Approach for the formal modeling of requirements, verification, and architecture of a multi-agent robotic system

Nadeem Akhtar
PhD, IRISA – University of South Brittany, Vannes, FRANCE
Assistant Professor, The Department of Computer Science & IT
The Islamia University of Bahawalpur, Bahawalpur, 63100, PAKISTAN

Anique Akhtar
Electrical Engineering, Graduate School of Science and Engineering
Koc University, Acarlar 26/10, Sariyer, Istanbul, Turkey

Abstract

It is important to have multi-agent robotic system specifications that ensure correctness properties of safety and liveness. As these systems have concurrency, and often have dynamic environment, the formal specification and verification of these systems along with step-wise refinement from abstract to concrete concepts play a major role in system correctness. Formal verification is used for exhaustive investigation of the system space thus ensuring that undetected failures in the behavior are excluded. We construct the system incrementally from subcomponents, based on software architecture. The challenge is to develop a safe multi-agent robotic system, more specifically to ensure the correctness properties of safety and liveness. We have proposed a development approach that allows for formal verification and evaluation during specification definition. The development process has been classified in to four major phases of requirement specifications, verification specifications, architecture specifications and implementation. This paper presents an overview of this approach.

Keywords: *Architecture Description Language (ADL), Correctness properties, Formal architecture, Formal verification, Liveness property, Multi-Agent robotic system, Safety property.*

1. Introduction

Today multi-agent robotic systems are not safe. Human lives can be lost due to errors in these systems, therefore it is important to have multi-agent robotic systems that are safe. Here by safe the emphasis is on correctness properties on the behavior of multi-agent robotic systems, the correctness properties that can be described by a combination of safety and liveness. How can safety and liveness properties be reinforced during the analysis, design, and implementation of a multi-agent robotic system? These safety and liveness properties are complimentary to each other and vital for system correctness, they can be satisfied by having a multi-agent robotic system development approach based on formal methods and languages, having major phases of requirement specifications, verification specifications, architecture specifications, and implementation [1].

Our area of research is formal methods including light-weight implementation of formal methods for the specification and verification of a multi-agent robotic system. Light-weight formal method has a degree of formalization that gives the flexibility to apply formal methods according to our implementation requirements. The objective of light-weight formalization is to provide the advantages of formalism and at the same time reduce the negative aspects of over formalization. The research domain focusing on formal methods, multi-agent systems, and robotics is identified in fig. 1.

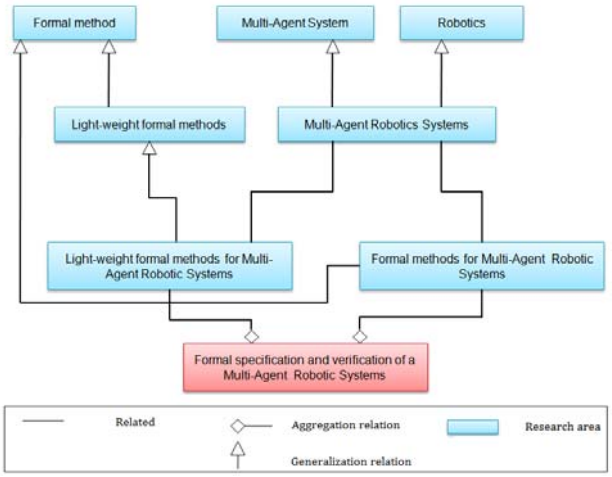

Fig. 1 Research domain

A multi-agent robotic system is distributed. A distributed system along with the interactions between its components presents a high-level of complexity, and results into complex possible system behavior in different scenarios. Complete understanding of the system behavior is required for the analysis, design, and implementation of such a system. An agent is considered as a computer system situated in some environment, capable of autonomous actions in this environment in order to meet its design objectives [18]. 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 and capabilities for solving the problem [8]. These are complex systems and their specifications involve many levels of abstractions. They have concurrency, and often have dynamic environments. To overcome the complexity problems in them and get significant results with formal analysis, we must cope with complexity at every stage of development: from the specification phase to the analysis, design and verification phase. The formal specification and verification of a multi-agent robotic system along with its step-wise refinement from abstract to concrete concepts plays an important role in system correctness. Safety and liveness properties have to be enforced during each development phase of requirement specifications, verification specifications, architecture specifications, and implementation.

2. The Problem Statement

How can a safe multi-agent robotic system be developed? Here by safe the focus is on correctness properties which can be described by a combination of safety and liveness. Thus the core question is how can safety and liveness properties be enforced during the development of a multi-agent robotic system?

3. Objective

The most challenging task in software specifications definition for robotic multi-agent systems is to ensure correctness. Safety and liveness properties are critical for system correctness. Multi-agent systems have concurrency, often have dynamic environments, therefore the formal specification and verification of these systems plays an important role in system correctness. Our objectives are formal specification, architecture, and design by considering functional and non-functional properties; the step by step refinement from abstract to concrete concepts and then formal verification of these specifications. It is important to address the following issues:

a) The formal specification of our multi-agent robotic system which has a dynamic architecture i.e. which can change during run-time;
b) To support the property-preserving transformations of agents from abstract to concrete specifications to code generation by stepwise refinement;
c) To support system verification using formal and light-weight formal methods and as a result to formally check the safety and liveness properties of the system.

In order to address the above issues, a formal approach is required which does not rely solely on immediate software development, but on continuous engineering, adaptation, and evolution of the software system.

Our objective is to propose a development approach that provides formal verification of safety and liveness properties, architecture description, and a service-oriented simulation based system implementation. It results into the development of a multi-agent robotic system that satisfies correctness properties of safety and liveness. Another objective is the formal specification, architecture, and implementation by considering the functional and non-functional properties; by refining in stepwise phases from abstract to concrete specifications along with the formal verification of these specifications.

This approach supports the efficient formal requirement specifications, verification specifications, architecture specifications, transformations, refinement from abstract to concrete concepts, and implementation of the system. The work aims to define and develop a formal architecture-based approach for the engineering of a multi-agent robotic system. The formal verification specifications i.e. verifying correctness properties of safety and liveness have been defined by labelled transition system based on finite state processes. For Formal architecture specifications, π-ADL dot NET based formal architecture is specified. The system is implemented by service-oriented architecture based simulation.

4. Contributions

Our contributions are (1) an approach based on a combination of methods to allow for formal verification and evaluation during development phases of requirement

specifications, verification specifications, architecture specifications, and implementation; (2) checking safety and liveness properties of correctness during each development phase; (3) a multi-agent robotic system case study to exemplify each phase of this approach; (4) a combination of process algebra and finite automata based techniques to define the formal specifications of our system and verifying each flow of concurrent executions.

5. Background studies

5.1 Formal methods

Formal methods are based on a solid mathematical foundation. With the passage of time as formal specification and verification techniques are getting more accomplished and mature, our capabilities to design and develop complex systems are also maturing and growing quickly. To overcome the complexity problems in multi-agent systems and get significant results with formal specifications, we must cope with complexity at each phase: requirement specification phase, architecture specification phase to design and implementation phases. We must assure formal verification during all phases. Formal verification can achieve complete exhaustive coverage of the system thus ensuring that undetected failures in the behavior are excluded. We can prove the correctness of agent software systems by formalizing critical components in the multi-agent development life-cycle. The reasons to have formal software engineering methods are:

a) Rigorous analysis of system properties;
b) Property-preserving transformations and error-free implementation
c) High quality of each phase of the development process;
d) Firm foundation for the adaptation and evolution process;
e) Continuous correctness especially as multi-agent robotic systems are concurrent and often have dynamic environments;
f) Formal specification and modeling of a multi-agent system architecture which can change at run-time;
g) Specification according to the functional and non-functional properties;
h) Property-preserving step by step transformations from abstract to concrete concepts, then stepwise refinement to implementation code;
i) Improved documentation and understanding of specifications.

5.2 Correctness properties: Safety and Liveness

The safety property is an invariant which asserts that "something bad never happens", which means that an acceptable degree of system working state is maintained. [9] have defined safety property S = {a1, a2, … , an} as a deterministic process that asserts that any trace having actions in the alphabet of S, is accepted by S. ERROR conditions are like exceptions which state what is not required, as in complex systems we specify safety properties by directly stating what is required. The liveness property asserts that "something good happens" which describes the states of a system that an agent can bring under certain conditions. Progress property P = {a1, a2, … , an} defines a property P which asserts that in an infinite execution of the system, at least one of the actions a1, a2, … , an will be often executed infinitely [6]. These properties play a vital role in system verification. Both safety and liveness properties are complementary to each other, safety alone or liveness alone is not sufficient to ensure system correctness.

5.3 Gaia multi-agent method

The Gaia [19] requirement specifications recognize the organizational structure as the core concept for the development of an agent system. A suitable choice of this organizational structure is required to meet both functional as well as non-functional requirements. It is based on organizational abstractions to drive the analysis and design of a multi-agent system, and it considers a multi-agent system as a computational organization consisting of interacting roles. These organizational abstractions play a significant role in the analysis, design, and implementation of a multi-agent system in a complex environment. For Gaia, the word method is used instead of methodology, as we consider the term methodology to be used for the study of methods. It has a concrete syntax, which can be extended to deal with the formal specification aspects of a multi-agent system, and it generates a number of models and specifications that can be used by different software development methods for implementation. After the completion of the design phase, we have a well-defined collection of agent roles to implement, and can define the agent and service model. During the specification definition we move from abstract to concrete concepts,

these abstract concepts conceptualize the system while the concrete concepts are used during the design phase, and are related to implementation.

The result of the design phase could be easily implemented in a technology neutral way. It captures the organizational structure of the system which allows for going systematically from the requirement analysis to a comprehensive design. For precise and unambiguous specifications, we need to formalize the specifications of each component and process.

Table 1: Abstract and concrete concepts in Gaia [19]

Abstract concepts	*Concrete concepts*
Roles Permissions Responsibilities Liveness properties Protocols Activities Safety properties	Agent types Services Acquaintances

a) A successful correct method has a well-defined formal basis. In fact the software engineer does not needs to know the existence of such formal basis, it is important to have a precise understanding of the terms and concepts used in a method [18].

b) Once we have designed the specification, we must be able to implement a correct system with respect to this specification. The next issue is to propose an approach to move from an abstract specification towards a concrete model. Manually refine the specification into an executable form via some formal refinement process.
   1) Directly execute the abstract specification along with its animation
   2) Translate the specification into a concrete model using automatic translation techniques.

c) It can play a significant role in the analysis and design of dynamic and open systems. In these systems components can join and leave the environment at runtime, and are composed of sub-components that may be different at design time and run time. They are a complicated class of systems to engineer [5] [7].

d) The organization structure is implicitly defined in the role and interaction models. These structures capture and represent the organization's communication and control structures.

5.4 Labelled Transition System (LTS)

Labelled transition system [9] is a technique for the formal verification and evaluation of concurrent systems. It is founded on model-checking for the verification of concurrency properties; it represents the system as a set of interacting finite state machines along with their properties; it exhaustively explores the system state space to prove correctness properties of safety and liveness, and it performs compositional reach ability analysis to exhaustively search for violations of these properties. [9] proposed an analysis tool LTSA [21] that generates labelled transition system consisting of parallel composition of asynchronous processes, interleaving interaction-shared actions.

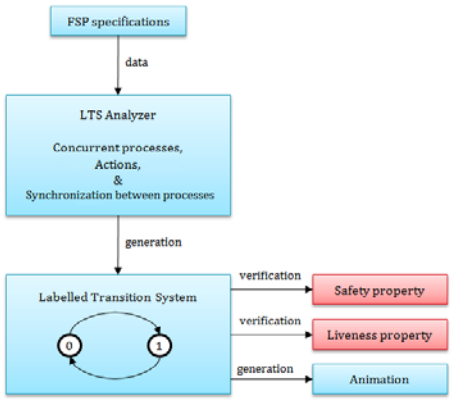

Fig. 2 The toolkit LTSA

As a result we are able to obtain a concurrent system in which there are processes working in parallel and there are synchronizations between different processes. LTSA also provides specification animation for an interactive exploration of system states.

5.5 π-ADL

The π-ADL [14] provides the software engineer with the fundamental structure and behaviors constructions for describing static as well as dynamic software architectures. It is an executable formal specification language and supports automated analysis as well as refinement of dynamic architectures. The π-ADL has as mathematical foundation the higher-order typed π-calculus [15] [11]. It is a well-formed higher-order calculus for defining dynamic and mobile architectural elements, which takes its foundation in work related to the use of π-calculus as a

semantic foundation for architecture description languages [3] [4]. According to [10], a natural solution for specifying dynamic behavior is π-calculus as it provides a computation model which is Turing-complete. It is an ideal choice for describing concurrent processes that communicate through message passing. In π-calculus every computation can take place but it is not always easy to demonstrate and express. π-ADL is a language having both structural and behavioral architecture-centric constructs, defined as an extensive version of the higher-order typed π-calculus. It achieves high architecture expressiveness and Turing completeness with the help of a simple formal syntax notation. As with any design of a language, the design of π-ADL makes tradeoffs between competing requirements and constituencies:

1) Making the language well suited for machine-automated processing for enactment, analysis, refinement and evolution vs. as a stand-alone language for humans: π-ADL is specified in a layered-approach with a core canonical abstract syntax and formal semantics, and different concrete human-oriented notations [13].
2) Making the language well suited for software architects to design large-scale software vs. making it automatic semantically tractable: π-ADL is based on a compositional approach [13].

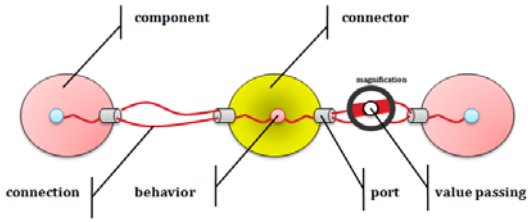

Fig. 3 Architectural concepts in π-ADL [14]

According to [13] the π-ADL design follows the following language design principles [12] [16] [17]:

1) Correspondence principle: the uses of names are consistent in π-ADL. Particularly there is a one to one relationship between the method of introducing names in declarations and parameter lists;
2) Abstraction principle: all major syntactic structures have abstractions defined over them e.g. π-ADL supports abstractions over behaviors as well as abstractions over data;
3) Data-type completeness principle: each data-type is a first-class citizen having no restrictions on its use.

6. Research methodology: The Proposed Approach

An approach has been proposed for the formal specification and verification of multi-agent robotic system. The requirements are specified, formally verified on the basis of safety and liveness properties, the architecture is specified, and the system is implemented. Our proposed approach is a combination of multi-agent methods, languages, and techniques; which takes into account the safety and liveness properties at each phase of development. This approach is exemplified by a case study of a multi-agent robotic system [1].

The proposed approach starts by the identification of components and sub-components of the system i.e. each and every part of the system that can be formally defined. Each component is formally verified and validated, particularly the critical components. The approach consists of four main development phases of requirement specification, requirement verification, architecture specification, and system implementation.

It is exemplified by a case study which is a multi-agent system composed of robotic agents. The mission is to transport goods from one storehouse to another. The robotic agents named as carrier agents transport goods from one storehouse to another. There is a possibility of collision between these carrier agents and collision resolution techniques are applied to avoid system deadlock.

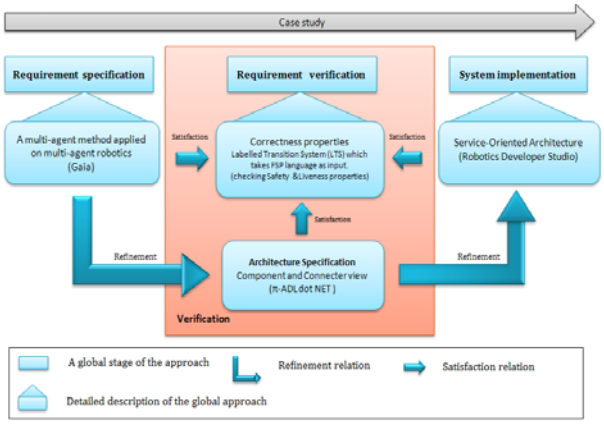

Fig. 4 A detailed view of the proposed approach

The requirements are specified by Gaia [19] based specifications. Gaia has a concrete syntax to express properties, and it is suitable to model behaviors. Formal verification of correctness properties of safety and liveness is done by defining the system as a labelled transition system which uses FSP as input. FSP based on process

algebra is a formal language which is specifically useful for specifying concurrent behavior. LTS has processes executing concurrently, with each process having one or more actions and synchronization between parallel processes by action sharing. During verification each sub-portion of the system is formally verified to make it consistent with the rest of the system, and at the end the system is verified as a whole. Transformations are made from the Gaia requirement specifications to LTS verification specifications for formal verification of the system. The architecture is specified by π-ADL dot NET language which provides a formal executable architecture model consisting of abstractions and behaviors. These architecture specifications describe the static, as well as dynamic aspects of architecture. The system is implemented by service-oriented based c# simulation implementation.

6.1 Requirement specification

The requirement specification phase starts with the identification of early requirements. It is followed by the identification of a multi-agent system as an organization. In this organization, there are multiple abstraction levels. Organizational rules are determined; these define the global constraints and rules within which the system has to work. The environmental model, which studies the environment and entities related to it, is defined. The role model has responsibilities and permissions, the responsibilities are expressed by safety and liveness properties. Agent roles are also defined. A single agent can have one or more roles but a single role cannot be performed by more than one agent. Safety and liveness properties are defined in this initial phase along with the definition of agent roles. At this phase these properties can be defined by regular expressions or by first-order predicate logic. Protocols are defined between agent roles which define the interactions between agents.

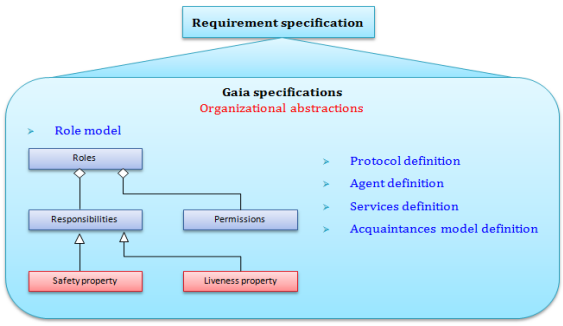

Fig. 5 Requirement specification phase

A services model is defined, where each service is defined according to input, output, pre-condition, and post-condition. The global diagram is defined which puts forward the main components along with the interactions.
These requirement specifications have a well-defined formal semantics. They capture the organizational structure of the system. They allow for going systematically from the requirement analysis to a comprehensive design. For precise and unambiguous specifications, we need precise mathematical semantics. Organization structures are defined implicitly inside these requirement specifications, within the role and interaction models. It is important to have a precise knowledge of the terms and concepts of a method. Once we specify the system, we would be able to implement a system that is correct with respect to our specifications. The next step is to move from abstract specifications to a concrete computational model. The organizational rules are defined, which put forth the global system properties and constraints, global relationships between roles, global relationships between protocols, and between roles and protocols.
During requirement specification phase, the safety properties are defined using first-order predicate logic, while liveness properties are defined using regular expressions. Here, it is to be noted that the Gaia role model does not have constructs for the formal verification of safety and liveness properties; therefore the formal verification of these properties is carried out in the next phase of our approach. The interaction model defines the protocols between agent roles; the services model puts forth the services, each service having input, output, pre-condition and post-conditions; the agent model identifies the agent instances; and at the end, the acquaintances model is defined which gives a global picture of agents along with their environment. The major emphasis throughout this phase is on the safety and liveness properties.

6.2 Requirement verification

Major emphasis is put on the requirement verification and the safety and liveness properties defined in the requirement specification phase are verified in this phase. The system is broken down into sub-components. Each component is verified by a light-weight formal model checking exhaustive method. This involves exhaustive verification of all the states, processes, and actions of each component along with its sub-component. After that all the

sub-components are assembled together and the system is verified as a whole [2].

Finite state process is a process algebra notation used for the concise description of component behavior particularly for the concurrent systems. It has strong artifacts for construction of concurrent processes, and therefore it is ideal for concurrent systems. It provides the constructs to formalize the specification of software components, each component consists of processes and each process has a finite number of states and is composed of one or more actions. The processes are modelled as a sequence of actions, and formal specification of dynamic behavioral aspects of the multi-agent robotic system are provided, correctness properties of safety and liveness are verified, along with progress property, deadlock freedom, and sequencing constraints. Concurrency exists between elementary calculator activities; processes are sequential or concurrent and there is management of the interactions, communication, and synchronization between processes.

The correctness properties of safety and liveness defined in requirement specification phase along with the multi-agent system environment are now defined as a labelled transition system. There are actions, processes, states, and transitions between states. It can be applied to a system at varying degrees ranging from a light-weight formal implementation to a concrete formal implementation. The verification specifications developed are a discrete system with a trace of actions; there are parallel processes with synchronization between them by action sharing.

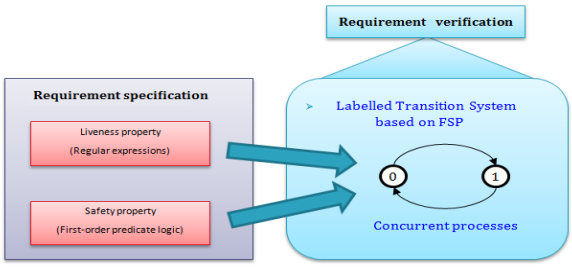

Fig. 6 Requirement verification phase

6.2.1 Moving from requirement specification to requirement verification

There is a satisfaction relation between requirement specification and requirement verification. This satisfaction relation is the formal verification of the two correctness properties of safety and liveness. This satisfaction relation is exemplified by a case study. The Gaia role model liveness and safety properties along with the organizational rules are specified in the form of finite automates for verification.

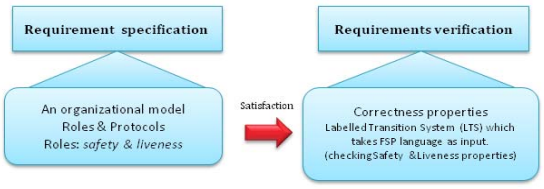

Fig. 7 Moving from requirement specification to requirement verification

6.3 Architecture specification

A formal architecture has been proposed which specifies the static, as well as the dynamic aspects of the system. In this architecture, the architectural elements are identified. All these architectural elements are separately specified and then connected together to represent the system as one unit.

The system architecture is based on π-ADL dot NET which is a dot NET extension of π-ADL. The components and their connectors are defined. It provides an executable model of system specifications consisting of abstractions and behaviors. We have an architecture that is formal, consisting of components and connectors, that executes and that can change dynamically during the executions. The π-ADL dot NET platform provides the core structure and behavior constructs for describing static, as well as the dynamic software architecture.

1) These architecture specifications provide a formal system having a mathematical foundation that can be used to describe static as well as the dynamic software architecture.
2) They have as formal foundation the higher-order typed π-calculus [15]. It is a well-formed higher-order calculus for defining communicating and mobile architectural elements.
3) They focus on the formal specification of architecture from the run-time perspective: the run-time structure, the run-time behavior, and the evolution of architecture over time.
4) They are executable i.e. a virtual machine executes the software architectures specifications.
5) They support multiple concrete syntaxes: both textual and graphical notations.
6) They support automated verification of properties by model checking.

6.3.1 Moving from requirement specification to architecture verification

The architecture specifications are based on requirement specifications. When we move from requirement to architecture then the safety and liveness properties should be preserved. The whole system is represented in the form of π-ADL dot NET with emphasis on safety and liveness properties.

6.3.2 Moving from architecture specifications to simulation implementation

The system is implemented as a simulation which reflects the architecture specifications; from π-ADL based system to SOA based robotic simulation system. There should be conformations between the properties of architecture specification and simulation implementation. We have not defined the rules and regulations that govern the transformation from π-ADL based architecture specifications to the SOA based simulation implementation.

6.4 System implementation

The system is implemented as a simulation. This simulation is based on SOA with each component as a service, with components having one or more sub-components with every sub-component implemented by a service. These services are loosely integrated and are orchestrated together by an orchestration service. As a result, we have a system that has reusable components.

This SOA based simulation is implemented by programming C# based Microsoft Robotics Developer Studio services. A refinement relation has been defined between the architecture specifications and these C# based services. It is an implementation of the LTS specifications in a simulation environment. In MRDS, each and every application is a service. An application is a composition of loosely-coupled concurrently executing components. For example: the carrier robot consists of a number of services orchestrated together. It has two wheels with a motor, sensors comprising of two bumpers for collision detection, and infrared laser for distance measurement and collision avoidance. Each of these motor, bumper, and infrared lasers is implemented by a service. There is a service for the orchestration of these sensors, motor, and actuator.

6.4.1 Moving from implementation to verification specification

The robotics simulation implementation specifications must satisfy the finite automata based LTS system. Both implementation and verification specifications should preserve the safety and liveness properties. These LTS properties should also be preserved during the simulation implementation.

The simulation is continuous with a continuous flow of actions. Each part of the simulation is a service, and there is also the orchestration of services. On the other hand, the LTS based system is a much lower abstraction level and is a discrete system. A system with concurrent processes and each process having discrete actions. There are discrete states and the system moves from one state to another. The continuous simulation must satisfy the discrete LTS system.

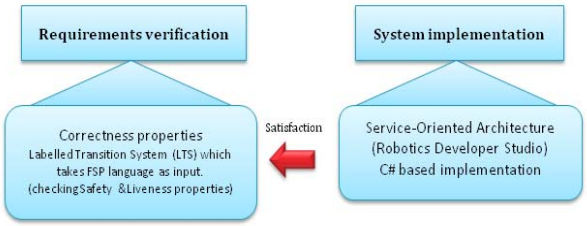

Fig. 8 Satisfaction relation from implementation to requirement verification

The safety and liveness properties should be preserved in both, and there should be a clear relationship between the two systems. The simulation specifications are able to satisfy the verification specifications. Practically, during our C# based robotics simulation model, we create a trace of actions that is equivalent to the trace of actions created by LTS specifications. A mapping of activities provides trace equivalence among requirement specification, verification specification, architecture specification, and implementations.

7. Future objectives

The work to be carried out by us in the future is classified into three axes as shown below in fig. 10.

1) The automated transformations from one model to another. The development of constructs and tools to generate code automatically from models.
2) Incorporating new versions of the proposed approach. Making improvements in the current approach and proposing new versions which have improved better constructs.
3) Making graphical tools which may lead to an easy drag and drop graphical programming interface for a robotic application development based on π-ADL

dot NET language. This programming interface allows novice programmers to program graphically without having knowledge of the underlying rigorous formal methods and languages.

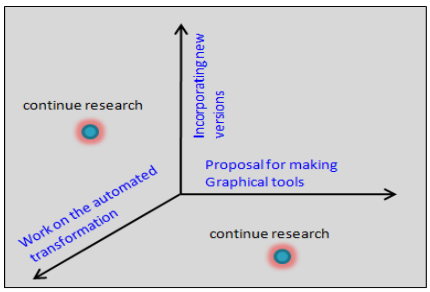

Fig. 9 Future work: three axes of work to be done

8. Concluding notes

The major contribution is the development of an ap-proach for a multi-agent robotic system that satisfy qualities of correctness i.e. safety and liveness property. An approach based on a combination of methods, languages, and technologies is proposed to support the efficient formal description, requirements gathering, formal specification definition, transformation and refinement from abstract to concrete concepts, verification of multi-agent robotic systems. The development also depends upon the degree of formalism, the more the degree of formalism, the more the cost is, as formalism needs time, expertise, and human resources. The proposed formal approach phases have key aspects of: organizational abstractions, organizational rules, requirement specifications, role model specifications, protocol definitions, formal requirement verification on the basis of correctness properties of safety and liveness, formal validation, LTS generation, formal static architecture specifications, formal dynamic architecture specifications, and SOA-based simulation implementation. This approach has models based on formal methods and it revolves around formal verification of correctness properties (i.e. safety and liveness) in each phase from early requirements to the implementation i.e. Gaia multi-agent method requirement specifications, FSP-based LTS verification specifications, π-ADL dot NET language based architecture specifications, and service-oriented architecture based C# implementation specifications. Through the goal of facilitating greater assurance to component's correctness, the processes can collaboratively integrate into distributed environment.

Acknowledgement

We are grateful to Prof. Dr. Muhammad Mukhtar, Vice Chanceller, The Islamia University of Bahawalpur for his support and motivation.
We would like to thank Prof. Dr. Flavio Oquendo for his guidance. He is a Full Professor of Computer Science and Software Engineering at the University of South Brittany, part of the European University of Brittany, France, where he leads the ARCHLOG research team on Software Architecture at the IRISA Research Lab.
We would like to thank Dr. Yann Le-Guyadec, Associate Professor of Computer Science and Software Engineering at the University of South Brittany, part of the European University of Brittany, France.

References

[1] Akhtar, N., "Contribution to the formal specification and verification of multi-agent robotic systems". *PhD thesis*, Ecole Doctorale, Lab. VALORIA, University of South Brittany, 2010.

[2] Akhtar, N., Guyadec, Y. L., and Oquendo, F., "FORMAL SPECIFICATION AND VERIFICATION OF MULTI-AGENT ROBOTICS SOFTWARE SYSTEMS: A Case Study". *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 09)*. Porto, Portugal, January 19-21. INSTICC Press, 2009.

[3] Chaudet, C., Oquendo, F., "A Formal Architecture Description Language Based on Process Algebra for Evolving Software Systems". *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00)*. IEEE Computer Society, Grenoble, 2000.

[4] Chaudet, C., Greenwood, M., Oquendo, F., Warboys, B., "Architecture-Driven Software Engineering: Specifying, Generating, and Evolving Component-Based Software Systems". IEEE Journal: Software Engineering, Vol. 147, No. 6, UK, 2000.

[5] Gasser, L., "Social conceptions of knowledge and action: DAI foundations and open systems semantics", *Artificial Intelligence*, vol. 47, pp. 107.138, 1991.

[6] Giannakopoulou, D., Magee, J. and Kramer. J., "Fairness and priority in progress property analysis", *Technical report*, Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queens Gate, London SW7 2BZ, UK, 1999.

[7] Hewitt, C., "Open information systems semantics for distributed artificial intelligence", *Artificial Intelligence*, vol. 47, pp. 79-106, 1991.

[8] Jennings, N., Sycara, K., and Wooldridge, M., "A roadmap of agent research and development". *Int. Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[9] Magee, J. and Kramer, J., "Concurrency: State Models and Java Programs". *John Wiley and Sons*, 2nd edition, 2006.

[10] Milner, R., "Communicating and Mobile Systems: The π-Calculus".

Cambridge University Press, 1999.

[11] Milner, R., Parrow, J. and Walker, D., "A Calculus of Mobile Processes". *Information and Computation*. 100(1), pp. 1-77, 1992.

[12] Morrison, R., "On the Development of S-algol". *Ph.D Thesis*, University of St Andrews, 1979.

[13] Oquendo, F., "Tutorial on ArchWare ADL – Version 2". *ArchWare European RTD Project IST-2001-32360*, 2005.

[14] Oquendo, F., "π-ADL: An Architecture Description Language based on the Higher Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures". *ACM Software Engineering Notes*, Vol. 29, No. 3, 2004.

[15] Sangiorgi, D., "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms". *PhD Thesis*, University of Edinburgh, 1992.

[16] Strachey, C., "Fundamental Concepts in Programming Languages", *Oxford University Press*, Oxford, 1967.

[17] Tennent, R.D., "Language Design Methods based on Semantic Principles". *Acta Informatica 8*, 1977.

[18] Wooldridge, M. and Jennings, N. R., "Intelligent agents: Theory and Practice". *Knowledge Engineering Review*, 10(2):115-152, 1995.

[19] Zambonelli, F., Jennings, N. R., and Wooldridge, M., "Developing Multiagent Systems: The Gaia Methodology". *ACM Transactions on Software Engineering and Methodology*, 12(3):317-370, 2003.

[20] FIPA, "Foundation for Intelligent Physical Agents", http://www.fipa.org/ .

[21] LTSA, "Labelled Transition System Analyser", http://www.doc.ic.ac.uk/ltsa/, 2006.

Dr. Nadeem Akhtar is Assistant Professor at the Department of Computer Science & IT, The Islamia University of Bahawalpur. He completed his PhD from the research Lab. VALORIA of Computer Science, University of South Brittany (UBS), France in 2010. His research areas are formal specification, formal architecture, and service-oriented architecture for robotics.