Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp736-744)

Modeling a fault tolerant multiagent system for the control of a mobile robot using MaSE methodology

MARÍA GUADALUPE ALEXANDRES GARCÍA[1] **RAFAEL ORS CAROT**[2] **LUCERO JANNETH CASTRO VALENCIA**[3]

[1, 2] **Computer science and Systems of Computers Department (DISCA)**
Universidad Politécnica de Valencia
Camino de Vera s/n. 46022 Valencia
SPAIN

http://www.disca.upv.es/rors

[3] **Posgrado Department**
Instituto Tecnológico de Hermosillo
Av. Tecnológico s/n 83240 Hermosillo Sonora
MEXICO

Abstract. - A multiagent system that tolerates failure in a hardware and software level in the distributed control system of a mobile robot is shown; it's made to guarantee the availability of the robot in the most efficient way. The multiagent system is modeled thru the formal MaSE methodology supported by its development tool, AgentTool; in such a way that a greater reliability is guaranteed. The multiagent system tolerates system failures in the robot's control systems through three types of agents that cooperate so that the mechanisms that detect failures in the input, output, processing and network control devices are activated; as well as the tasks that constitute the robot's control system, these agents also activate the failure-isolation mechanisms and reconfigure the system by means of interactions between the agents that are supported in the design of the physical architecture of the robot's control system, In our system, the agents are designed such that if they recover from the failure, the agents reconfigure the control system to the state prior to the failure, if they are not able to recover the failure, the robot's control system continues working due to the double connection and to the duplicity of the tasks and devices, because the implemented design in the physical architecture of the system allows it.

Key Words: - Model, Multiagent, MaSE, System of distributed control, mobile robot, AgentTool, Failure tolerant .

1 Introduction

Current mobile robotic systems have had a great development and have been built with efficiency and are being used in different areas such as: agriculture, manufacture industry, oil industry, nuclear waste treatment, volcano exploration, medic laboratories, high-risk material management, automotive industry, recovery of victims of catastrophes, cardio surgeon assistants, surveillance, planetary exploration, bomb detectors. The work done by these machines is precise, exact, and laborious, must work 24 hours a day, that's why it is very important to designed them in a way they can tolerate the failures.

Nowadays in a robotics system the failure-tolerance to the hardware level is based mainly in repetition that allows offering a service according to the specifications, regardless of the failures, which allows us to guarantee the availability of the computational system with no interruption, as long as it is required to be maintained working. By duplicating the critical software and hardware systems we assure that we have a failure-tolerant system, to best guarantee the availability of the system it's been thought to include in the robot's control system a layer of distributed intelligent agents, failure-

tolerant, in such a way that with the communication capabilities offered by the paradigm of agents, effects over the hardware and software is giving a better liability to the robot. This layer it is implemented using a set of agents that can be climb easily due to the fact that every agent it's associate to a node or a task that integrates the system. The goal is a new focusing into the failure-tolerant systems in distributed robotic systems with the implementation of intelligent agents.

Inside the context of artificial intelligence, the multiagent systems have been characterized by their offering of a possible solution to the development of complex problems with distributed characteristics. When approaching the development of multiagent systems it's doubtless a noticeable increase in the complexity as well as the necessity to adapt existent techniques, or occasionally, the development of new techniques and tools. It is evident that the development of any kind of software needs the existence of methods and tools that make easier the acquirement of liable final products. In that line, in the last years have appeared different papers that try to propose multiagent systems' development processes. In the last years a great advance in the design of applications has been observed, getting as a result intelligent applications, capable of working in an autonomous way and to take their own decisions. The development of these applications is based in the agent technology, and takes the software-engineering concepts at the time of structuring the development process, and the artificial intelligence, when it is required to give the programs response capabilities when facing certain events. From the point of view of this technology, the distributed systems change to multiagent systems. The multiagent technologies represent something new and exciting to the analysis, design and development of complex software. Now, all the conceptualization of the agents and the multiagent systems land in the appliance in a system that solves a certain problem. In order to do this, it is required to develop it, creating a model doing an analysis and design in such a way that its implementation is relatively easy and liable.

2 Formal definition of the multiagent system's failure tolerance

In the last years, the technology of agents has received way more attention due to the advantages that the multiagent systems have in complex and distributed environments. A multiagent system must provide efficiency, must be trustful, robust and secure. Nevertheless, developing multiagent systems is a complicated process and there is no guarantee that the systems that results from the initial requirements will work properly according to the desired behavior. In order to produce big and complex systems that work in an efficient and trustful way, as well as having extension capabilities and keep their functionality and security, it's needed to be able to have a methodology that guides us in the different stages of the system's development and allows us to make the model properly. Without this, it is impossible to develop complex software, even more if the problem's solving is implemented with a multiagent system.

Due to the complexity of the problem to solve, it has been required to select a multiagent systems-oriented methodology that helps us to model the proposed system. This methodology was selected based on certain characteristics that must have according to our necessities, this methodology is MaSE (Multiagent System Software Engineering)

2.1 Formal propose

The failure-tolerant-agents that work in the distributed robotics systems are:

The Agent Node (AN) whose mission is the Node-mode failure tolerance and belongs to a determinate Node (N) of the distributed system (SD), a Node is a hardware device that is made of Z devices that may be: sensors, actuators, microcontrollers, memories, etc.

The Agent task that belongs to one of the many tasks that completes the system is the one related to the failure-tolerance in a system level, which is found in every system node. (Such as recovery of task, reconfigure the system if the node or the failed task accomplishes to recover).

The Agent System (AS) that belongs to the distributed system which mission is the one related with the failure-tolerance to in a system level, its found in every system nodes (it terminates to the nodes or tasks that contains a failure, reconfigure the system in order to keep working with the nodes or tasks that are still active or, makes a safe stop in case the system cant be recover, carries a failure register of all the nodes and on the tasks, reconfigures the system if the node or the failure task achieve to recover).

Be *SD*, a composed distributed system by a set of nodes $N = \{N_i\}$, where each N_i can be composed by many devices $[D_{i,z}]$. On the other hand, a set of tasks is executed over SD, $T = \{T_j\}$.

Definition 1: Be $N = \{N_i\}$, where i is the number of nodes of the distributed system.

Definition 2: Be $T= \{T_j\}$, where j is the number of tasks that are executed in the system.
Definition 3: Be $[D_{i,z}]$, where z is the number of devices that N_i will have.
From these definitions, the next thing can be made:
Definition 4: Be a distributed system SD, formed by the double: SD = {N, T}, to this SD it is tried to equip to him with certain tolerance-failure characteristics. For this issue it's proponed the usage of the distributed artificial intelligence paradigm, with which it is possible to speak of a new focus of the FTS in SD failure-tolerant systems with the implementation of intelligent agents.

$$AITF = \{AN_i, AT_j, AS\}$$

Now, will be defined the agents tolerant to failure, that will work in SD.
The Agent Node $(AN_i) \in N_i$, whose task is the one related with the failure tolerance in a node level. (That works inside the node).
The Agent Task $(AT_j) \in T_j$, whose task is the one related with the failure tolerance in a task level. (How to recover the possible tasks of the errors that they can suffer)
The Agent System (AS) $\in$ SD, whose task is the one related with the failure-tolerance in a level system (wich task's should be complete in the system and on what nodes)
With this, a SD tolerance to failure it is defined like:
Definition 5: A tolerance to failure distributed system TFDS it is defined as the double.

$$TFDS = \{SD, AITF\}$$

3 The model of the multiagent system with MaSE

MaSE methodology consists in seven steps represented on the figure 1 [1]. The first three steps represents the phase of analysis, the last four represent the phase of design.

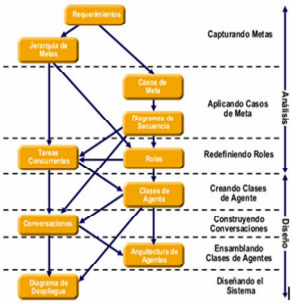


Fig. 1

3.1 Goals capture for modeling the failure-tolerant multiagent in the mobile robot distributed control system (SMA TF SCDRM)

The first step on MaSE is the *goal capture*, that takes the initial specifications of the system and it transforms them in a structured set of goals that the system must accomplished in order to properly work according to the problem that he must solve. This stage is based on the goals because they are a high stable structure for build the system model. The tolerant to failure multiagent system requirements for the distributed control of a mobile robot witch is our model problem; lead us to deduce the general goal: tolerate the failure in the distributed control system of a mobile robot in a hardware and software level. In order to determine its intention each scenario listed the requirements that identify the goals. As an example of methodology use in our model is shown in figure 2:

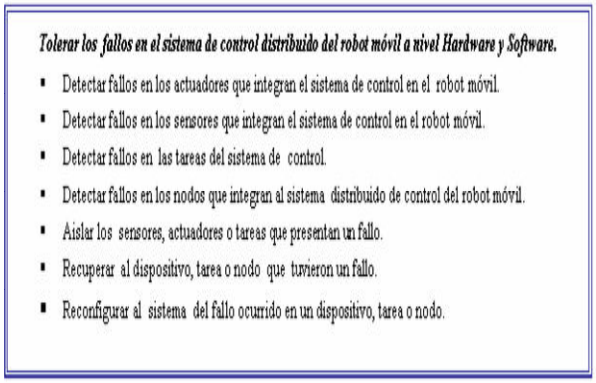


Fig .2

The final step on this phase is to structure the goal, making an analysis by importance and building a *goals hierarchy diagram*. Figure 3 shows the *hierarchy diagram*.

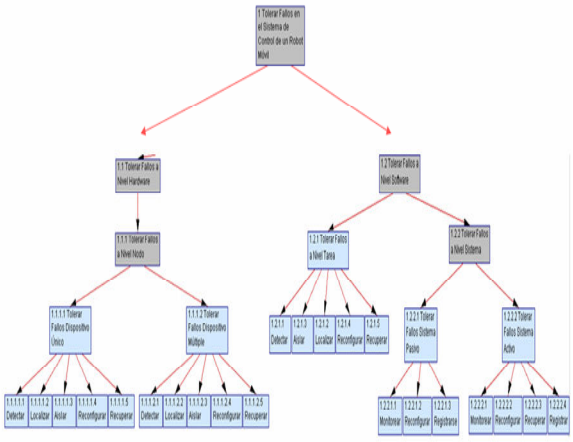
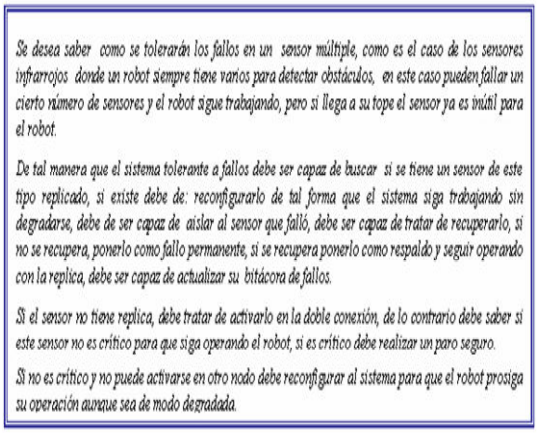


Fig. 3

3.2 Applying goal cases in the SMMTF SCDRM

Once all goals have been captured and explicitly declared, you have the basis for modeled analysis model [1]. The next thing to do is to clearly identify the scenarios' requirements, which are detailed in goal cases. An easy translatable example in a goal cases is shown in figure 4, which is an identified scenario to accomplish failure tolerance of a multiple output/input device that integrates the mobile robot's control system; this shall pass to be a goal case.



Se desea saber como se tolerarán los fallos en un sensor múltiple, como es el caso de los sensores infrarrojos donde un robot siempre tiene varios para detectar obstáculos, en este caso pueden fallar un cierto número de sensores y el robot sigue trabajando, pero si llega a su tope el sensor ya es inútil para el robot.

De tal manera que el sistema tolerante a fallos debe ser capaz de buscar si se tiene un sensor de este tipo replicado, si existe debe de: reconfigurarlo de tal forma que el sistema siga trabajando sin degradarse, debe de ser capaz de aislar al sensor que falló, debe ser capaz de tratar de recuperarlo, si no se recupera, ponerlo como fallo permanente, si se recupera ponerlo como respaldo y seguir operando con la replica, debe ser capaz de actualizar su bitácora de fallos.

Si el sensor no tiene replica, debe tratar de activarlo en la doble conexión, de lo contrario debe saber si este sensor no es crítico para que siga operando el robot, si es crítico debe realizar un paro seguro.

Si no es crítico y no puede activarse en otro nodo debe reconfigurar al sistema para que el robot prosiga su operación aunque sea de modo degradada.

Fig. 4

The scenario in figure 4 gives many information segments. First, it illustrates an action trajectory throughout the system; second, introduces some new concepts to the goal. For this particular case, the control system is distributed thru Nodes (figure 5) that control the robot, the tasks and devices are distributed to different Nodes. The Goal cases are valuable inside MaSE since they help plot communication trajectories that will become conversations between the different agents that constitute the system.

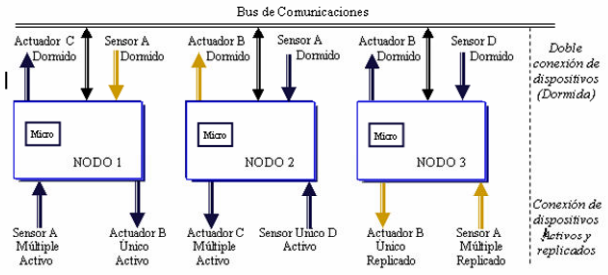


Fig. 5

3.3 Sequence diagrams in SMA TF SCDRM

Applying goal cases requires the previously identified scenarios and restructures them to be able to make a sequence diagram [1]. In MaSE, the different processes are different agent roles. The events between roles are called messages [2]. The sequence diagrams give a high level view of how different roles work reciprocally to accomplish their goals, and are useful while building each role's tasks.

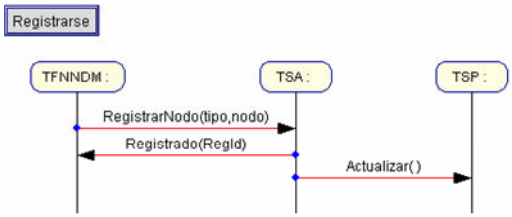


Fig. 6

Figure 6 shows the sequence diagram that represents a series of events sent between the multiple device node level failure tolerator (TFNNDM) and the active system role tolerator (TSA) and passive system tolerator (TSP), which are required for the multiple device failure tolerator registries. These events ought to be contained in their respective role conversation.

The next step on MaSE is to transform the hierarchically structured goals in a more useful way to build the multiagent system, these being the roles [1]. The roles are the basic blocks of agent constructions and represent the system's goals during the design phase. When associating each goal with a role, the goals will be accomplished, because each role will be executed by a type of agent.

3.4 Transform Goals in Roles in SMA TF SCDRM

The next step is to transform the structured goal in a more useful way to build multiagent systems being the roles [1]. When associating each goal with a role, goals will be accomplished, because each role will be executed by a type of agent. In this model the goal hierarchy was taken in order to create the roles shown in figure 7. The parentheses indicate the goals associated to each role.

- Tolerar Fallos Dispositivo Único (1.1.1.1, 1.1.1.1.1, 1.1.1.1.2, 1.1.1.3, 1.1.1.1.5)
- Tolerar Fallos Dispositivo Múltiple (1.1.1.2, 1.1.1.2.1, 1.1.1.2.2, 1.1.1.2.3, 1.1.1.2.5)
- Tolerar Fallos Nivel Tarea (1.2.1, 1.2.1.1, 1.2.1.2, 1.2.1.3, 1.2.1.4, 1.2.1.5)
- Tolerar Fallos Sistema Pasivo (1.2.2.1, 1.2.2.1.1,1.2.2.1.2,1.2.2.1.3)
- Tolerar Fallos Sistema Activo (1.2.2.2, 1.2.2.2.1, 1.2.2.2.2, 1.2.2.2.3, 1.2.2.2.4)
- Tolerador Dispositivo (1.1.1.1.4, 1.1.1.2.4)
- Reconfigurador Sistema (1.2.2.2.2, 1.2.2.2.3)

Fig. 7

Figure 8 shows the role model of the proposed SMA. In this way, we assure that each of the proposals of the goal hierarchy diagram are designated to a goal [1]. Roles may have one or more concurrent tasks associated to them. The lines between the tasks denote the communication protocols that occur between these. The arrows indicate which task is the one that begins the action and which one is the responder. The solid arrows (red) indicate an external communication between the two tasks from different roles or two tasks from different instances of the same role. The external protocols involve messages being sent between roles and will become messages in a conversation between the types of agents that do certain actions. The dotted arrows (blue) denote communication between two tasks that belong to the same instance of the role.

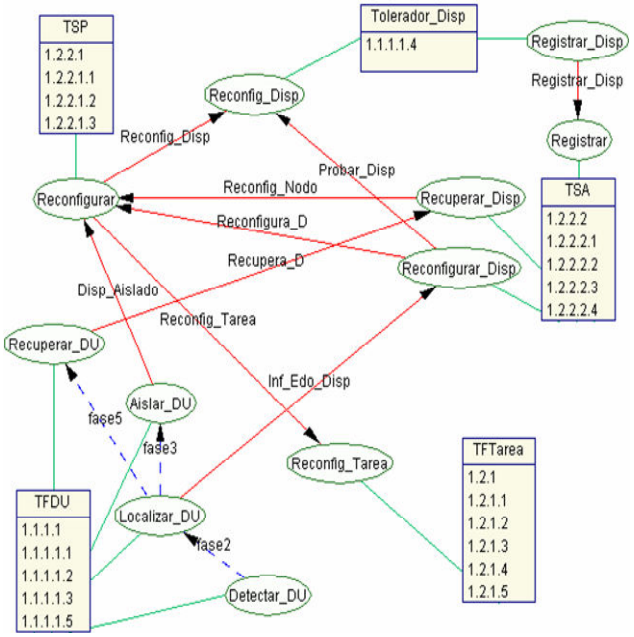


Fig. 8

The model is made of five roles, with their respective tasks. The single device failure TOLERATOR TDFU is responsible of tolerating failures in single devices (DU) (a camera) of the SCDRM, for this it is necessary to carry out certain tasks in order to accomplish the goal defined objectives that integrate it. The role TSA and TSP are responsible of reconfiguring the devices when a failure in the DU occurs, besides, the TSA role registers the node agents. The TFT roles (task level failure TOLERATOR) and Tolerador_Dev (Device level failure TOLERATOR) have the function to reconfigure the task and the device respectively according to the actions sent by the TSA and TSP roles. The interaction between the different tasks that constitute the roles model is given by the protocols shown in figure 8.

3.5 The concurrent tasks in the proposed model

After the roles have been created, the tasks can be associated with each role. Each goal associated with a role may have a task that details how the goal must be accomplished [1]. By using concurrent tasks models, you help define the inner behavior of the agents and define the interactions with other agents related to these inner processes.

Figure 9 shows the task Reconfigurar_Disp of the TSA role, the tasks allows the active system active (ASA) take the pertinent actions to reconfigure the device (unique or multiple sensor or actuator) when it fails in a node. The first thing it tries to do is to active the device in its replica and to check the functionality of the double connection.

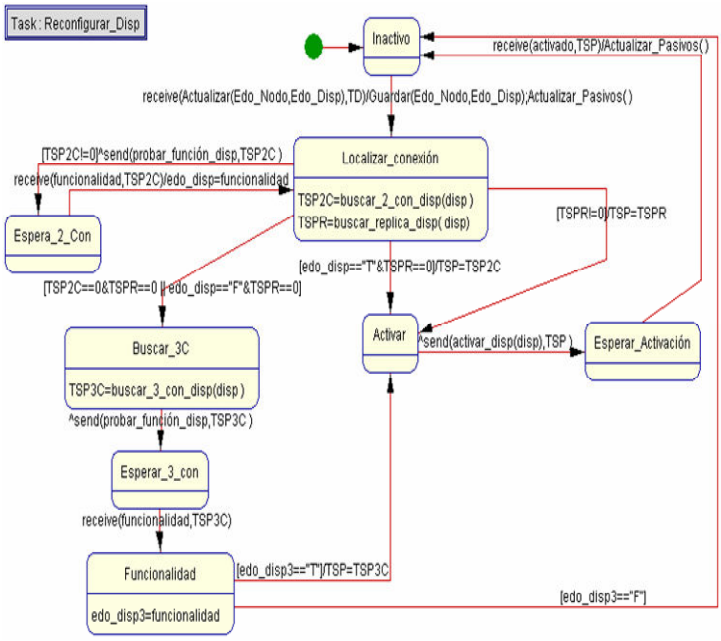


Fig. 9

3.6 Creating agent types in SMA TF SCDRM

In the design phase, the first step is to create the types of agents that will integrate the multiagent system, from the roles. The product of this phase is an *Agent Types diagram*, which shows the types and conversations between them [1]. This diagram is the first design of the MaSE agent that shows the complete multiagent system. Each role must be executed by a type, but it is possible that a role is carried out by two types of agents or one

type can represent various roles and change dynamically.

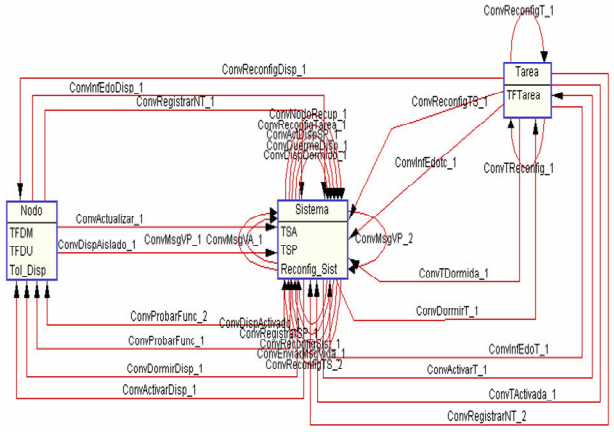


Fig. 10

Figure 10 shows the agent types diagram:

1. The node type: Tolerates the failures in the single and multiple devices, this type of agent carries out three roles (TFDM, TFDU and Tolerator_Dev).
2. The system type: Reconfigures the active TOLERATOR, passive tolerator and reconfigures the SCDRM when a failure occurs in a device or node that integrates the robot's control system. This type contains three roles (TSA, TSP and Reconfigure_Syst).
3. The task type: Tolerates the failures in existent tasks in the SCDRM, contains the TFT role (Task failure TOLERATOR).

The fact that an agent type contains more than one role does not mean that it will carry them out at the same time. The role that a type carries out can be changed in execution time according to the actions in the system. To our model, for example, the node type agent begins with the role Tolerator_Dev, because the first task it must carry out is to register with the active system tolerator (TSA), and then it will carry out the single device failure tolerator (TFDU) this type (that will become an agent) is in charge of a single device. The TSA and TSP roles cannot be carried out at the same time by the type, according to the initial requisites of the system.

3.7 Building conversations SMA TF SCDRM

The agents that integrate this SMA are able to communicate thru structured messages. The structured sequence of messages is called Agent Conversation [1]. A conversation consists of two conversation diagrams, one for the type that initiates the conversation and one for the one who responds to the conversation. Figures 11 and 12 show the communication diagrams for the ConRegister_Dev Conversation (Conversation Register Device). The node type initiates the conversation sending the *registered* message (parent.TFNTS, parent.Type) to the system type, this one receives it, registers the node agent, refreshes itself and refreshes the rest of the agent systems replicated in each node of the SCDRM, at the end it sends the identifier to the node agent thru the registered message (regID), the node agent receives it and refreshes itself.

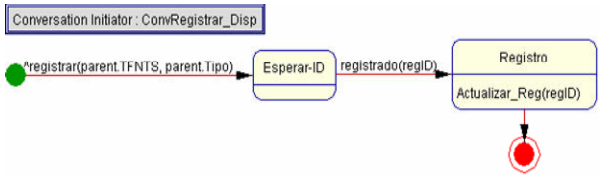


Fig. 11

Fig. 12

The infinite cycles, deadlock and other errors in the communication can cause trouble in the MAS, even worse, the system can keep on working while there is a catastrophic problem and that was not perceived by the conversation designer. Because of this, it is necessary to explore the trajectories so that the conversation can be valid [3], it is required to be formally verified. Once the conversations have been verified, one can be sure that the agents will communicate as expected. The AgentTool tool provides a module that assures the validity and interoperability of the conversations, this module accomplishes as well that the communication protocol politic can be satisfied [4].

3.8 Transformation from analysis phase to design phase

The transformation process that MaSE provides is correct and robust for the generation of models of the design without the loss of information from the analysis phase. The formal transformation systems reduce mistakes that happen during the design, in figure 13 you can see the transformation of the analysis phase from the error model and the tasks to the design phase in the type Agent model, inner components and conversations.

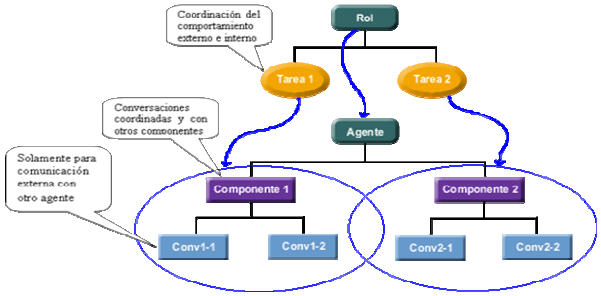


Fig. 13

In MaSE it is assured that the only way of modeling the agent's structure organization, by the means of components[5] and conversations in the design phase, is by capturing all the data present in the analysis models and they preserve the basic idea of a conversation. Figure 14 shows the architecture of the Node and System types, in figures 15 and 16 it is shown the architecture of the tasks type. This architecture is obtained from Roles Model (figure 8)

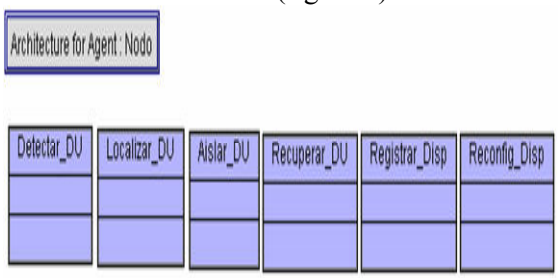


Fig. 14

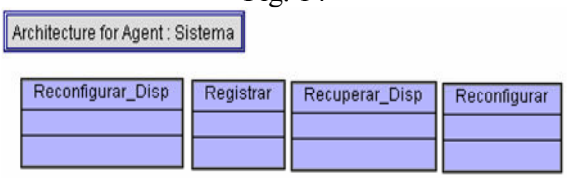


Fig.15

Fig.16

The second stage is focused in the components and the state diagram shows where it starts and where it ends, this phase also equals the external events in the different components that become the initial messages of a conversation. The state diagram for the Reconfig_Dev component is shown in figure 17. Letter S represents where the conversation [6]begins and letter L at the end of a transition represents the end of conversation.

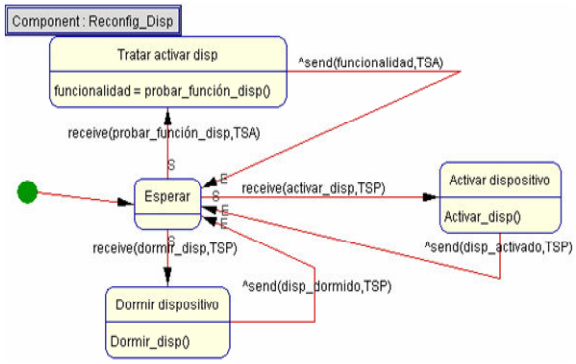


Fig. 17

The state diagram of the *Locate_DU* component is shown in figure 18.

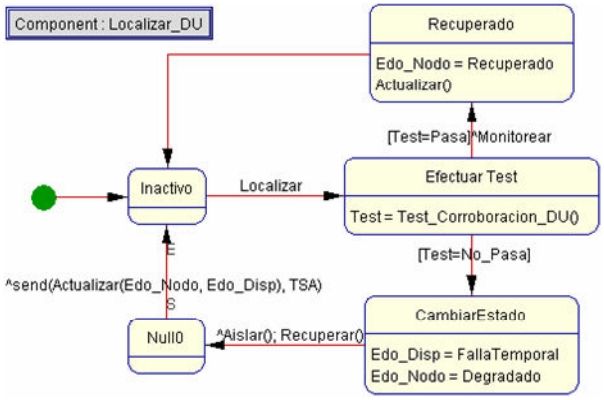


Fig. 18

There is a new null state added during the transformation stage. This state is the result of dividing the transition in the inner (isolate and recover) and external *(send (refresh (Edo_Nodo, Edo_Disp), TSA))*, that allowed a clear limiting of where the conversation start and end. At the end of the three stages of the transformation process, the inner components of the agent's architecture are shown in figures 19, 20 and 21 of the node, system and task agents. The superior part of each of the inner component represents the name of the component, the second division contains the attributes contained in the component's state diagram and the third division represents the functions that are found in the component. The transformation process creates a process for each conversation related to each component; this is because each conversation, just like the tasks, is executed in a control string. The methods and attributes can be eliminated, modified and even added.

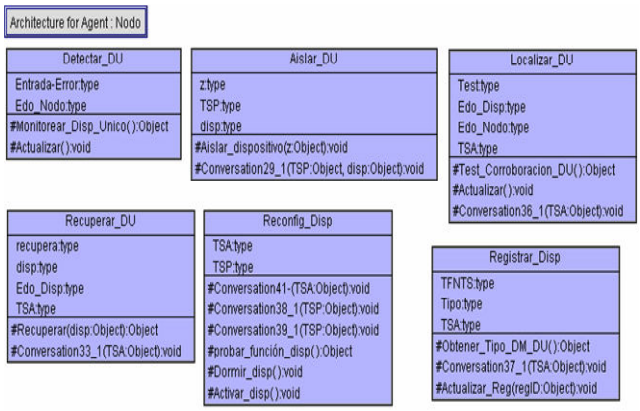


Fig. 19

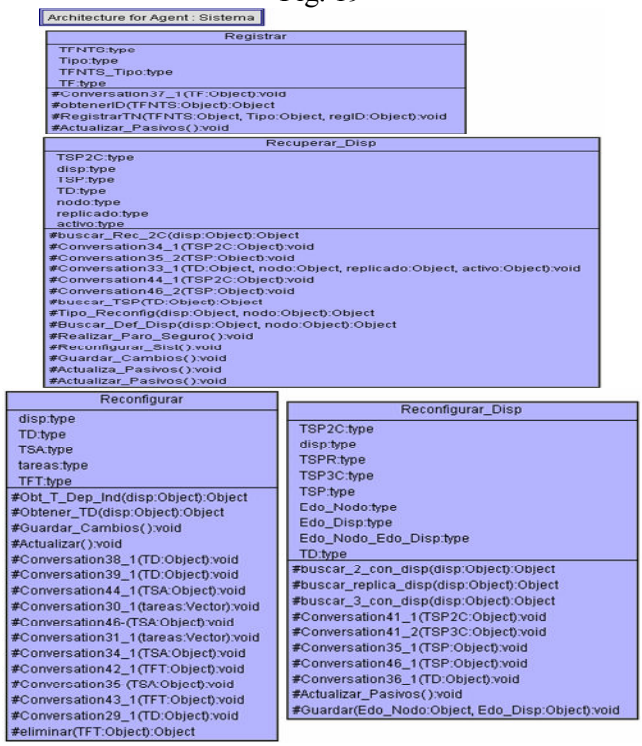


Fig. 20

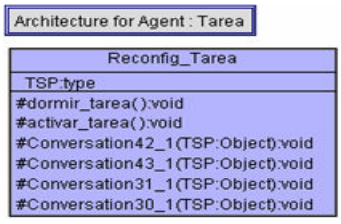


Fig. 21

3.9 The unfold diagram

The final stage of the MaSE takes the types of agents and directs them as true agents. It uses the unfold diagram to show the numbers, types and locations of the agents inside the system. This is the simplest stage because most of the work has been done in the previous stages.

A system must be analyzed in an unfold diagram before it can be coded. This is because of the differences between the agents and the agent types. An agent requires the information like a hostname and an address to participate in any communication out of the system in which it resides.

An unfold diagram also offers the designer other opportunities to adjust the system. The agents can be sorted by different machines configurations, to use in a better way the processing power or bandwidth.

Figure 22 shows an example of an unfold diagram for the SMA TF SCDRM (it can have multiple configurations), in it, it is shown three nodes; in node 1 it contains the node agent for single device (ANDU1) and the task agent 1 (AT1) that must work with a single device, besides it contains the node agent for multiple device 1 (ANDM1), the task agent 2 (AT2) that works with a multiple device and lastly contains the passive system agent 1 (ASP1) in charge of that node; node 2 contains ANDU2, AT3, ANDM2, AT4 and the active system agent (ASA); node 3 contains inactive ANDU1, in charge of the single device (copy of node 1 content) and the corresponding AT1 copies from node 1's AT1. Evenly, the ANDM2 and AT4 are found replicated, corresponding to ANDM2 and AT4 from node 2; lastly, it contains the ASP2 in charge of that node. The figure shows the required conversation in case the single device from node 1 fails (monitored by ANDU1). In this diagram, all conversation of the agent type diagram must be included. For effects of visibility, only a few conversations are shown. The 3D boxes are agents and the lines connecting them represent conversation between agents. Any conversation between agent types appears between those types' agents. Besides, a dotted box indicates that the agents are contained in the same physical platform. In some cases, the system requirements can specify a certain number of components or machines in which they reside. Otherwise, the designer must consider the messages traffic when putting agents in particular machines. Obviously, the communication speed between agents will depend on the net they're communicating thru. In some cases, the agents can be put in the same machine. When putting many agents in one equipment the advantages of the distribution obtained when using the agent diagram are destroyed. Another consideration is the processing power of one particular equipment and the required by a specific agent. If an agent has a high CPU requirement, it can be put in a machine only for

himself. One of MaSE strength is that these modifications can be done after having designed and generate a variety of system configurations, altogether with the reunion of the data from the operation. One final consideration is the automatic code generation. This methodology and AgentTool are basically used in agent system engineering. All the steps work towards that goal.

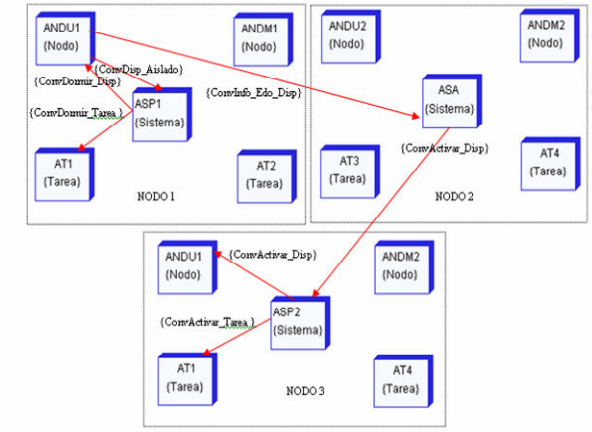


Fig. 22

4 Conclusion

In this paper a tolerant to failure distributed and hierarchy intelligent agents of a multiagent system in the control system of a mobile robot is modeled, trying to give to the robot a better guarantee of functionality, the system was modeled with the formal development methodology MaSE.

Failure to tolerance it's obtained thanks to the associate part of an agent with each node or a task that integrates the system mobile robot, when the agent is designed be a node or a task, it makes independent from the rest components of the system, besides the system agent is in charge of supply the failure tolerance in a level system, making with this a bigger trustworthiness.

The diagrams that represent the tasks and the diagrams that represent the communication have been validated thru the validation programs that are in the AgentTool with out having too much trouble.

What is concern to the physics architecture of the robot system control, the correct functionality have been validated modeling the multiagent system thru the behavior at the time of modeling and formal validated.

The principal characteristics of this tolerant to failure model over a distributed control architecture whether if is reactive, deliberative or hybrid, this can be resume in: simplicity, scalability, transparency, sistemitdad of the system and binnacle is achieve of the different type of failure that where presented during the operability of the system, witch one where capable of restoring and still they continue.

Besides, obtained a fundamental characteristics that is the increase of trustworthiness of the system.

References:

[1] Scott A. DeLoach, Analysis and Design using MaSE and agentTool, 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001) Miami University, Oxford, Ohio, March 31-April 1, 2001

[2] Mark F. Wood, Captain, USAF, Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems Thesis Degree of Master of Science in Computer Science, School of Engineering and Management Air Force Institute of Technology Air University, AFIT/GCS/ENG/00M-26, March 2000.

[3] Clint H. Sparkman, 1st Lieutenant, USAF, Transforming analysis models into design models for the multiagent systems engineering (MaSE) methodology Thesis Degree of Master of Science in Computer Science, School of Engineering and Management Air Force Institute of Technology Air University, AFIT/GCS/ENG/01M-12,March 2001.

[4] Timothy H. Lacey, Captain, USAF, A formal methodology and technique for verifying communication protocols in a multiagente environment, Thesis Degree of Master of Science in Computer Science, School of Engineering and Management Air Force Institute of Technology Air University, AFIT/GCS/ENG/00M-12.

[5] David J. Robinson, 1st Lieutenant, USAF, A component based approach to agent specification Thesis Degree of Master of Science in Computer Science, School of Engineering and Management Air Force Institute of Technology Air University, AFIT/GCS/ENG/00M-22, March 2000.

[6] Scott A. DeLoach & Mark Wood. Developing Multiagent Systems with agentTool, Intelligent Agents VII-Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000). Springer Lecture Notes in AI, Springer Verlag, Berlin, 2001.