

SoS. These organizational abstractions are important in the design and implementation of complex systems like SoS.

The phase of the detailed design in Gaia methodology consists of agent and services model, these models implement and instantiate agent roles. Gaia methodology consists of models (i.e. role, interaction, agent, services, and acquaintance models) and multiple levels of organizational abstractions between these models. The role model and interaction model constitute the analysis phase that plays the most important part and that can be extended by using LTS and Event-B. The analysis phase consists of preliminary role and interaction models. The design phase consists of a detailed role model, detailed interaction model, agent, services and acquaintance model.

### 2.3. Safety and Liveness Properties

Safety property is an invariant which asserts that "nothing bad happens i.e. that is an acceptable state of affairs is maintained". Safety property defines the constraints and boundaries under which the system has to function. Liveness property can be defined as “something good happens i.e. that describe the states of the system that an agent must bring about given certain conditions".

According to Gaia’s role model [5] "liveness properties are defined by expressions defining the execution trajectories through the activities and interactions associated with the role. An activity is an agent unit of action that does not involve interaction with any other agent. A Protocol is an activity that requires interaction with other agents". Temporal logic and regular expression based formal specifications are used in Gaia for specifying liveness and safety properties [4].

### 2.4. Model Checking

The proposed mail transport multi-agent SoS is a safety-critical system. Safety-critical systems are getting more complex and are playing a critical role in saving human lives and financial losses. The reliability of the system is important in the design process. The Gaia requirement specifications are translated into FSP specifications and then LTS is generated and model checked. According to [24] "Model checking is a computer-assisted method for the analysis of dynamical systems that can be modeled by state-transition systems".

Model checking [23] [24] [25] is one of the most practical formal methods for automatic, systematic and exhaustive verification. A mathematical model of the system is created and a comprehensive review of the model is performed. It includes checking all states and transitions in the model i.e. exhaustive analysis of the model. It creates an abstract representation of a system with all the possible states and transitions. The correctness of the model is verified through model checking techniques. The model takes input and systemically checks whether the system properties are ensured by the system. It is widely used in verification of critical engineering projects i.e. verification of software systems of airplanes, spacecraft, trains, subway trains, nuclear reactors, and satellites. Its overall goal is to improve the quality of verification by specifying and checking correctness properties.

According to [25] "model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model".

Model Checking is a verification technique that explores and verifies all possible system states in an exhaustive manner. A model checker systematically checks all possible system states. It performs exhaustive state-space analysis of the system. It works in a brute-force manner to check each possible state. It proposes counter example. A counter example proposes an execution path that leads from the initial state to a state that violates the property being verified [25].

### 2.5. Finite State Processes and Labelled Transition System (LTS)

Finite State Processes (FSP) is a process algebra notation for the formal, precise, rigorous, and concise description and modeling of software behavior. It has specialized constructs for parallelism, so it is ideal for specifying parallel and concurrent systems. A component consists of one or more processes and each process has a finite number of states and is composed of one or more actions. There is concurrency between processes, actions, therefore it is important to manage the communication, interactions, and synchronization between processes.

[6] Proposed LTS Analyzer i.e. toolkit for LTS. A

model of the system is built as a composition of finite state machines. The properties of these machines are specified. LTS analyzer performs compositional reach-ability analysis to exhaustively search for violations of correctness properties. FSP formalizes the specification of software components and LTS verifies the system-level concurrency properties. In multi-agent SoS an important objective is deadlock detection and prevention, and there exist significant possibilities of deadlock.

### 2.6. Event-B and Rodin

Event-B [7] [8] is a formal approach for constructing a precise, accurate, and rigorous model of a system. It is based on set theory with a few extensions. It uses set theory as a notation for formal modeling based around abstract machine notation; it uses refinement to represent systems in multiple abstraction levels; and uses mathematical proof obligations to validate consistency between different refinement levels.

Event-B can be used for the construction of complex discrete systems. The multi-agent SoS behavior can be abstracted by a number of successive states inter-mixed with jumps that cause sudden state changes. In reactive systems, transitions are occurring concurrently and rapidly, resulting into large number of concurrent changes occurring at a very high frequency. Despite a high number and frequency of changes, such systems are intrinsically discrete. They are also called transition systems.

Event-B involves modeling and formal reasoning by constructing a mathematical model which will be analyzed by mathematical proofs. Modeling is accompanied by reasoning. Model of a program also contains proofs that are related to the properties of the program. Multi-agent SoS models are complex systems and they require a high degree of correctness. Complex models are made by step-wise refinement. An Event-B model consists of contexts and machines. According to [7] "Context specifies the static aspects of the model. It contains sets, constants, axioms, and theorems. Machine specifies the dynamic aspects of the model. A machine has a state, which is defined by means of variables. A variable is constrained by an invariant i.e. if V are the variables of the machine then the invariants are I(V). Invariants are specified to hold whenever variable values change".

In order to design the multi-agent SoS a discrete model is made of the real system; this model is designed at some level of abstraction; then periodically this discrete model is detailed with the help of a number of refinements. [7] describes this discrete dynamic model as "consisting of a number of states, and transitions that are triggered under certain circumstances. Each event is composed of a guard and an action. The guard is a predicate built on the variables and state constants. It specifies the condition under which the event may occur. An event may be executed only when its guard holds. The action, signifies the way in which state variables evolve when the event occurs. An event have parameters. They can be used to model array of events or communication channel in the composition of machines".

Rodin [8] [9] platform is an integrated development environment for Event-B [7] [8]. It allows to model refinement and mathematical proof. It integrates modeling as well as exhaustive proving. It is open-source, and is extensible with plug-ins. In order to use formal modeling effectively, good tool support is of critical importance. Rodin has made a large contribution in making theorem proving a practical tool for software engineering. Rodin platform provides techniques used in programming, formal modeling to formal verifications. Instead of compilation, Rodin is centered on proof obligation generation.

### 2.7. Managing the Complexity of Closed Models

The number of transactions in the multi-agent SoS is very large, and a large number of variables are required to describe the state of such a system. This complexity is managed by using three fundamental concepts: refinement, decomposition, and generic instantiation. These concepts are linked together. A multi-agent SoS model is refined and then it is decomposed, and, it is decomposed further to refine it. Finally, a generic model is constructed which can then be instantiated.

Luckcuck et al. [26] review on formal specification and verification on autonomous robotic systems, have greatly helped us in choosing the appropriate formal methods and techniques for the formal analysis, modeling, and model-checking of the mail transport SoS.

Farrell et al. [27] describe Integrated Formal Methods (iFM) which defines the integration of multiple formal methods, or a formal method with a semi-formal method, or a formal method with a non-formal method. iFM can specify static and dynamic behavior of a system for easy

analysis. It addresses the challenges of integrating formal methods in different types of multi-agent systems by using iFM. This work helped us in integrating Gaia, FSP, LTS, and Event-B based formal specifications.

Ferrando et al. [28] has designed and developed constructs for runtime verification to recognize anomalous environmental interactions. It identifies the previous invalid formal verification specifications.

[29] describes an agent, written in GWENDOLEN language [30] controlling a driverless car in a group of vehicles. AJPF model checker [31] verifies safety properties of a car joining and leaving a group of cars, and for maintaining a safe distance during platooning. Verification of the timing properties is implemented by Uppaal [32][33]. This work has helped us in focusing on safety and liveness properties for formal verification.

[34] describes the RoboChart notation and its oolset and Isabelle/HOL robotic systems verification. Ferrando et al. [35] defines "runtime verification to recognize anomalies in agent environmental interactions. Therefore it identifies when the previous formal verification that has been carried out on an autonomous system (with some environmental assumptions) becomes invalid". This work helps us formally modeling agent interactions.

## RESEARCH METHODOLOGY

Our methodology has been used for the specification, modeling, model-checking, and verification of a mail transport system based on multi-agent SoS.

### 3.1. Mail Transport System based on Multi-Agent SoS

A mail transport system is considered which is composed of agents, these agents constitute multiple multi-agent systems working together as SoS. Each multi-agent system in itself is a complete autonomous system which can work as its own. Each multi-agent system can be fundamentally different from any other multi-agent system of the SoS. These multi-agent systems combine to form the SoS which has an emergent behavior. This behavior of the SoS is different from the behavior of the constituent multi-agent systems.

An approach for the analysis, design, formal modeling and verification of a multi-agent SoS based Mail Transport System is proposed. The approach is centered on the formal verification of the correctness properties of safety and liveness. This SoS has a number of multi-agent systems managing different sub-functions i.e. classification of letters and parcels, organization of letters and parcels of each specific city, management of the salaries of the employees, record of the available stationary, management of the damaged envelopes and parcels, management of the envelops and parcels without the destination addresses; calculation of time duration for the transport of envelops, packages and parcels; management of the transport automobiles.

The designed and developed system is centered on three fundamental areas (1) Multi-Agent Systems (2) SoS, and (3) Formal modeling and verification. In all these three areas disciplined, rigorous, organized, and formal methods are used. As shown in the block diagram of fig.2 for the analysis and design of the multi-agent SoS Gaia multi-agent methodology is selected. Therefore in the analysis, design, and detailed design of this mail transport SoS the key aspect is the organizational abstractions i.e. there are multiple levels of refinement. For the formal modeling and verification multiple methods are used together, in order to perform a rigorous, exhaustive verification. LTS are constructed for model checking, after which formal proof obligations are generated and also manually constructed by using Event-B method. Rodin platform is used for this exhaustive proof based formal verification. In these formal proofs a number of contexts and machines are constructed, having a number of refinement layers starting from an abstract model and then gradually refining it into a fully functional exhaustive complete model, thus having a number of refinement layers i.e. 10 to 15 refinement layers.

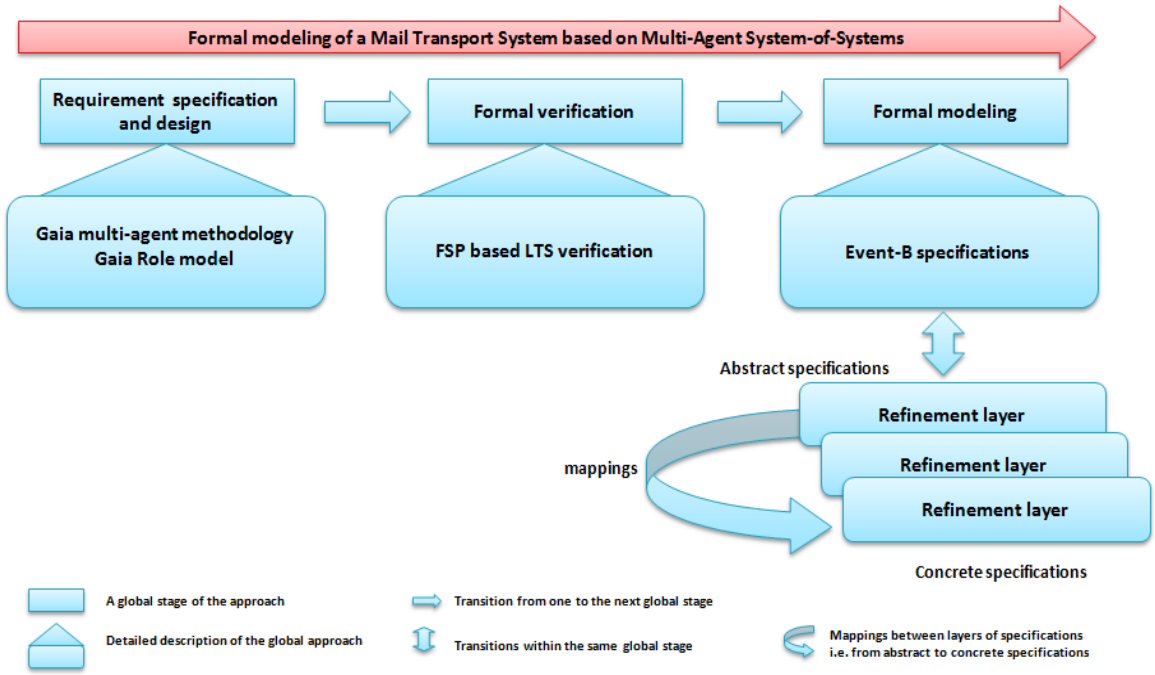



**Figure 2. The Proposed Approach**

### 3.2. Agent Roles of Multi-Agent SoS

The Gaia role model provides the organizational roles of the proposed multi-agent SoS. Each agent can have one or more roles. Each role specifies the functional properties (i.e. behavior) as well as the non-functional properties (i.e. constraints) of the role. The two fundamental properties formally specified in the agent roles model are the liveness and safety properties of correctness. The liveness property is specified by using regular expression. The protocols and activities constitute the liveness property. A protocol defines an action between two agents, whereas an activity is an action of a single agent. The safety property is specified by first-order predicate logic. Here in this paper two of the major roles are presented.

#### 3.2.1. Role of Mail Transport Agent

The role model of the mail transport agent that transports mail boxes between stock-houses.

| Role Schema: **Mail Transport Agent** |
|---|
| **Description:**<br>This role involves an agent that transports mail boxes from Stock-house A to Stock-house B |
| **Protocols and Activities:**<br>`waitForLoading, loadMailAgent, readPosition, movetoNext, readUnloadSensor, readLoadSensor, waitForUnloading, unloadMailAgent` |
| **Permissions:**<br>reads: *position_sensor (external)*<br>*changes: position (internal)* |
| **Responsibilities:**<br>***Liveness:***<br>**Transport** = `(waitForLoading.loadMailAgent.`***Move***`.waitForUnloading.unloadMailAgent.`***Move***`)+`<br>***Move*** = `(readPosition.movetoNext)+`<br>`| (readUnloadSensor)`<br>`| (readLoadSensor)` |

| Role Schema: **Mail Transport Agent** |
|---|
| ***Safety:***<br>**(Agent_Full(m) ∧ can_movetoNext(ps)) ∧ (Agent _Full(m) ∧ Agent_collision_sensor(m))**<br>**∨**<br>**(Agent_Empty(m) ∧ can_movetoNext(ps)) ∧ (Agent_Empty(m) ∧ can_movetoPrevious(ps))**<br>where m is for mail agent and ps for position sensor |

As specified in liveness property the protocols are waitForLoading, loadMailAgent, waitForUnloading, unloadMailAgent and the activities (i.e. underlined) are readPosition, movetoNext, readUnloadSensor, readLoadSensor. As specified in liveness property the mail transport agent waits for loading operation, then it reads the position sensor which gives the exact position of the mail agent in its route, then it moves to the next road partition. It repeats this move operation until the agent reads an unload-sensor. Then it stops and waits for the un-loader agent to get unloaded.

### 3.2.2. Role of Loader Agent

The role model of a loader agent that loads the mail transport agent with mail.

| Role Schema: **Load** |
|---|
| **Description:**<br>This role of the loader agent which involves loading empty carriers with mailboxes |
| **Protocols and Activities:**<br>`waitForCarrierPresence, waitForLoading, waitForStoremanager, loadCarrier` |
| **Permissions:**<br>reads: *carrier_position // position of carrier*<br>*carrier_isempty // whether the carrier is full or empty*<br>changes: *material_count // quantity of material in the storehouse* |
| **Responsibilities:**<br>***Liveness:***<br>**Load** = ***WaitForEmptyCarrier***`.waitForStoremanager.waitForLoading.loadAgent`<br>***WaitForEmptyCarrier*** = `(waitForAgentPresence)+`<br>***Safety:***<br>**carrier_isempty ∧ (material_count ≠ 0) ∧ (carrier_position = loadsign)** |

As specified in liveness property the protocols are waitForLoading, waitForStoremanager, loadCarrier and the activity (i.e. underlined) is waitForCarrierPresence. As specified in liveness property the mail loader agent waits for the empty mail transport agent. It gets mailboxes from the store-manager and as soon as the empty carrier agent arrives, the loader agent loads it with the mailbox.

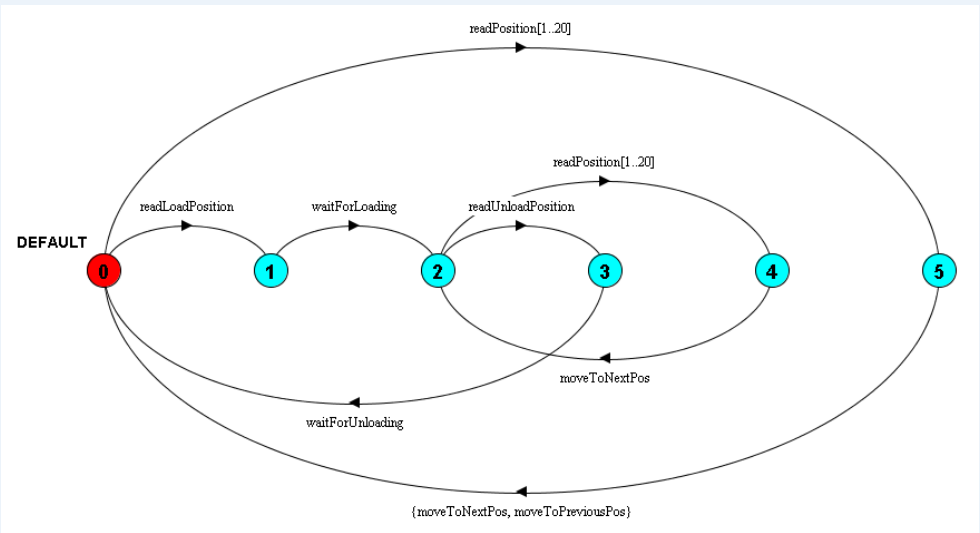
## 3.3. Construction of Labelled Transition System (LTS) for Formal verification

The liveness and safety properties of correctness are specified, modeled and verified by constructing LTS. The FSP based formal specifications are written which are then compiled and transformed into a labelled transition system by using the tool LTSA. This labelled transition system is exhaustively run, tested, and simulated.

### 3.3.1. Multi-Agent SoS Mail Transport

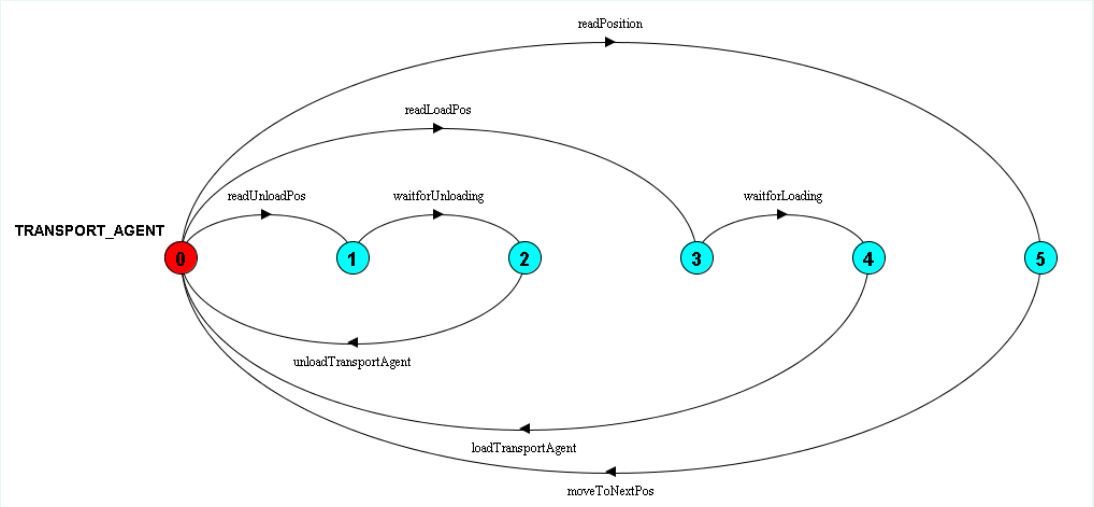
The FSP specification and the generated LTS of the mail transport agent. The environment is divided into equal sized blocks, each block is labelled with a location number. The transport agent reads its location number and moves to the next location.

```
1    /// ----------------- MultiAgent SoS -----------------///
2    range Pos = 1..20   /// Twenty Position Blocks
3    TRANSPORT_AGENT = MOVE_EMPTY,
4    MOVE_EMPTY = (  /// Empty Transport Agent
5           readPosition[p:Pos] -> [2] -> MOVE_EMPTY
6         | readLoadPosition  -> waitForLoading -> MOVE_MAIL
7     ),
8    MOVE_MAIL = (   /// Transport Agent carrying Mail
9           readPosition[p:Pos]  -> moveToNextPos -> MOVE_MAIL
10        | readUnloadPosition -> waitForUnloading -> MOVE_EMPTY
11   ).
```




The mail transport agent reads its current position, and moves to the next position. If the transport agent reads the load position then it waits at this position for the loader agent to load it with mails, and if the transport agent reads the un-loader position then it waits at this position for the un-loader agent to unload the mails.

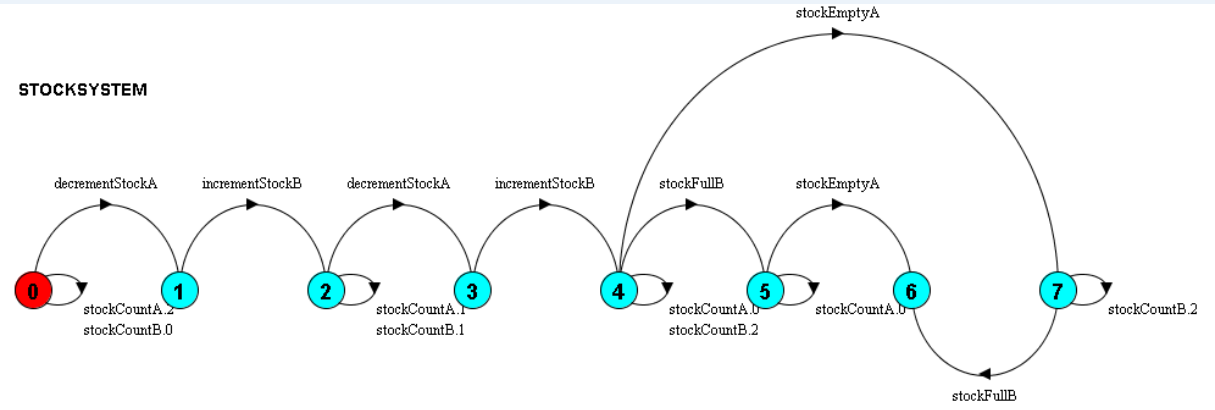
```
1    /// -------- FSP specifications independent of the road topology ------ ///
2    TRANSPORT_AGENT =      ( readPosition -> moveToNextPos -> TRANSPORT_AGENT
3         | readLoadPos -> waitforLoading -> loadTransportAgent -> TRANSPORT_AGENT
4         | readUnloadPos -> waitforUnloading -> unloadTransportAgent -> TRANSPORT_AGENT
5    ).
```



### 3.3.2. Mail Stock Management

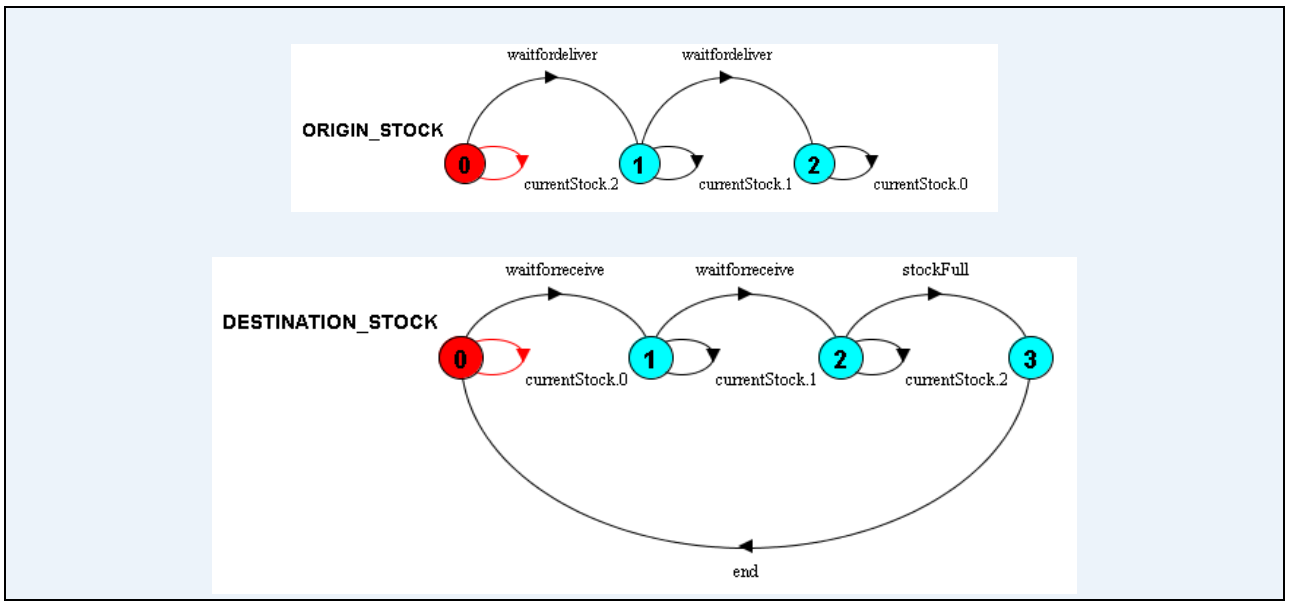
The management of mail stock between storehouse-A and storehouse-B. In order to display the LTS in this paper, the smallest case is specified in which there are only two boxes of stock. The stock management ensures that there is no loss of stock during the transport i.e. it keeps a record of the count of the stock.

```
1    /// ------ Stock management: (sum of Stock = constant) --------///
2    const MaxSt = 2      /// Maximum number. of Stock = 2 for this case
3    range St = 0..MaxS   /// Range of Mail stock
4
5    STOCK_A_MNGMNT = STOCK_A[MaxSt],
6    STOCK_A[s:St] = ( stockCountA[s] -> STOCK_A[s]
7                    | when(s == 0) stockEmptyA -> STOP
8                    | when(s > 0) decrementStockA -> send -> STOCK_A[s-1]).
9
10   STOCK_B_MNGMNT = STOCK_B[0],
11   STOCK_B[s:St] = ( stockCountB[s] -> STOCK_B[s]
12                   | when(s == MaxS) stockFullB -> STOP
13                   | when(s < MaxS) receive -> incrementStockB -> STOCK_B[s+1]).
14
15   ||STOCKSYSTEM = (STOCK_A_MNGMNT || STOCK_B_MNGMNT)
16                     /{ decrementStockA/receive, incrementStockB/send }.
```




The management of stock (i.e. envelops, packages etc) between the two storehouses i.e. storehouse-A and storehouse-B.

```
1    /// ------------------------ STOCK ----------------------///
2    const MaxS = 2    /// Maximum no. of Stock = 2 for this case
3    range St = 0..MaxS  /// Range of Mail stock
4
5    ORIGIN_STOCK = STOCKA[MaxS],
6    STOCKA[v:St] = ( when(TRUE) currentStock[v] -> STOCKA[v]
7                   | when(v >= 1) waitfordeliver -> STOCKA[v-1]
8                  ).
9
10   DESTINATION_STOCK = STOCKB[0],
11   STOCKB[v:St] = ( when(TRUE) currentStock[v] -> STOCKB[v]
12                  | when(v < MaxS) waitforreceive -> STOCKB[v+1][Clarke, 2018 #13]
13                  | when(v == MaxS) stockFull -> end -> DESTINATION_STOCK
14                ).
```




The requirement specifications of the SoS is completed by specifying agent roles. The agent roles are verified by LTS.

## CONCLUSION

This work is centered on formal modeling and verification of the liveness and safety properties of correctness of a mail transport system based on multi-agent SoS. This work has combined the advantages of (1) distributed autonomous systems i.e. multi-agent systems (2) systems having very large size and high degree of complexity like SoS (3) formal modeling and verification that ensures highest degree of correctness.

**The future work is classified into three axes:**

1. Requirement specifications by using Gaia multi-agent methodology. Integrating formal constructs of regular expression in liveness property and first-order predicate calculus in safety property of Gaia multi-agent methodology.
2. Automated transformations from requirement specification model to verification model (i.e. from role model requirement specifications to LTS based verification specifications). In addition to this automated transformation the development of formal constructs, techniques and tools for automated code generation from models.
3. Event-B based Proof obligation generation. Modeling. Event-B context, machines, and events. Making graphical tools for role definition, role transformation into LTS based state space evaluation.

- **Short-term objectives**

Graphical programming interface for role model creation, roles transformation into LTS based verification specifications.

- **Long-term objectives**

A complete formal approach from the role model specification, verification specification, and theorem proving using Event-B.

## REFERENCES

1. Oquendo, F., "Software-intensive System-of-Systems: Concepts, Paradigm, and Challenges". IRISA – The Computer Science Research Institute of Brittany, France, 2013.
2. M. Wooldridge, and N. Jennings, "Intelligent agents: Theory and practice," The Knowledge Engineering Review, vol. 10, no. 2, pp. 115-152, 1995.
3. N. R. Jennings, K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," Autonomous Agents and Multi-Agent Systems, vol. 1, no. 1, pp. 7-38, 1998.
4. M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design,"

Autonomous Agents and Multi-Agent Systems, vol. 3, no. 3, pp. 285-312, September 01, 2000.
5. F. Zambonelli, N. R. Jennings, and M. Wooldridge, “Developing multiagent systems: The Gaia methodology,” ACM Trans. Softw. Eng. Methodol., vol. 12, no. 3, pp. 317-370, 2003.
6. J. Magee, and J. Kramer, Concurrency State Models and Java Programs, pp. 374, New York, NY: John Wiley and Sons, 1999.
7. J.-R. Abrial, Modeling in Event-B: System and Software Engineering: Cambridge University Press, 2010.
8. J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, “Rodin: an open toolset for modelling and reasoning in Event-B,” International Journal on Software Tools for Technology Transfer, vol. 12, no. 6, pp. 447-466, November 01, 2010.
9. M. Jastram, and P. M. Butler, Rodin User's Handbook: Covers Rodin v.2.8: CreateSpace Independent Publishing Platform, 2014.
10. F. Giunchiglia, J. Mylopoulos, and A. Perini, "The Tropos Software Development Methodology: Processes, Models and Diagrams," Agent-Oriented Software Engineering III. pp. 162-173.
11. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, vol. 25. Egnlewood Cliffs, NJ, USA: Prentice-Hall, 1995, p. 27.
12. L. C. Jain and D. Srinivasan, Innovations in Multi-Agent Systems and Application. Springer, 2010.
13. D. Ye, M. Zhang, and A. V. Vasilakos, ``A survey of self-organization mechanisms in multiagent systems,'' IEEE Trans. Syst., Man, Cybern., Syst., vol. 47, no. 3, pp. 441-461, Mar. 2017.
14. A. Dorri, S. S. Kanhere and R. Jurdak, "Multi-Agent Systems: A Survey," in IEEE Access, vol. 6, pp. 28573-28593, 2018. doi: 10.1109/ACCESS.2018.2831228
15. M. Wooldridge, An Introduction to Multiagent Systems. New York, NY, USA: Wiley, 2009.
16. A. P. Garcia, J. Oliver, and D. Gosch, "An intelligent agent-based distributed architecture for smart-grid integrated network management," in Proc. IEEE 35th Conf. Local Comput. Netw. (LCN), pp. 1013-1018, Oct. 2010,
17. S. D. J. McArthur et al., "Multi-agent systems for power engineering applications. Part I: Concepts, approaches, and technical challenges,'' IEEE Trans. Power Syst., vol. 22, no. 4, pp. 1743-1752, Nov. 2007. doi: 10.1109/TPWRS.2007.908471
18. H. Rezaee and F. Abdollahi, ``Average consensus over high-order multiagent systems,'' IEEE Trans. Autom. Control, vol. 60, no. 11, pp. 3047-3052, Nov. 2015.
19. L. Ma, H. Min, S. Wang, Y. Liu, and S. Liao, "An overview of research in distributed attitude coordination control," IEEE / CAA J. Autom. Sinica, vol. 2, no. 2, pp. 121-133, Apr. 2015.
20. J. Qi, R. Vazquez, and M. Krstic, "Multi-agent deployment in 3-D via PDE control,'" IEEE Trans. Autom. Control, vol. 60, no. 4, pp. 891-906, Apr. 2015.
21. R. Merris, "Laplacian matrices of graphs: A survey,'" Linear Algebra Appl., vols. 197-198, pp. 143-176, Jan./Feb. 1994.
22. C. Godsil and G. F. Royle, Algebraic Graph Theory, vol. 207. Springer, 2013
23. Edmund M. Clarke Jr., Orna Grumberg , Daniel Kroening, Doron Peled, Helmut Veith, Model Checking, 2nd Edition, pp.424, Cyber Physical Systems Series, The MIT Press, ISBN:9780262038836, 2019.
24. Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. Handbook of Model Checking (1st ed.). pp.1210, Springer Publishing Company, Incorporated, ISBN:9783319105741, 2018.
25. Christel Baier and Joost-Pieter Katoen. Principles of Model Checking (Representation and Mind Series). The MIT Press, ISBN:9780262026499, 2008.
26. Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon, and Michael Fisher. Formal Specification and Verification of Autonomous Robotic Systems: A Survey. ACM Comput. Surv. vols. 52, no. 5, Article 100, pp.1-41, ACM, New YOrk, USA, October 2019 DOI: https://doi.org/10.1145/3342355
27. M. Farrell, M. Luckcuck, and M. Fisher. Robotics and Integrated Formal Methods: Necessity meets Opportunity. In C. Furia and K. Winter, editors, Integr. Form. Methods, volume 11023 of LNCS, page 10. Springer, 2018.
28. A. Ferrando, L. Dennis, D. Ancona, M. Fisher, and V. Mascardi. Recognising Assumption Violations in Autonomous Systems Verification. In Auton. Agents Multiagent Syst., 2018.
29. M. Kamali, S. Linker, and M. Fisher. Modular verification of vehicle platooning with respect to decisions, space and time. In Work. Formal Techniques for Safety-Critical Systems, volume 1008 of CCIS, pages 18–36. Springer, 2018.
30. L. Dennis and B. Farwer. Gwendolen : A BDI Language for Verifiable Agents. Log. Simul. Interact. Reason., pages 16–23, 2008.
31. L. Dennis, M. Fisher, M. Webster, and R. Bordini. Model checking agent programming languages. Autom. Softw. Eng., 19(1):5–63, 2012.
32. A. Aniculaesei, D. Arnsberger, F. Howar, and A. Rausch. Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. Theor. Comput. Sci., 232:79–90, 2016.
33. D. G. De La Iglesia and D. Weyns. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. Trans. Auton. Adapt. Syst., 10(3):15, 2015.
34. S. Foster, J. Baxter, A. Cavalcanti, A. Miyazawa, and J.Woodcock. Automating Verification of State Machines with Reactive Designs and Isabelle/UTP. arXiv Prepr. arXiv1807.08588, 2018.
35. A. Ferrando, L. Dennis, D. Ancona, M. Fisher, and V. Mascardi. Recognising Assumption Violations in Autonomous Systems Verificaion. In Auton. Agents Multiagent Syst., 2018.