Petri net and Rewriting Logic Based Formal Analysis of Multi-Agent Based Safety-Critical Systems

Ammar Boucherit[a,*], Laura M. Castro[b], Abdallah Khababa[c], Osman Hasan[d]

[a] *Computer Science Department, University of El-Oued, Algeria*
[b] *Computer Science Department, A Coruña University, Spain*
[c] *Computer Science Department, University of Ferhat Abbas Sétif, Algeria*
[d] *School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Pakistan*

Abstract

The formal design and development of multi-agent systems has attracted a considerable attention over the past decades because of their extensive use in safety-critical applications. This paper presents an efficient, hybrid and scalable formal development approach for safety-critical systems based on the multi-agent paradigm. In fact, we aim in this paper to benefit from the advantages of existing tools and techniques for each development stage and then integrate them in one unified approach. In particular approach, we advocate using Petri nets and rewriting logic to facilitate the formalization of multi-agent based systems, as well as we have integrated both the model checking and property-based testing techniques in the verification and testing stages. For illustrating the utilization and effectiveness of the proposed approach, we use it to analyze a simple automated distributing machine.

Keywords: Multi-Agent System, Model-checking, Property-Based Testing, Petri net, Rewriting Logic.

1. Introduction

Over the past decades, the technological expansion in computer science and electronic domains have led to a variety of innovative software systems; later, so-called the revolution of electronic digital computer. Thereafter, software systems are become increasingly present in our environment for personal use, such as microwaves, mobile phones, washing machines, digital cameras, as well as collective such as military, medical and aeronautics. In fact, these systems have not only facilitated to make our daily activities more comfortable, but it is becoming so clear that our lives are shaped and increasingly controlled by software systems

*corresponding author

Email addresses: ammar-boucherit@univ-eloued.dz (Ammar Boucherit), lcastro@udc.es (Laura M. Castro), akhababa@univ-setif.dz (Abdallah Khababa), osman.hasan@seecs.nust.edu.pk (Osman Hasan)

till the extent that we cannot even imagine our world without them. However, such software systems are also responsible for an ever-increasing number of serious errors especially when it comes to safety-critical systems.
In this context, the use of multi-agent systems (MAS) has proven to be one of the most relevant approach to promote the development of real-time [1, 2] and complex systems [3, 4]. Nevertheless, due to the absence of available agent-specific safety analysis methods, it is so risky to use alone and trust agent-oriented methods in situations where safety is critical [5, 6]. Hence, the main interest in developing software for such systems is not just confined to clearly defining the system functions, interaction between components and properties of interest, but primordially on how one can identify and eliminate system errors in each development stage in order to ensure a successful implementation of the overall system.
To meet the high quality level and safety objectives satisfying the rising complexity of safety-critical systems, the use of adequate design approaches that support decentralized processing as well as rigorous analysis techniques and methods necessary to specify, design, implement and assess safety-related software systems, is gaining more and more importance.

Motivation

The present work is motivated by a desire to bridge the gap between formal specification, verification and testing during the design and development of multi-agent based critical systems. In this regard, the following issues must be addressed to increase the trustworthiness of safety-critical systems :

i. The choice of the formalism for behavioral modeling of the safety-critical system.
ii. The choice of a verification technique to ensure the correctness of the proposed model before proceeding to the system realization.
iii. The choice of the code testing technique to ensure the absence of error before its implementation.

In this context, Multi-Agent Systems (MAS) represent an innovative paradigm for modeling, simulating and designing complex distributed systems [7, 8]. Their effectiveness is due to the ability of agents to represent the system entities, their behaviors and interactions. In fact, an agent has proactive and reactive features that are very useful in decision making. In addition, a key characteristic of MAS is the collective intelligence or distributed intelligence. In this context, it is well known in the domain of MAS that a single ant or bee (agent) is not smart, but their colonies (MAS) are characterized by numerous additional capabilities [9], such as solving complex tasks and finding the shortest path to the food source. Moreover, MAS can continue to function even with the failure of one of its components (agent), without degrading the system as a whole. This last feature is extremely important in the case of safety-critical systems. Therefore, the use of multi-agent systems is very advantageous for modeling safety-critical systems[10, 11, 12]. Nevertheless, given the lack of a formal, standardized method for the development of agent-based systems, software engineering for agent-based critical systems is quite challenging.
Then, Petri nets [13] represent a well-defined theoretical model of concurrency in which the temporal ordering and causal relationships between system actions can be easily represented.

Therefore, Petri nets are widely accepted for modeling and analyzing complex, dynamic and concurrent systems, like command and control systems [14], industrial real-time applications [15] and safety-critical systems [16, 17]. In addition and in the context of MAS, Petri nets are one of the most reliable formalisms for simulation and analysis of both behavioral modeling and failure analysis of MAS [18]. One of the distinguishing features of Petri nets is that they provide a graphical representation of the dynamic behavior of systems by enabling visualization of the state changes in the modeled system that greatly facilitates the analysis process [19]. These features of Petri nets are the key reason behind their wide popularity in the context of MAS [20, 21]. However, this potential of Petri nets should not deter the inherited difficulties and the increasing need for formal methods [22] and tools facilitating the analysis of large scale Petri net models.
For this purpose, we advocate the use of Maude [23] on the specification stage as a formal specification language based on rewriting logic [24] for describing Petri nets based MAS models. However, the existing rewriting logic based framework for petri nets [25] needs some enhancements, namely, additional operators to facilitate counting or testing the number of tokens in a Petri net place. In addition, Petri nets require automatic tools to facilitate the transcription of their models into Maude specifications. Such operation is usually time consuming and error-prone in the case of large scale and complex models when performed manually. In order to overcome these limitations, we propose to benefit from this work [26], which presents an efficient algorithm allowing the automatic generation of Maude specifications for multi-agent system models based on Petri nets. Moreover, the choice of using Maude is also motivated by the availability of an integrated tool set that facilitates the formal verification of Petri net models, such as Maude model-checker [27], the reachability analysis tool and Maude's inductive theorem prover (ITP) [28].
Similarly, we advocate the use of property-based testing (PBT) [29], which is an automatic verification technique of the proposed code skeletons based on random generation of test sequences (i.e. test scenarios). Contrary to the classic testing, the PBT technique is used to automatically test code skeletons or algorithms before implementation.
The main objective of this work is on the formal analysis of MAS based safety-critical systems by the proposition of a new approach that covers almost all the development stages. To the best of our knowledge, both the enhanced Petri net specification and the combination of model checking with PBT techniques has never been used in similar context before.
The remainder of this paper is structured as follows: Section 2 discusses some related works. Then, Section 3 presents a literature review about MAS formalization. In Section 4, we briefly present our proposed approach for the formal analysis of multi-agent based safety-critical systems followed by a simple case study to show its feasibility in Section 5. Finally, Section 6 concludes the paper and presents some perspectives for future work.

2. Related works

We mean by safety-critical systems all systems in which failures may lead to catastrophic consequences, such as loss of human life, significant damage to the environment, and/or financial disasters. In such a case, multi-agent systems have been widely used to better

developing [30, 31], simulating [32], securing [33] and analyzing [34] of critical systems.
Firstly, in [10], the main objective was to propose a formal methodology in order to ensure the correctness properties of safety and liveness of the Mail Transport System. While specifying Gaia based multi-agent requirement and design specifications, authors used regular expression in specifying the liveness properties and the first-order predicate calculus to specify the safety property. In the verification stage, The Finite State Processes (FSP) and Labelled Transition System (LTS) are used. Then, Event-B is exploited for creating exhaustive proofs.
Secondly, a multi-agent approach is proposed in [12] to ensure the reliability problem of an Air Traffic Control (ATC). Therefore, an agent-based decision-aided system was developed to help controllers in using their multiple software tools in critical situations such as the unavailability of some tools due to technical incidents. Practically, a simulation environment is used to test the MAS and demonstrate its significance to air traffic controllers.
Thirdly, a simulation-based hazard analysis method using multi-agent modeling has been presented in [35] to explore the effects of deviant node behaviour within a System of System (SoS). In comparison with traditional hazard analysis techniques, the presented method in this paper has demonstrated its powerfulness to reveal hidden and undiscovered hazards.
In addition, a formal development of a hospital MAS by refinement in Event-B has been presented in [36]. In this work, authors focused on modelling and verification of some safety critical activities such as consistent updates of patient data and handling emergencies. The main properties of MAS have been formalized and demonstrated using Event-B. However, due to the autonomous agent behaviour, the highly dynamic nature of a hospital and volatile error-prone communication environment, ensuring correctness of these activities was not a trivial task.
Moreover, in [37], agent-based modeling has been investigated to study and simulate the interdependencies in critical infrastructures. In fact, authors had proposed to adopt UML for better representing the interdependencies between the different components of the critical system. Thereafter, it has been exploited to simulate such complex systems through multi-agent paradigm.
Analyzing the characteristics of the presented works, it is possible to conclude that a promising approach for developing multi-agent based safety-critical system will be very advantageous if it based on Petri nets as a well-defined theoretical model of concurrency on the modeling stage. Then, the use of rewriting logic at the specification stage will provide many benefits for both specification and verification purposes because of the Maude associated formal analysis tools. Finally, the integration of a powerful technique to ensure the absence of code error during the system implementation will raise the use of such approach in critical system context.

3. Multi-Agent Based Systems Formalization : Literature Review

Formalization is a process that starts with the construction of a formal model (a precise description of the system structure, services and relationships) of the targeted system. Its purpose is to assist designers in proving the conformity of the system with its specification.

In this context, a large number of approaches, programming languages and graphical (or textual) formalisms have been used to describe all aspects of multi-agent systems such as CASL [38], SLABS [39], XABSL [40] and others [41, 42]. Nevertheless, it would be difficult to present an exhaustive list of all these works, and we therefore recall only a few works that we consider very important from the point of view of their relevance and their widespread usage.

3.1. Specification Approaches

On the basis of specification formalisms and languages, one can easily find that Petri nets are still considered as one of the most used formalisms to help designers formalize and simulate MAS models [21, 43]. For instances, Petri nets and derivatives have been widely used to describe the social and behavioral aspects of agents [44, 45], interaction protocols [46] as well as cooperation and coordination in multi-agent systems [47]. Indeed, Petri nets have also been favored for the analysis of multi-agent systems because of the exact mathematical definition of their semantics of execution as well as the theory developed around them. The corresponding Petri net models of the multi-agent based systems are evaluated using the existing analysis methodologies and simulation techniques for Petri nets [48, 49].
Similarly, Maude has been used in several works for the formal specification of multi-agent systems [50, 51]. In addition, Maude was used to implement a (simplified language of) cognitive agent programming language 3APL in [52]. This work has been enhanced for prototyping the BUpL agent programming language (Belief Update programming language) and for executing agent programs [53]. Moreover, Maude is used in [54, 55] to formalize their agent based models, which are then simulated and verified using associated tools. However, the main disadvantage of all these works is that the translation of the models of the multi-agent systems to a Maude specification is done manually and in an ad hoc manner. It can therefore be quite difficult to ensure that this specification is a true description of the models of multi-agent systems studied especially in the case of complex models.
In this context, UML [56, 57] is widely used to describe the different aspects of multi-agent systems. In addition, the UML language has been a subject of several extensions to capture other aspects of multi-agent systems. For example, AML [58], MAS-ML [59] and AUML [60]. Moreover, many other works have emerged to develop a special Object Constraint Language (OCL) [61, 62] allowing to reduce the ambiguity and the misunderstanding of the models by specifying additional constraints on the behavior of the components of the studied system.
Similarly, architecture description languages (ADLs) are a particular type of languages that provide a concrete syntax for specifying system architectures in a more abstract, flexible and robust way than other traditional approaches. They have been widely used for the description of architectures of multi-agent systems [63, 64].
Except some works that propose a formal UML (or ADL) based description for MAS [65, 66], the major problem with most ADLs and UML is that they lack formal semantics, explicit support for running a description of the architecture, and ensuring compliance between a system and a user-specified architecture. For instance, some works were focused on the translation or combination of UML [67, 68] and ADLs [69, 70] into Petri nets to benefit

from the existing formal analysis tools. Other authors attempted to translate UML diagrams [71, 72] and software architecture description language models [73, 74] into rewriting logic. Moreover, some authors proposed rewriting logic (resp, Petri net) based ADLs [75, 76] (resp, [77, 78]) in order to obtain a formal description and simulation mechanism for the studied system architecture.

3.2. Verification Approaches

The verification of multi-agent systems is gaining more importance [79, 80]. For instance, a considerable amount of work has been done to verify the behavior of agents [81, 82], analyzing interaction protocols and their communication languages [83, 84] as well as to simulate and control security in multi-agent systems [85, 86]. However, we can see that the existing research works are generally directed towards the creation of verification tools for multi-agent systems, otherwise, they are based on existing model checkers. Table 1 presents a summary of the most known model-checkers based works, alongside the used formalism for system and/or property specification.

Work	**System Specification**	**Property Specification**	**Verification Tool**	**Year**
Wooldridge et al. [87]	MABLE	LORA	SPIN	2002
Benerecetti et al. [88]	Binary Decision Diagrams	MATL	NuMAS	2002
Bordini et al. [89, 90]	AgentSpeak(F)	LTL	SPIN	2003
Gammie et al. [91]	Binary Decision Diagrams	CTL / LTL	MCK	2004
Lomuscio et al. [92]	Interpreted Systems Programming Language (ISPL)	Alternating-time Temporal Logic (ATL)	MCMAS	2006
Belala et al. [93]	Multi-Formalisms	LTL	Maude Model-Checker	2006
Riemsdijk et al. [52]	3APL	LTL	Maude Model-Checker	2006
Lomuscio et al. [94]	Interpreted Systems	CTLK	NuSMV	2007
Rodion et al. [95]	Alloy	Alloy's relational algebra	Alloy Analyzer	2007
Boudiaf et al. [55]	Rewriting Theories	LTL	Maude Model-Checker	2008
Nabialek et al. [96]	Multi-Formalisms	CTLKD	VerICS	2008
Astefanoaei et al. [97]	Normative MAS Language	LTL	Maude Model-Checker	2009
D'Souza et al. [98]	State Chart Diagrams	CTL	NuSMV2	2012
Laouadi et al. [54]	Agent UML	LTL	Maude Model-Checker	2017
Jeremy et al. [99]	Interpreted Systems	CTLK	MCMAS	2017

Table 1: Summary of works using model-checkers for MAS verification

3.3. Synthesis

The specification of a multi-agent system involves the identification of a large number of entities and their relationships. For this, several types of formalisms have been used to construct the most appropriate system model to verify the properties in question and not to be limited — if necessary — on the use of a single formalism so to manage the different perspectives of the system [100].
In this context, we can easily deduce that Petri nets and rewriting logic are quite well-suited for MAS modeling and specification. At the verification stage, Maude is widely used as a formal semantic framework in many works for the verification of multi-agent systems and therefore its use is very appropriate and endorsed.
In addition, within the scope of enhancing multi-agent based systems quality, numerous other testing based works have been realized [101, 102]. However, it is better if testing strategies are combined (used jointly) with model checking based ones in order to better guarantee the system implementation quality.
Based on the literature review about MAS formalization, the fundamental steps of a formal analysis approach of multi-agent based systems can be summarized in the following three points:

i. Clearly define the structure and behavior of the system using the appropriate modeling formalisms and/or the most adapted formal specification languages.
ii. Identify the risks and errors of the system — which may be catastrophic or expensive — to support and better verify its corresponding properties using formal verification techniques.
iii. Ensure that the system implementation meets the specification and that the proposed code will not make errors using advanced testing techniques.

4. Proposed Formal Analysis Approach of Multi-Agent Based Safety-Critical Systems

The proposed approach for the formal development and analysis of behavioral aspect of multi-agent based safety-critical systems contemplates the following phases of software development: modeling, specification, validation, and testing. The next subsections provide an overview of each stage of the proposed approach.

4.1. Modeling Stage

The goal of this first stage is to construct the Petri net model formalizing the dynamic behavior of the studied system. In addition, the designer uses rewriting logic, which is a very expressive logic to give formal semantics when specifying Petri net models. Depending on the system studied, a system may have several operational phases, generally: startup, initialization, processing etc. Usually, each operational phase may have its own model. Therefore, it is necessary in the case of some complex systems to built more than one model, and according to the property or aspect to be verified, the designer must prepare an independent (separate) Petri net model before the complete model is developed.

4.2. Specification Stage

This stage is intended to prepare two specification levels :

1) System Specification: The main objective of the specification level is to translate the proposed model into rewriting logic by using an enhanced proposed specification for Petri nets based on rewriting logic.
2) Properties Specification: If the aim of system specification level, is to give a more or less abstract description of the system then the system can be formally defined by its properties. In this step, we refer to the created model and its specification to prepare a module that defines the set of predicates expressed in standard LTL propositional logic. These predicates are considered by the Maude's model-checker tool as the set of verified properties in the system. Then, the set of properties to be checked must be expressed in LTL as well.

4.3. Verification Stage

The verification stage is used to show that the system satisfies the desired properties and that it exhibits a stable behavior, and/or certifies that the probable malfunctions of the system will not causes damages.
The following two verification tools may be used in this stage:

- *LTL Model-Checker* : It checks the properties of a model by expressing them in linear temporal logic (LTL).

- *Search Tool* : This tool may overcome some of the shortcomings of the model-checking technique. For instance, it allows designers to check the absence/existence of some worst-cases — offered by experts in the field — that may not be expressible in LTL. In addition, it presents all the possible scenarios for the such worst-cases if they exist, which is not the case for the model-checker counter example (single scenario).

4.4. Testing Phase

The more complex a software system is, the more difficult (or unfeasible) its formal verification becomes. In addition, even if the model has been completely verified by model checking techniques, its full realization also needs to be tested. In the final stage of our approach, we advocate to use property-based testing to test the proposed implementation of the SUT. Therefore, we define all desired properties in addition to those that have not been verified in earlier phases as universally quantified expressions, and devote as much time as available to automatically generate, run, and evaluate them as test cases. In this step, we propose to use QuickCheck's capabilities to produce shunk (i.e. simplified) counterexamples when a specific test case is found to show how that the realization of the system violates a given property.

5. Case Study

5.1. System Description and Modeling

To show the practical usefulness and effectiveness of the proposed approach, we consider a simple automated distributing machine placed in a hotel to provide coffee, cakes and beverage items to hotel visitors. The visitor has to use his magnetic hotel visitor card to gain services from the machine.

Such a machine is daily programmed to deliver a cup of coffee, cake with a bottle of water and juice (***Service 1***) to the visitor upon the first usage of his card. Otherwise, the machine delivers only coffee and cake (***Service 2***). For that, the machine tests the inserted card to check its validity and determines if it is the first use for a card or no. Thereafter, the corresponding service is delivered.

For the sake of simplicity, we assume that the visitor card contains a bar code with 14 digits where the addition of digits in positions (1, 2, 4, 8) equals to 20. In addition, the card codes of the present visitors in the hotels are inserted daily in a list into the memory of the machine and when a valid visitor card is used, it will be automatically deleted from this list. Such a mechanism facilitates the process to deliver ***Service 1*** or ***Service 2*** to a visitor.

Therefore, in both cases (***Testing card validity*** and ***Determining services***), two algorithms are used for verification. Figure. 1 shows the Petri net that models the described machine, and Table 2 presents the meaning of places and transitions.

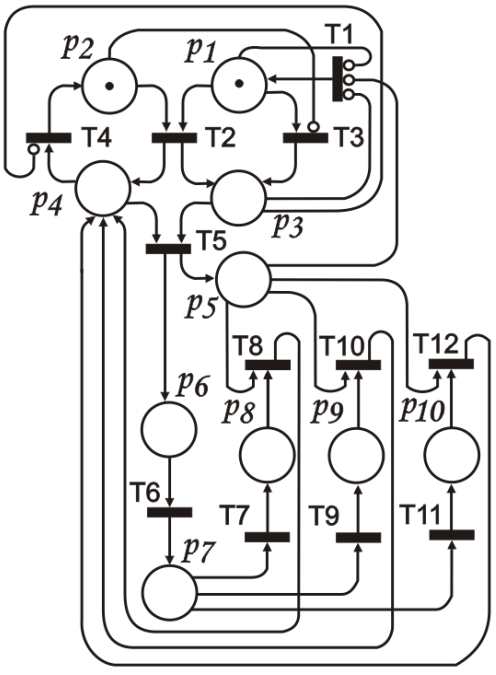

Figure 1: Petri Net model of the automatic distributor

Place labels with meaning	
P1	A visitor is coming and waiting for the machine to start up
P2	Machine in the state Idle (sleep)
P3	A visitor is ready (to insert his card)
P4	Machine is ready for use
P5	A visitor is using the machine (waiting test and/or services)
P6	Card is inserted
P7	Card is tested
P8	Invalid card is rejected
P9	Delivering service 1 and rejecting card
P10	Delivering service 2 and rejecting card

Transition labels with meaning	
T1	Add new visitor to use the machine (only if there is no one waiting, ready or using the machine)
T2	A visitor pushes the start button of the machine
T3	Changing visitor state to ready (if the machine is not in the sleep state)
T4	The machine returns to the Idle state (if there is no visitor in ready state)
T5	Inserting card
T6	Testing card
T7	Rejecting invalid card
T8	Visitor takes invalid card and goes away
T9	Accepting card and delivering service 1
T10	Visitor takes his card with service 1 and goes away
T11	Accepting card and delivering service 2
T12	Visitor takes his card with service 2 and goes away

Table 2: Meaning of Petri net places and transitions

5.2. System and Properties Specification

We should note that the existing rewriting logic semantics for Petri nets does not support (without other additional operators) neither the specification of inhibitor arcs nor counting the number of tokens in a specific place. Therefore, we propose an enhanced specification that alleviates such two limitations.

5.2.1. System Specification

The corresponding rewrite theory of the Petri net model is illustrated in the following modules :

```
fmod PETRI_NET is
protecting INT .
sorts PlaceName Places Marking .
subsort Places < Marking .
ops P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 : -> PlaceName .
op place(_,_) : PlaceName Int -> Places [ctor] .
op __ : Marking Marking -> Marking [ctor assoc comm id:null] .
op null : -> Marking .
endfm
mod PN_ADM is
protecting PETRI_NET .
rl[T1]: place(P1,0) place(P3,0) place(P5,0) => place(P1,1) place(P3,0) place(P5,0) .
rl[T2]: place(P1,1) place(P2,1) place(P3,0) place(P4,0) => place(P1,0) place(P2,0)
        place(P3,1) place(P4,1) .
rl[T3]: place(P1,1) place(P2,0) place(P3,0) => place(P1,0) place(P2,0) place(P3,1) .
rl[T4]: place(P2,0) place(P3,0) place(P4,1) => place(P2,1) place(P3,0) place(P4,0) .
rl[T5]: place(P3,1) place(P4,1) place(P5,0) place(P6,0) => place(P3,0) place(P4,0)
        place(P5,1) place(P6,1) .
rl[T6]: place(P6,1) place(P7,0) => place(P6,0) place(P7,1) .
rl[T7]: place(P7,1) place(P8,0) => place(P7,0) place(P8,1) .
rl[T8]: place(P8,1) place(P4,0) place(P5,1) => place(P8,0) place(P4,1) place(P5,0).
rl[T9]: place(P7,1) place(P9,0) => place(P7,0) place(P9,1) .
rl[T10]: place(P9,1) place(P4,0) place(P5,1) => place(P9,0) place(P4,1) place(P5,0).
rl[T11]: place(P7,1) place(P10,0) => place(P7,0) place(P10,1) .
rl[T12]: place(P10,1) place(P4,0) place(P5,1) => place(P10,0) place(P4,1) place(P5,0).
endm
```

In this specification, the static aspect (the signature) of the system is defined in the functional module PETRI_NET. This module has been imported in the system module PN_ADM, in which we add a rule for each transition to complete describing the dynamic aspect of the machine.

5.2.2. Properties Specification

In the case of our system, the following properties have to be verified :

- *System evolution:* This property ensures that the system is always (or able to be) active and it will not be blocked in a deadlock situation.

- *Serving visitors:* This property ensures that whenever a visitor is present and wants to take something from the machine, he must be served, either by Service 1, Service 2 or his card is rejected if it is not valid.

- *Determining services:* This property ensures that each visitor gets Service 1 at the first time and Service 2 after that. In fact, this property is very important for the hotel due to the financial loss it can incur. Therefore, it must be rigorously verified.

Then, two modules for the specification of system properties are prepared. The first module, PN_ADM_PREDS, describes of the set of properties that are assumed to be verified in the model and that will be used as a reference in the process of verification by the model checking tool. However, in the second module, ACS-CHECK, the designer expresses the relevant properties to be checked in the system model. This module contains the properties that are expressible in LTL only. These modules are given as follows:

```
mod PN_ADM_PREDS is
protecting PN_ADM .
including SATISFACTION .
subsort Marking < State .
ops New-V-Coming V-Ready V-Use-Machine : -> Prop .
ops Machine-Sleep Machine-Ready Machine-In-Use : -> Prop .
ops Reject-Card Service-1 Service-2 : -> Prop .
var A : Marking .
*** ---------- SOME IMPORTANT PROPERTIES -----------
eq place(P1,1) A |= New-V-Coming = true .
eq place(P3,1) A |= V-Ready = true .
eq place(P5,1) A |= V-Use-Machine = true .
eq place(P2,1) A |= Machine-Sleep = true .
eq place(P4,1) A |= Machine-Ready = true .
eq place(P6,1) A |= Machine-In-Use = true .
eq place(P7,1) A |= Machine-In-Use = true .
eq place(P7,1) place(P5,1) A |= Reject-Card = true .
eq place(P7,1) place(P5,1) A |= Service-1 = true .
eq place(P7,1) place(P5,1) A |= Service-2 = true .
endm

mod ADM-CHECK is
inc PN_ADM_PREDS .
inc MODEL-CHECKER .
inc LTL-SIMPLIFIER .
op initial : -> Marking .
eq initial = place(P1,1) place(P2,1) place(P3,0) place(P4,0) place(P5,0) place(P6,0)
             place(P7,0) place(P8,0) .
ops no-deadlock serving : -> Prop .
eq no-deadlock = [](New-V-Coming \/ V-Ready \/ V-Use-Machine \/ Machine-Ready \/
                 Machine-Sleep \/  Reject-Card \/ Service-1 \/ Service-2 ) .
eq serving = [](P-Use-Machine) -> <> (Reject-Card \/ Service-1 \/ Service-2)) .
endm
```

5.3. Verification

Starting from the initial state of the Petri net, the Maude LTL model-checker has been used for the verification of two properties.

- *System evolution:* This property is defined in the module `ADM-CHECK` as `"no-deadlock"`. After verification, the LTL model-checker confirmed the correctness of this property as follows:

```
Maude> reduce in ADM_CHECK : modelCheck(initial, no-deadlock) .
       rewrites: 653 in -42042743134ms cpu (31ms real) (~ rewrites/second)
       result Bool: true
```

- *Serving visitors:* This property is defined as `"serving"` in the module `ADM-CHECK`. The LTL model-checker also confirmed the correctness of this property.

```
Maude> reduce in ADM_CHECK : modelCheck(initial, serving) .
       rewrites: 70 in -41991903084ms cpu (0ms real) (~ rewrites/second)
       result Bool: true
```

5.4. Testing

As per the proposed approach, we used PBT to test the "*Determining service*" property mentioned in Section 5.2.2. In fact, this property is not expressible as a LTL formula and cannot be verified even by the Maude "*search tool*" since it is related to the implementation and not to the system model. However, it can be translated into a number of declarative properties, such as:

```
property_first_time_client_gets_first_service() ->
  ?FORALL(Client, valid_client_card(),
          proper_first_service(sut:use_machine(Client))).

property_following_times_client_gets_second_service() ->
  ?FORALL({Client, Uses}, {valid_client_card(), nat()},
           proper_second_service(
             repeat(Uses+1, sut, use_machine, [Client]))).
```

These two functions aim to test the two possibilities when it comes to the service control: either the client should get the first service when using the machine, or else the card had already been used and so the service should be the second type. These statements should hold *for all* the *valid client card*s that we test.

In this first formulation, the kind of testing to be performed is *positive testing*, since only *valid* client cards are generated to be used as input. The `valid_client_card` function is a data generator that, to comply with the description in Section 5.1, could be implemented as follows:

```
valid_client_card() ->
  ?SUCHTHAT({P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14},
            {nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),
             nat(),nat()}, P1+P2+P4+P8 == 20).
```

where `nat()` is a QuickCheck data generator that produces random natural numbers. Our customized generator produces sequences of 14 natural numbers, but only returns those that fulfill the condition `P1+P2+P4+P8 == 20`.
The additional helper functions `proper_first_service` and `proper_second_service` are used to check the result of invoking the functionality under test on the SUT (`sut:use_machine(Client)`). We include them here to illustrate how using the Erlang version of QuickCheck allows us to implement these kinds of test codes, regardless of the actual technology used to implement the SUT . In fact, the properties needed to be defined in a declarative manner, using the typical artefacts of a functional language (such as pattern-matching, heavily exploited here), like Erlang :

```
proper_first_service(coffee) -> true;
proper_first_service(cake)  -> true;
proper_first_service(water)  -> true;
proper_first_service(juice)  -> true;
proper_first_service(_)      -> false.

proper_second_service(coffee) -> true;
proper_second_service(cake)  -> true;
proper_second_service(_)      -> false.
```

Finally, we take advantage of recursion to implement one last helper to invoke the SUT a random number of times, to simulate the card being used n times after the first time. Again, *for all* ns that we generate, the above property should hold.

```
repeat(1, Mod, Op, Args) ->
  erlang:apply(Mod, Op, Args);
repeat(N, Mod, Op, Args) ->
  erlang:apply(Mod, Op, Args),
  repeat(N-1, Mod, Op, Args).
```

Running these properties as many times as desired will increase our confidence in the SUT implementation if we are unable to find a counterexample. Of course this is not as strong as *proving* that the SUT is correct, but it is the best that can be done using testing.

```
> eqc:quickcheck(properties:property_first_time_
    client_gets_first_service()).
.....................................................................
.....................................................................
OK: Passed 100 test(s).

> eqc:quickcheck(properties:property_following_times_
    client_gets_second_service()).
.....................................................................
.....................................................................
OK: Passed 100 test(s).

> eqc:quickcheck(proper:numtests(1000,
    properties:property_first_time_client_gets_first_service())).
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
OK: Passed 1000 test(s).
```

Of course, we could also formulate the previous two properties to perform *negative testing* as well, i.e. generate valid *and* invalid inputs, and state that only if the input is valid, the property is expected to hold. These new properties are as follows:

```
property_first_time_client_gets_first_service() ->
  ?FORALL(Client, any_client_card(),
    ?IMPLIES(valid_client(Client),
             proper_first_service(sut:use_machine(Client)))).

property_following_times_client_gets_second_service() ->
  ?FORALL({Client, Uses}, {any_client_card(), nat()},
    ?IMPLIES(valid_client(Client),
             proper_second_service(repeat(Uses+1,
                                  sut, use_machine, [Client])))).
```

where the new data generator `any_client_card` will now only produce sequences of 14 natural numbers, and another helper function, `valid_client_card` would be used for determining whether the body of the function is expected to be true or not:

```
any_client_card() ->
  ?LET(Card = {P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14},
              {nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),nat(),
               nat(),nat(),nat()},Card).

valid_client({P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14}) -> P1+P2+P4+P8 == 20.
```

If the SUT had a similar functionality to that of `valid_client_card` then it should be tested against this helper function in a property that states that *for all* client cards, both the helper function and the SUT implementation should generate the same output for a given card.

Last but not least, we could also test the behaviour of the SUT when a card is invalid. In doing so, we would generate invalid card data on purpose, and test that invoking the SUT always returns the no service message. This property can be expressed as follows:

```
property_invalid_client_gets_no_service() ->
  ?FORALL({Client, Uses}, {invalid_client_card(), nat()},
    begin
      FirstOutput = sut:use_machine(Client),
      FollowingOutput = repeat(Uses+1, sut, use_machine, Client]),
      not proper_first_service(Output)
      andalso not proper_second_service(Output)
    end).
```

5.5. Discussion and Motivation

Based on the above-mentioned analysis, we now highlight the main advantages of the proposed approach. In the modeling stage, we justify the selection of the Petri net among many other formalisms, such as UML diagrams, and architecture description languages (ADL) based on the following points:

- **Expressiveness:** In addition to the intuitive graphical representation, the expressiveness of Petri nets, makes it a very suitable formalism for modeling and analyzing a rich variety of systems and applications including, concurrent, distributed and critical system architectures.

- **Formal Semantics:** Since Petri Net is one of the most important formalisms having a well-defined formal semantic and theoretical foundation, it is widely used to formalize not only systems but even other modeling techniques and description languages that are used to describe software architecture (see, Sec. 3.1).

- **Formal analysis:** Petri net has gained wide acceptance as a modeling technique in both industrial and academic areas because of the existence of a large number of analysis tools [103, 104] and techniques [105] for Petri net based models.

The specification language used in our approach is Maude. This choice is motivated by the expressiveness, modularity and parametrization of Maude. Moreover, the rewriting logic allows us to naturally express both flat and hierarchically composite Petri net models. Finally, the availability of a large number of integrated tools, such as Maude model-checker [27] and Maude search tool, facilitate the formal verification of Petri net models in the verification stage. These tools cannot be used to analyze infinite-state systems, and for such systems Maude supports reachability analysis and model checking, including model checking for a useful class of temporal logic formulas. Furthermore, due to the reflective nature of its logic and its implementation, such a library can be easily extended by the user with new

model checking strategies.
Because of the excellent results obtained in the context of MAS [101], we advocate the use of property-based testing (PBT) for the testing stage. Indeed, testing is the last checkpoint at which software errors can be detected before a system goes into operation. However, it leaves all responsibility of designing and writing test scenarios to a human developer, who is limited by time constraints and usually based on his/her own experience and views on the SUT [106]. PBT has proven to play a vital role — as an automatic technique — in detecting corner cases that usually slip through human-conducted testing [107]. Moreover, PBT supports randomness for the generation of test sequences (i.e. test scenarios) themselves not only to alleviate the human factor, but also to compromise effort and correctness [108].

Consequently, we can say that we have proposed a generic approach that covers all software development stages. Thus, we have used rewriting logic as a unifying semantic framework in the specification stage, Maude model checking tool to check model properties, and finally QuickCheck's property-based testing at the implementation level, to raise the confidence on the realization of the given system.

6. Conclusion and Future Work

In this paper, we have presented a new user-friendly approach for the formal analysis of behavioral software architecture of multi-agent based safety-critical systems. The proposed approach has been tested on a case study and results have been found to be quite promising. The ultimate objectives of our approach were :

i. Using the Petri net formalism in the modeling stage to prepare an appropriate model of the given system architecture. Then, the rewriting logic is used for the specification of petri net in an enhanced version that naturally allows the specification of inhibitor arcs and count tokens in a Petri net place.
ii. Combining model-checking and PBT techniques to ensure the absence of any error or unexpected behavior in the specification and implementation of the software under design.

In the future, we plan to develop and implement an algorithm for the automatic generation of rewriting logic specification of Petri nets; according to the enhanced semantics. Such algorithm will enlarge and facilitate the use of our approach and permit developers to benefits from the Maude associated formal analysis tools while developing their safety-critical systems.

References

[1] Petr Skobelev. Multi-agent systems for real-time adaptive resource management. In *Industrial Agents*, pages 207–229. Elsevier, 2015.

[2] Andrey Glaschenko, Anton Ivaschenko, George Rzevski, and Petr Skobelev. Multi-agent real time scheduling system for taxi companies. In *8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary*, pages 29–36, 2009.

[3] Mohammed Chennoufi, Fatima Bendella, and Maroua Bouzid. Multi-agent simulation collision avoidance of complex system: application to evacuation crowd behavior. *International Journal of Ambient Computing and Intelligence (IJACI)*, 9(1):43–59, 2018.

[4] Nicolás F Soria Zurita, Mitchell K Colby, Irem Y Tumer, Christopher Hoyle, and Kagan Tumer. Design of complex engineered systems using multi-agent coordination. *Journal of Computing and Information Science in Engineering*, 18(1):011003, 2018.

[5] Abla Chaouni Benabdellah, Imane Bouhaddou, and Asmaa Benghabrit. Supply chain challenges with complex adaptive system perspective. In *World Conference on Information Systems and Technologies*, pages 1081–1093. Springer, 2018.

[6] Davide Calvaresi, Mauro Marinoni, Arnon Sturm, Michael Schumacher, and Giorgio Buttazzo. The challenge of real-time multi-agent systems for enabling iot and cps. In *Proceedings of the international conference on web intelligence*, pages 356–364. ACM, 2017.

[7] Alexander Wendt, Stefan Wilker, Marcus Meisel, and Thilo Sauter. A multi-agent-based middleware for the development of complex architectures. In *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*, pages 723–728. IEEE, 2018.

[8] Oussama Arki, Abdelhafid Zitouni, Eddine Dib, and Ahmed Taki. A multi-agent security framework for cloud data storage. *Multiagent and Grid Systems*, 14(4):357–382, 2018.

[9] Dehuri Satchidananda, Ghosh Susmita, and Cho Sung-bae. *Integration of swarm intelligence and artificial neural network*, volume 78. World Scientific, 2011.

[10] Nadeem Akhtar and Shafiq Hussain. Formal modeling of a mail transport system based on multi-agent system-of-systems. *Journal of Information Communication Technologies and Robotic Applications*, pages 68–79, 2019.

[11] Zhou Liu, Zhen Chen, Haishun Sun, and Yanting Hu. Multi agent system based process control in wide area protection against cascading events. In *2013 IEEE Grenoble Conference*, pages 1–6. IEEE, 2013.

[12] Minh Nguyen-Duc, Zahia Guessoum, Olivier Marin, Jean-François Perrot, and Jean-Pierre Briot. *A multi-agent approach to reliable air traffic control*. na, 2008.

[13] Wolfgang Reisig. *A primer in Petri net design*. Springer Science & Business Media, 2012.

[14] Yao-hong ZHANG, Jian-cai FAN, and Xiao-lin LIAO. Simulation method of command and control process based on petri net [j]. *Journal of System Simulation*, 7, 2012.

[15] Alessandro Fantechi and Stefano Pepi. Petri nets modeling for the schedulability analysis of industrial real time systems. In *AMARETTO@ MODELSWARD*, pages 5–13, 2016.

[16] Santanu Ku Rath et al. Analysis and modeling of a safety critical system using petri-net model. In *2016 1st India International Conference on Information Processing (IICIP)*, pages 1–6. IEEE, 2016.

[17] Lalit Kumar Singh and Hitesh Rajput. Dependability analysis of safety critical real-time systems by using petri nets. *IEEE Transactions on Control Systems Technology*, 26(2):415–426, 2017.

[18] Mirgita Frasheri, Lan Anh Trinh, Baran Cürüklü, and Mikael Ekström. Failure analysis for adaptive autonomous agents using petri nets. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 293–297. IEEE, 2017.

[19] Wil van der Aalst and Eike Best. *Application and Theory of Petri Nets and Concurrency: 38th International Conference, PETRI NETS 2017, Zaragoza, Spain, June 25–30, 2017, Proceedings*, volume

10258. Springer, 2017.

[20] Maksym Figat and Cezary Zieliński. Methodology of designing multi-agent robot control systems utilising hierarchical petri nets. *arXiv preprint arXiv:1906.11614*, 2019.

[21] Fu-Shiung Hsieh. A hybrid and scalable multi-agent approach for patient scheduling based on petri net models. *Applied Intelligence*, 47(4):1068–1086, 2017.

[22] Slavomír Šimoňák and Martin Tomášek. Acp semantics for petri nets. *Computing and Informatics*, 37(6):1464–1484, 2019.

[23] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, and Carolyn Talcott. Two decades of maude. In *Logic, Rewriting, and Concurrency*, pages 232–254. Springer, 2015.

[24] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96(1):73–155, 1992.

[25] Mark-Oliver Stehr, José Meseguer, and Peter Csaba Ölveczky. Rewriting logic as a unifying framework for petri nets. In *Unifying Petri Nets*, pages 250–303. Springer, 2001.

[26] Ammar Boucherit, Abdallah Khababa, and Laura M Castro. Automatic generating algorithm of rewriting logic specification for multi-agent system models based on petri nets. *Multiagent and Grid Systems*, 14(4):403–418, 2018.

[27] Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The maude ltl model checker. *Electronic Notes in Theoretical Computer Science*, 71:162–187, 2004.

[28] Joe Hendrix, José Meseguer, and Ralf Sasse. Maude itp 2.0 tutorial. Technical report, Technical report, University of Illinois at Urbana-Champaign, 2008.

[29] Zoe Paraskevopoulou, Cătălin Hriţcu, Maxime Dénès, Leonidas Lampropoulos, and Benjamin C Pierce. Foundational property-based testing. In *International Conference on Interactive Theorem Proving*, pages 325–343. Springer, 2015.

[30] Davide Calvaresi, Giuseppe Albanese, Mauro Marinoni, Fabien Dubosson, Paolo Sernani, Aldo Franco Dragoni, and Michael Schumacher. Timing reliability for local schedulers in multi-agent systems. In *RTcMAS@ IJCAI*, pages 1–15, 2018.

[31] Sylvanus A Ehikioya and Cong Zhang. Real-time multi-agents architecture for e-commerce servers. *International Journal of Networked and Distributed Computing*, 6(2):88–98, 2018.

[32] Leonel Aguilar, Maddegedara Lalith, and Muneo Hori. Time critical mass evacuation simulation combining a multi-agent system and high-performance computing. *Multi-agent Systems*, page 69, 2017.

[33] Manisa Pipattanasomporn, Hassan Feroze, and Saifur Rahman. Securing critical loads in a pv-based microgrid with a multi-agent system. *Renewable Energy*, 39(1):166–174, 2012.

[34] Ching Louis Liu, Edmund Kazmierczak, and Tim Miller. A safety analysis method for multi-agent systems. *International Journal of Computer and Information Engineering*, 9(10):2310–2319, 2017.

[35] Rob Alexander and Tim Kelly. Supporting systems of systems hazard analysis using multi-agent simulation. *Safety science*, 51(1):302–318, 2013.

[36] Inna Pereverzeva, Elena Troubitsyna, and Linas Laibinis. Formal development of critical multi-agent systems: A refinement approach. In *2012 Ninth European Dependable Computing Conference*, pages 156–161. IEEE, 2012.

[37] Valeria Cardellini, Emiliano Casalicchio, and Emanuele Galli. Agent-based modeling of interdependencies in critical infrastructures through uml. In *Proceedings of the 2007 spring simulation multiconference-Volume 2*, pages 119–126. Society for Computer Simulation International, 2007.

[38] Lachlan Birdsey, Claudia Szabo, and Katrina Falkner. Casl: a declarative domain specific language for modeling complex adaptive systems. In *Proceedings of the 2016 Winter Simulation Conference*, pages 1241–1252. IEEE Press, 2016.

[39] Ji Wang, Rui Shen, and Hong Zhu. Agent oriented programming based on slabs. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, volume 1, pages 127–132. IEEE, 2005.

[40] Max Risler and Oskar von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in xabsl. In *AAMAS08-workshop on formal models and methods for multi-robot systems, Estoril, Portugal*. Citeseer, 2008.

[41] Vinitha Hannah Subburaj and Joseph E Urban. Applying formal methods to specify security requirements in multi-agent systems. In *2018 Federated conference on computer science and information systems (FedCSIS)*, pages 707–714. IEEE, 2018.

[42] Adam D Barwell, Christopher Brown, Kevin Hammond, Wojciech Turek, and Aleksander Byrski. Using program shaping and algorithmic skeletons to parallelise an evolutionary multi-agent system in erlang. *Computing and Informatics*, 35(4):792–818, 2017.

[43] Shiladitya Pujari and Sripati Mukhopadhyay. Petri net: A tool for modeling and analyze multi-agent oriented systems. *International Journal of Intelligent Systems and Applications*, 4(10):103, 2012.

[44] Rajib Kumar Chatterjee, Neha Neha, and Anirban Sarkar. Behavioral modeling of multi agent system: high level petri net based approach. *International Journal of Agent Technologies and Systems (IJATS)*, 7(1):55–78, 2015.

[45] Carlos Silva, Eder Gonçalves, Graçaliz Dimuro, Glenda Dimuro, and Esteban de Manuel Jerez. Modeling agent periodic routines in agent-based social simulation using colored petri nets. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 644–650. IEEE, 2013.

[46] Borhen Marzougui and Kamel Barkaoui. Interaction protocols in multi-agent systems based on agent petri nets model. *Int J Adv Comput Sci Appl*, 4(7), 2013.

[47] Fu-Shiung Hsieh. Developing cooperation mechanism for multi-agent systems with petri nets. *Engineering Applications of Artificial Intelligence*, 22(4-5):616–627, 2009.

[48] Rajib Kr Chatterjee, Anirban Sarkar, and Swapan Bhattacharya. Modeling and analysis of agent oriented system: Petri net based approach. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP'11)*, pages 17–23, 2011.

[49] Said Brahimi, Ramdane Maamri, and Zaidi Sahnoun. Dynamic verification of hierarchical multi-agent plans. *Multiagent and Grid Systems*, 13(2):113–142, 2017.

[50] Ahmed T Dib and Zaïdi Sahnoun. Formal specification of multi-agent system architecture. In *ICAASE*, pages 65–72, 2014.

[51] Nadeem Akhtar and Malik M Saad Missen. Contribution to the formal specification and verification of a multi-agent robotic system. *arXiv preprint arXiv:1604.05577*, 2015.

[52] M Birna Van Riemsdijk, Frank S De Boer, Mehdi Dastani, and John-Jules Ch Meyer. Prototyping

3apl in the maude term rewriting language. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 95–114. Springer, 2006.

[53] M Birna van Riemsdijk, L Aştefănoaei, and Frank S de Boer. Using the maude term rewriting language for agent development with formal foundations. In *Specification and Verification of Multi-agent Systems*, pages 255–287. Springer, 2010.

[54] Mohamed Amin Laouadi, Farid Mokhati, and Hassina Seridi-Bouchelaghem. A formal framework for organization-centered multi-agent system specification: A rewriting logic based approach. *Multiagent and Grid Systems*, 13(4):395–419, 2017.

[55] Noura Boudiaf, Farid Mokhati, and Mourad Badri. Supporting formal verification of dima multi-agents models: towards a framework based on maude model checking. *International Journal of Software Engineering and Knowledge Engineering*, 18(07):853–875, 2008.

[56] Bernhard Bauer and James Odell. Uml 2.0 and agents: how to build agent-based systems with the new uml standard. *Engineering applications of artificial intelligence*, 18(2):141–157, 2005.

[57] Darshan S Dillon, Tharam S Dillon, and Elizabeth Chang. Using uml 2.1 to model multi-agent systems. In *Software Technologies for Embedded and Ubiquitous Systems*, pages 1–8. Springer, 2008.

[58] Radovan Cervenka and Ivan Trencansky. *The Agent Modeling Language-AML: A Comprehensive Approach to Modeling Multi-Agent Systems*. Springer Science & Business Media, 2007.

[59] Enyo José Tavares Gonçalves, Mariela I Cortes, Gustavo Augusto Lima Campos, Yrleyjander S Lopes, Emmanuel SS Freire, Viviane Torres da Silva, Kleinner Silva Farias de Oliveira, and Marcos Antonio de Oliveira. Mas-ml 2.0: Supporting the modelling of multi-agent systems with different agent architectures. *Journal of Systems and Software*, 108:77–109, 2015.

[60] Vinitha Hannah Subburaj and Joseph E Urban. Specifying security requirements in multi-agent systems using the descartes-agent specification language and auml. In *Information technology for management: Emerging research and applications*, pages 93–111. Springer, 2018.

[61] Nils Przigoda, Christoph Hilken, Robert Wille, Jan Peleska, and Rolf Drechsler. Checking concurrent behavior in uml/ocl models. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 176–185. IEEE, 2015.

[62] OMG Ocl. 2.0 specification. *Final Adopted Specification ptc/03-10*, 14, 2005.

[63] Hongjun Guan. Prevention and control model of enterprise business risk based on multi-agent. *Journal of Convergence Information Technology*, 5(7):148–154, 2010.

[64] N Darragi, EM El-Koursi, and S Collart-Dutilleul. Architecture description language for cyber physical systems analysis: a railway control system case study. *Computers in Railways XIV: Railway Engineering Design and Optimization*, 135:227–237, 2014.

[65] Haralambos Mouratidis, Manuel Kolp, Paolo Giorgini, and Stephane Faulkner. An architectural description language for secure multi-agent systems. *Web Intelligence and Agent Systems: An International Journal*, 8(1):99–122, 2010.

[66] Manfred Broy and María Victoria Cengarle. Uml formal semantics: lessons learned. *Software and Systems Modeling*, 10(4):441–446, 2011.

[67] João M Fernandes. Combining petri nets and uml for model-based software engineering. CEUR Workshop Proceedings, 2012.

[68] Sima Emadi and Fereidoon Shams. Transformation of usecase and sequence diagrams to petri nets.

In *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, volume 4, pages 399–403. IEEE, 2009.

[69] Xavier Renault, Fabrice Kordon, and Jérôme Hugues. From aadl architectural models to petri nets: Checking model viability. In *Object/Component/Service-Oriented Real-Time Distributed Computing, 2009. ISORC'09. IEEE International Symposium on*, pages 313–320. IEEE, 2009.

[70] Xavier Renault, Fabrice Kordon, and Jerome Hugues. Adapting models to model checkers, a case study: Analysing aadl using time or colored petri nets. In *Rapid System Prototyping, 2009. RSP'09. IEEE/IFIP International Symposium on*, pages 26–33. IEEE, 2009.

[71] Nasreddine Aoumeur and Gunter Saake. Integrating and rapid-prototyping uml structural and behavioural diagrams using rewriting logic. In *CAiSE*, pages 296–310. Springer, 2002.

[72] Farid Mokhati and Mourad Badri. Generating maude specifications from uml use case diagrams. *Journal of Object Technology*, 8(2):319–136, 2009.

[73] Zhibin Yang, Kai Hu, Dianfu Ma, and Lei Pi. Towards a formal semantics for the aadl behavior annex. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 1166–1171. IEEE, 2009.

[74] Peter Csaba Ölveczky, Artur Boronat, and José Meseguer. Formal semantics and analysis of behavioral aadl models in real-time maude. In *Formal Techniques for Distributed Systems*, pages 47–62. Springer, 2010.

[75] David Garlan, Bradley R Schmerl, and Shang-Wen Cheng. Software architecture-based self-adaptation. *Autonomic computing and networking*, 1:31–55, 2009.

[76] Sahar Smaali, Aicha Choutri, and Faiza Belala. K semantics for dynamic software architectures. In *Proceedings of the International Arab Conference on Information Technology (ACIT '2013)*, 2013.

[77] Zhenhua Yu and Yuanli Cai. Object-oriented petri nets based architecture description language for multi-agent systems. *IJCSNS*, 6(1):123–131, 2006.

[78] Meuse NO Junior, Paulo Maciel, Ricardo Lima, Angelo Ribeiro, Cesar Oliveira, Adilson Arcoverde, Raimundo Barreto, Eduardo Tavares, and Leornado Amorin. A retargetable environment for power-aware code evaluation: An approach based on coloured petri net. *LNCS*, 3728:49, 2005.

[79] Muaz A Niazi, Amir Hussain, and Mario Kolberg. Verification & validation of agent based simulations using the vomas (virtual overlay multi-agent system) approach. *arXiv preprint arXiv:1708.02361*, 2017.

[80] Panagiotis Kouvaros and Alessio Lomuscio. Automatic verification of parameterised multi-agent systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 861–868. IFAAMAS, 2013.

[81] Alireza Souri and Nima Jafari Navimipour. Behavioral modeling and formal verification of a resource discovery approach in grid computing. *Expert Systems with Applications*, 41(8):3831–3849, 2014.

[82] Wing Lok Yeung. Behavioral modeling and verification of multi-agent systems for manufacturing control. *Expert Systems with applications*, 38(11):13555–13562, 2011.

[83] Hamza Mazouzi, Amal El Fallah Seghrouchni, and Serge Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 517–526. ACM, 2002.

[84] Marco Alberti, Davide Daolio, Paolo Torroni, Marco Gavanelli, Evelina Lamma, and Paola Mello. Specification and verification of agent interaction protocols in a logic-based system. In *Proceedings of*

the 2004 ACM symposium on Applied computing, pages 72–78. ACM, 2004.

[85] Evren Bulut, Djamel Khadraoui, and Bertrand Marquet. Multi-agent based security assurance monitoring system for telecommunication infrastructures. In *Proceedings of the Fourth IASTED International Conference on Communication, Network and Information Security*, pages 90–95. ACTA Press, 2007.

[86] Alexia Zoumpoulaki, Nikos Avradinis, and Spyros Vosinakis. A multi-agent simulation framework for emergency evacuations incorporating personality and emotions. In *Hellenic Conference on Artificial Intelligence*, pages 423–428. Springer, 2010.

[87] Michael Wooldridge, Michael Fisher, Marc-Philippe Huget, and Simon Parsons. Model checking multi-agent systems with mable. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 952–959. ACM, 2002.

[88] Massimo Benerecetti and Alessandro Cimatti. Symbolic model checking for multi-agent systems. *Proc. MoChart*, 2:1–8, 2002.

[89] Rafael H Bordini, Michael Fisher, Carmen Pardavila, Willem Visser, and Michael Wooldridge. Model checking multi-agent programs with casp. In *International Conference on Computer Aided Verification*, pages 110–113. Springer, 2003.

[90] Rafael H Bordini, Michael Fisher, Carmen Pardavila, and Michael Wooldridge. Model checking agentspeak. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 409–416. ACM, 2003.

[91] Peter Gammie and Ron Van Der Meyden. Mck: Model checking the logic of knowledge. In *International Conference on Computer Aided Verification*, pages 479–483. Springer, 2004.

[92] Alessio Lomuscio and Franco Raimondi. Mcmas: A model checker for multi-agent systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 450–454. Springer, 2006.

[93] Faiza Belala and A Boucherit. A contribution to the formal checking of multi-agents systems. In *IEEE International Conference on Computer Systems and Applications, 2006.*, pages 9–16. IEEE, 2006.

[94] Alessio Lomuscio, Charles Pecheur, and Franco Raimondi. Automatic verification of knowledge and time with nusmv. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1384–1389. IJCAI/AAAI Press, 2007.

[95] Rodion Podorozhny, Sarfraz Khurshid, Dewayne Perry, and Xiaoqin Zhang. Verification of multi-agent negotiations using the alloy analyzer. In *International Conference on Integrated Formal Methods*, pages 501–517. Springer, 2007.

[96] Magdalena Kacprzak, Wojciech Nabiałek, Artur Niewiadomski, Wojciech Penczek, Agata Półrola, Maciej Szreter, Bożena Woźna, and Andrzej Zbrzezny. Verics 2007-a model checker for knowledge and real-time. *Fundamenta Informaticae*, 85(1-4):313–328, 2008.

[97] Lacramioara Astefanoaei, Mehdi Dastani, John-Jules Ch Meyer, and Frank S de Boer. On the semantics and verification of normative multi-agent systems. *J. UCS*, 15(13):2629–2652, 2009.

[98] Supriya D'Souza, Abhishek Rao, Amit Sharma, and Sanjay Singh. Modeling and verification of a multi-agent argumentation system using nusmv. *arXiv preprint arXiv:1209.4330*, 2012.

[99] Jeremy Kong and Alessio Lomuscio. Symbolic model checking multi-agent systems against ctl* k specifications. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*,

pages 114–122. IFAAMAS, 2017.

[100] Eric Ramat. Introduction à la modélisation et à la simulation à événements discrets. *Modélisation et simulation multi-agents: Applications pour les sciences de lhomme et de la société*, pages 50–73, 2006.

[101] Clara Benac Earle and Lars-Åke Fredlund. A property-based testing framework for multi-agent systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1823–1825. IFAAMAS, 2019.

[102] Álvaro Carrera, Carlos A Iglesias, and Mercedes Garijo. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *Information Systems Frontiers*, 16(2):169–182, 2014.

[103] Karsten Wolf. Petri net model checking with lola 2. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 351–362. Springer, 2018.

[104] Monika Heiner, Martin Schwarick, and Jan-Thierry Wegener. Charlie–an extensible petri net analysis tool. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 200–211. Springer, 2015.

[105] Yi-Sheng Huang, Ta-Hsiang Chung, and Pin-June Su. Synthesis of deadlock prevention policy using petri nets reachability graph technique. *Asian Journal of Control*, 12(3):336–346, 2010.

[106] H. Zhu, P.A.V. Hall, and J.H.R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.

[107] J. Derrick, N. Walkinshaw, T. Arts, C. Benac Earle, F. Cesarini, L.-A. Fredlund, V. Gulias, J. Hughes, and S. Thompson. Property-based testing - the protest project. *LNCS*, 6286 LNCS:250–271, 2010.

[108] Alex Groce, Gerard Holzmann, and Rajeev Joshi. Randomized differential testing as a prelude to formal verification. In *29th International Conference on Software Engineering (ICSE'07)*, pages 621–631. IEEE, 2007.

Author's bio

Ammar Boucherit is an associate professor at the University of El-Oued, Algeria. He holds his PhD degree in Computer Science from University of Ferhat Abbas, Sétif, Algeria. His research focused on three areas: formal design of computer systems, software architecture and multi-agents systems.

Laura M. Castro is an associate professor at the University of A Coruña, Spain. She is a member of the MADS (Models and Applications of Distributed Systems) research group. She gives lectures on Software Architecture, Software Verification and Validation. Her research focuses on software testing (automated, model and property-based testing), applied to software in general, and distributed, concurrent, functional systems in particular.

Abdallah Khababa is currently a full professor at the University of Ferhat abbas Sétif, Algeria. He holds a Ph.D. in computer science from the University of Ferhat Abbas Sétif. His main areas of interest include cognitive science and artificial intelligence, knowledge representation with ontologies and human-machine interface and its usability.

Osman Hasan is the senior head of department of electrical engineering SEECS and the director of the system analysis and verification (SAVe) Lab at NUST SEECS. His main research interests include formal verification, interactive theorem proving and higher-order logic. Prior to joining NUST SEECS, He did his PhD and post-doctoral fellowship from the hardware verification group at Concordia University, Montreal Canada.