International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 7553-7560
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Towards a Rewriting Logic-Based Framework for Specifying FIPA Request Protocols

Mohamed Amin Laouadi[1]*, Djamel Nessah[2], Abdelaziz Lakhfif[3],

[1] Department of Computer Science, Faculty of Sciences, Setif 1 University - Ferhat Abbas, Setif, 19000, Algeria
* **Corresponding Author Email:** mohamed.laouadi@univ-setif.dz - **ORCID :** 0000-0002-6215-2769

[2] ICOSI Laboratory, Department of Computer Science, University Abbes Laghrour-Khenchela, Khenchela 40002, Algeria
Email: Nessah_djamel@univ-khenchela.dz - **ORCID :** 0009-0004-8671-7017

[3] Department of Computer Science, Faculty of Sciences, Setif 1 University - Ferhat Abbas, Setif, 19000, Algeria
Email: abdelaziz.lakhfif@univ-setif.dz - **ORCID :** 0000-0002-3287-5456

Article Info:

DOI: 10.22399/ijcesen.4007
Received : 05 August 2025
Accepted : 06 October 2025

Keywords

Interaction Protocols,
FIPA Request,
rewriting logic,
Real-Time Maude,
First Aid.

Abstract:

This paper presents a practical and systematic approach for formally specifying interactions in multi-agent systems (MAS) based on the well-established FIPA request protocol. It aims to simplify and streamline the process of translating MAS interaction descriptions expressed with Agent UML diagrams into precise, unambiguous formal specifications using the Real-Time Maude language, which is founded on rewriting logic—a powerful and expressive formalism for specifying and analyzing concurrent and real-time systems. This integration enables enhanced analysis, rigorous verification, and validation of agent behaviors in complex and dynamic scenarios. The effectiveness and usability of the proposed approach are demonstrated through a detailed first aid case study, highlighting its practical applicability and benefits for designing reliable MAS interactions.

1. Introduction

Interaction is one of the key aspects of multi-agent systems, it ensures cooperation and negotiation between agents. Implementing the interaction requires an infrastructure that includes communication languages and interaction protocols. According to FIPA (Foundation of Intelligent Physical Agents), an interaction protocol is a common pattern of communication, so the specification and implementation of the protocol must be independent of the application domain and agent internal architecture. To improve the development of interaction between agents, we want to formalize the interaction protocols using a formal language that is well adapted. We are interested in FIPA request interaction protocol since it was the most used protocols for the design of MAS. Formalizing the FIPA request interaction protocol is very important for both analysis and design activities. Furthermore, the Multi-Agent System (MAS) design requires the involvement with formal languages. Among these languages: Real-Time Maude (RT-Maude) [1]. The formal specifications will eliminate ambiguities in the interpretation of the models. For that, the integration of Agent UML (AUML) and Real-Time Maude will enable the formal validation of the FIPA request protocol.The purpose of our approach is to translating AUML diagrams into a formal specification to integrate the validation of the FIPA request protocol's consistency starting from the analysis phase. The remainder of the paper is organized as follows. Section 2 provides a brief overview of key related works. Section 3 presents the interactional protocol FIPA request. In section 4 we describe briefly rewriting logic basics. Sections 5 present the proposed approach along with the transformation process. However, Section 6 presents a first aid case study to demonstrate the transformation and validation procedures. In section 7, we conclude and offer some future work directions for further research.

2. Related Works

In works [2] and [3] authors presented a formalization process of FIPA protocols and a formal framework has been proposed. In these

frameworks, several interaction concepts are considered.

To incorporate the formal validation of the consistency of these FIPA interaction protocols, several modules have been developed since the analysis phase.

In the present work we continue in the same way of both approaches [2] and [3] but with more details and applying the result of the translation process proposed on a real example, an essential idea that has not been done yet. Additionally, by utilizing the interactional protocol FIPA request, the suggested formal framework for MAS, reduces the possibility of misunderstandings between users and developers.

3. The Interactional Protocol FIPA Request

The Interactional Protocol FIPA Request allows one agent to request another to perform some action. The representation of this protocol is given in Fig. 1.After considering the request, the Participant decides whether to accept it or not. "Refused" becomes true and the Participant communicates a refusal if a refusal choice is made. If not, "agreed" turns into reality. The Participant communicates a "agree" if the criteria specify that an express agreement is essential (i.e., "notification necessary" is true). Depending on the situation, the "agree" might not be required [4].

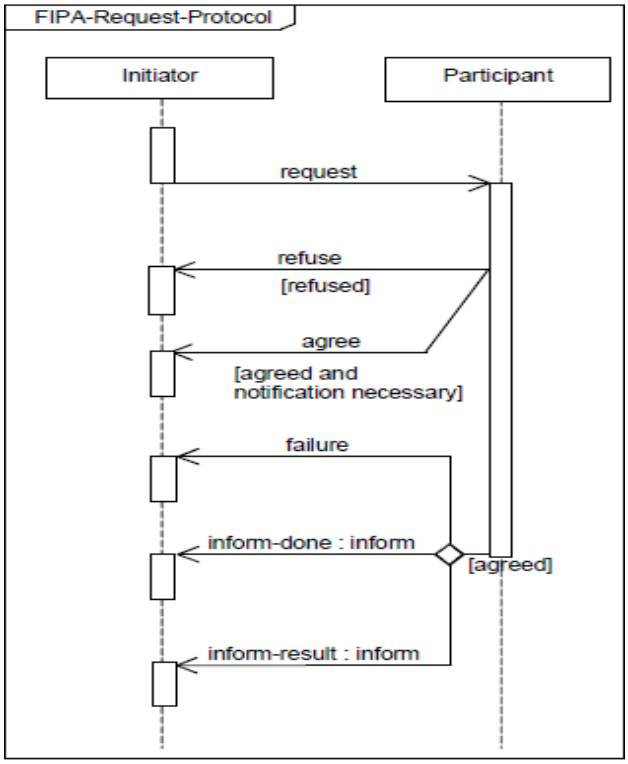

Figure 1. *FIPA request interaction protocol [4].*

4. Rewriting Logic and Real-Time Maude

According to Eker et al. [5] rewriting logic is a highly expressive and natural framework designed for specifying concurrent systems, parallelism, communication, and interaction. It effectively models state and concurrent computations, particularly suited for concurrent object-oriented programming. Maude [6] is a language for the specifation, simulation, and model checking rewrite theories, which are the specification units of rewriting logic (see the survey [7] and [8, 9, 10]).

Maude's rewriting logic is realized via rewrite theories that combine equational specifications with rewrite rules that handle local state transformations. It supports complex rewriting modulo associativity, commutativity, and identity axioms, complemented by a strategic language layer that controls rule application for enhanced user-defined behavior control.

Regarding to the language performance, Maude was ranked second (after Haskell) as the best performance language in a recent comparative analysis of well-known algebraic, functional and object-oriented languages performed in [11]. This blend makes Maude a powerful tool for specifying, executing, verifying, and analyzing complex concurrent systems, programming languages, and protocols.

Real-Time Maude [12] is an extension of Maude developed to utilize the principles of real-time rewrite theory.

Recent developments have further enhanced Maude’s capabilities and applications. Durán et al. [13] demonstrate how Maude supports formal specification, verification, and declarative programming of open distributed systems, enabling scalable and reliable system designs that leverage rewriting logic's inherent concurrency and modularity. Moreover, bridging semantic gaps between qualitative and quantitative models in distributed systems has been advanced through work by Liu et al. [14], who apply Maude for combined qualitative-quantitative semantics facilitating automated reasoning on complex system properties.

5. The FIPA Request Formal Interactional Framework

The developed framework (Fig. 2) consists of multiple formal modules, with different categories (functional, object-oriented, and timed object-oriented modules). Due to space constraints, only the main modules of the framework are presented. The STATE-CHART module (Fig. 3) defines the types of actions and conditions an agent can use to specify operations related to its states.

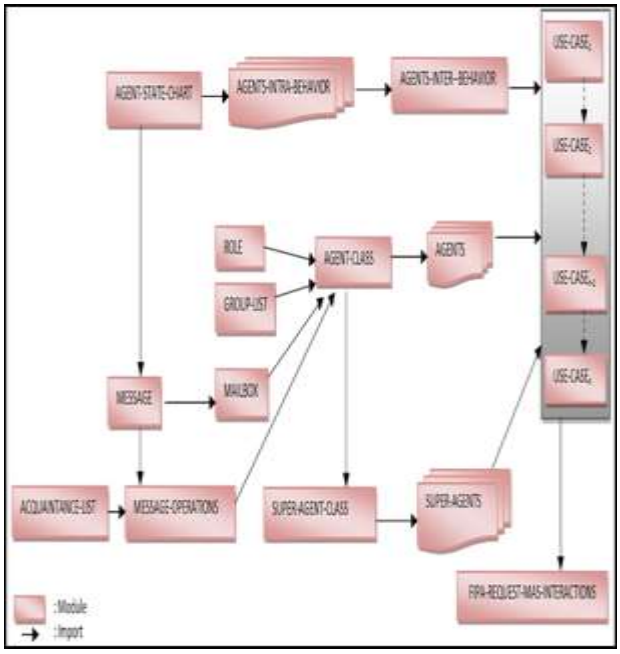

Figure 2. *The FIPA Request formal framework modules.*

```
(fmod STATE-CHART is
sorts State NameS.
sorts Condition Action.
 op State : NameS -> State .
 op IsInternalAct : Action -> Bool .
 op IsReceivingAct : Action -> Bool .
 op IsSendingAct : Action -> Bool .
 op TargetS : State Condition -> State .
 op ActionToAccomplish : State Condition ->Action.
endfm)
```

Figure 3. *The functional module STATE-CHART*

Individual agent behavior is represented via the INTRA-BEHAVIOR modules, which import the STATE-CHART module. Based on the INTRA-BEHAVIOR modules, the INTER-BEHAVIOR module establishes relations to control interactions between various agents.The IDENTIFICATION and ACTION modules are imported within the MESSAGE module (Fig. 4), handles the agent identification mechanism and defines the structure of messages exchanged between agents.

```
(omod MESSAGE is
    protecting STATE-CHART .

    sorts Identif  Message Content.
    subsort Oid < Identif .
    subsort Action  < Content .

    op :_:_ : Identif Content Identif -> Message.
    op GetIdent : Message -> Identif .
    vars A A1 : Identif . var Cont : Content .
    eq GetIdent (A : Cont : A1) = A .
endom)
```

Figure 4. *The object oriented module MESSAGE.*

Communicating agents typically have a Message Box to store messages received from other agents, along with a list of their acquaintances. To manage these, we suggest the MESSAGE-BOX and LIST-OF- ACQUAINTANCE modules, which deal with agents' Message Boxes and acquaintance lists, respectively.

We define the MESSAGE-OPERATIONS module to explain the sending and receiving operations (Fig. 5), which imports the LIST-OF- ACQUAINTANCE and MESSAGE modules.

```
(omod MESSAGE-OPERATIONS is
    pr LIST-OF-ACQUAINTANCE  MESSAGE .
    subsort Acquaintance < Identifier .

    op SendMssg : Message -> Msg .
    op ReceiveMssg : Message -> Msg .

endom)
```

Figure 5. *The object oriented module MESSAGE-OPERATIONS*

The functional module ROLE-CONCEPT (Fig. 6) represent the concept of role that agents can performs within groups.

```
(fmod ROLE-CONCEPT is
    sort Role .
********** User Part
    …
Endfm)
```

Figure 6. *The functional module ROLE-CONCEPT.*

The concept of a group is represented in the module LIST-OF-GROUP (Fig. 7).

```
(omod LIST-OF-GROUP is
    inc STRING .
    sorts ListofGroup  Group .
    subsort Group < ListofGroup .
    subsort String < ListofGroup .

    op _:_: Group ListofGroup -> ListofGroup .
    op EmptyListofGroup : -> ListofGroup .
    var Grp : Group .  var GrofL : ListofGroup .

    eq Grp : EmptyListofGroup = Grp .

endom)
```

Figure 7. *The object oriented module LIST-OF-GROUP.*

In the object-oriented module CLASS-OF-AGENT (Fig. 8), the basic attributes (CurrentS, PlayR, GrL, AcqL, and MBx,) of the class structure for agents are defined. which represent the agent's current state, role, group list, acquaintance list, and Message Box,

respectively. The module imports the STATE-CHART, ROLE-CONCEPT, MESSAGE BOX, MESSAGE-OPERATIONS, and LIST-OF-GROUP modules.

```
(omod CLASS-OF-AGENT is
pr STATE-CHART ROLE-CONCEPT .
pr MESSAGE-BOX LIST-OF- GROUP .
pr MESSAGE-OPERATIONS.

class Agent | CurrentS : State, PlayR : Role,
GrL : ListofGroup,  AcqL : ListofAcquaint,
MBx : Message-Box  .        ---(1)

endom).
```

Figure 8. *The object oriented module CLASS-OF-AGENT.*

To represent the supervisor of each agents group, we define the SupervisorAgent class in the object-oriented module SUPERVISOR-CLASS-OF-AGENT (Fig. 9). This class includes the attribute Responsibility (line 1) and imports the CLASS-OF-AGENT module, serving as a subclass of the Agent class (line 2).

```
(omod SUPERVISOR-CLASS-OF-AGENT is
ex CLASS-OF-AGENT  .
sort Supervisor .
subsort String < Supervisor.
class SupervisorAgent | Commitment : Supervisor. --
-(1)
subclass SupervisorAgent  <  Agent .          ---(2)

endom)
```

Figure 9. *The object oriented module SUPERVISOR-CLASS-OF-AGENT.*

The timed object-oriented module FIPA-USE-CASEi (Fig. 10) has the same name as the associated use case and corresponds to any use case that is depicted in the various Agent UML Protocol diagrams. These modules contain various rewrite rules that specify scenarios in which agents interact; these interactions might be conditional or unconditional, instantaneous or tick-based.

```
(tomod FIPA-USE-CASEi is
inc  CLASS-OF-AGENT .
inc  SUPERVISOR-CLASS-OF-AGENT.
inc  INTER-BEHAVIORS.
rl [1] : Config1 => Config2.
…..
rl [K] : Config2k-1 => Config2k.
endtom)
```

Figure 10. *The Timed object oriented Module FIPA-USE-CASEi*

The timed object-oriented module FIPA-REQUEST-MAS-INTERACTIONS (Fig. 11), which represents all system interactions, imported all FIPA-USE-CASEi modules.

```
(tomod FIPA-REQUEST-MAS-INTERACTIONS  is
inc FIPA-USE-CASE1.
…
inc FIPA-USE-CASEn.
endtom)
```

Figure 11. *The Timed object oriented Module FIPA-REQUEST-MAS-INTERACTIONS.*

Fig. 12, where the Timer message is defined (line 1), illustrates the tick rule that ensures the system's time progression.

```
crl [tickrule] :
 {Timer(TimeO)          --(1)
        < A : Agent |   PlayR: Initiator>
REST:Configuration} =>
  { Timer(TimeO monus 1)
< A : Agent | >  REST:Configuration }
in time 1 if (TimeO > zero).
```

Figure 12. *The RT-Maude Tick Rule form.*

6. Practical Case Study: FIRST AID

We have chosen the first aid case study to validate our approach because it can be built according to FIPA request interaction protocol for MAS development.

We generalized the model developed by CHU Thanh Quang [13] which models land rescue activities to apply to various other cases (road accident; fire; earthquake; household accidents (gas leak, etc.).

The decomposition of the MAS chosen for first aid application is illustrated in Fig. 13. This application involves two types of agents: (1) the Fireman, acting as a Supervisor agent, and (2) the Fire-Station and Hospital agents.

When an incident occurs, the Fire-Station agent will be informed via its toll-free number from then on: a group of firemens agents hurries to the place to secure it and take care of the victims if there is one. Once there the firemens inform the nearest hospital agent of the number of victims and their conditions.

6.1 AUML used Diagrams

6.1.1 AUML Protocol Diagram between Fire-Station and Fireman:

This diagram (Fig. 14) allows us to see the interactions between the FireStation, and Fireman agents according to the FIPA Request protocol.

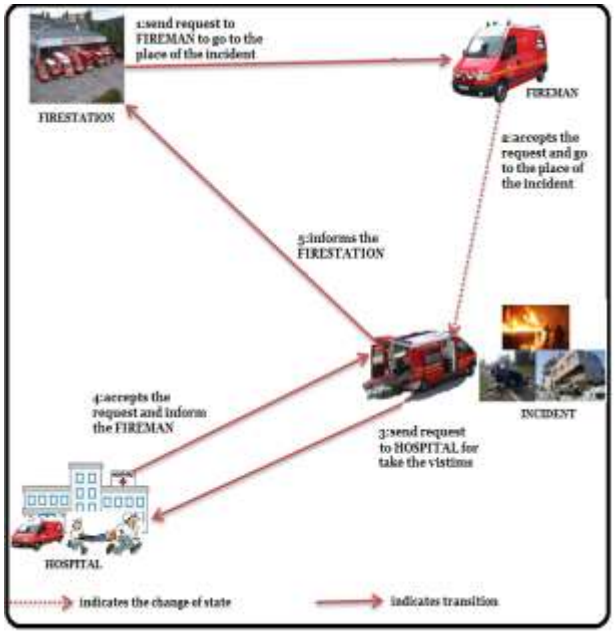

Figure 13. *MAS based Decomposition of the First-Aid example.*

The Fire-Station agent receives a call informing him of an incident from a witness which triggers the protocol by sending the message with the performative "Request" to Fireman who accepts the request by responding with an "Agree" message. When the operation is complete, Fireman sends an "inform" message to his Fire Station to inform him that the mission is complete and therefore the protocol ends.

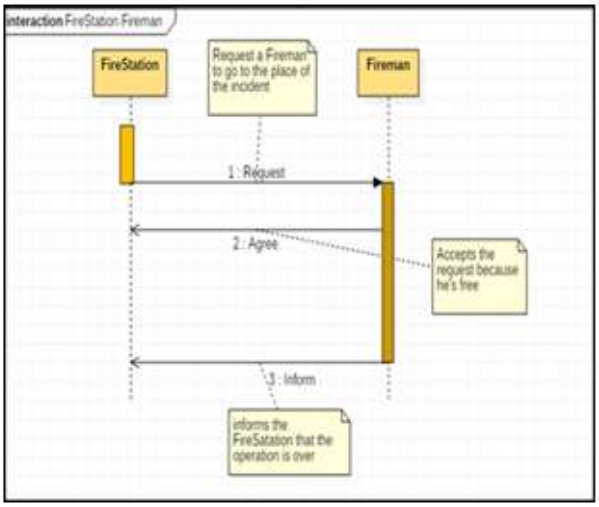

Figure 14. *FIPA Request Protocol Diagram between Fire-Station & Fireman.*

6.1.2 AUML Protocol Diagram between Fireman and Hospital:

As illustrated in Fig. 15. The Fireman supervisor agent begins the protocol by sending a message with the performative "Request" to instruct Hospital agent to pick up the victims it is transporting. The Hospital agent accepts the request by responding with an "Agree" message, then sends an "inform" message to let the Fireman know that the victims are being taken care of.

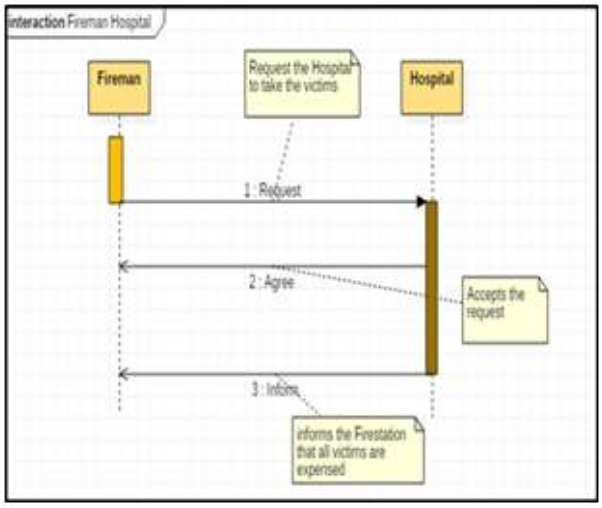

Figure 15. *FIPA Request Protocol Diagram between Fireman and Hospital.*

6.2 Application of the Translation Process

All of the modules mentioned above are implied in the generated description. (STATE-CHART, MESSAGE, MESSAGE-OPERATIONS, …), which don't change with the other modules' definitions: AGENTS (Fig. 16), SUPERVISOR-AGENTS (Fig. 17), INTER-BEHAVIORS (Fig. 18), and FIPA-REQUEST-MAS-INTERACTIONS (Fig. 19) ... etc.

```
(omod AGENTS is
    ex CLASS-OF-AGENT .
    inc STRING NAT .
    subclass FireStation  <  Agent .
    subclass Hospital  <  Agent .
class FireStation | CurrentS : State, PlayR : Role,
GrL : ListofGroup, AcqL : ListofAcquaint,
MBx : MailBox  .
class Hospital | CurrentS : State, PlayR : Role, GrL :
ListofGroup, AcqL : ListofAcquaint,
MBx : MailBox  .
  endom)
```

Figure 16. *The object oriented module AGENTS.*

```
(omod SUPERVISOR-AGENTS is
    ex SUPERVISOR-CLASS-OF-AGENT .
    subclass Fireman  <  SupervisorAgent .
    class Fireman | Commitment : Supervisor .

endom)
```

Figure 17. *The object oriented Module SUPERVISOR-AGENTS*

.

```
(fmod INTER-BEHAVIORS is

    pr FIRESTATION-INTRA-BEHAVIOR.
    pr FIREMAN-INTRA-BEHAVIOR .
    pr HOSPITAL-INTRA-BEHAVIOR .

op CorrespondingS : State -> State .          ---[1]
op CorrespondingCond : Action -> Condition .  ---[2]

******** FireStation
    eq CorrespondingS(AgentState(StartCA)) =
      AgentState(StartF) .
    eq CorrespondingCond(SendAlert ) =
      ReceiveAlert  .

******* Fireman
    eq CorrespondingS(AgentState(StartF)) =
      AgentState(WaitResponseCA) .
    eq CorrespondingCond(SendAcceptF ) =
      ReceiveAcceptF .

******* Hospital
    eq CorrespondingS(AgentState(StartH)) =
      AgentState(WaitF) .
    eq CorrespondingCond(SendAcceptH ) =
      ReceiveAcceptH  .
…
endfm)
```

Figure 18. *The functional Module INTER-BEHAVIORS.*

```
(tomod FIPA-REQUEST-MAS-INTERACTIONS is

    including NAT-TIME-DOMAIN .
    including AGENTS .
    including SUPERVISOR-AGENTS .
    including INTER-BEHAVIORS .
********* User Part
    subsort String < Identifier .
    ops Event : Identifier State Condition -> Msg .
    op UpDateAcqL : MailBox -> ListofAcquaint .

    vars A1 A2 : Identifier .      vars S1 S2 : State .
    vars MBx1 : MailBox .  vars ACL1 :
ListofAcquaint .
    vars Cond1 Cond2 : Condition .
    var Act1 : Action .
    var M1 : Message .
    var Cont1 : Content .
eq UpDateAcqL(MBx1) =
if  MBx1 == EmptyMessageBox then
EmptyAcquaintanceList
    else
     GetIdent(FrontMBox(MBx1)) :
     UpDateAcqL(DeleteMBox(MBx1))
fi .
----------|(FireStation)|---------
crl [Sending-Case1] :
            Event(A1, S1, Cond1)
< A1 : FireStation | CurrentS : S1, AcqList : ACL1 >
          =>
 < A1 : FireStation | CurrentS :
TargetState(S2,Cond2), AcqList : TailA(ACL1) >
SendMessage(A1 : ActionToAccomplish(S1, Cond1)
: HeadA(ACL1))
          Event(HeadA(ACL1), CorrespondingS(S1),
CorrespondingCond(ActionToAccomplish(S1,
Cond1)))
 if (IsSendingAction(ActionToAccomplish(S1,
Cond1)) == true) .
---
endtom)
```

Figure 19. *The Timed OO Module FIPA-REQUEST-MAS-INTERACTIONS.*

6.3 Validation of the First Aid based FIPA Request

In order to validate the First Aid scenario, we analyzed a preliminary configuration in which the FireStation agent begins by receiving a call, which eventually leads to fulfilling the requirements of the other agents.

Fig. 20, presents the timed object oriented module FIRST-AID, which imports the module FIPA-REQUEST-MAS-INTERACTIONS and contains an Initial Configuration. This later describes agents in their initial states.

All agents are shown in their success states in the resultant configuration (Fig. 21), signifying that the FireStation's restrictions have been met.

7. Conclusions

Several interaction protocols for describing multi-agent systems existed. However, the majority of them offer informal or semi-formal descriptions. In this work, we propose a new and general approach,

```
(tomod FIRST-AID is
 extending FIPA-REQUEST-MAS-
INTERACTIONS.

--------|Validation Part (Initial Configuration)|-------
op Initstate : -> GlobalSystem .
eq Initstate =
{
Event("FireStation", AgentState(StartCA),
ReceiveCall)  Event("Fireman1",
AgentState(StartWorkF), No-Pb)

Event("Fireman1", AgentState(InRoadF), IsAlive)

< "FireStation" : FireStation | PlayR : FireStation,
CurrentS : AgentState(StartCA), ListofGroup:
"Groupe-1", AcqL : "Fireman1", MBx :
EmptyMessageBox >

< "Fireman1" : Fireman | PlayR : Fireman,
Commitment : "Supervisor1", CurrentS :
AgentState(StartF),  GrL : "Groupe-2", MBx :
EmptyMessageBox, AcqL :("FireStation" :
"Hospital") >
 < "Hospital" : Hospital | PlayR : Hospital,
CurrentS : AgentState(StartH), GrL : "Groupe-3",
MBx : EmptyMessageBox,  AcqL : "Fireman1" >
  } .

endtom)

(trew Initstate with no time limit .)
```

Figure 20. *Initial Configuration.*

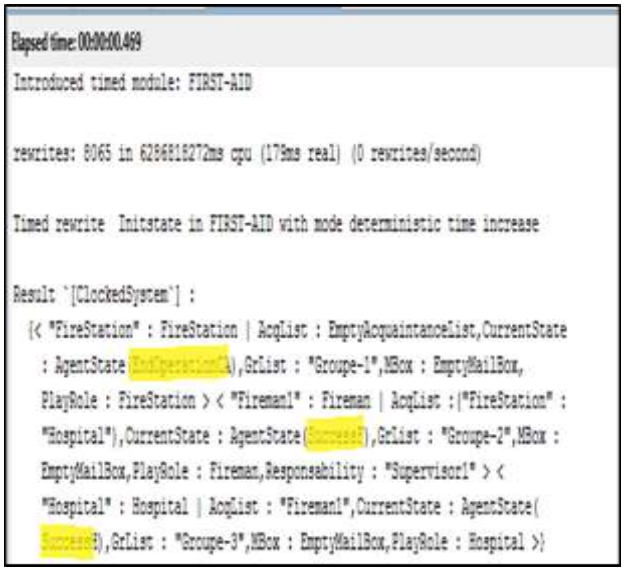

Figure 21. *Execution of Initial Configuration.*

that makes it easier to formally describe and validate multi-agent systems. It is based on the FIPA request interaction protocol.Our approach first captures the different aspects (functional, static, and dynamic) of multi-agent systems from an interactional perspective using AUML diagrams and then translates these graphical descriptions into a formal representation in RT-Maude.Employing formal notations to specify MAS using FIPA request interaction protocol enables the creation of precise interaction descriptions and provides stronger support for their verification and validation processes.As we have stressed, the results reported in this paper are preliminary. We plan to the expansion of this formal interactional framework to encompass the various interactional concepts that are offer in others agent- interaction protocols. Additionally, we want to extend our approach by including RT-Maude's capabilities to verify specific FIPA request interaction protocol features.

Author Statements:

- **Ethical approval:** There is no connection between the study and the use of humans or animals.
- **Conflict of interest:** According to the researchers, they have no known financial conflicts or affiliations that could have impacted the research described in this article.
- **Acknowledgement:** According to the writers, they have no institutions or individuals to thank.
- **Author contributions:** The authors of this work declare their equality.
- **Funding information:** According to the authors, there is no funding to acknowledge.
- **Data availability statement:** The corresponding author can provide the data supporting the findings of the research upon request. Due to ethical and privacy concerns, the data are not publicly accessible.

References

[1] Ölveczky, P. C. (2014, April). Real-Time Maude and its applications. In International Workshop on Rewriting Logic and its Applications (pp. 42-79). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-12904-4_3

[2] Mokhati, F., Sahraoui, B., Bouzaher, S., & Kimour, M. T. (2010). A tool for specifying and validating agents' interaction protocols: From Agent UML to Maude. *Object Technology*, *9*(3). http://www.jot.fm/issues/issue_2010_05/article2/

[3] Mokhati, F., Boudiaf, N., Badri, M., & Badri, L. (2007). Translating AUML Diagrams into Maude Specifications: A Formal Verification of Agents Interaction Protocols. *J. Object Technol.*, *6*(4), 77-102. http://www.jot.fm/issues/issue_2007_05/article2

[4] FIPA Request Interaction Protocol Specification. 2002. http://www.fipa.org/specs/fipa00026/index.html

[5] Eker, S., Martí-Oliet, N., Meseguer, J., Rubio, R., & Verdejo, A. (2023). The Maude strategy language. Journal of Logical and Algebraic Methods in Programming, 134, 100887. https://doi.org/10.1016/j.jlamp.2023.100887

[6] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., & Talcott, C. (2007). All about maude-a high-performance logical framework: how to specify, program, and verify systems in rewriting logic (Vol. 4350). Springer.

[7] Meseguer, J. (2012). Twenty years of rewriting logic. The Journal of Logic and Algebraic Programming, 81(7-8), 721-781. https://doi.org/10.1016/j.jlap.2012.06.003

[8] Liu, S. (2019). Design, verification and automatic implementation of correct-by-construction distributed transaction systems in Maude (Doctoral dissertation, University of Illinois at Urbana-Champaign)

[9] Liu, S., Ölveczky, P. C., Zhang, M., Wang, Q., & Meseguer, J. (2019, April). Automatic analysis of consistency properties of distributed transaction systems in Maude. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 40-57). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-17465-1_3

[10] Ölveczky, P. C. (2016, September). Formalizing and validating the P-Store replicated data store in Maude. In International Workshop on Algebraic Development Techniques (pp. 189-207). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-72044-9_13

[11] Garavel, H., Tabikh, M. A., & Arrada, I. S. (2018, June). Benchmarking implementations of term

rewriting and pattern matching in algebraic, functional, and object-oriented languages: The 4th rewrite engines competition. In International Workshop on Rewriting Logic and its Applications (pp. 1-25). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-99840-4_1

[12] Olveczky, P. C. (2007). Real-time maude 2.3 manual. Department of Informatics, University of Oslo, 180.

[13] Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., & Talcott, C. (2024, September). Programming open distributed systems in Maude. In Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming (pp. 1-12). https://doi.org/10.1145/3678232.3678237

[14] Liu, S., Meseguer, J., Ölveczky, P. C., Zhang, M., & Basin, D. (2022). Bridging the semantic gap between qualitative and quantitative models of distributed systems. Proceedings of the ACM on Programming Languages, 6(OOPSLA2), 315-344. https://doi.org/10.1145/3563299

[15] Amouroux, E., Chu, T. Q., Boucher, A., & Drogoul, A. (2007, November). GAMA: an environment for implementing and running spatially explicit multi-agent simulations. In Pacific Rim International Conference on Multi-Agents (pp. 359-371). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01639-4_32