Obtención de diagramas de objetivos para sistemas Teleo-Reactivos: una aproximación metodológica

José M. Morales[1], Pedro Sánchez[1], Bárbara Álvarez[1] , Antonio Sánchez García[2] y Elena Navarro[3]

[1] DSIE, Universidad Politécnica de Cartagena, España
[2] Sociedad Anónima de Electrónica Submarina, Cartagena, España
[3] LoUISE, Universidad de Castilla-La Mancha, España
josemiguel.morales@upct.es

Resumen. Este artículo presenta un método para obtener un diagrama TRiStar partiendo de la descripción textual de un sistema Teleo-Reactivo. El método se ilustra con un ejemplo clásico en la literatura de sistemas Teleo-Reactivos: el recolector de latas. El uso de este método permitirá facilitar la especificación y la reutilización de sistemas Teleo-Reactivos.

Palabras clave: Teleo-Reactivo, TRiStar, metodología.

1 Introducción

El paradigma Teleo-Reactivo (TR) de Nilsson es un enfoque orientado a objetivos para modelar sistemas autónomos reactivos en el que se dirige al sistema a conseguir sus objetivos (Teleo) reaccionando ante los cambios del entorno (reactivo) [1]. Morales et al. proponen una extensión para el lenguaje de especificación i*, llamada TRiStar, para adaptarlo a las particularidades de los sistemas TR [2][3]. TRiStar ha demostrado ser más efectivo y eficiente que la notación original [4].

El proceso de creación de un modelo a partir de una descripción textual de un sistema puede llegar a ser complicado, sobre todo a medida que se abordan sistemas de mayor complejidad. La literatura recoge múltiples propuestas de recomendaciones metodológicas para facilitar este tipo de pasos, siendo la metodología TROPOS [5] el ejemplo más paradigmático en el caso de los diagramas i* para su uso en el ámbito de la programación orientada a agentes. El caso de los diagramas TRiStar no es una excepción.

El objetivo de este artículo es proporcionar una serie de pasos metodológicos para obtener un diagrama TRiStar a partir de la descripción en lenguaje natural del System to Be (STB). En las siguientes secciones aportaremos algunas nociones sobre el paradigma TR y la notación TRiStar. En la sección 2 mostraremos los pasos que proponemos para obtener un diagrama TRiStar correcto. Por último la sección 3 contiene algunas conclusiones y trabajos futuros.

1.1 El paradigma Teleo-Reactivo

El paradigma Teleo-Reactivo fue definido por Nilsson como una forma robusta de guiar a un agente software hacia sus objetivos (ver [6] para una revisión sistemática de la literatura). Los programas TR tienen la habilidad de reaccionar de forma robusta ante cambios en las condiciones del entorno gracias al procesamiento continuo de las percepciones que proporcionan los sensores. Los programas TR aprovechan los cambios propicios y se recuperan ante cambios adversos. Se definen como un conjunto de reglas condición / acción ordenadas por prioridad que están continuamente sensorizando el entorno. La acción correspondiente a la regla con mayor prioridad de entre todas aquellas cuya condición es verdadera será la que finalmente se ejecute. La ejecución de dicha acción puede llevar a que se hagan verdaderas las condiciones de reglas de mayor prioridad o incluso el objetivo final del STB. A continuación mostramos un sencillo ejemplo de un programa TR adaptado de [7]:

```
Robot:
    !holding && allCansCollected -> nil
      holding -> Deliver
      True -> Collect

Collect:
      seeCan -> Fetch
      True -> rotate

Fetch:
      touching -> grasp
      True -> forward

Deliver:
      atDepot -> ungrasp
      True -> GoDepot

GoDepot:
      seeDepot -> forward
      True -> rotate
```

Se trata de un robot desarrollado para recoger latas del suelo y llevarlas a un depósito. El robot es capaz de rotar (`rotate`), avanzar (`forward`) y abrir (`ungrasp`) o cerrar (`grasp`) una pinza. Además incluye sensores que le permiten saber si está viendo (`seeCan`), tocando (`touching`) o sosteniendo (`holding`) una lata o bien si el depósito está a la vista (`seeDepot`). También es capaz de saber si se encuentra en el depósito (`atDepot`). Cada vez que lleva una lata al depósito va actualizando un contador. Cuando ese contador llega a un número determinado, `allCansCollected` se hace verdadero.

El programa arranca ejecutando el objetivo principal del robot (`Robot`). Al principio la única regla cuya condición es verdadera entre las dos de Robot es la segunda, que es siempre cierta. Por tanto el robot ejecuta el subgoal `Collect` y aplicando el

mismo razonamiento ejecuta la acción correspondiente a la segunda regla y comienza a rotar. Es de esperar que a causa de esa rotación en algún momento entre en su campo de visión una lata haciendo que la condición `seeCan` se vuelva verdadera. Como la primera regla de `Collect` tiene más prioridad dejará de ejecutarse la acción `rotate` y el robot pasará a ejecutar el subgoal `Fetch`. Una vez más la regla que se ejecuta es la correspondiente a la condición `True` y el robot comenzará a moverse hacia adelante como consecuencia de la ejecución de `forward`. Ese movimiento debería ir acercando el robot a la lata hasta entrar en contacto con ella y haciendo verdadera la condición `touching`. En ese momento se activará la primera regla de `Fetch` y el robot cerrará la pinza (`grasp`) cogiendo la lata y haciendo que se cumpla `holding`. El hecho de que `holding` sea verdadera provocará que la primera regla de `Robot` tome el control haciendo que, de forma similar a la mostrada hasta ahora, el robot lleve la lata hasta el depósito (`GoDepot`).

Supongamos ahora que alguien desea facilitarle el trabajo al robot y le coloca una lata en la pinza. La condición `holding` se haría verdadera y el robot procedería a llevarla al depósito ejecutando el subobjetivo `Deliver`, aprovechando la ayuda prestada. Si por el contrario esa misma persona le quita la lata de la pinza, `holding` se volverá falso y el robot comenzará inmediatamente a buscar una nueva lata para llevarla al depósito, recuperándose ante la situación adversa. Las ventajas del paradigma TR se ven mucho más claras planteándose el diagrama de estados equivalente que contemple todas las posibles transiciones.

1.2 TRiStar

La notación i* proporciona muchas ventajas para especificar gráficamente sistemas orientados a objetivos. Sin embargo, a la hora de especificar sistemas TR se han encontrado una serie de debilidades. Por esta razón, Morales et al. decidieron desarrollar TRiStar, una extensión de i* que supera dichas debilidades [2]. Para probar que la nueva notación era más efectiva y eficiente que la original se llevó a cabo una familia de experimentos cuyos resultados fueron publicados en [4].

Como ya se ha mencionado anteriormente, un diagrama TRiStar hereda y extiende la notación i*. La figura 1 muestra un resumen de los elementos gráficos usados en i*. Esos elementos permiten modelar descomposiciones jerárquicas de objetivos (elipses), las acciones ejecutadas en cada objetivo (hexágonos) y las condiciones que disparan la ejecución de dichas acciones (enlaces a rectángulos). Además, el comportamiento de cada agente se encapsula en una elipse sombreada.

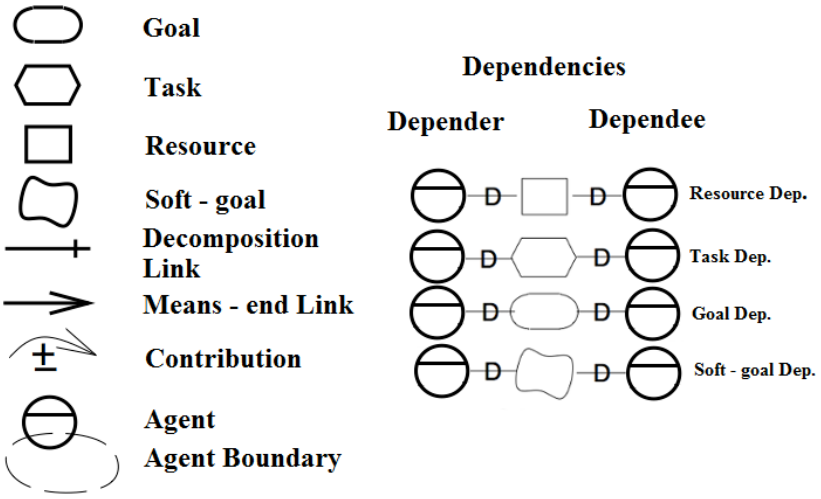

Fig. 1. Elementos gráficos de i*

Las adiciones realizadas por TRiStar a la notación i* original se resumen en la figura 2. En [2] se puede encontrar una descripción completa de la notación. Los elementos añadidos son los siguientes:

- Recursos lógicos: este tipo de recursos se usan para representar combinaciones booleanas de otros recursos. Además del recurso en sí, se necesita una tabla en el diagrama para contener la expresión booleana equivalente.
- Enlaces de descomposición con prioridad: i* no proporciona una representación para la prioridad entre los distintos subobjetivos y tareas de un mismo objetivo. Para establecer esas prioridades TRiStar añade una serie de marcas al enlace de descomposición original. Cuantas menos marcas tenga el enlace, menor será la prioridad del subobjetivo o tarea enlazado.
- Dependencia sobre enlace de descomposición: i* ofrece enlaces de dependencia a recursos, objetivos y tareas, pero no sobre un enlace de descomposición. Mediante este enlace se puede representar la relación condición / acción presente en las reglas de un programa TR. El recurso representa una percepción que actúa como una condición en la regla representada por el enlace de descomposición entre un objetivo y su subobjetivo.
- Dependencia de recurso lógico: este enlace establece la relación entre un recurso lógico y los recursos utilizados en su expresión booleana.

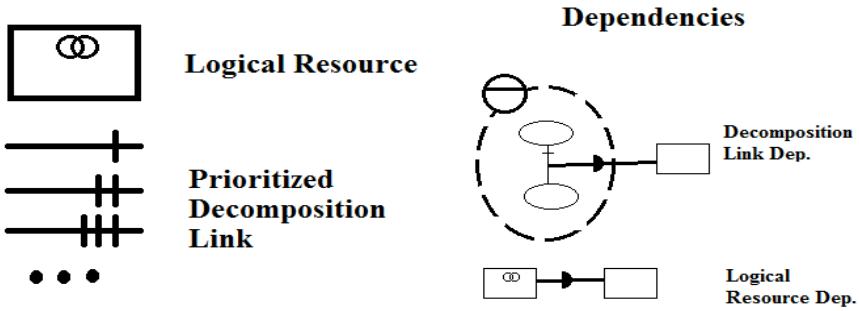

Fig. 2. Elementos gráficos añadidos a TRiStar

Estas extensiones permiten modelar gráficamente sistemas TR, lo que facilita al personal no técnico interesado entender y reutilizar las especificaciones TR. La figura 3 muestra el diagrama TRiStar correspondiente al ejemplo del recolector de latas descrito en la sección anterior e incluye ejemplos de todos los elementos enumerados. La sección 2 analiza dicho ejemplo en detalle.

Obtener un programa TR a partir de un diagrama TRiStar es un proceso simple y directo que se describe con gran detalle en [4]. Por lo tanto, podemos considerar que obtener el diagrama TRiStar de un sistema es equivalente a obtener su programa TR.

2 El método

En esta sección usaremos el ejemplo del robot recolector descrito en la sección anterior para mostrar los pasos principales que guiarán el proceso de obtener un diagrama TRiStar como el de la Figura 3 partiendo de la descripción del STB en lenguaje natural. La descripción de nuestro ejemplo es la misma que establecimos en la sección 1:

"El STB es un robot capaz de encontrar un número determinado de latas en su entorno, llevarlas a un depósito y dejarlas allí. Para ello cuenta con una cámara que le permite identificar las latas y el depósito; un GPS que le permite saber si se encuentra en el depósito; un motor que le permite moverse hacia adelante y rotar y una pinza que le permite saber si está tocando una lata y cogerla."

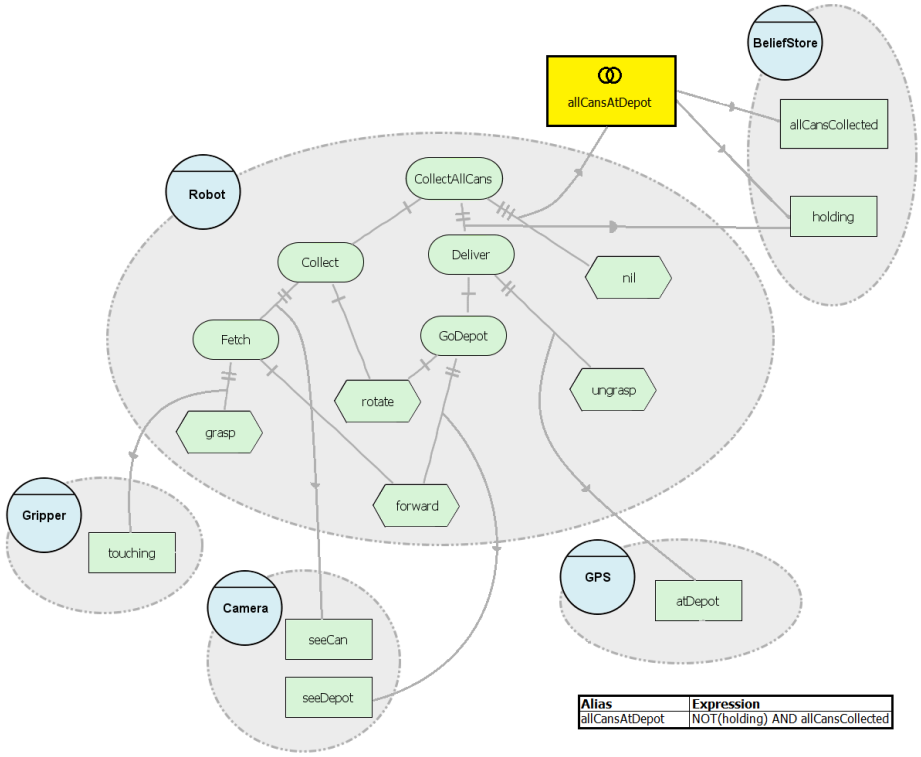

Fig. 3. Diagrama TRiStar del robot recolector de latas

2.1 PASO 1: Identificar el Objetivo Principal

Lo primero que debemos clarificar al diseñar un sistema TR es qué va a hacer el sistema. Eso es a lo que llamamos el Objetivo Principal (OP). Debemos encontrar una frase que resuma el propósito del STB de la forma más simple posible usando si es necesario relaciones booleanas. En nuestro ejemplo el OP podría ser:

"Llevar todas las latas al depósito y dejarlas allí".

2.2 PASO 2: Identificar los elementos

Una vez que hemos identificado el OP tenemos que identificar los elementos de los que disponemos para conseguir dicho OP. Necesitamos identificar tres tipos de elementos:

- Sensores disponibles y percepciones que dichos sensores proporcionan. Para diseñar sistemas TR necesitamos percepciones booleanas, es decir, percepciones que nos digan si una condición es verdadera o falsa. Probablemente necesitaremos escribir wrappers para los sensores off-the-self que pueden encontrarse en el mercado.

- Actuadores disponibles y las acciones que dichos actuadores permiten realizar.
- Creencias: tomadas de TeleoR [8], es usual usar en sistemas TR creencias que pueden afectar al procesamiento de la misma manera que si fueran percepciones. Esas creencias se almacenan en la `BeliefStore`.

En el ejemplo del recolector se identifican tres sensores: la cámara (`Camera`), el GPS y la pinza (`Gripper`). La cámara proporciona las siguientes percepciones:

- `seeCan`: esta percepción se hace verdadera cuando la cámara reconoce una lata frente al robot.
- `seeDepot`: esta percepción se hace verdadera cuando la cámara reconoce el depósito frente al robot.

El GPS proporciona una única percepción:

- `atDepot`: esta percepción se hace verdadera cuando el robot se encuentra dentro del perímetro del depósito.

La pinza por su parte proporciona también una única percepción:

- `touching`: esta percepción se hace verdadera cuando la pinza está tocando un objeto y por tanto se encuentra en la posición adecuada para ser agarrado por la pinza.

Pasemos ahora a identificar los actuadores. En nuestro ejemplo los actuadores son dos: la pinza y el motor. Las acciones que el robot es capaz de ejecutar usando el motor son las siguientes:

- `rotate`: provoca que el robot gire sobre sí mismo permitiendo así que la cámara barra el espacio alrededor del robot.
- `forward`: el robot avanza en la dirección en la que lo ha dejado la última acción rotate ejecutada.

La pinza, por su parte, permite al robot realizar las siguientes acciones:

- `grasp`: cierra la pinza agarrando aquello que esté a su alcance en ese momento.
- `ungrasp`: abre la pinza soltando lo que pudiera tener agarrado.

Por último, las creencias identificadas en el ejemplo del recolector son las siguientes:

- `holding`: cuando `touching` es verdadero y el robot ejecuta `grasp`, `holding` se hace verdadero representando que el robot está sosteniendo la lata que acaba de coger. Se hará falso de nuevo cuando el robot ejecute `ungrasp`, soltando la lata.
- `allCansCollected`: esta creencia se hace verdadera en el momento en que el robot recoge la última lata y se desplaza hasta el depósito.

2.3 PASO 3: Descomponer el OP en subobjetivos

Se debe descomponer el OP en subobjetivos tratando de responder a la pregunta "¿CÓMO puede alcanzarse el OP?". Esos subobjetivos deben descomponerse en nuevos subobjetivos o en acciones de las que proporcionan los actuadores disponibles. Se puede considerar descomposiciones alternativas representando soluciones diferentes al mismo problema. Al final de este paso, todos los subobjetivos deben estar descompuestos en acciones. Si esto no es posible quizá sea necesario adquirir nuevos actuadores. En este momento algunas de las descomposiciones alternativas pueden descartarse por falta de actuadores apropiados.

A la hora de dividir los objetivos y subobjetivos hay que tener siempre en cuenta las percepciones que proveen los sensores de los que disponemos. El sistema necesita saber cuándo un subobjetivo ha sido alcanzado y la única forma de saberlo es mediante sus sensores.

En este paso podemos empezar a dibujar el diagrama TRiStar representando el STB como un agente. El OP se dibuja como un objetivo (una elipse con el nombre del objetivo en su interior) en la parte de arriba del interior del agente que representa el STB. Debajo del OP se dibujan todos los subobjetivos en los que se ha dividido el OP usando la misma representación. Las acciones que se hayan identificado deben dibujarse como tareas TRiStar (hexágonos). Una vez dibujados todos los subobjetivos y tareas, hay que conectarlas usando enlaces de descomposición como los que se ven en la Figura 1.

Si se están considerando descomposiciones alternativas para un determinado objetivo se debe usar enlaces tipo mean-end (ver Figura 1). La Figura 4 muestra un ejemplo de uso de esos enlaces. Para alcanzar el `Goal1` se han considerado dos posibilidades: `SubgoalAlt_1` y `SubgoalAlt_2`. Cada altenativa se descompone en sus propios subobjetivos y tareas. En algún momento una de esas alternativas será elegida para ser implementada.

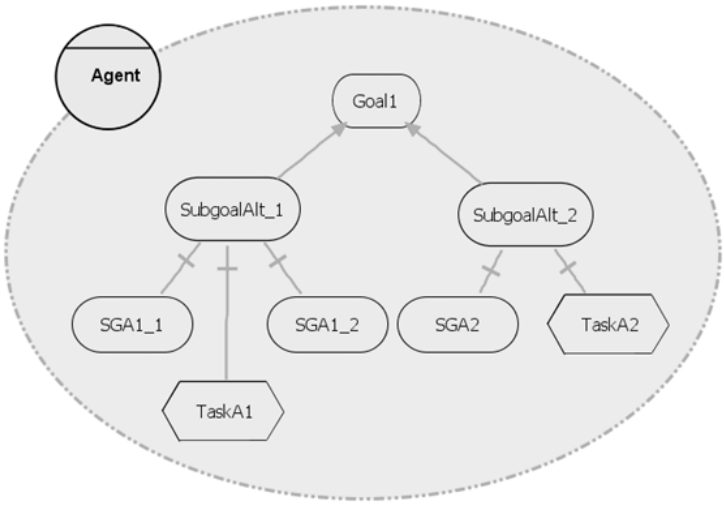

Figura 4. Descomposiciones alternativas

La Figura 5 muestra cómo quedaría el diagrama TRiStar al final de este paso.

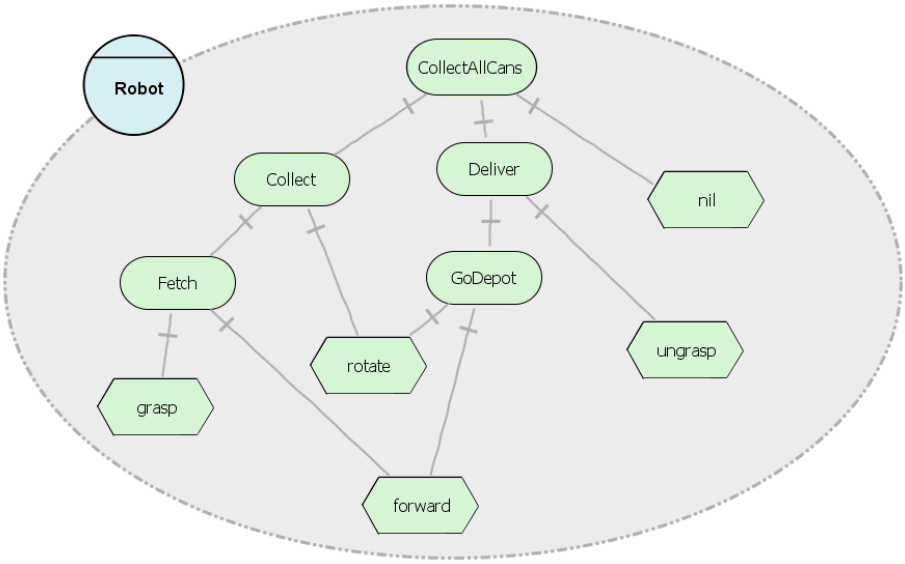

Figura 5. Recolector de latas tras paso 3.

Obsérvese cómo se ha dibujado el OP en la parte alta del agente `Robot` y cómo se ha dividido en dos subobjetivos (`Collect` y `Deliver`) y la tarea `nil`, que representa la consecución del OP y por tanto el robot puede quedar en reposo. `Collect` se ha dividido a su vez en el subobjetivo `Fetch` y la tarea `rotate`: en algún momento durante la rotación el robot quedará mirando una lata; en ese momento podrá avanzar a cogerla, que es lo que hace el subobjetivo `Fetch`. Por su parte, `Deliver` se subdivide en `GoDepot` y la tarea `ungrasp`: cuando el robot llegue al depósito (subobjetivo `GoDepot` alcanzado) el robot puede soltar la lata.

2.4 PASO 4: Priorización

En este paso se introducen las prioridades entre los diferentes subobjetivos y tareas. En los sistemas TR el orden en el que los subobjetivos se van alcanzando es importante. En el ejemplo del colector de latas, el robot necesita coger la lata antes de poder llevarla al depósito. TRiStar usa enlaces de descomposición con prioridad para establecer el orden en que cada tarea o subobjetivo debe ser alcanzado. Cuantas menos marcas tiene un enlace, antes debe ser iniciada la tarea enlazada. La prioridad que se establece de este modo será la que determine el orden de las reglas en el programa TR resultante.

La figura 6 muestra el diagrama del colector de latas incluyendo las prioridades entre las tareas. Obsérvese cómo el enlace entre el OP y `Collect` tiene una sola marca

mientras que el enlace con `Deliver` tiene dos. Para llevar una lata al depósito el robot necesita cogerla previamente.

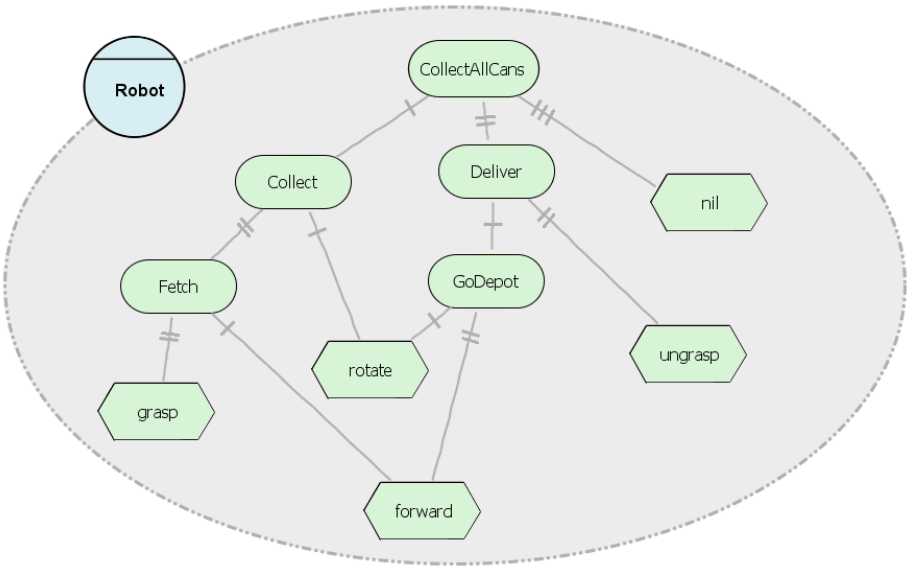

Figura 6. Recolector de latas tras el paso 4.

2.5 PASO 5: Monitorizar el entorno

El STB necesita saber cuándo ha alcanzado el OP o uno de sus subobjetivos. La única forma que tiene de estar al tanto de los cambios en su entorno es a través de las percepciones que le proporcionan los sensores disponibles. Por esa razón necesitamos establecer una correspondencia entre cada subobjetivo y una combinación booleana de las percepciones y creencias disponibles.

En el ejemplo del recolector de latas el OP puede representarse por la siguiente expresión:

```
allCansCollected AND NOT(holding)
```

Además el sistema necesita saber cuándo se ha alcanzado un subobjetivo y cuándo debe dejar de ejecutar una tarea para pasar a ejecutar la siguiente. Este proceso debe realizarse con todos los subobjetivos y tareas. Si nos encontramos con que no somos capaces de establecer una de estas correspondencias probablemente necesitemos nuevos sensores que nos proporcionen otras percepciones o quizá debamos descartar las alternativas que incluyan esos subobjetivos o tareas. La expresión que representa la consecución de un subobjetivo es la misma que la de la consecución de su subobjetivo (o tarea) más prioritario.

En el ejemplo del recolector el subobjetivo `Collect` se puede dar por alcanzado cuando el robot tiene una lata agarrada en la pinza. Podemos por tanto usar la creencia

`holding` para representar la consecución de dicho subobjetivo y el disparador para iniciar la siguiente tarea a realizar: llevarla al depósito.

Los sensores cuyas percepciones se usan en el STB se representan como agentes TRiStar. Las percepciones que proveen como recursos dentro del agente que representa al sensor. Véase como ejemplo la percepción `touching` que proporciona la pinza (`Gripper`) en la Figura 3.

Las expresiones booleanas que combinan dos o más percepciones o creencias se representan como recursos lógicos, como `allCansAtDepot` en la Figura 3. Las expresiones lógicas que corresponden a esos recursos lógicos se pueden encontrar en la tabla que hay en la esquina inferior izquierda de la Figura 3. Nótese cómo cada recurso lógico se enlaza con las percepciones o creencias usadas en la expresión. En nuestro caso, `allCansAtDepot` corresponde tal y como se muestra en la tabla a la expresión `allCansCollected AND NOT(holding)`. Por esa razón el recurso lógico está enlazado a las creencias `allCansCollected` y a `holding`.

Cuando se alcanza un subgoal (*subA*) el sistema debe intentar alcanzar el siguiente subgoal (*subB*). La condición que representa la consecución de *subA* es al mismo tiempo el disparador para empezar a intentar alcanzar *subB*. Esto se representa en TRiStar con un enlace de dependencia entre el recurso que representa la consecución de *subA* y el enlace de descomposición que acaba en *subB*. Por eso el OP siempre se enlaza a un enlace de descomposición que acaba en la tarea `nil`: cuando se alcanza el OP no hay que hacer nada más. Como se explicó anteriormente `holding` significa que el subobjetivo `Collect` se ha alcanzado y que se debe comenzar el subobjetivo `Deliver`. Véase en la Figura 3 cómo el recurso `holding` está enlazado al enlace de descomposición que hay entre el OP y `Deliver`.

Al final de este paso el diagrama TRiStar está completo. El diagrama final correspondiente al robot recolector de latas es el que se muestra en la Figura 3.

2.6 Los cinco pasos resumidos

Los cinco pasos que se necesitan para obtener un diagrama TRiStar a partir de la descripción textual de un sistema TR son:

1. Identificar el Objetivo Principal.
2. Identificar los elementos: sensores (y sus percepciones), actuadores (y sus acciones) y creencias.
3. Descomponer el Objetivo Principal en subobjetivos y tareas y esos subobjetivos en nuevos subobjetivos y tareas hasta que no queden subobjetivos.
4. Establecer la prioridad entre los subobjetivos y las tareas.
5. Monitorizar el entorno usando las percepciones y creencias identificadas para decidir cuándo se ha alcanzado un objetivo y por tanto hay que empezar a ejecutar la siguiente tarea.

3 Conclusiones y trabajos futuros

Hemos presentado cinco pasos metodológicos para obtener un diagrama TRiStar a partir de la descripción textual de un sistema Teleo-Reactivo. Además hemos demostrado que esos pasos pueden aplicarse fácilmente mostrando un ejemplo completo.

En un futuro próximo pretendemos diseñar una familia de experimentos para probar la utilidad de esta metodología. En cada uno de esos experimentos presentaremos una descripción textual de un sistema TR a dos grupos de sujetos. Uno de los grupos intentará obtener el programa TR que implementa el sistema propuesto directamente a partir de la descripción. Al otro se le explicará este método y se le pedirá que obtenga el diagrama TR. Mediremos y compararemos la corrección de los resultados obtenidos y el tiempo usado en conseguirlos.

Aunque las transformaciones necesarias para obtener un programa TR a partir de un diagrama TRiStar están claras, una herramienta que automatizara esta transformación resultaría muy útil. Además, esa herramienta debería permitir dibujar el diagrama TRiStar de manera similar a como lo hace OpenOME, una herramienta open source para dibujar diagramas i* [9].

Agradecimientos

Este artículo es el resultado de la investigación llevada a cabo bajo el Programa de Investigación para Grupos de Excelencia Científica de la Fundación Séneca (Agencia para la Ciencia y la Tecnología de la Región de Murcia, ref. 19895/GERM/15) y ha sido apoyado parcialmente por el proyecto del CDTI OCEAN MASTER (Multipurpose Autonomous System for different Environment and Roles) en el programa FEDER INTERCONECTA 2015.

Los autores desean agradecer a SAES [10] su apoyo en el desarrollo de esta investigación.

Referencias

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 and 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 (2016), pp. 82-100.
5. Bresciani, P., Perini, A., Giorgini, P. et al, “Tropos: An Agent-Oriented Software Development Methodology”, Autonomous Agents and Multi-Agent Systems (2004) 8: 203. https://doi.org/10.1023/B:AGNT.0000018806.20944.ef

6. J. M. Morales, P. Sánchez and D. Alonso, “A systematic literature review of the Teleo-reactive paradigm”, Artificial Intelligence Review, 42 (2014), pp. 945–964.
7. P.Sánchez, D. Alonso, J. M. Morales and P. J. Navarro, “From Teleo-Reactive specifications to architectural components: A model-driven approach”, The Journal of Systems and Software, 85 (2012), pp. 2504-2518.
8. 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.
9. OpenOME official website, Web:https://se.cs.toronto.edu/trac/ome/wiki, last accesed 2018/03/08.
10. SAES official website, Web:http://www.electronica-submarina.com, last accesed 2018/03/08.