A Method to Derive TRiStar Diagrams from Textual Descriptions of Teleo-Reactive Systems

José Miguel Morales, Pedro Sánchez
Universidad Politécnica de Cartagena
Cartagena, Spain
josemiguel.morales, pedro.sanchez@upct.es

Antonio Sánchez
Sociedad Anónima de Electrónica Submarina
Cartagena, Spain
a.sanchez@electronica-submarina.com

***Abstract*—This paper presents a method for obtaining a TRiStar diagram starting from a textual description of a Teleo-Reactive system. The steps in the method are exemplified using a Unmanned Underwater Vehicle used for deactivateing naval mines. Using this method will make the specification and reuse of Teleo-Reactive systems much easier.**

I. Introduction

Nilsson's Teleo-reactive (TR) paradigm is a goal-oriented approach for modelling autonomous reactive systems which directs the system to achieve its goal (Teleo) by reacting to changes in the environment (reactive) [1]. Morales et alter propose an extension for the i* specification language, called TRiStar, in order to adapt it to the particularities of TR systems [2] [3]. TRiStar has proven to be more effective and more efficient than the original notation [4].

The objective of this paper is to propose some methodological steps in order to obtain a TRiStar diagram starting from the description of the System To Be (STB) in natural language. In the next subsections we will give some notions on the TR paradigm and TRiStar notation. Section II will show the methodological steps we are proposing for getting a proper TRiStar diagram. And last, Section III contains some conclusions and future work.

A. *The Teleo-reactive paradigm*

The Teleo-reactive paradigm was defined by Nilsson as a robust way to guide a software agent towards its goals (see [5] for an exhaustive survey). TR programs have the ability to react robustly when conditions in the environment change, thanks to the continuous computation of the perceptions given by the sensors. TR programs take advantage of propitious changes and recover from adverse changes. TR programs are defined as a set of prioritized condition/action rules that continuously sense the environment. The action of the rule with the highest priority among all the rules whose condition is true, is executed. That execution may lead the system to satisfy the conditions of other rules with higher priority or, eventually, the final goal of the STB. Here follows a simple example of a TR program:

```
UUV:
  MineDeactivated && nextToOwnship -> nil
  MineDeactivated && !nextToOwnship -> ComeBack
  true -> DeactivateMine

ComeBack:
  OwnshipAhead -> forward
  true -> search

DeactivateMine:
  NextToMine -> deactivate
  true -> FindMine

FindMine:
  MineAhead -> forward
  true -> search
```

The previous code corresponds to a very simplified Unmanned Underwater Vehicle (UUV) which is submerged from a mothership in a zone where a naval mine is known to be; the vehicle looks for the mine, deactivates it and goes back to the mothership.

The program starts executing the main goal of the vehicle (`UUV`). At the beginning, among the three rules in that goal, only the condition of the third rule is true (in fact, it is always true). Then, the action `DeactivateMine` is executed. Only the condition of the second rule in `DeactivateMine` is true and that is why the action `FindMine` starts its execution. Analogously the action that finally executes is the task `search`. When the UUV starts searching, it makes some movements in order to find a particular element. As a consequence of those movements it may eventually face a mine and the condition of the first rule (`MineAhead`) becomes true. As the first rule has a higher priority than the second one, the UUV stops searching and starts going forward.

Now suppose that a stream in the water helps the vehicle and it arrives to the mine before than expected. The condition `nextToMine` becomes true and the vehicle can proceed with the deactivation of the mine. If the stream changes and the vehicle is thrown away, the program will recover itself executing again the `search` task until the mine is found again. The advantages of using the TR paradigm can be seen more clearly if you think of the equivalent statechart that considers all the possible transitions.

This example will be used all along this paper and is deeply described in section II.

B. *TRiStar*

i* notation provides a lot of advantages for graphically specifying goal-oriented systems, but some weaknesses have

ISSN 2305-7254

been described when using i* for specifying TR systems. This is why Morales et al. decided to develop TRiStar, an extension of i* which overcomes those weaknesses [2]. In order to prove that the new notation was more effective and efficient than the original one when specifying TR systems, a family of experiments was carried out and its results published in [4].

As stated before, a TRiStar diagram inherits and extends i* graphical notation. Fig. 1 shows a summary of the graphical elements used in i*. Those elements allow the modeling of: hierarchical decomposition of goals (ellipses), the actions taken in every goal (hexagons) and the conditions that enable the execution of those actions (links to rectangles). In addition, the behavior of every agent is encapsulated inside a shadowed ellipse.

The additions made by TRiStar to the original i* notation are summarized in Fig. 2. See [2] for a complete description. The new elements are the following:

- Logical Resources: this type of resources are used to represent boolean combinations of other resources. In addition to the resource, a table containing the equivalent boolean expression is needed in the diagram.
- Prioritized Decomposition Links: i* doesn't provide a representation for priorities among the different subgoals and tasks of a goal. To stablish priorities, TRiStar adds a number of marks to the i* decomposition link. The fewer marks, the lower priority of the linked subgoal or task.
- Decomposition Link Dependency: i* offers dependency links on resources, goals and tasks, but not on decomposition links. Through this link we can represent the condition / action relation present in the rules of a TR program. The resource represents a perception which acts as a condition in the rule represented by the decomposition link between a goal and its subgoal.
- Logical Resource Dependency: this link gives the relationship among a logical resource and the resources used in its boolean expression.

These extensions allow graphically modelling TR systems, which makes it easier for stakeholders and other non-technical people the understanding and reuse of TR specifications. Fig. 3 shows the TRiStar diagram corresponding to the UUV example described in the previous section. In that figure, examples of use of all those elements can be observed. Section II deeply analyzes that example.

Obtaining the TR program which corresponds to a TRiStar diagram is a simple and direct process and is described in [4]. For that reason, obtaining a TRiStar diagram can be considered equivalent to obtaining a TR program.

II. The Method

In this section we will use the UUV example described in the previous section in order to show the main steps that will guide the process to obtain a TRiStar diagram like that in Fig. 3 starting from a description of the STB in natural language. The description of our example is the same as we stated in section I:

"The STB is an UUV which is submerged from a mothership in a zone where a naval mine is known to be; the vehicle looks for the mine, deactivates it and goes back to the mothership."

A. STEP 1: Identifying the Main Goal

The first thing we need to clarify when designing a TR system is what the system is going to do. That is what we call the Main Goal (MG). Try to find a sentence that summarizes the purpose of the STB as simply as possible. Boolean relationships such as *and*, *or* and *not* between concepts may be used in the MG. In the UUV example, the MG is:

"To deactivate the naval mine and go back to the mothership".

B. STEP 2: Identifying the elements

Once we have identified the MG we need to identify which elements are available to achieve that MG. Three types of elements need to be identified:

- Sensors available to be used in the STB and the perceptions those sensors can provide. For TR systems we need boolean perceptions, i.e. perceptions that tell us if a condition is true or false. We will probably need to write some wrapper for the off-the-self sensors that can be found in the market.
- Actuators available to be used in the STB and the actions those actuators are able to make.
- Beliefs: taken from the TeleoR approach [7], it has

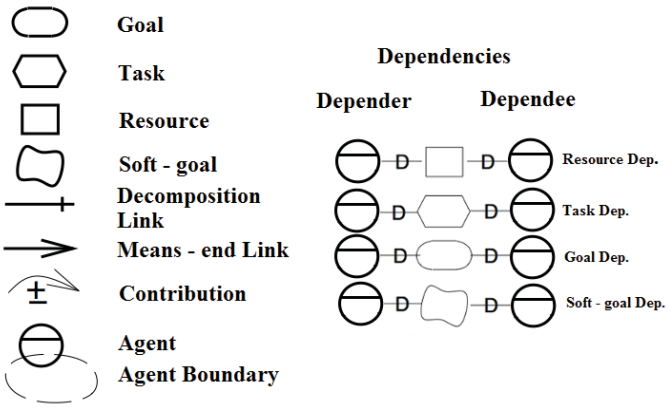

Fig. 1. i* graphical elements

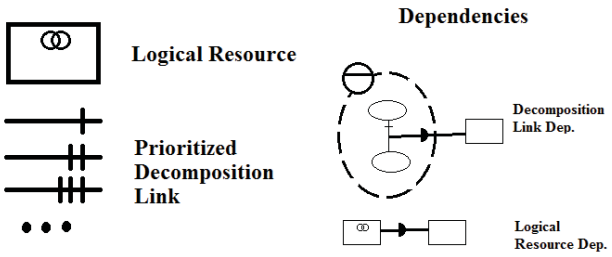

Fig. 2. TRiStar added graphical elements

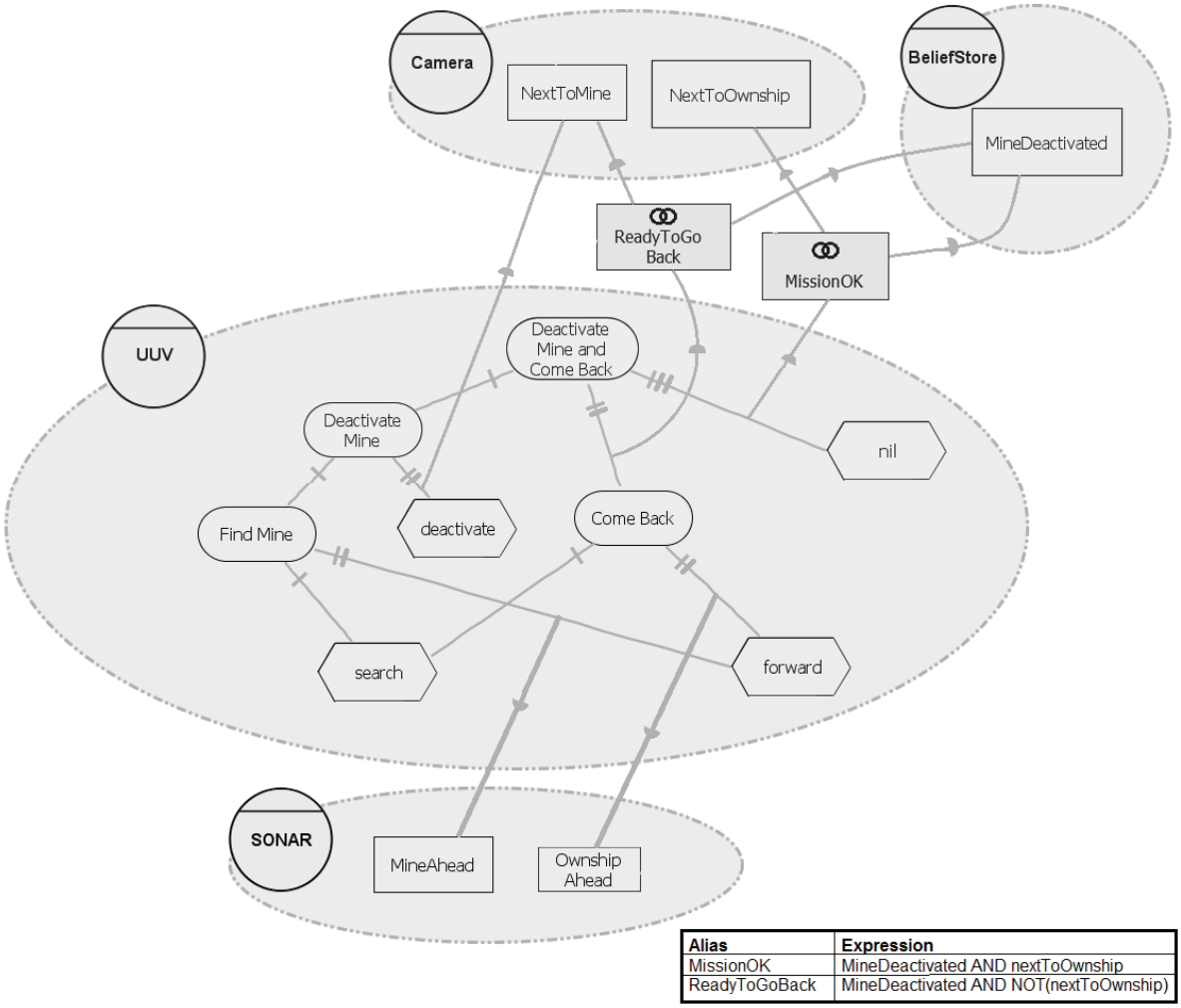

Fig. 3. UUV TRiStar diagram

become usual in TR systems to use beliefs that may affect the processing just as if they were perceptions. Those beliefs are stored in the `BeliefStore`.

In the UUV example the identified sensors are the `SONAR` and the `Camera`. The `SONAR` provides the following perceptions:

- `MineAhead`: this perception becomes true when the `SONAR` detects a naval mine in front of the UUV. The `SONAR` processes the acoustic signal received and a wrapper states if there is an object within its range that could be a naval mine.
- `OwnshipAhead`: this perception becomes true when the `SONAR` detects the mothership just in front of the UUV. In the same way as the previous perception, the `SONAR` processes the acoustic signal received and the wrapper states if the mothership is within its range.

The perceptions provided by the `Camera` agent are the following:

- `NextToMine`: this perception becomes true when the `Camera` detects that the UUV is near enough to the mine in order to deactivate it. The wrapper of the `Camera` will state if the captured images correspond to a mine within the range of the `Arm`.
- `NextToOwnship`: this perception becomes true when the `Camera` detects that the UUV is near enough to the mothership in order to be recovered. The wrapper of the `Camera` will state if the captured images correspond to the mothership being at a distance that allows its recovery.

On the other hand, the actuators identified in the UUV example are the `Engine` and the `Arm`. The actions that the UUV is able to make using the `Engine` are the following:

- `forward`: this action allows the UUV to move forward following a rectilinear trajectory in the direction it is facing.

- `search`: this action allows the UUV to turn over itself following a pattern designed to maximize the possibilities of finding a naval mine. While the UUV follows this pattern the `SONAR` is at the same time trying to detect the mine.

The only action that the UUV is able to make using the `Arm` is `deactivate`. Through this action the UUV will deactivate the naval mine by, for example, cutting the moors of a moored mine.

At last, the only identifed belief in the UUV example is `MineDeactivated`. When the mine is finally deactivated, this belief becomes true in the `BeliefStore`.

C. STEP 3: Decomposing MG into subgoals

Decompose the MG into subgoals trying to answer to the question: "HOW can the MG be achieved?". Take each of those subgoals and try to decompose them again into new subgoals or actions allowed by the available actuators. Alternative decompositions may be considered representing different solutions to the same problem. At the end of this step, all the subgoals must be decomposed into actions. Otherwise, new actuators may be needed. At this moment, some of the alternative decompositions may be discarded due to the lack of appropriate actuators.

When dividing goals and subgoals keep always in mind the perceptions that the available sensors are providing. Take into account that the system needs to know when a goal or subgoal has been achieved. The only way for knowing that is through the sensors of the system.

In this step we can start drawing the TRiStar diagram by representing the STB as an agent. Draw the MG as a TRiStar goal (an ellipse with the name of the goal in it) in the upper part of the agent that represents the STB. Just below the MG draw all the subgoals you have divided the MG into using the same representation (ellipses). If you have identified any action, draw them as TRiStar tasks (hexagons). Connect the MG and the new subgoals and tasks using decomposition links (see Fig. 1).

If you are considering alternative decompositions for a certain goal, use means – end links for each of them (See Fig. 1). Fig. 4 shows an example of the use of means – end links for this purpose. In order to achieve `Goal1`, two possibilities have been considered: `SubgoalAlt_1` and `SubGoalAlt_2`. Each alternative is decomposed in its own subgoals and tasks. Eventually, one of those alternatives will be chosen to be implemented.

Fig. 5 shows the diagram of the UUV example at the end of this step. See how we have drawn the MG in the upper part of the UUV agent. We have divided it into two subgoals: `Deactivate Mine` and `Come Back`. The task `nil` represents in this case the achievement of the goal: as the goal has been achieved, the system has nothing more to do. Then, `Deactivate Mine` has been divided into the subgoal `Find Mine` and the task `deactivate`. Both `Find Mine` and `Come Back` subgoals have been divided into the tasks `search` and `forward`. In the first case, the task `search` will eventually put the UUV facing a mine. At that moment the UUV can move forward to reach it. In the second case the behavior is similar but facing the mothership. This difference will be solved in step 5 using the sensors of the system.

D. STEP 4: Prioritization

In this step, priorities among the different subgoals and tasks are introduced. In TR systems, the order in which the subgoals are being achieved is very important. In the UUV example, the vehicle obviously needs to find the mine prior to deactivating it. TRiStar uses prioritized decomposition links to stablish the order in which each subgoal or task has to be achieved. The fewer marks in the link, the earlier that task has to be initiated. The priority stablished in this way will determine the order of the rules in the resulting TR program.

Fig. 6 shows the UUV TRiStar diagram including the priorities among the subgoals. See how the decomposition link

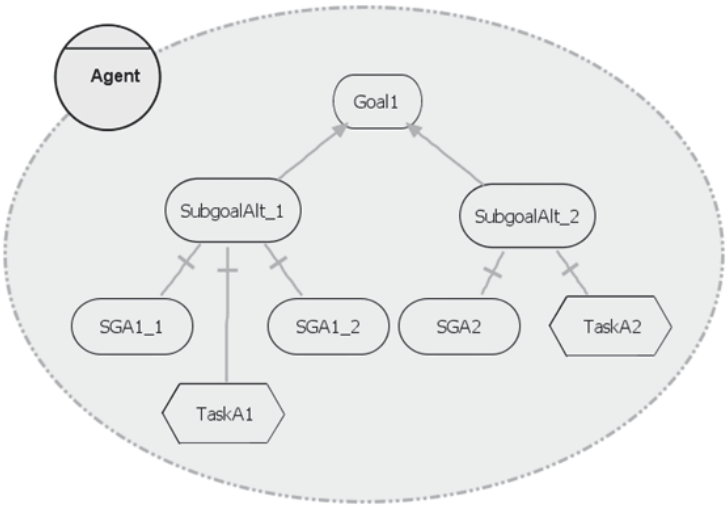

Fig. 4. Alternative decompositions

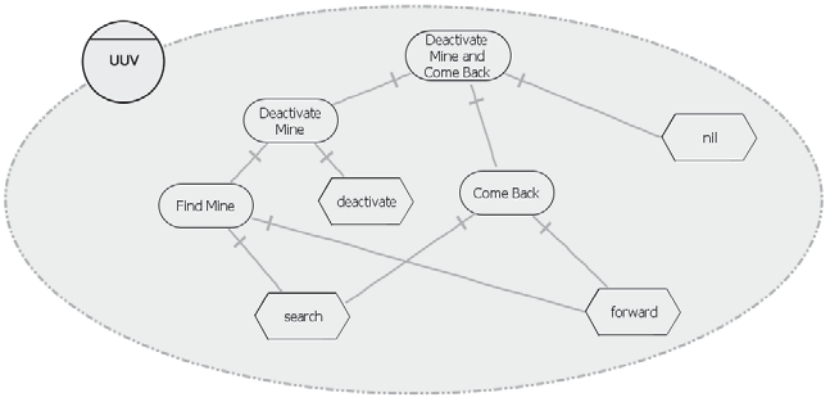

Fig. 5. UUV after step 3

between `Deactivate Mine` and `Find Mine` has only one mark and that between `Deactivate Mine` and `deactivate` has two marks. When the system tries to achieve `Deactivate Mine` it needs to find where the mine is and move next to it in order to deactivate it using the `Arm`.

E. STEP 5: Monitoring the environment

The STB needs to know when it has achieved the MG or any of the subgoals. The perceptions given by the available sensors are the only way of being aware of the changes in the environment. For this purpose, we need to stablish a correspondance between each subgoal and a boolean combination of the perceptions given by the sensors of the STB.

In the UUV example, the MG can be represented by the following expression:

```
mineDeactivated AND nextToOwnship
```

In addition, the system needs to know when a subgoal has been achieved and when to stop executing a task and start executing the next one. The same process must be accomplished with the rest of subgoals and tasks. If a correspondance for any of the subgoals or tasks cannot be stablished we may need to get new sensors or discard the alternatives which include those subgoals or tasks. The expression representing the achievement of a subgoal will be the same as the one representing the achievement of its subgoal with the highest priority (or the accomplishment of its task with the highest priority).

In the UUV example, the subgoal `Find Mine` can be considered as achieved when the vehicle is located next to the mine. Then, we can use the perception `NextToMine` to represent the achievement of the subgoal `Find Mine` and the trigger for the next task to be done once the mine has been found: deactivate it.

The sensors whose perceptions are being used in the STB will be represented in the diagram as TRiStar agents. The perceptions provided by those sensors will be represented as resources inside the agent representing the corresponding sensor. See for example the agent `Camera` and the resource `NextToMine` inside it in Fig. 3.

Boolean expressions combining two or more perceptions will be represented as TRiStar logical resources. See `ReadyToGoBack` or `MissionOK` in Fig. 3 as examples. The boolean expresions corresponding to those logical resources can be found in the table at the lower left corner of the figure. Notice how each logical resource is linked to the perceptions used in those expresions through dependency links. For example, looking at the table in Fig. 3 we can see that `ReadyToGoBack` corresponds to the expression:

```
mineDeactivated AND NOT(nextToOwnship)
```

For that reason, the logical resource `ReadyToGoBack` in Fig. 3 is linked to both `MineDeactivated` and `NextToOwnship` resources.

When a subgoal (*subA*) has been achieved, the system must go on trying to achieve the next subgoal (*subB*). The condition that represents the achievement of *subA* is at the same time the trigger to start trying to achieve *subB*. This is represented in a TRiStar diagram as a dependency link between the resource that represents the achievement of *subA* and the decomposition link ending in *subB*. That is why the MG is always linked to a decomposition link ending in the `nil` task: when the MG has been achieved, nothing more has to be done. As stated before in the UUV example, `NextToMine` means that the `FindMine` subgoal has been achieved and that the `deactivate` task must begin. See in Fig. 3 how resource `NextToMine` is linked to the decomposition link between the subgoal `DeactivateMine` and the task `deactivate`.

Although both `Find Mine` and `Come Back` subgoals share the same tasks (`search` and `forward`) the difference between them is in the perceptions they use to decide that the system doesn't need to continue executing those tasks. In `Find Mine` the system stops searching when the `SONAR` detects a mine in front of the UUV (perception `MineAhead`). Nevertheless in `Come Back` the system stops searching when the `SONAR` detects the mothership in front of the UUV (perception "`OwnshipAhead`").

At the end of this step the TRiStar diagram is completed. The complete TRiStar diagram for the UUV example has been already shown in Fig. 3.

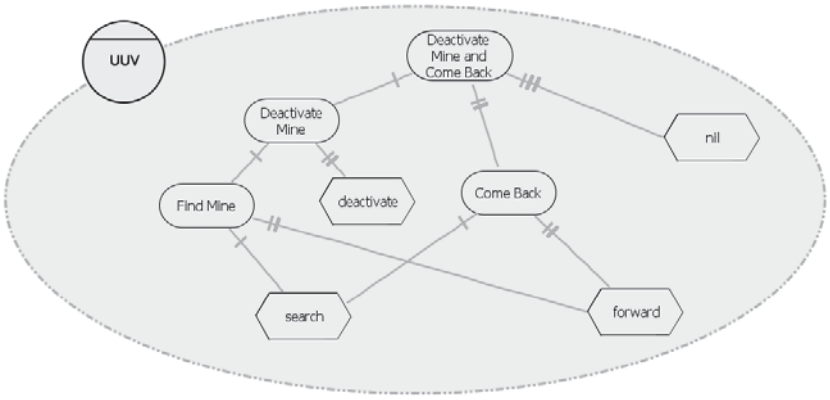

Fig. 6. UUV after step 4

F. THE METHOD IN A NUTSHELL

The five steps needed to get a TRiStar diagram starting from the textual description of a TR system are the following:

1) Identify the Main Goal.
2) Identify the elements: sensors (and their perceptions), actuators (and their actions) and beliefs.
3) Decompose the Main Goal into subgoals and tasks and those subgoals into new subgoals or tasks until there are no subgoals left.
4) Stablish the priority among the subgoals and tasks.
5) Monitor the environment using the identified perceptions and beliefs to state when a subgoal has been achieved and the next task needs to be started.

III. CONCLUSIONS AND FUTURE WORK

We have presented five methodological steps to obtaining a TRiStar diagram starting from a textual description of a Teleo-Reactive system. In addition we have proven that this steps can be easily applied by showing a complete example.

In the near future we pretend to design a family of experiments to test the usefulness of this methodology. In each of those experiments we will present a textual description of a TR system to two group of subjects. One group will try to obtain the TR program which implements the proposed system directly from the description. The other will be told to use these methodological steps to obtain the correspondant TRiStar diagram. We will measure and compare the correctness of the obtained results and the time used to get them.

Although the transformations needed to obtain a TR program from a TRiStar diagram are clearly stated, a tool that allows making this transformation automatically would be really useful. In addition, that tool should allow drawing the TRiStar diagram in a manner similar to that of OpenOME, an open source tool to draw i* diagrams [7].

ACKNOWLEDGEMENT

This paper is the result of the research carried out under the Research Program for Groups of Scientific Excellence of the Seneca Foundation (Agency for Science and Technology of the Region of Murcia, ref. 19895/GERM/15) and has been partially supported by the CDTI's OCEAN MASTER (Multipurpose Autonomous System for different Environment and Roles) project in the FEDER INNTERCONECTA 2015 program.

The authors wish to thank SAES (Sociedad Anónima de Electrónica Submarina) [8] for their generosity and support for this research.

REFERENCES

[1] N. J. Nilsson, "Teleo-reactive programs for agent control", *J. Artif. Intell. Res.* 1(1), 1993, pp. 139–158.
[2] J. M. Morales, E. Navarro, P. Sánchez and D. Alonso, "TRiStar: an i* extension for Teleo-reactive systems requirements specifications", *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, April 13-17 2015. Salamanca (Spain) , pp. 283-288.
[3] E. Yu, "Towards modelling an reasoning support for early-phase requirements engineering", *Proc. of the 3rd IEEE International Symposium on Requirements Engineering*, January 6-8, 1997. Washington D.C. (USA), pp. 226-235.
[4] J. M. Morales, E. Navarro, P. Sánchez and D. Alonso, "A family of experiments to evaluate the understandability of TRiStar and i∗ for modeling teleo-reactive systems", *The Journal of Systems and Software*, 114, pp. 82-100.
[5] J. M. Morales, P. Sánchez and D. Alonso, "A systematic literature review of the Teleo-reactive paradigm", *Artificial Intelligence Review*, 42, pp. 945–964.
[6] K. L. Clark and P. J. Robinson, "Robotic agent programming in TeleoR", *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 26, 2015. pp. 5040-5047.
[7] OpenOME official website, Web: https://se.cs.toronto.edu/trac/ome/wiki
[8] SAES official website, Web: http://www.electronica-submarina.com.