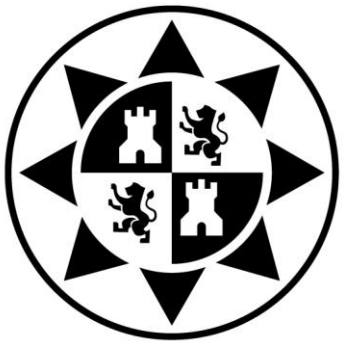
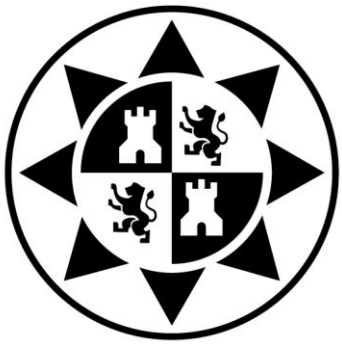


Universidad
Politécnica
de Cartagena

DESARROLLO AUTOMATIZADO DE SISTEMAS TELEO-REACTIVOS A PARTIR DE OBJETIVOS: UN ENFOQUE BASADO EN COMPONENTES Y DIRIGIDO POR MODELOS

José Miguel Morales Illán

2016



Universidad
Politécnica
de Cartagena

DESARROLLO AUTOMATIZADO DE SISTEMAS TELEO-REACTIVOS A PARTIR DE OBJETIVOS: UN ENFOQUE BASADO EN COMPONENTES Y DIRIGIDO POR MODELOS

José Miguel Morales Illán

Directores:

Dr. Pedro Sánchez Palma

Dr. Diego Alonso Cáceres

2016

Queda autorizada la reproducción integral de esta tesis
a efectos de investigación, mediante declaración escrita
del interesado.

Universidad Politécnica de Cartagena

DT-16

CONFORMIDAD DE SOLICITUD DEAUTORIZACIÓN DE DEPÓSITO DE TESIS DOCTORAL POR EL/LA DIRECTOR/A DE LA TESIS

D. Pedro Sánchez Palma y D. Diego Alonso Cáceres Directores de la Tesis doctoral - Desarrollo automatizado de sistemas Teleo-Reactivos a partir de objetivos: un enfoque orientado a componentes y dirigido por modelos-

INFORMAN:

Que la referida Tesis Doctoral, ha sido realizada por D. José Miguel Morales Illán, dentro del programa de doctorado Tecnologías de la Información y Comunicaciones, dando nuestra conformidad para que sea presentada ante la Comisión de Doctorado para ser autorizado su depósito.

La rama de conocimiento en la que esta tesis ha sido desarrollada es:

- ☐ Ciencias
- ☐ Ciencias Sociales y Jurídicas
- ✓ Ingeniería y Arquitectura

En Cartagena, a 9 de febrero de 2016

[redacted] OS DIRECTORES DE LA TESIS [redacted]

Fdo.: Pedro Sánchez Palma Fdo.: Diego Alonso Cáceres

COMISIÓN DE DOCTORADO

Universidad Politécnica de Cartagena

DT-17

CONFORMIDAD DE DEPÓSITO DE TESIS DOCTORAL POR LA COMISIÓN ACADÉMICA DEL PROGRAMA

D. Fernando Quesada Pereira, Presidente/a de la Comisión Académica del Programa "Tecnologías de la Información y Comunicaciones"

INFORMA:

Que la Tesis Doctoral titulada, "DESARROLLO AUTOMATIZADO DE SISTEMAS TELEO-REACTIVOS A PARTIR DE OBJETIVOS: UN ENFOQUE ORIENTADO A COMPONENTES Y DIRIGIDO POR MODELOS", ha sido realizada, dentro del mencionado programa de doctorado, por D. José Miguel Morales Illán, bajo la dirección y supervisión del Dr. D. Pedro Sánchez Palma y del Dr. D. Diego Alonso Cáceres.

En reunión de la Comisión Académica de fecha 12/02/16, visto que en la misma se acreditan los indicios de calidad correspondientes y la autorización del Director de la misma, se acordó dar la conformidad, con la finalidad de que sea autorizado su depósito por la Comisión de Doctorado.

La Rama de conocimiento por la que esta tesis ha sido desarrollada es:

- ☐ Ciencias
- ☐ Ciencias Sociales y Jurídicas
- ✓ Ingeniería y Arquitectura

En Cartagena, a 15 de Febrero de 2016

EL PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA

[redacted]

Fdo:----------

COMISIÓN DE DOCTORADO

UNIVERSIDAD
POLITÉCNICA DE
CARTAGENA

COMITÉ DE DIRECCIÓN DE LA EINDOC

UNIVERSIDAD POLITÉCNICA
- CARTAGENA -
NEGOCIADO DE POSGRADO Y DOCTORADO
19 FEB 2016
SALIDA Nº 088

Sr. D. José Miguel Morales Illán

Visto el informe favorable del Director de Tesis y el V°B° de la Comisión Académica del Programa de Doctorado "Tecnologías de la Información y Comunicaciones" para la presentación de la Tesis Doctoral titulada: **"Desarrollo automatizado de sistemas teleo-reactivos a partir de objetivos: un enfoque basado en componentes y dirigido por modelos"** en la modalidad de "compendio de publicaciones" solicitada por D. José Miguel Morales Illán, el Comité de Dirección de la Escuela Internacional de Doctorado de la Universidad Politécnica de Cartagena, en reunión celebrada el 18 de febrero de 2016,considerando lo dispuesto en el artículo 23 del Reglamento de Estudios Oficiales de Doctorado de la UPCT, aprobado en Consejo de Gobierno el 17 de diciembre de 2015,

ACUERDA

Autorizar la presentación de la Tesis Doctoral a D. José Miguel Morales Illán en la modalidad de "compendio de publicaciones".

Contra el presente acuerdo, que no agota la vía administrativa, podrá formular recurso de alzada ante el Sr. Rector-Magnífico de la Universidad Politécnica de Cartagena, en el plazo de un mes a partir de la notificación de la presente.

Cartagena, 19 de febrero de 2016

EL DIRECTOR DE LA ESCUELA
INTERNACIONAL DE DOCTORADO

[redacted]

Fdo.: Pablo Fernández Escámez

Título: Desarrollo automatizado de sistemas Teleo-Reactivos a partir de objetivos: un enfoque basado en componentes y dirigido por modelos.

Autor: José Miguel Morales Illán.

Directores: Dr. Pedro Sánchez Palma.

Dr. Diego Alonso Cáceres.

© 2016 José Miguel Morales Illán

El trabajo de tesis que a continuación se presenta se acoge a la modalidad de "*tesis por compendio de publicaciones*" del Departamento de Tecnologías de la Información y las Comunicaciones de la Universidad Politécnica de Cartagena, de acuerdo con la normativa vigente (reglamento de estudios oficiales de doctorado de la Universidad Politécnica de Cartagena aprobado por Consejo de Gobierno de 17 de diciembre de 2015). Consta de cuatro artículos en revistas internacionales indexadas en ISI-JCR segundo cuartil. Tres de dichos artículos han sido publicados en la revista "*Journal of Systems and Software*" (editorial Elsevier) y el otro en la revista "*Artificial Intelligence Review*" (editorial Springer-Verlag).

Los artículos incluidos en este compendio son:

[1] Sánchez, P., Alonso, D., Morales, J. M., & Navarro, P. J. (2012). *From Teleo-Reactive specifications to architectural components: a model-driven approach*. Journal of Systems and Software, 85(11), 2504-2518. (JCR-ISI, IF 1.135, Q2). DOI: 10.1016/j.jss.2012.05.067

[2] Morales, J. M., Sánchez, P., & Alonso, D. (2014). *A systematic literature review of the Teleo-Reactive paradigm*. Artificial Intelligence Review, 42(4), 945-964. (JCR-ISI, IF 2.111, Q2). DOI: 10.1007/s10462-012-9350-2

[3] Morales, J. M., Navarro, E., Sánchez, P., & Alonso, D. (2015). *A controlled experiment to evaluate the understandability of KAOS and i* for modeling Teleo-Reactive systems*. Journal of Systems and Software, 100, 1-14. (JCR-ISI, IF 1.352, Q2). DOI: 10.1016/j.riai.2015.06.003

[4] Morales, J. M., Navarro, E., Sánchez, P., & Alonso, D. (2016). *A family of experiments to evaluate the understandability of TRiStar and i* for modeling Teleo-Reactive systems*. Journal of Systems and Software. (JCR-ISI, IF 1.352, Q2). DOI: 10.1016/j.jss.2015.12.056

RESUMEN

Esta tesis doctoral se presenta bajo la modalidad de compendio de publicaciones. Está formada por un total de cuatro artículos publicados en revistas del segundo cuartil del *Journal Citation Reports.*

El artículo "*A systematic literature review of the Teleo-Reactive paradigm*" ofrece una completa revisión sistemática de la literatura existente sobre el paradigma Teleo-Reactivo desde su presentación por el profesor Nils Nilsson en el año 1994. Su papel en esta tesis es el de servir de estado del arte de dicho paradigma, ofreciendo una buena perspectiva de la evolución de los sistemas Teleo-Reactivos desde su formulación hasta el presente.

Para poder desarrollar sistemas Teleo-Reactivos a partir de objetivos, surgió la necesidad de especificar los requisitos de estos sistemas usando el lenguaje más apropiado. Ese es uno de los objetivos principales del artículo "*A controlled experiment to evaluate the understandability of KAOS and i* for modeling Teleo-Reactive systems*". Como resultado de dicho trabajo se decidió utilizar i* dado que el experimento realizado mostró que las especificaciones realizadas con dicho lenguaje resultaban ligeramente más comprensibles que las realizadas con KAOS.

Aunque i* resultaba más comprensible a la hora de especificar requisitos para sistemas Teleo-Reactivos, también presentaba ciertas debilidades. Estas debilidades han sido descritas detalladamente en el artículo "*A family of experiments to evaluate the understandability of TRiStar and i* for modeling Teleo-Reactive systems*", en el que además se propone una extensión al lenguaje que permite superarlas. La extensión propuesta se denomina TRiStar y fue inicialmente presentada en [Morales15]. TRiStar ha demostrado superar los problemas de comprensibilidad identificados en i* en el modelado de sistemas Teleo-Reactivos mediante una familia de experimentos realizada con estudiantes de últimos cursos de grado y con desarrolladores software experimentados, cuyos resultados se exponen exhaustivamente en el artículo mencionado. En él se describe, además, un mecanismo que permite obtener mediante transformación de modelos el programa Teleo-Reactivo equivalente a un diagrama TRiStar dado.

TRiStar permite, por lo tanto, partiendo de los objetivos de un sistema Teleo-Reactivo obtener un diagrama que especifique su comportamiento. Ese diagrama puede ser transformado en un programa Teleo-Reactivo equivalente. Y siguiendo las transformaciones descritas en "*From Teleo-Reactive specifications to architectural components: a model-driven approach*" se puede obtener a partir del programa Teleo-Reactivo el modelo de componentes y la máquina de estados que describe el comportamiento de cada uno de esos componentes. Con estos elementos y usando un framework como el descrito en [Iborra09] se cerraría el proceso de desarrollo del sistema Teleo-Reactivo.

Como resultado de las investigaciones realizadas en el transcurso de esta tesis, y aunque no forma parte del compendio, hay un quinto artículo [Sánchez16] que está en segunda revisión en el *Journal of Systems and Software* en el que se estudian las posibilidades de introducir requisitos de tiempo real cuando se sigue el enfoque Teleo-Reactivo desde el modelado a la implementación de un sistema. Tras realizar un estudio del tipo de restricciones temporales que se pueden imponer desde el punto de vista Teleo-Reactivo, se considera la posibilidad de utilizar TeleoR [Clark14] para incorporar dichas restricciones y se proponene una serie de extensiones a TRiStar para permitir representar requisitos temporales. Estas extensiones dan lugar a lo que hemos llamado TRiStar+.

ABSTRACT

This doctoral dissertation has been presented in the form of *thesis by publication*. It is comprised of four articles indexed in the second quartile of the *Journal Citation Reports*.

The article "*A systematic literature review of the Teleo-Reactive paradigm*" offers a complete systematic review of the existing literature on the Teleo-Reactive paradigm since Prof. Nils Nilsson presented it in 1994. It plays the role of state of the art of that paradigm, showing a perspective of the evolution of Teleo-Reactive systems from their formulation to present time.

In order to develop Teleo-Reactive systems starting from its goals, there is the need of specifying the requirements of these systems using the most adequate language. That is one of the main objectives of the article "*A controlled experiment to evaluate the understandability of KAOS and i* for modeling Teleo-Reactive systems*". As a result, we decided to use i* because the experiment showed that i* specifications where slightly more understandable than those made using KAOS.

Although i* was more understandable when specifying requirements for Teleo-Reactive systems, the experiment also showed some shortcomings. These shortcomings have been deeply described in the article "*A family of experiments to evaluate the understandability of TRiStar and i* for modeling Teleo-Reactive systems*". In this article, an extension to i* is proposed in order to overcome the identified limitations. The proposed extension is named TRiStar and was initially presented at [Morales15]. TRiStar has shown to be more understandable than i* when modeling Teleo-Reactive systems through a family of experiments done with last year students and experienced software developers, whose results are described in the aforementioned article. In that article, a mechanism to obtain a Teleo-Reactive program starting from a TRiStar diagram is also described.

Therefore, TRiStar allows obtaining a diagram which specifies the behavior of a Teleo-Reactive system starting from its goals. That diagram can be transformed into an equivalent Teleo-Reactive program. Then, following the transformations described in "*From Teleo-Reactive specifications to architectural components: a model-driven approach*", a component model and the state machine describing the behavior of each of those components can be obtained. With these elements and using a framework as that described in [Iborra09], the development process of the Teleo-Reactive system would be finished.

As a result of the research carried out during this dissertation there is another article, which is not comprised in the compilation, in second revision at the *Journal of Systems and Software* [Sánchez16]. In that article, after making a study of the type of timing constraints from the TR perspective, we consider the possibility of using TeleoR [Clark14] for incorporating such constraints. Some extensions on TRiStar notation are proposed to represent temporal requirements. Those extensions have been named TRiStar+.

ÍNDICE

AGRADECIMIENTOS

En primer lugar, y como no podía ser de otra manera, quiero agradecer a mis directores, los doctores Pedro Sánchez Palma y Diego Alonso Cáceres, su dedicación y acertados consejos a lo largo del proceso de elaboración de esta tesis. Sin su apoyo en el campo académico nada de esto hubiera sido posible.

Sin embargo, no es sólo en el campo académico en el que he necesitado ayuda en el tortuoso periplo que en mayor o menor medida es siempre la redacción de una tesis. En el campo afectivo y, más aún, en el logístico siempre ha estado ahí Irene para ocuparse de nuestros hijos cuando *Papá tenía que trabajar en la tesis...* Espero que Miguel e Inés puedan perdonarme algún día todas las horas de juego que les he robado para sacrificarlas en el altar de la Ciencia. Mi padre y en especial mi madre, me han animado continuamente a seguir adelante con este trabajo y me consta que son, probablemente, las dos personas que más se enorgullecen de que al fin haya llegado este momento. No quiero terminar este capítulo sin agradecerle a María Dolores las muchas manos que nos echa a lo largo de la semana a Irene y a mí para poder dedicarnos a estos menesteres.

Volviendo al campo académico, la Dra. Elena Navarro se ha comportado en muchas ocasiones como una tercera directora de mi tesis. Quiero agradecerle de todo corazón el gran esfuerzo realizado como coautora de dos de los artículos de este compendio y mostrarle mi más profundo reconocimiento al trabajo realizado. Vaya desde aquí mi agradecimiento también al Dr. Pedro J. Navarro por sus aportaciones al primer artículo científico en el que participé. Para terminar con los colaboradores en la redacción de los artículos he dejado para el final a mi buena amiga Esther Corbalán, sin cuyo apoyo en el estudio estadístico del último artículo yo hubiera sido incapaz de sacarlo adelante.

Mención aparte creo que merece el personal y la directiva de la empresa SAES, en la que desarrollo la mayor parte de mi vida laboral. Desde el primer momento mostraron una actitud de plena colaboración empezando por la participación en experimentos hasta la flexibilidad mostrada para permitirme asistir a congresos y seminarios necesarios para completar mi formación en el programa de doctorado. Antonio Arnao, Carlos Alonso, Consuelo Lázaro, Antonio Sánchez, David Rebollo, Juan Pedro Cánovas o Antonio Padilla son sólo algunos de los que han tenido que soportar mis rollos sobre sistemas Teleo-Reactivos a lo largo de todos estos años.

Por último, quiero agradecer a los alumnos de la UPCT, tanto a los que participaron en los experimentos como a los que acuden a mis clases, el haberme inoculado el veneno de la docencia. Todo esto empezó por vuestra culpa.

No quiero terminar sin nombrar a mis amigos, Pablo, Emilio, Ángel, Tomás, Paco, Cristóbal y JuanSe que, sabiéndolo ellos o no, han estado ahí cuando yo más lo necesitaba.

Gracias a todos.

"Tú, que me lees, ¿estás seguro de entender mi lenguaje?"

Jorge Luis Borges, *La Biblioteca de Babel*

A Irene, a Miguel y a Inés

OBJETIVOS

Los objetivos principales de esta tesis doctoral son los siguientes:

- Escoger un lenguaje de especificación de requisitos para sistemas Teleo-Reactivos entre los lenguajes GORE (Goal Oriented Requirements Engineering) más extendidos.
- Una vez escogido ese lenguaje, proponer extensiones si es necesario para mejorar la eficacia y la eficiencia del lenguaje a la hora de especificar sistemas Teleo-Reactivos.
- Las especificaciones en ese lenguaje extendido deben permitir obtener el programa Teleo-Reactivo correspondiente al sistema especificado mediante transformaciones de modelos.
- Partiendo de un programa Teleo-Reactivo, obtener una arquitectura de componentes que permita su implementación así como una descripción del comportamiento de dichos componentes a través de máquinas de estados.
- Favorecer el "*diseño para la reutilización*": es posible guiar el proceso de catalogación de componentes a partir de la especificación dada en los programas Teleo-Reactivos.
- Favorecer el "*diseño desde la reutilización*": para el desarrollador es más directo identificar del catálogo qué componentes se adaptan al comportamiento del *software-to-be*.
- Realizar un estudio de las posibilidades que ofrece el paradigma Teleo-Reactivo para la consideración de restricciones temporales en los requisitos.
- Analizar las posibilidades que ofrece TeleoR para incorporar los requisitos temporales identificados y demostrar la viabilidad metodológica mediante un ejemplo.

ESTADO DEL ARTE

Uno los artículos que conforman este compendio ("*A systematic literature review of the Teleo-Reactive paradigm*") ofrece una completa revisión sistemática de la literatura existente sobre el paradigma Teleo-Reactivo desde su presentación por el profesor Nils Nilsson en el año 1994.

La ingeniería de requisitos orientada a objetivos (GORE, por sus siglas en inglés) ha demostrado ser muy útil en el proceso de la ingeniería de requisitos [Lamsweerde01]. Las distintas propuestas GORE (ver [Kavakli05] para una introducción exhaustiva) se centran en el 'por qué' del *system-to-be* especificando la motivación y el razonamiento que justifica la especificación de requisitos. Un modelo orientado a objetivos puede ser especificado de muchas maneras pero todas ellas utilizan grafos dirigidos y refinamiento iterativo de objetivos.

Las referencias básicas para entender el desarrollo de software dirigido por modelos siguen siendo [Atkinson03] y [Selic03] mientras que para el desarrollo de software basado en componentes lo son [Szypersky02] y [Lau07]. La utilidad de desarrollar sistemas reactivos, particularmente en robótica, basados en componentes reutilizables está muy generalizada en la comunidad científica. [Brugali09] y [Brugali10] proporcionan una introducción a la ingeniería del software para robótica basada en componentes. Explican en detalle los principios de diseño y los aspectos básicos del desarrollo de sistemas robóticos mediante componentes software reutilizables y mantenibles. A este respecto, el grupo de investigación DSIE tiene más de una década de experiencia [Iborra09] en el desarrollo de sistemas reactivos con inputs de la ingeniería del software (por ejemplo, desarrollo de línea de producto software, patrones de diseño, frameworks de arquitectura, desarrollo de software dirigido por modelos o desarrollo de software basado en componentes). En particular, DSIE ha desarrollado un enfoque integrado para el desarrollo de sistemas reactivos basado en el uso de frameworks, patrones de diseño y generación de código a través de transformaciones de modelos.

PUBLICACIONES

FROM TELEO-REACTIVE SPECIFICATIONS TO ARCHITECTURAL COMPONENTS: A MODEL-DRIVEN APPROACH

RESUMEN

El enfoque Teleo-Reactivo diseñado por el profesor Nils Nilsson ofrece un modelo de programación de alto nivel que permite el desarrollo de sistemas reactivos como por ejemplo, vehículos robóticos. Los programas Teleo-Reactivos se escriben de manera que facilitan a los ingenieros definir el comportamiento del sistema teniendo en cuenta sus objetivos y los cambios que puedan producirse en su entorno. Este artículo presenta un enfoque sistemático que hace posible derivar modelos arquitectónicos con descripciones estructurales y de comportamiento partiendo de programas Teleo-Reactivos. El desarrollo de sistemas reactivos puede por tanto beneficiarse significativamente de la combinación de dos enfoques: (1) el enfoque Teleo-Reactivo, orientado a la descripción del sistema desde el punto de vista de sus objetivos y del estado de su entorno y (2) el enfoque arquitectónico, que se orienta al desarrollo de software basado en componentes, en el que las decisiones están condicionadas por la necesidad de reutilizar soluciones ya probadas con anterioridad. La integración de este trabajo en un entorno de desarrollo que proporciona generación de código mediante transformaciones de modelos abre nuevas posibilidades en el desarrollo de este tipo de sistemas. La propuesta se valida mediante un caso de estudio representativo del dominio y una encuesta realizada a estudiantes de postgrado.

The Journal of Systems and Software 85 (2012) 2504–2518

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

From Teleo-Reactive specifications to architectural components: A model-driven approach

Pedro Sánchez*, Diego Alonso, José Miguel Morales, Pedro Javier Navarro

Systems and Electronic Engineering Division (DSIE), Universidad Politécnica de Cartagena, Campus Muralla del Mar s/n, Cartagena, Spain[1]

ARTICLE INFO

Article history:
Received 8 December 2011
Received in revised form 9 April 2012
Accepted 22 May 2012
Available online 9 June 2012

Keywords:
Teleo-Reactive programs
Component-based software development
Reactive systems
Robotics
Model-driven software development

ABSTRACT

The Teleo-Reactive approach designed by N.J. Nilsson offers a high-level programming model that permits the development of reactive systems, such as robotic vehicles. Teleo-Reactive programs are written in a manner that allows engineers to define the behaviour of the system while taking into account goals and changes in the state of the environment. This article presents a systematic approach that makes it possible to derive architectural models, with structural descriptions and behaviour, from Teleo-Reactive Programs. The development of reactive systems can therefore benefit significantly from a combination of two approaches: (1) the Teleo-Reactive approach, which is oriented towards a description of the system from the standpoint of the goals identified and the state of the environment and (2) the architectural approach, which is oriented towards the design of component-based software, in which decisions are conditioned by the need to reuse already tested solutions. The integration of this work into a development environment that allows code to be generated via model transformations opens up new possibilities in the development of this type of systems. The proposal is validated through a case study that is representative of the domain, and a survey carried out with post-graduate students.

© 2012 Elsevier Inc. All rights reserved.

1. Introduction

Reactive systems possess specific characteristics that make them particularly sensitive to the architectural decisions that are made during the course of their construction. Aspects such as task planning, concurrency management, fault tolerance, and distributed communication need to be addressed as far as possible through the solutions that the literature has shown to be most suited to systems of this kind. Moreover, reactive systems are event-driven since they have to react to external/internal stimuli. These computer-based systems are usually employed to control systems in which failures can be costly and endanger lives. One representative instance of reactive systems is the mobile robotics domain. This domain embraces applications in which robotic vehicles act and move autonomously in semi- and non-structured environments. Their field of application is immense, and very many systems have in fact been designed, both for outdoor use (agriculture, forestry engineering, mining, construction, rescue, etc.) and for indoor use (vacuum cleaners, care assistance, museum guides, etc.). The growing sophistication of these applications has led to the need for more sophisticated languages and platforms with which to express, validate and implement them. Research into mobile robotics has traditionally focused on the development of high-level algorithms for navigation, location, mapping, etc. More recently, in the last decade, there have been highly significant advances in this domain through the use of the *component* concept as the basic architectural design element (Brugali and Scandurra, 2009, 2010).

The Systems and Electronic Engineering Division (DSIE) research group has spent recent years (Iborra et al., 2009; Sánchez et al., 2011) searching for solutions to the development of reactive systems, and particularly robotic systems, using the model-driven software development (MDSD) approach (Atkinson and Kühne, 2003; Bézivin, 2005). The use of components, following the component based software development (CBSD) paradigm (Szyperski, 2002; Crnkovic et al., 2002), is a well known approach to composing systems out of reusable building blocks, with a significant reduction in the development cost and time-to-market since developers can assemble such systems from a set of reusable components rather than building them from scratch. CBSD is tightly coupled with the study and practice of software architecture (Taylor et al.,

* Corresponding author.
E-mail addresses: pedro.sanchez@upct.es (P. Sánchez), diego.alonso@upct.es (D. Alonso), josemiguel.morales@upct.es (J.M. Morales), pedroj.navarro@upct.es (P.J. Navarro).
[1] Tel.: +34 968326460; fax: +34 968325973.

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.05.067

2009), since the way in which components are developed and connected influences the properties of the final system. Thus, architectural issues should be taken into account when designing the components that will finally make up the application, as noted in Medvidovic et al. (2007).

The reuse of existing solutions in current robotic systems is mostly focused on the level of algorithms and libraries, and does not usually take place at a high abstraction level. In Pastor et al. (2010) we recently introduced an MDSD-based approach that separates the component-based description of real-time applications from their possible implementations on different platforms. This separation is supported by automatic integration of the code obtained from the input models into object-oriented frameworks. In this respect, the management of models from the early stages of software construction and the support for an automatic transformation process into code in the MDSD framework have proven useful in the development of reactive systems, as shown in Section 7.

Also, some authors argue that software development should be directed towards what the user wishes to obtain from it, i.e., its *goals*, and that these goals should drive the requirements engineering process (Lamsweerde, 2009). Since reactive systems may benefit from this approach, we opted for the *Teleo-Reactive (TR) paradigm* (Nilsson, 1994), a goal-oriented approach for the modelling of systems that manages their actions, outputs and status in response to stimuli from within or outside them. A very recent work (Castro et al., 2012) follows a similar approach in order to derive structural architecture from system goals, using the i* (iStar) language for modelling system requirements. However, we think that the TR paradigm is best suited for modelling the goals that a reactive system should achieve, and in our proposal we generate both the architectural structure and the behaviour of the components.

The TR paradigm has provided very interesting results in the robotics domain, as will be shown later. This approach, as applied to reactive systems, is both a viable and an exciting prospect, since TR programs inherently recover from errors and unexpected events while proceeding towards their goals. TR programs are written in a production-rule-like language. Thus, the TR programming model is a high-level approach to implementing systems that react dynamically to changes in their environment. The models that are generated are therefore very suitable for describing robotic systems from a goal-oriented perspective. It is important to distinguish between a reactive and a proactive perspective in robotic system development. A reactive robot responds to changes in the environment in accordance with a set of rules. In contrast, a proactive robot will work to achieve a specialized goal. The reactive robot has sensors that are used to perceive the current state of both itself and its environment. An action performed by this robot will usually promote a new state in the environment. The perception that the robot has of the environment obviously needs to be appropriate to fulfil the predefined set of goals. That is the perspective that will be considered in this article.

Architectural design is not a trivial task because it depends on the expertise of the developers and on how they understand the requirements. We propose a model transformation approach to derive an architectural design (structure and behaviour) from TR programs. The idea presented in this article is a significant step forward that combines both approaches for the development of reactive systems: the TR paradigm to specify the overall application behaviour, and CBSD to specify the application architecture. The main contributions of this article are:

- The establishment of a bridge between the specifications of TR programs with the development of component-based software. The modelling of applications is therefore based purely on goals, and the developer will not need to know either the details of the implementation platform, or the modelling languages for specifying components, both of which usually require a certain amount of experience.
- It provides a translational semantics of TR programs based on components and state machines.
- It favours '*design for reuse*': guided cataloguing of components is possible using the specification provided in the TR programs.
- It favours '*design from reuse*': it is more direct for the developer to use a catalogue to identify which components can be adapted to the behaviour of the *software-to-be*. This is because the components are specific and can be catalogued at a higher abstraction level, which is closer to the problem domain.

The results are demonstrated by focusing on a case study of a mobile robot. However, all the solutions that have been developed can be extrapolated to other reactive systems which can also be modelled using the TR approach. Since the proposed approach has TR programs as input artefacts, it can be easily extended and applied to domains in which the TR paradigm could also be applied. All of these objectives are very ambitious, but they have been shown to be feasible and have been achieved incrementally thanks to the formal and theoretical foundations of MDSD, and the technological support provided by the Eclipse development environment.

The rest of this article is organized as follows: Section 2 introduces the TR approach, along with various definitions and a robotic example. Section 3 presents the proposed systematic approach for transforming TR programs into component-based models, while Section 4 provides an overview of the global approach into which the proposal is integrated. Section 5 presents some details of the implementation carried out using the Eclipse technology. Section 6 presents a survey-based evaluation of the approach. Section 7 describes the state of the art. And last but not least, Section 8 presents conclusions and future research.

2. Teleo-Reactive programs

As Nilsson (1994) states, a Teleo-Reactive program (TR program) is a mid-level agent control program that robustly directs an agent towards a goal in a manner that continuously takes into account the system-changing perceptions of a dynamic environment. In other words, TR programs are a set of reactive rules that continuously sense the environment and trigger actions whose continuous execution eventually leads the system to satisfy a goal. The main advantage of TR programs is their ability to react robustly to changes in their environment owing to the continuous computation of sensing values. TR programs provide engineers with an intuitive approach within which to write goal-directed programs. A TR program is defined as a set of prioritized condition/action rules. A TR program interpreter should constantly re-evaluate the triggering condition set for each rule, and should execute the action corresponding to the highest-priority rule with a satisfied precondition.

A TR program is usually denoted by Nilsson (1994) as:

$$K_1 \rightarrow a_1$$
$$K_2 \rightarrow a_2$$
$$\dots$$
$$K_m \rightarrow a_m$$

where K_i are the conditions in sensory inputs and in a model of the world, and a_i are actions in the world or which change the model of the world. The list of rules is evaluated from the top for the first rule whose condition is *true*, and its corresponding action is then executed. An action a_i may consist of a single action, or may itself be a TR program.

Nilsson states that if a TR program is *complete* (i.e., the condition K_1 or K_2 or ... or K_m is a tautology) and respects the *regression* property (i.e., each condition K_i is a regression of some higher condition through an action a_i) then the system implementing the TR program will always achieve its goal. TR programs differ substantively from conventional production systems because actions can be durative rather than discrete. A durative action is one that can continue indefinitely, and its execution will therefore continue as long as the condition is true (for example, move the robot forward is a durative action, as opposed to a discrete action like move the robot forward 10 m). As an example of a TR program, a robot that might be rotating until it detects the target (sensoring input) and starts to execute the (durative) action *move_forward* could be written as:

target_detected → move_forward

true → rotate

The effects of an action on its environment occur as a consequence of the interaction of the system. They are not modelled in the TR program but rather present in the changes detected in the sensor inputs. Some authors, such as Gubisch et al. (2008), extend the TR paradigm by additionally considering the simultaneous invocation of activities by following the syntax:

$K_i \rightarrow a_j, a_k$

In this case, when K_i holds, both activities are activated, and similarly, when K_i does not hold, the activities are simultaneously deactivated. The parallel execution of actions a_i and a_j must continue to satisfy the regression property.

Definition. A *basic TR program* is a *goal* defined by the structure (S, A, f), where $S \cup \{\text{True}\}$ is a set of sensor inputs, $A \cup \{\text{nil}\}$ is a set of durative actions, and $f : S \times \mathcal{P}(A) \rightarrow p$ is a function that defines a set of rules having the form $s \rightarrow \mathcal{P}(A)$ with priority p, where s is a well formed formula, over the set S using relational (<, <=, >, >=, <>) and Boolean operators (**and**, **or**, **not**), and $\mathcal{P}(A)$ is a powerset over A that represents the set of all possible simultaneous executions of actions over the set A.

Definition. Given two rules, r_1 and r_2, of a TR program, r_1 *is said to have a higher priority than* r_2 *if and only if* $(r_1) > f(r_2)$.

Various authors have considered the possibility of including hierarchies for the definition of TR programs, which favours modularity, greater ease of reuse, refinement, tests, etc. We also think it would be useful for developers to be able to structure TR programs using hierarchies, and we have therefore extended the definition shown above.

Definition. In a *hierarchical TR program*, actions can also be basic TR programs. Each TR program that is referenced in a rule is called a *subgoal*. The root of the hierarchy of a TR program is called the *goal* of the system-to-be.

In a general TR program, conditions, actions and goals may have free variables that are bound when the TR program is called. It is important to highlight that unlike in conventional programming where a called subroutine assumes control of the execution until completion, a TR program continues to constantly evaluate its set of conditions even when a subgoal has been called. Consequently, if another rule condition that has higher priority becomes *true*, the subgoal is immediately terminated.

As a case study we consider a TR program for a robot, as depicted in Fig. 1 (taken and adapted from Dongol et al., 2010), which has been developed to clear cans from the floor by moving them to a depot. The robot is able to rotate, to scan the environment for cans or the depot, to move forward, and to close or open its gripper. The robot includes sensors that evaluate whether or not the robot is holding, seeing, and touching a can. For example, if the robot sees a can, then it moves forward while it can see the can. Provided that the environment does not move the can, the robot will eventually touch it. Using its current location, the robot is able to see the depot and to know whether it is at the depot or not.

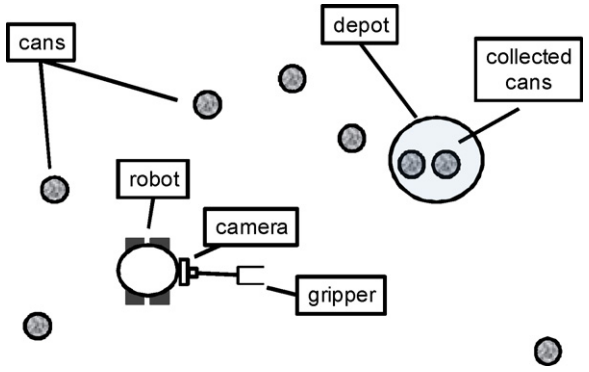


Fig. 1. The can collector robot and its environment. Taken and adapted from Dongol et al. (2010).

The TR program derived from the above specification is the following:

S = {holding, at_depot, see_depot, see_can}
A = {forward, rotate, grasp, ungrasp}
f:
holding and at_depot → ungrasp (highest priority)
holding and see_depot → forward
holding → rotate
see_can and touching → grasp
see_can → forward
True → rotate (lowest priority)

The guards *holding*, *see_can*, *at_depot*, *see_depot*, and *touching* are Boolean variables whose values are equivalent to the values of the corresponding sensors. The primitive actions of the robot are 'rotate', 'forward', 'grasp', and 'ungrasp', and these control the robot's basic movements.

The equivalence between the hierarchical and the plain specification of a TR program is straightforward. Following is hierarchical version of the TR program for the robot:

S = {holding, at_depot, see_depot, see_can}
A = {forward, rotate, grasp, ungrasp}
f:
Robot:
holding → **Deliver**
True → **Collect**
Collect:
see_can → **Fetch**
True → rotate
Fetch:
touching → grasp
True → forward
Deliver:
at_depot → ungrasp
True → **Go_depot**
Go_depot:
see_depot → forward
True → rotate

We can see that the TR program includes one root goal ('Robot') and four subgoals. The fulfilment of each subgoal leaves the robotic system in a particular state. The actions (subgoals) Deliver and Collect cause the robot to respectively deliver and collect cans. Furthermore, Deliver is expanded into the primitive action 'ungrasp' and the subgoal Go_depot. Since the subgoal Fetch is the last action of the subgoal Collect, both subgoals are fulfilled at the same time, and the system will consequently hold the same state (*holding*). Here, completion of the first action Deliver represents the accomplishment of the goal, while completion of the second action Collect represents the achievement of the subgoals that cause the robot to make progress towards enabling the action Deliver.

Completion of the 'ungrasp' activity represents completion of Deliver (because *holding* becomes false), while completion of Go_depot enables 'ungrasp'.

A better understanding of the TR rule activation mechanism can be gained from Fig. 2, which shows a timing diagram corresponding to an example of typical robot execution in the version of the TR program without subgoals. Actions 'grasp' and 'ungrasp' were substituted by 'on_electromagnet' and 'off_electromagnet', since we had a robot without a grasping mechanism, as described in Section 6. The timing diagram was obtained with the Xilinks simulation software after generating the electrical circuit for the input TR program, given that Nilsson (1994) described the equivalence between both representations. The changes in the conditions and the actions carried out by the robot are shown at the bottom.

At the outset, the only line that is activated is the one corresponding to 'rotate', as its guard condition is always true. Whenever the line *see_can* is activated, 'forward' is also activated since its guard condition is *see_can* and the corresponding rule has higher priority than "forward". The next line to be activated is 'touching'. This causes compliance with another, higher-priority rule and hence activation of the action 'on_electromagnet', since both *touching* and *see_can* are activated. Soon after the electromagnet is switched on, the condition *holding* becomes true, and so a new, higher-priority rule activates the 'rotate' line again until the condition *see_depot* is verified. This triggers another change to a higher-priority rule whose condition is true, and hence the action 'forward' is executed and the action 'rotate' terminated. Finally, the line *at_depot* is activated, which verifies the highest-priority rule of all, causing deactivation of 'forward' and activation of 'off_electromagnet'.

3. From TR programs to architectural components

As noted above, TR programs are an effective approach by which to continuously perform a set of activities in order to achieve particular goals, and to react to changes in the environment. However, despite the fact that TR programs are an effective framework for robotics, they have not been fully exploited. One of the greatest deficiencies lies in the absence of a method that would allow a TR specification to be translated into architectural models from which an application could be generated. As is shown in Section 7, various authors have tackled the definition, formalization and simulation of TR programs. Although highly significant in terms of their contributions to the area of artificial intelligence, all these works are limited in that for the implementation of robotic systems they either reject the reuse of previously developed software artefacts (Estublier and Vega, 2005), ignore the importance of the design of their software control architecture (Shaw and Clements, 2006), or avoid the restrictions imposed by the execution infrastructures, which are typical of real-time systems (Laplante, 2004).

The utility of developing reactive systems, particularly in robotics, based on reusable components is generally accepted. In Brugali and Scandurra (2009, 2010), an introduction to component-based robotics software engineering is provided. They explain in detail the main design principles and implementation issues in the development of robotic systems through reusable and maintainable software building blocks. In this respect, the DSIE research group has more than a decade of experience (Iborra et al., 2009) in the development of reactive systems with inputs from software engineering (for example, software product line development, design patterns, architectural frameworks, MDSD and CBSD). In particular, the DSIE research group has developed an integrated approach for the development of reactive systems based on the use of frameworks, design patterns and code generation through model transformations. This is part of a global development approach which is dealt with in detail in Section 4.

The present article presents a solution combining the best aspects of both approaches: on the one hand, the benefits of modelling reactive systems using the TR approach, and on the other, the development of systems in the context of MDSD and CBSD. Our approach starts by specifying the TR program which describes the goals that the system must attain. We first describe the component types we have considered for translating the TR program, and then we outline the patterns that can be used to derive the components of the applications. And finally we propose an implementation of the TR semantics using state machines.

3.1. Component types considered

The following two types of components are considered when translating a TR specification to components:

(a) **Boundary components** serve as a bridge between the robot and its environment, both to collect information from the environment via its sensors, and to act on it in response to the execution of actions.

Taking the example of the can collector robot, components of this type would include Motor, the Camera used to detect objects, etc. The behaviour of these components is as stated in their specifications, as these have either been developed by third parties or been reused or developed specifically for the application. The invocation of the services provided by boundary components will normally be associated with the execution of domain-specific algorithms (e.g., to control a motor and to detect obstacles through computer vision), or monitoring of the changes that occur in the environment.

(b) **TR components**, which implement the behaviour of a TR program (sub)goal. A TR component has a generic interface, as shown in Fig. 3. Unified Modelling Language (UML) notation is used throughout the article where components are modelled as simple rectangles, the lollipop symbol is used to indicate an implemented interface, ports are depicted as small squares on the sides of components, and a required interface is modelled using the socket notation. The I_TR interface allows the TR component to be activated and deactivated by another TR component, which is generated from a (sub)goal of higher priority. That is to say, after a TR component is activated, all the rules defined in the (sub)goal are evaluated in order to execute the highest-priority active (its condition is true) rule. The execution of one of these rules will lead to one of the following three actions: (1) nothing will occur (*nil* action), (2) the activation of another TR component, or (3) the invocation of a boundary component service. The updating of the state of the environment, which determines the execution of the TR component, is carried out through the services (provided or required) of the boundary components with which it is connected. The state of a TR component is therefore given by the reference to the rule that is currently active.

3.2. Deriving components from TR programs

Fig. 4 depicts the general case for translating TR programs to component diagrams. As noted earlier, a TR program is a set of rules in which actions or subgoals are executed. A boundary component will be created for each primitive action, while a new TR component will be created for each invoked subgoal. This procedure will be recursively applied to each of the subgoals of the TR program.

As an example, Fig. 5 shows the component-based model that resulted from translating the TR program from the example of the can-collector robot following the procedure described above. As

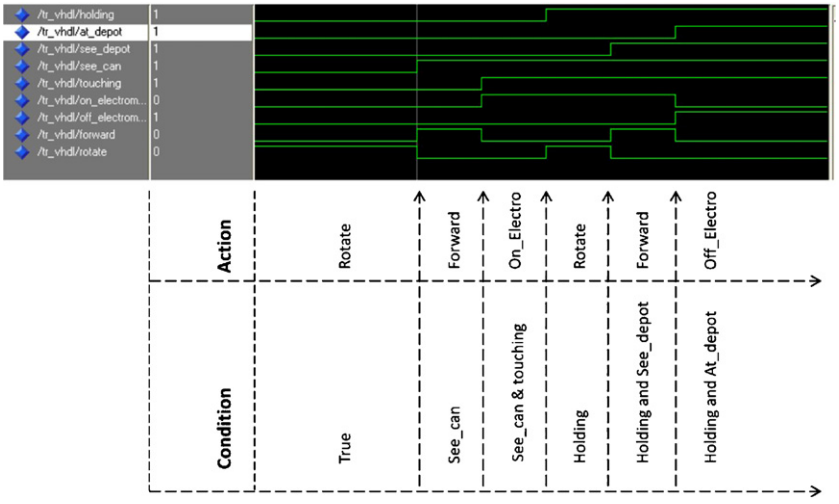


Fig. 2. Example of activation/deactivation of TR rules and of the invocation of primitive actions.

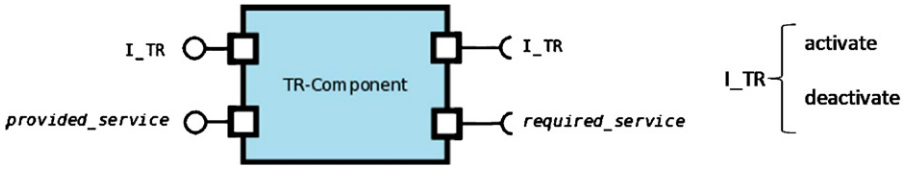


Fig. 3. Basic schema of a TR component.

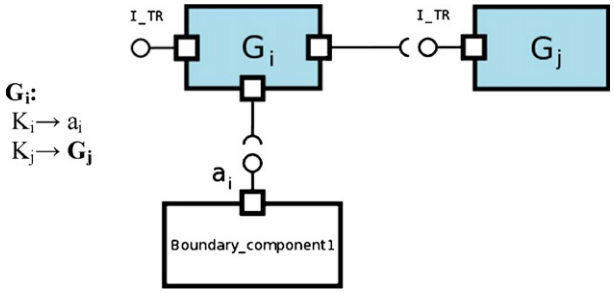


Fig. 4. Correspondence between (sub)goals, actions and components (general case).

can be seen, a TR component (shaded component) has been created for each subgoal. The main goal in the hierarchy is Robot, and the activation or deactivation of this TR component therefore represents the initiation or interruption of the system execution. The process of constructing the component model therefore follows this basic rule: each TR component is connected, via the I_TR interface, with each of the TR components corresponding to the subgoals that are invoked in its rules.

As explained above, the activation of a TR component entails initiating the execution of the associated TR rules. In the example, the activation of Collect thus implies executing the rules shown in the previous section, which will eventually lead to the activation of

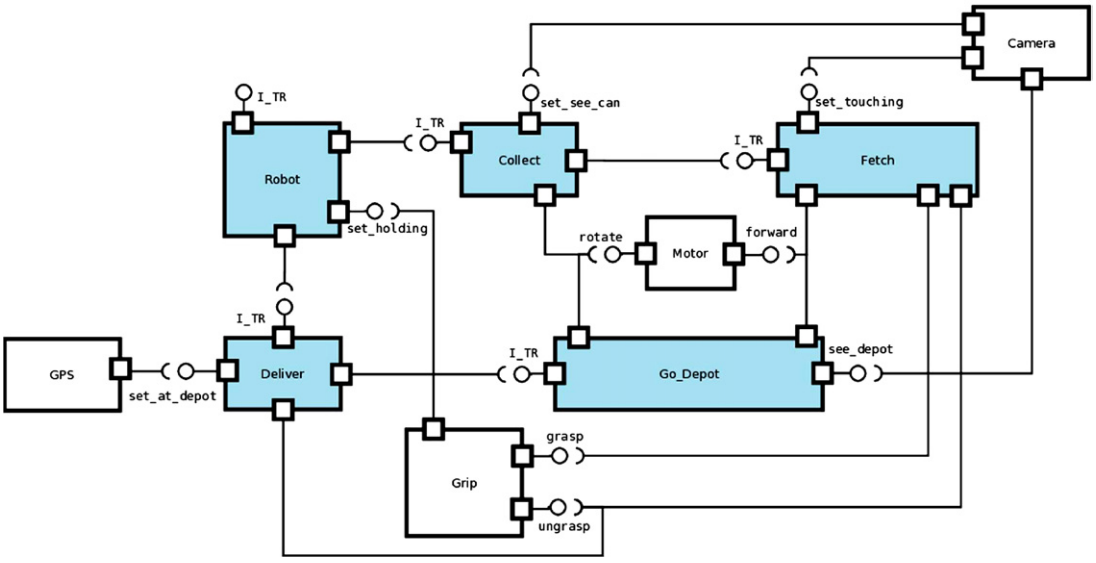
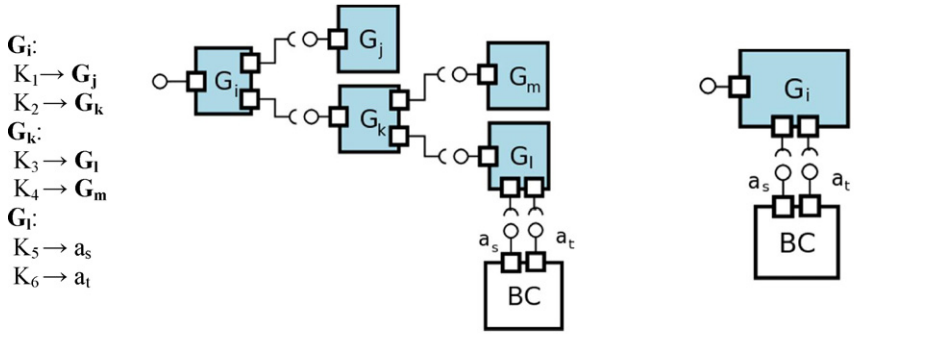


Fig. 5. Component model for the robot example.



(a) Version of the TR program with all the subgoals (b) Version of the TR program collapsed into one goal

Fig. 6. From subgoals and actions to components. Two limit cases.

Fetch. Following the semantics of a TR program, when a higher priority (sub)goal contains a rule which becomes true, this invokes the deactivation of the activated TR component of lesser priority which, if it proceeds, will send the deactivation command to the other TR components of lesser priority in the chain. For example, when Fetch is active, if the condition of the other rule of the Robot goal is made true (*holding* = true because, for example, a can is manually placed in the robot's gripper), then 'deactivate' is invoked in the TR component Collect, which in turn deactivates the TR component Fetch.

In the process of generating the architectural model, it is important to bear in mind two factors that affect the resulting design: (1) the number of TR components that are to be included in the design and (2) what boundary components will be providing the services corresponding to the actions included in the TR rules.

As regards the first factor, it is important to note that the number of TR components will be equal to the number of subgoals in the TR program. The developer can change the number of subgoals (thus flattening the structure to a greater or lesser extent by replacing subgoals with the sets of rules into which they break down), in obedience to different criteria, thus influencing the resulting architectural model. What solution is adopted will be determined by the degree of granularity desired in the architectural model. Fig. 6 shows two possible situations, one of which considers the initial version of the TR program (Fig. 6a), while the other considers a totally flattened model in which the subgoals are eliminated leaving a single TR component containing all the rules (Fig. 6b).

As regards the second factor, the idea is that the number of boundary components may vary depending on the available catalogue of components providing the required functionality and the way in which they provide it. For example, there may be a single component that encapsulates the entire Motor and Grip functionality, or else a component for each one of them as in the example in Fig. 5. Similarly, changes in the state of the environment may be detected either by periodic sampling of the sensorizing devices or else by reception of a break indicating a change in that state. Obviously, one situation requires a different solution from the other. For instance, *holding* becomes true when the Grip component makes the Robot variable set_holding true.

Upon applying these rules, it is possible to systemize the architectural model's generation process, as detailed in Section 5. The proposal presented here has a number of particularly attractive advantages:

(1) It promotes the reuse of TR components. Developers can have a catalogue of potentially reusable goals, together with the associated TR components, at their disposal. Since these goals are defined in the problem domain, their reuse is highly favoured for various robotic applications.

(2) Developers have a good knowledge of the functionality of the components, since they are defined at the same abstraction level as the TR program. This favours the rapid development of applications and the possibility of involving potential system users in the design, even if they are not experts in implementation aspects.

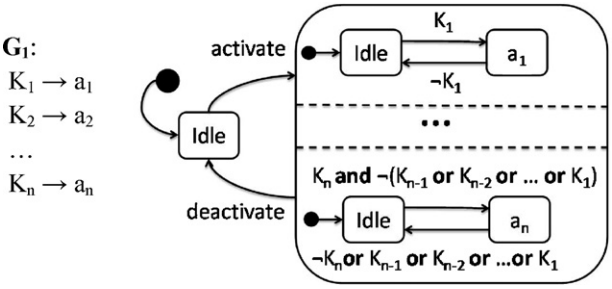


Fig. 7. Pattern to transform a TR program without subgoals.

3.3. Deriving state machines from TR programs

The next question that must be answered is how to translate the TR behaviour to other representations that facilitate the generation of code. The solution adopted in our case was to generate an implementation of the behaviour of TR rules using UML state machines, and then associate a state machine with each TR component. In this way we get a complete development process (described in Section 4), taking advantage of the infrastructure developed by the DSIE research group so as to generate executable code from architectural and behavioural models (i.e., components, their relationships and their behaviour expressed through state machines). More details on this infrastructure and the possibilities it offers from a methodological point of view can be found in Pastor et al. (2010). Section 4 gives a brief description of the global process and the tools involved.

The process of transforming TR programs into components plus state machines thus consists of two steps: (1) definition of the components (boundary and TR) that are to be generated and their connections (as detailed above) and (2) the detailed design of each TR component using state machines. The behaviour of the state machines must correspond to the TR goal from which the TR component has been generated.

The state machines considered include two states (see Fig. 7): an 'idle' state representing the TR component when at rest, and a super-state containing as many orthogonal regions as there are rules in the (sub)goal that represents the TR component. Both states

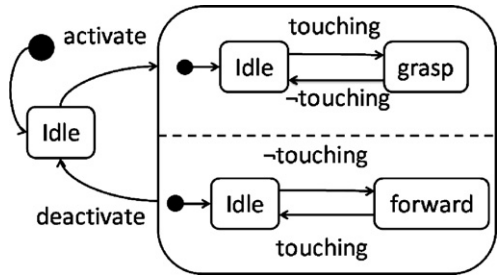


Fig. 8. State machine generated by applying the pattern to the subgoal Fetch from the example.

are connected to two transitions: one corresponding to the activation of the TR component (from 'idle' to the super-state), and another corresponding to its deactivation (from the super-state to 'idle'). Each of the orthogonal regions of the super-state also has two states:

- An 'idle' state in which no action is executed.
- A state in which a_i will be executed in the *do* part and which will be labelled with the name of the action, for simplicity's sake.

There are two transitions between these two states:

- One from 'idle' state to 'a_i' state whose guard condition is that of fulfilling K_i and of not fulfilling any of the conditions of the higher priority rules in the (sub)goal:

$$K_i \ \textbf{and} \ \neg(K_{i-1} \ \textbf{or} \ K_{i-2} \ \textbf{or} \ \ldots \ \textbf{or} \ K_1)$$

- Another transition from a_i state to 'idle' state whose guard condition is the negation of the previous guard, i.e., that K_i is not fulfilled, or that any of the conditions corresponding to a higher priority rule in the (sub)goal is fulfilled, which is to say:

$$\neg K_i \ \textbf{or} \ K_{i-1} \ \textbf{or} \ K_{i-2} \ \textbf{or} \ \ldots \ \textbf{or} \ K_1$$

In all the orthogonal regions corresponding to the implementation of a TR component, the initial state is 'idle'. When a TR component is activated (by another TR component or by the operating system in the case of the TR component that corresponds to the root goal), its state then changes to the super-state, where the procedure of generating the transition guards ensures that the action corresponding to the highest priority rule that is active is executed. The concurrent regions are necessary to determine, in parallel, which rule should be activated according to the priority and the rule's activation conditions. The way in which the transitions' guards are generated, and the use of orthogonal regions, are determined by the need to implement the behaviour of TR programs, as described in Section 2. Fig. 8 depicts the state machine of the subgoal Fetch from the robot example, resulting from the application of the pattern shown in Fig. 7.

Fig. 9 corresponds to a TR program that includes both actions and subgoals in its rules. In this case, the rules include activation of goal G_i and there is a separate TR component for this goal. The region managing subgoal G_i includes activation of the component when the condition is fulfilled, and its deactivation when the 'activateG_i' state is abandoned, either because the condition is no longer fulfilled or because the 'deactivate' transition has been triggered. The rest of the state machine is generated following the general case depicted in Fig. 7. This pattern would apply recursively for any TR program.

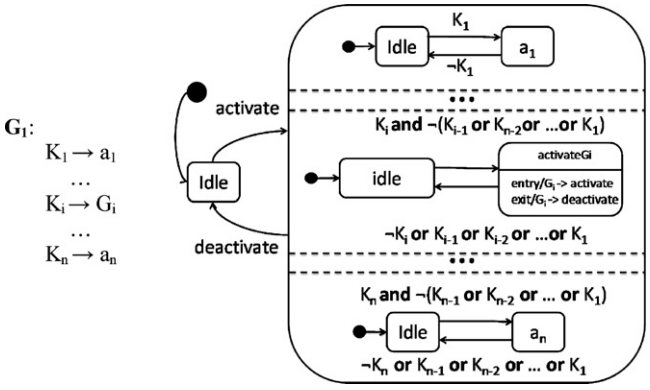


Fig. 9. Pattern to transform a TR program with subgoals.

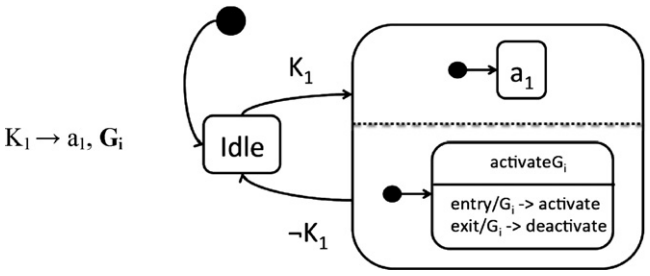


Fig. 10. Translation between a TR rule with parallel execution (action and subgoal) and the equivalent state machine.

As noted earlier, one very interesting possibility offered by TR programs is for the TR rules to include parallel execution of one or more actions or subgoals. Fig. 10 shows how to take advantage of the concurrent nature of state machines to transform a rule of this kind which includes the parallel execution of an action and a subgoal. Note that the execution of a_1 is affected only by condition K_1 and that this execution is concurrent with the activation of the TR component associated with G_i. When condition K_1 becomes false, a_1 will be interrupted and the TR component concerned will be deactivated. From there we can readily deduce how to represent any other case that includes only parallel actions, only parallel subgoals, a combination of several and so on.

In order to simplify the explanation, we have excluded the additional concurrent regions that are needed to model the interactions of each TR component with its environment (e.g., read sensors and command actuators). For example, the TR component Fetch should include a concurrent region with which to update the information from the environment, and to do this it must be connected to the boundary component Camera in order to determine whether or not the *see_can* condition is true. It is up to the software developer to adjust the minimum updating periods needed to guarantee a correct functioning of the program while meeting the system requirements.

4. Description of the development approach

As was mentioned previously, the DSIE research group has experience in the development of software using the component-based development paradigm for reactive systems including the robotic domain (Iborra et al., 2009), among others (home automation systems and wireless sensor and actuator networks). In this context, we have defined a development framework (Pastor et al., 2010) based on (1) a tool-chain for the design of components and (2) a set of implementation frameworks and additional support tools which will be used by the tool-chain. The application developer designs the architecture on the basis of the components that

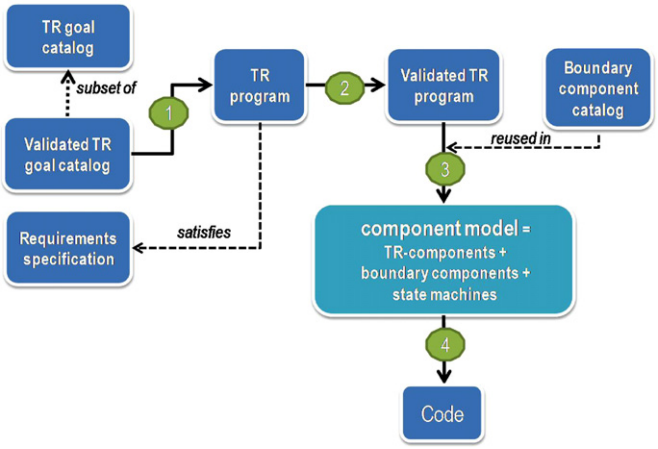


Fig. 11. General approach of the proposal.

fulfil the functional requirements of the application, and selects the framework (from a catalogue) that will provide the desired non-functional requirements and the run-time support for the application. These frameworks may have been designed ad hoc, or they may incorporate third-party solutions, and they may further include configuration and instantiation tools as well as tools for validation and verification of certain properties in the final application.

This development environment is fully supported by the technological framework provided by the MDSD tools, so that there is: (1) a modelling language with which to work with components and state machines; (2) a model transformation which instantiates the destination framework in the most appropriate manner and enables execution from the application model; (3) a model transformation which generates a temporary analysis model for real-time analysis tools; and (4) a transformation which generates a deployment model that the user can use to decide the application's deployment in distributed environments. The problem of generating code from the component-based architectural model of an application, then, has been technologically resolved.

The work presented in this article thus constitutes a step further in this development process, in that it facilitates the transition between the specification of what the system should do (requirements) and how it should do it (architectural design). Goal-based languages, and the TR paradigm in particular, provide a higher-level perspective that is closer to the problem domain than component-based language. These languages are also easier for domain experts to understand and correct. In this respect, we believe that the inclusion and use of TR programs in our development framework will facilitate both its use and the refinement of the functionality of the applications.

As shown above, the implementation of the semantics of a TR program using a component-based language considers TR components, which incorporate the behaviour of TR rules. They are components with a regular structure like the other components in the application (i.e., a common port for activation/deactivation of the component and a state machine that is derived according to the rules provided in the TR program). In terms of the component modelling language and the implementation framework, there is no difference between the origins of the components of which the framework is formed. This is an essential characteristic, since it assures uniformity of approach and tools.

Fig. 11 shows the general approach of the proposal. The developer may have a catalogue (*TR goal catalogue*) containing goals available, some of which have been previously validated (*Validated TR goal catalogue*) with regard to their behaviour. In order to carry out this validation, the developer is furnished with a tool designed by the authors to emulate a TR program, given the direct equivalence between TR programs and electrical circuits described by Nilsson (1994). The developer uses this catalogue (step 1) to compose the TR program, paying attention to the set of requirements of the software-to-be.

The developer can use the same emulator (step 2) to validate the TR program in an integral manner (*Validated TR program*). Once this step has been carried out, the developer generates (step 3) a component model as described in this article, possibly reusing boundary components from the *Boundary component catalogue*. This is done by using a model-to-model transformation (as described in the following section) in the Eclipse environment, whose result is integrated into the development process supported in the frameworks developed by the authors, so that the application's final code can be generated (step 4).

5. Implementation

This section describes the tool-chain developed to integrate the TR modelling of the application into the global component-based software development framework described above. Specifically, the tool-chain corresponds to step number 3 in Fig. 11. These tools have been developed and integrated into the free, open-source Eclipse platform using some of the plug-ins that support the MDSD approach in Eclipse.

As noted earlier, MDSD revolves around three central ideas (Bézivin, 2005): everything is a model, a model conforms to its metamodel (which is itself a model that defines a modelling language), and models evolve through meta-models by means of model transformations (which are themselves also treated as models). Metamodels define the main concepts of the domain and the relationships existing among them. Software development approaches normally comprise several metamodels, each of which defines the level of abstraction at which the software is being developed, or a specific view of it.

The Eclipse-based tool-chain developed (see screenshot in Fig. 12) comprises two metamodels (one for modelling Teleo-Reactive programs and another for component-based applications), a textual editor to facilitate the creation and validation of TR programs, and also a model transformation that embeds the correspondences between TR programs and components (as described in Section 3). The metamodel for TR programs is shown in Fig. 13. The metamodel for component-based applications, which was developed by the DSIE research group, is described in Iborra et al. (2009). This metamodel is part of the development approach described in Section 4, which is already available and thus falls outside the scope of this article. The complete ATL code of the transformations between TR program metamodel and component-based metamodel can be reviewed in Appendices A and B.

All the developed tools have been integrated into Eclipse using the following MDSD-related plug-ins: EMF (Eclipse Modeling Framework; Steinberg et al., 2008), which adds MOF support to Eclipse; EMF OCL (Object Constraint Language, OCL; OMG, 2006), which provides a formal language to define constraints and queries on models; ATL (Atlas Transformation Language; Jouault et al., 2008), which adds a declarative transformation language; and Xtext (The Eclipse Foundation, 2011), which provides a framework for creating textual model editors from metamodels.

The user of the tool-chain starts by using the Xtext model editor (see Fig. 12, window 1) to create the TR program. This textual model editor enables the definition of TR programs using the same original syntax, as defined by Nilsson (1994). Once the program is finished, the user executes the ATL transformations (arrows in Fig. 12) to generate the component model (see Fig. 12, window 2)

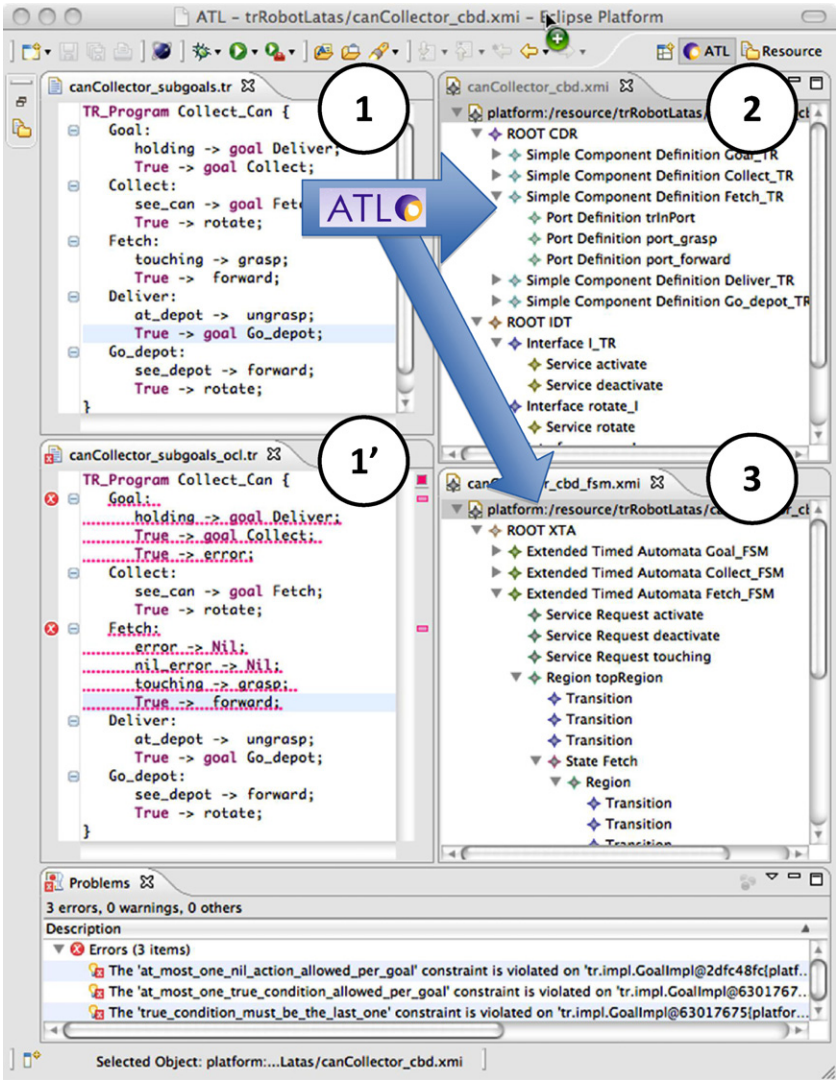


Fig. 12. Screenshot of the Eclipse tool-chain with the textual model editor (window 1), generated component model (window 2) and generated state machine model (window 3). An ill-formed TR program is shown (window 1′) together with the violated OCL constraints.

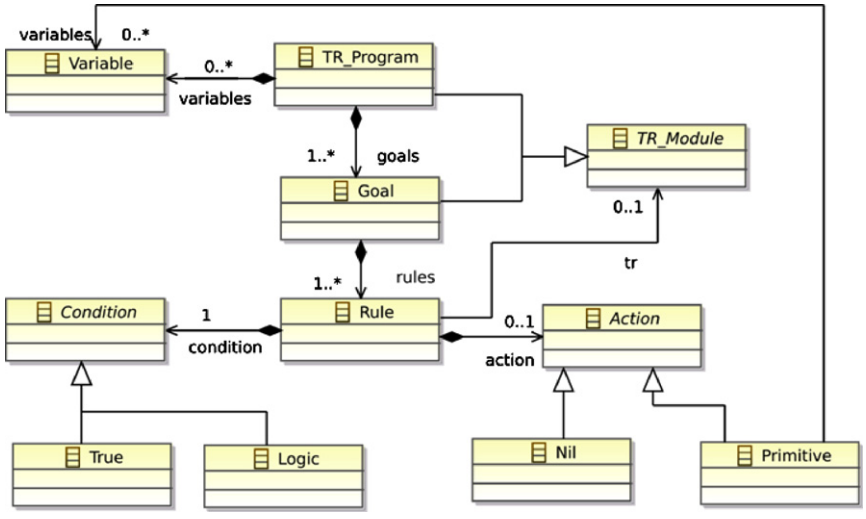


Fig. 13. Metamodel for TR programs. As defined in the MOF standard, metamodels can be depicted using the well-known UML class diagram representation.

and the state machines (see Fig. 12, window 3) that implement the semantics of the input TR program, as described in Section 3. The Xtext editor not only provides syntax colouring, but also facilitates, among other things, the editing and syntax validation of models and the detection of basic restrictions by means of additional OCL constraints. These constraints make it possible to detect ill-formed models, that is models that conform to the metamodel but are not correct. For instance, with the metamodel it is possible to create a goal that has two rules with true conditions but is not correct. With MDSD technologies OCL constraints can be used to detect such ill-formed models. One such ill-formed model is shown in Fig. 12 (window 1′), where we can see which elements violate the defined OCL constraints. Some of the added constraints are:

```
invariant at_most_one_nil_action_allowed_per_goal:
 rules->select(i : Rule | if i.action.oclIsUndefined() then false
  else i.action.oclIsTypeOf(Nil) endif)->size() < 1;
invariant true_condition_must_be_the_last_one:
 let a : Rule  rules->select(i : Rule | i.condition.oclIsTypeOf(True))->
  first() in if a.oclIsUndefined() then true else a  rules->last() endif;
invariant at_most_one_true_condition_allowed_per_goal:
 rules->select(i : Rule | i.condition.oclIsTypeOf(True))->size() < 1;
```

Fig. 13 depicts the metamodel for TR programs, which corresponds to the definitions provided in Section 2. It states that a TR program contains a set of Goals, each of which is composed of a set of Rules which comprise a Condition (either the True condition or a general-purpose Logical condition) and either an Action (Primitive or the Nil action) or a Goal (or TR program, since it is also considered a TR-Module). It also considers that Primitive actions can have Variables, though this fact is not used in the example of the can collector robot.

Once the TR program is well-formed, two ATL model transformations (see Appendices A and B) are in charge of automatically generating a component-based (structure and behaviour) representation by applying the rules described in Section 3. ATL transformations are defined as sets of declarative rules that define which model elements they process (field 'from' in the ATL source code), and which model elements they generate (field 'to'). Since it is a declarative language, relative order among rules is not important in ATL.

The transformation from TR programs to architectural components (Appendix A) comprises five rules. From a TR program, Rule RootElement (lines 6–19) generates a component repository in which all the components generated by the rest of the rules will be stored, as well as the I_TR interface that contains the 'activate' and 'deactivate' services used by TR components. Rule Action2Iface (lines 20–24) generates an interface with one service for each primitive action in the TR program, while rules Rule2TR_Port (lines 32–40) and Rule2Port (lines 41–45) generate, respectively, ports that require the I_TR interface or the interface created in the corresponding Action2Iface rule. Lastly, rule Goal2SCD (lines 25–31) transforms a Goal into a TR component that includes (1) a port providing the I_TR interface (with which to activate and deactivate the component); (2) as many ports requiring the I_TR interface as there are Goals invoked from the rules contained in the Goal that is being processed at that moment (Rule2TRPort rule in the ATL code); and (3) as many ports as there are primitive actions that appear in the rules contained in the Goal that is being processed at that moment (Rule2Port rule in the ATL code).

Regarding state machine creation, the transformation comprises four rules (see Appendix B). Rule RootFSM (lines 6–9) generates the repository for storing all the components' state machines. Rule Goal2FSM (lines 10–23) generates the overall structure of the state machine generated from each goal, while rule Rule2Region (lines 29–40) generates a region from each rule in a goal. Both rules (Goal2FSM and Rule2Region) generate all the states and transitions shown in the figures in Section 3, since they follow a regular structure in which only the conditions of the transitions differ from goal to goal. Rule Logic2Event (lines 24–28) is in charge of generating the events that trigger the transitions of the state machine, depending on the conditions of the input TR program.

6. Evaluation of the proposal

The success of the proposal was measured by conducting a survey among postgraduate students at the Universidad Politécnica de Cartagena in Spain. The survey was set up according to Pfleeger and Kitchenham (2001). The first step was to set the survey objective. The objective of our evaluation was to determine by how much the time and effort required to derive a component-based architectural model from a TR program was reduced using the proposed tool and the development approach by comparison with manual component modelling. The advantage of selecting post-graduate students to evaluate the proposal is that they are on hand for training in the use of the environments developed and the evaluation interests them because of the opportunity to participate in the use of the tools produced by research work (some of these students are going on to do PhDs). The drawback of the evaluation method is firstly that the number of opinions received is not representative enough for statistical purposes, and secondly that the tool is intended for a more specialized kind of user who is more demanding in terms of utilities and the robustness of the environment. But even so, it was decided to go ahead with the experiment for the feedback and opinions, which have helped to improve the tools still further. As noted below in the conclusions, our future plans include a survey targeting a more specific group of users for development in real time.

All the students had received previous training (as part of the robotics subject on the 'information and communication technologies' master's course) in real-time systems, reactive system modelling (UML, SysML), and a basic introduction to the concepts and tools associated with MDSD and CBSD. In addition, all students were taught the TR formalism and the implementation of the robotic examples.

The 24 students were randomly divided into two groups of equal size. As detailed above, the objective of the evaluation was to discover to what extent it was more efficient to derive the component model of a reactive system (particularly a robotic system) using the automatic TR component derivation mechanism (Group I), than to manually derive the component model using a component and state machine editor (Group II). It is important to note that the Group I students had to make decisions as to what boundary components to use, at what level to group subgoals so as to have more or less TR components, and so on. For that reason, although they did not concern themselves with component model generation or the behaviour of state machines (which is automatic), they had to make design decisions, in the course of which they could make mistakes. Therefore, although the results are much better than without the environment, there are decisions that cannot be made entirely without the intervention of the developer.

Table 1
Questions in the survey.

#	Question
Q1	How many attempts were necessary to attain a correct version? (1–3 or >3)
Q2	How difficult was it to model the case study? (1: not difficult–5: too difficult)
Q3	How much time was needed to obtain the correct model? (minutes)
Q4	Were you able to successfully incorporate the modification? (yes/no)
Q5	What was your degree of confidence in the architectural model before testing the robot? (1: none–5: total)
Q6	What level of experience do you consider is required to model similar case studies? (1: very low–5: very high)

The basic problem of the above-mentioned example of the can collector robot was defined for each group. Group II was only provided with the tool-chain for component-based development and the implementation framework (Pastor et al., 2010). Thus, Group II had to generate the component model from the TR program manually. Each group was composed of 6 pairs of students to resolve the problem, and each pair used the technology placed at their disposal. Whenever they developed a component model proposal (with the corresponding state machines), they were able to put it into practice by installing the code they had generated in the robot, which had been prepared for the case study. Each pair of students was given the opportunity to try out the model that they had constructed up to a total of three times. If they did not manage to implement it correctly by the third attempt, they recorded the trial as a failure. Every time a model was tested, another pair of students verified that its behaviour was as expected. Those pairs that found the solution were asked to make a minor modification to the requirements that had to be incorporated in the model constructed. In this case, they were only allowed one attempt to incorporate the modification.

The mobile robot used in the survey (see Fig. 14) belongs to the family of robots used for initiation, teaching and rapid prototyped robotic applications called the "LabVIEW Robotics Starter Kit" produced by National Instruments. The kit includes the mobile robot and the LabVIEW Robotics software with which to program it.

We drafted a number of questions (see Table 1) to gauge the level of success in deriving the component model from Teleo-Reactive specifications.

The questions were composed with reference to Pfleeger and Kitchenham (2001) and have the following properties: they are neutral, in other words we have avoided the use of wording that might influence the answer; they adequately cover the topic; the order of the questions is independent, meaning that the answer to one does not influence the response to the next; and they represent unbiased and mutually exclusive response categories. After pre-testing the questionnaire (i.e., reliability, understandability, validity, etc.) the students filled it in. No time limit was imposed for filling in the questionnaire. Before undertaking any detailed analysis, the responses were vetted for consistency and completeness, as recommended by Pfleeger and Kitchenham (2001). The students received the questionnaire along with an explanation of the purpose of the survey.

Table 2 shows the statistical results for each of the 6 pairs of students that participated. As this table shows, the outcomes of the study as reflected in the survey indicate that the proposal helped to provide a better design of the robotic system, and hence to reduce time and costs.

The results of the evaluation indicate that those students who used the proposal presented in the article clearly benefitted from it. It should be highlighted that all the students in Group I were able to complete the initial case study in less than the given time, and were even able to model the initial problem. Moreover, the perceived degree of difficulty of the tools is much less than that of the students from Group II, who perceived the tools as much more complex to use.

Table 2
Results of the evaluation.

	Q1	Q2	Q3	Q4	Q5	Q6
Group I	2	2	50	Y	4	2
	1	1	40	Y	4	1
	2	2	45	Y	5	2
	3	3	60	Y	3	3
	2	1	48	Y	5	2
	1	2	57	Y	4	1
Average	1.8	1.8	50	Y	4.1	1.8
Group II	>3	4	85	N	2	5
	3	4	76	Y	3	4
	2	3	70	Y	3	3
	3	5	87	N	2	4
	>3	5	79	N	1	5
	3	4	76	Y	2	3
Average	3	4.1	78.8	N/Y	2.1	4

7. Related work

Since the original work by Nilsson (1994), various authors have tackled the construction of reactive systems using the Teleo-Reactive paradigm. Hayes (2008) and Dongol et al. (2010) provide a formalization of the semantics of TR programs and a set of rules with which to reason about them. This work provides a reasoning framework to support the verification of TR programs, through time-interval semantics. Nilsson extended his own work (Benson and Nilsson, 1995) by introducing an autonomous agent architecture that integrates the ability to produce robust and flexible systems and to react appropriately in dynamic environments with the ability to plan and learn. Later, Nilsson (2001), the same author presented the "Triple-Tower T-R Agent", a three-level (Perception, Model and Action) architecture for developing robots. The proposal is an interesting framework for addressing the synthesis of control systems for robots and other reactive systems. In Coffey and Clark (2006) this last work provided a basis for a hybrid robot control architecture based on agent control schemes (Beliefs–Desires–Intentions, BDI) complete with TR programming. In Broda et al. (2000), a construct process is presented in which TR programs can be constructed systematically. This work is particularly interesting in that it provides methodological steps for the definition of TR programs. Hawthorne and Anthony (2010), on the same subject, present details of a software engineering strategy for driving program development and reducing the occurrence of problems.

In Vargas and Morales (2009) a TR program learning process for mobile robots in different dynamics and unknown environments is introduced. The learning process is created by using simple grammar induction algorithms, enabling them to incrementally learn more complex tasks on top of others that have been learned previously. Some authors have explored the possibility of implementing TR programs using high level programming languages such as the native C++ code (Weiglhofer, 2011), or interpreters using a framework for developing parallel programs (Zelek and Levine, 1995). Many other papers do not focus specifically on robotics. Marinovic et al. (2010a) argue that ECA (Event–Condition–Action) policies are too primitive to use when the agents managed are humans, and that the use of TR programs to model and integrate the roles of humans in pervasive services is more appropriate. These authors also demonstrate the benefits of adopting the TR paradigm in the

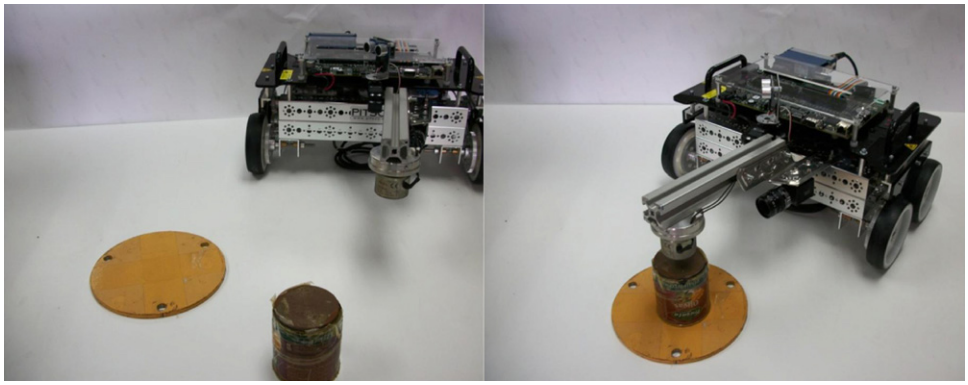


Fig. 14. Robot used in the practices and in evaluation of the proposal: frontal view of robot moving towards detected can (left); robot finalizing process of placing can in depot (right).

management of workflow-based systems (Marinovic et al., 2010b) and policy-based systems (Twidle et al., 2010). In Broda and Hogger (2005) the TR paradigm is used as a basis to define "Teleo-Reactive agents" that behave autonomously under the control of policies, and are predisposed by those policies to achieve goals. Payne (2008) presents a flexible policy language that will help designers of predictably resilient reconfigurable systems to govern the component reconfigurations that may take place.

Furthermore, the interest in utilizing MDSD and CBSD in the development of reactive systems (and real-time systems in general) is patent in various initiatives, some of which are highly relevant in terms of the businesses and projects that support the idea. Prominent among these are the SAE (Society of Automotive Engineers) AADL standard (Feiler et al., 2006), the AUTOSAR open architecture (AUTOSAR, 2011) and the European project OpenEmbeDD (OpenEmbedded, 2011). In all these cases there are highly diverse companies operating in different sectors, but all concerned with reactive system development. In addition to the particular efforts of individual enterprises, there are large European projects in progress involving several enterprises from different sectors that are developing reactive systems, in an attempt to create suitable standards and support tools that will enable them to start designing their system using the new technologies. It is worth highlighting those R&D projects that are financed with European funds—"Robot Standards and Reference Architectures" (RoSTa) and "Best Practice in Robotics" (BRICS)—which are devoted to developing software for robotic systems using the MDSD and CBSD paradigms. Be it said, however, that in both cases the basis is an abstraction level provided by components, while this article proposes a solution in which the design is conceived from the point of view of the goals that the robot should attain.

All of the above-mentioned works complement the proposal presented here, since they address either the domain of artificial intelligence or the use of formal techniques for reasoning about TR specifications, or they deal with application of the paradigm to the construction of systems of various kinds, not only robotics. None of these works meet our requirements, which are: to produce a component-based software architecture, to provide a complete behaviour description, and to permit the generation of code. The contribution of this article is significant in that it is the first solution that enables TR programs to be translated to architectural models so as to reduce the conceptual leap between specification and implementation of the solution. Since there is a great deal of literature concerning component-based software development, as well as new mechanisms for code generation in the context of frameworks and model driven development, the solution presented here can obviously be carried forward by any developers who find it necessary to combine both approaches.

In this context, our article makes an additional contribution to the production of tools, methods and techniques to raise the abstraction level in the development of reactive systems, while taking advantage of the reusability provided by the component model, and the automatic programming facilities from models that offer an MDSD approach.

8. Conclusions and future work

The TR approach offers a high-level programming style for the development of reactive systems such as robotic vehicles. TR programs are written in a way that allows engineers to define behaviour taking account of goals and changes in the state of the environment, which is particularly useful in the case of mobile robotics. Nowadays TR implementations for reactive system development are a highly promising subject of study.

This article has demonstrated the feasibility of translating TR specifications into component-based representations. This gives engineers the opportunity to carry on working on code generation without losing the benefits of both approaches. The results are highly satisfactory, both as regards the viability of the tools developed and as regards the degree of acceptance shown in the evaluation carried out with a group of post-graduate students.

One of the contributions of this work that we should like to highlight is the translation of a TR program into architectural components, of both boundary and TR types, described in Section 3. This translation is sufficiently general to allow implementation using third-party component languages. In this connection, one of our intended future projects is an alternative implementation to the one described here, using the OSGi service platform or the Fractal (Blair et al., 2009) component model. The extensions to the basic Teleo-Reactive formalism that we are currently considering include: the use of agents to coordinate the execution of independent TR programs (as suggested in Benson and Nilsson, 1995), the use of discrete actions and a model of the world that could vary according to the actions to be executed (including an additional formalism that would allow the change in state to be represented), and the use of parameters of any type in the invocation of goals (currently only included in the invocation of actions corresponding to boundary components).

In the area of Domain Engineering, we intend to keep on enhancing the proposal with a catalogue that will be sufficiently representative of goals (extracted from more complex examples) in order to evaluate their capacity for reuse. This, along with the development of a tool integrated in the Eclipse environment itself, will allow uniform and traceable management that is interrelated with these goals. Moreover, owing to the very nature of

the TR program, if it can be translated directly into an equivalent electronic circuit, it would be possible to consider co-design techniques to select which parts of the application to implement on hardware and which parts to deploy in a conventional computing system. Finally, we plan to extend the evaluation of the proposal to professional developers of reactive systems, as well as to researchers in other research groups that have experience in applying software engineering techniques to the development of robotic systems.

Acknowledgements

The authors wish to thank the Fundación Séneca of the Murcia Region (ref. 15374/PI/10) and the CICYT EXPLORE (ref. TIN2009-08572), Ministry of Education and Science, Spain, for partially supporting this work. They also want to thank N. Dulay (Imperial College, London) for his explanations about the use of TR programs in implementing workflow-based and human-centric pervasive systems.

Appendix A. ATL transformation from TR program to architectural components

```
1  module tr2v3cmm;
2  create v3M : v3MM from trM : trMM;

4  helper def : trInterface : v3MM!Interface   OclUndefined;
5  helper def : app : v3MM!ComplexComponentDefinition   OclUndefined;

6  rule RootElement {
7   from f : trMM!TR_Program
8   to t : v3MM!ROOT_CDR (componentDefinitions<-f.goals),
9     ri: v3MM!ROOT_IDT (interfaces<-trI,
        interfaces<-trMM!Primitive.allInstances()),
10    ra: v3MM!ROOT_APP (application<-app),
11    app: v3MM!ComplexComponentDefinition(name<-f.name,
       components<-f.goals->collect(i|thisModule.resolveTemp(i,'comp'))),
12    trI : v3MM!Interface (name<-'I TR', services<-Set{act, deact}),
13    act : v3MM!Service (name<-'activate'),
14    deact : v3MM!Service (name<-'deactivate')
15  do {
16   thisModule.trInterface<-trI;
17   thisModule.app<-app;
18  }
19 }
20 rule Action2Iface {
21  from f : trMM!Primitive
22  to t : v3MM!Interface (name<-f.name+'_I', services<-s),
23    s : v3MM!Service(name<-f.name)
24 }
25 rule Goal2SCD {
26  from f : trMM!Goal
27  to t : v3MM!SimpleComponentDefinition(name<-f.name+'_TR', ports<-trIP,
        ports<-f.rules),
28    trIP : v3MM!PortDefinition (name<-'trInPort',
        providedInterfaces<-thisModule.trInterface),
29    comp : v3MM!Component(name<-f.name+' TR',definitions<-t,
        ports<-comp_p,
        ports<-f.rules->collect(i|thisModule.resolveTemp(i,'trOP_c'))),
30    comp_p : v3MM!Port(name<-'trInPort', definitions<-trIP)
31 }
32 rule Rule2TR_Port {
33  from f : trMM!Rule (f.action.oclIsUndefined())
34  to trOP : v3MM!PortDefinition (name<-'trOutPort_'+f.tr.name,
        requiredInterfaces<-thisModule.trInterface),
35   trOP c : v3MM!Port (name<-'trOPort '+f.tr.name, definitions<-trOP),
36   link : v3MM!AssemblyLink(endA<-trOP c,
        endB<-thisModule.resolveTemp(f.tr,'comp_p'))
37  do {
38   thisModule.app.portLinks<-link;
39  }
40 }
41 rule Rule2Port {
42  from f : trMM!Rule (f.tr.oclIsUndefined())
43  to p : v3MM!PortDefinition (name<-'port_'+f.action.name,
        requiredInterfaces<-f.action),
44    trOP_c : v3MM!Port (name<-p.name, definitions<-p)
45 }
```

Appendix B. ATL transformation from TR program to finite state machines

```
module tr2v3cmm_fsm;
create v3M : v3MM from trM : trMM;

helper context trMM!Goal def: guardTxt (until : trMM!Rule) : String
 self.rules.asSequence().subSequence(self.rules.first(),
 self.rules.indexOf(until)-1)->collect(i|i.condition.expression+' or ')
 ->flatten();

rule RootFSM {
 from f : trMM!TR_Program
 to t : v3MM!ROOT_XTA (xtas<-f.goals)
}
rule Goal2FSM {
 from f : trMM!Goal
 to t : v3MM!ExtendedTimedAutomata(name<-f.name+'_FSM',topRegion<-tr,
      events<-Set{ea,ed}, events<-f.rules->select(i|
      i.condition.oclIsTypeOf(trMM!Logic))->collect(i|i.condition),
      events<-f.rules->select(i|i.condition.oclIsTypeOf(trMM!Logic))
      ->collect(i|thisModule.resolveTemp(i.condition,'nt'))),
   ea : v3MM!ServiceRequest(name<-'activate'),
   ed : v3MM!ServiceRequest(name<-'deactivate'),
   tr : v3MM!Region(name<-'topRegion',vertices<-Set{iniS,idleS,mainS},
   transitions<-Set{tIni,tAct,tDeact}),
   iniS : v3MM!InitialPseudostate,
   idleS : v3MM!State(name<-'Idle'),
   mainS : v3MM!State(name<-f.name, regions<-f.rules),
   tIni : v3MM!Transition(source<-iniS,target<-idleS),
   tAct : v3MM!Transition(source<-iniS,target<-mainS, firedBy<-ea),
   tDeact : v3MM!Transition(source<-mainS,target<-idleS, firedBy<-ed)
}
rule Logic2Event {
 from f : trMM!Logic
 to t : v3MM!ServiceRequest(name<-f.expression),
   nt : v3MM!Signal(name<-'not ('+f.expression+')')
}
rule Rule2Region {
 from f : trMM!Rule
 to t : v3MM!Region (vertices<-Set{iniS,idleS,mainS},
      transitions<-Set{tIni,tAct,tDeact}),
   iniS : v3MM!InitialPseudostate,
   idleS : v3MM!State(name<-'Idle'),
   mainS : v3MM!State(name<-'Main'),
   tIni : v3MM!Transition(source<-iniS,target<-idleS),
   tAct : v3MM!Transition(source<-iniS,target<-mainS,
      firedBy<-f.condition, guardsAnd<-actG),
   actG : v3MM!UserDefinedGuard(
      description <- f.refImmediateComposite().guardTxt(f)),
   tDeact : v3MM!Transition(source<-mainS,target<-idleS, firedBy<-
      thisModule.resolveTemp(f.condition,'nt'), guardsOr <- deactG),
   deactG : v3MM!UserDefinedGuard(
      description <- f.refImmediateComposite().guardTxt(f))
}
```

References

Atkinson, C., Kühne, T., 2003. Model-driven development: a metamodelling foundation. IEEE Software 20 (5), 14–18.

AUTOSAR, 2011. The AUTOSAR Project., http://www.autosar.org.

Benson, S., Nilsson, N., 1995. Reacting, planning and learning in an autonomous agent. Machine Intelligence, vol. 14. Oxford University Press Inc., New York, NY, USA.

Bézivin, J., 2005. On the unification power of models. Journal of Systems and Software 4 (2), 171–188.

Blair, G., Coupaye, T., Stefani, J., 2009. Component-based architecture: the fractal initiative. Annals of Telecommunication, vol. 64. Springer-Verlag.

Best Practice in Robotics (BRICS), Collaborative Project funded under EU's FP7. Available from: www.best-of-robotics.org/.

Broda, K., Hogger, C.J., 2005. Determining and verifying good policies for cloned Teleo-Reactive agents. Computer Systems Science and Engineering 20 (4), 249–258.

Broda, K., Hogger, C.J., Watson, S., 2000. Constructing Teleo-Reactive robot programs. In: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI), Germany, pp. 653–657.

Brugali, D., Scandurra, P., 2009. Component-based robotic engineering. Part I: Reusable building blocks. IEEE Robotics and Automation Magazine 16 (4), 84–96.

Brugali, D., Scandurra, P., 2010. Component-based robotic engineering. Part II: Systems and models. IEEE Robotics and Automation Magazine 17 (1), 100–112.

Castro, J., Lucena, M., Silva, C., Alencar, F., Santos, E., Pimentel, J., 2012. Changing attitudes towards the generation of architectural models. Journal of Systems and Software 85 (3), 463–479.

Coffey, S., Clark, K., 2006. A hybrid Teleo-Reactive architecture for robot control. In: Multi-Agent Robotic Systems.

Crnkovic, I., Hnich, B., Jonsson, T., Kiziltan, Z., 2002. Specification, implementation, and deployment of components. Communications of the ACM 45 (10), 35–40.

Dongol, B., Hayes, I.J., Robinson, P.J., 2010. Reasoning about real-time Teleo-Reactive programs. Technical Report SSE-2010-01. Division of Systems and Software Engineering Research, School of Information Technology and Electrical Engineering, The University of Queensland, QLD, Australia. http://www.itee.uq.edu.au/~sse.

Estublier, J., Vega, G., 2005. Reuse and variability in large software applications. SIGSOFT Software Engineering Notes 30 (5), 316–325.

Feiler, P., Gluch, D., Hudak, J., 2006. The Architecture Analysis & Design Language (AADL): An Introduction. CMU Software Engineering Institute.

Gubisch, G., Steinbauer, G., Weiglhofer, M., Wotawa, F., 2008. A teleo-reactive architecture for fast, reactive and robust control of mobile robots. Lecture Notes in Compute Science 5027, 541–550.

Hawthorne, J., Anthony, R., 2010. A methodology for the use of the Teleo-Reactive programming technique in autonomic computing. In: Proceedings of the 11th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, UK, pp. 245–250.

Hayes, I.J.,2008. Towards reasoning about Teleo-Reactive programs for robust real-time systems. In: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems (SERENE'08). ACM, New York, NY, USA, pp. 87–94.

Iborra, A., Alonso, D., Ortiz, F., Pastor, J.A., Sánchez, P., Álvarez, B., 2009. Experiences using software engineering in the design of service robots. IEEE Robotics and Automation Magazine 16 (1), 24–33.

Jouault, F., Allilairea, F., Bézivin, J., Kurtev, I., 2008. ATL: a model transformation tool. Science of Computer Programming 72 (1–2), 31–39.

Lamsweerde, A., 2009. Requirements Engineering: From Goals to UML Models to Software Specifications. John Wiley & Sons Ltd., England.

Laplante, P., 2004. Real-Time Systems Design and Analysis. Wiley-IEEE Press.

Marinovic, S., Twidle, K., Dulay, N., Sloman, M., 2010a. Teleo-Reactive policies for managing human-centric pervasive services. In: International Conference on Network and Service Management (CNSM), Canada, pp. 80–87.

Marinovic, S., Twidle, K., Dulay, N., 2010b. Teleo-Reactive workflows for pervasive healthcare. In: Proceedings of First IEEE PerCom Workshop on Pervasive Healthcare, Germany, pp. 316–321.

Medvidovic, N., Dashofy, E.M., Taylor, R.N., 2007. Moving architectural description from under the technology lamppost. Information and Software Technology 49 (1), 12–31.

National Instruments. LabVIEW. http://sine.ni.com/ds/app/doc/p/id/ds-217/lang/es.

Nilsson, N.J., 1994. Teleo-Reactive programs for agent control. Artificial Intelligence Research 1 (1), 139–158.

Nilsson, N.J., 2001. Teleo-Reactive programs and the Triple-Tower architecture. Electronic Transactions on Artificial Intelligence 5, 99–110.

OMG, 2006. Object Constraint Language (OCL) Specification v2.0., http://www.omg.org/spec/OCL/2.0/.

OpenEmbedded, 2011. The OpenEmbedded Project., http://www.openembedded.org.

Pastor, J.A., Alonso, D., Sánchez, P., Álvarez, B., 2010. Towards the definition of a pattern sequence for real-time applications using a model-driven engineering approach. 15th International Conference on Reliable Software Technologies (Ada-Europe 2010). Lecture Notes in Compute Science 6106, 167–180.

Payne, R.J., 2008. RPL: a policy language for dynamic reconfiguration. In: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems, SERENE'08, pp. 73–78.

Pfleeger, S., Kitchenham, B., 2001. Principles of survey research. ACM SIGSOFT Software Engineering Notes 26, 16–18.

Robot Standards and Reference Architectures (RoSTa), Coordination Action funded under EU's FP6. Available from: http://www.robot-standards.eu/.

Sánchez, P., Jiménez, M., Rosique, F., Álvarez, B., Iborra, A., 2011. A framework for developing home automation systems: from requirements to code. Journal of Systems and Software 84 (6), 1008–1021.

Shaw, M., Clements, P., 2006. The golden age of software architecture. IEEE Software 23 (2), 31–39.

Steinberg, D., Budinsky, F., Paternostro, M., Merks E, E.M.F., 2008. Eclipse Modeling Framework. Addison-Wesley Professional.

Szyperski, C., 2002. Component Software: Beyond Object-Oriented Programming. Addison-Wesley.

Taylor, R., Medvidovic, N., Dashofy, E., 2009. Software Architecture: Foundations, Theory, and Practice. John Wiley & Sons Ltd.

The Eclipse Foundation, 2011. Xtext., http://www.eclipse.org/Xtext/.

Twidle, K., Marinovic, S., Dulay, N., 2010. Teleo-Reactive policies in Ponder2. In: IEEE International Symposium on Policies for Distributed Systems and Networks, USA, pp. 57–60.

Vargas, B., Morales, E.F., 2009. Learning navigation Teleo-Reactive Programs using behavioural cloning. In: IEEE International Conference on Mechatronics (ICM'09), pp. 1–6.

Weiglhofer, M., 2011. Extended Teleo-Reactive Compiler., http://www.ist.tugraz.at/staff/weiglhofer/projects/trcompiler/.

Zelek, J.S., Levine, M.D., 1995. Teleo-reactive autonomous mobile navigation. In: Canadian Conference on Electrical and Computer Engineering, vol. 1, pp. 477–480.

Pedro Sánchez is an associate professor of computer science at the Universidad Politécnica de Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include model-driven engineering for real-time systems. Sánchez has a PhD in computer science from the Technical University of Valencia.

Diego Alonso is an associate professor of computer science at the Universidad Politécnica de Cartagena and a member of the DSIE (Division of Systems and Electronic Engineering) research group. His research interests focus on the application of the model-driven engineering approach to the development of component-based reactive systems with real-time constraints. He has a PhD with "Doctor Europaeus" Mention from the Universidad Politécnica de Cartagena.

José Miguel Morales is an assistant professor and a PhD student in computer science at the Universidad Politécnica de Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include real time systems, and specification and design of reactive systems. Morales has a master's in information technology engineering from the Universidad de Murcia.

Pedro Javier Navarro is an associate professor of computer science at the Universidad Politécnica de Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include image processing techniques for detecting defects in texture and advanced robotics and intelligent transportation systems with special emphasis on computer vision. Navarro has a PhD in computer vision from the Technical University of Cartagena.

A SYSTEMATIC LITERATURE REVIEW OF THE TELEO-REACTIVE PARADIGM

RESUMEN

N.J. Nilsson definió el enfoque Teleo-Reactivo en el año 1994. Desde entonces muchos investigadores han usado dicho enfoque bien aplicándolo a un dominio determinado o extendiéndolo para añadir nuevas capacidades a la definición original. Este artículo proporciona una revisión sistemática de 53 estudios basados en el paradigma Teleo-Reactivo publicados previamente en revistas o actas de congresos. El objetivo de este artículo es identificar, seleccionar y sintetizar toda esta actividad investigadora de alta calidad relacionada con el uso del paradigma Teleo-Reactivo. La literatura ha sido sistemáticamente revisada para ofrecer una visión del estado actual de este campo de estudio y para identificar los principales resultados obtenidos gracias al enfoque Teleo-Reactivo. Por último, el artículo detalla los desafíos y dificultades que deben ser superador para asegurar futuros avances en el uso de esta técnica.

Artif Intell Rev
DOI 10.1007/s10462-012-9350-2

A systematic literature review of the Teleo-Reactive paradigm

Jose Luis Morales · Pedro Sánchez · Diego Alonso

© Springer Science+Business Media B.V. 2012

Abstract The Teleo-Reactive approach designed by N. J. Nilsson offers a high-level programming model for the development of reactive systems such as robotic vehicles. Teleo-Reactive programs are written in a way that allows engineers to define behaviour taking account of goals and changes in the state of the environment. Since Nilsson's original definition, published in 1994, various researchers have used the Teleo-Reactive paradigm, either applied to a particular domain or extended by adding more capabilities to the original definition. This article provides a systematic literature review of 53 previous Teleo-Reactive-based studies in journals, conference proceedings and the like. The aim of this paper is to identify, appraise, select and synthesize all this high-quality research evidence relating to the use of the Teleo-Reactive paradigm. The literature has been systematically reviewed to offer an overview of the present state of this field of study and identify the principal results that have been obtained thanks to the Teleo-Reactive approach. Finally, this article details the challenges and difficulties that have to be overcome to ensure further advances in the use of this technique.

Keywords Teleo-Reactive formalism · Reactive systems · Systematic review

1 Introduction

This paper reviews several studies on the Teleo-Reactive paradigm, as defined by Nilsson (1994), to evaluate progress and the outlook for future research on the development of reactive systems using this paradigm. A reactive system must respond to changes occurring in

J. L. Morales · P. Sánchez (✉) · D. Alonso
DSIE Research Group, Universidad Politécnica de Cartagena, 30202 Cartagena, Spain
e-mail: pedro.sanchez@upct.es

J. L. Morales
e-mail: joseluis.morales@upct.es

D. Alonso
e-mail: diego.alonso@upct.es

Published online: 30 June 2012

its environment by taking actions and observing a set of rules. The actions taken by a reactive system must produce a change in its environment. These computer-based systems are commonly used to control critical systems where failures can cost money or lives. One representative instance of a reactive system is the mobile robotics domain. This domain embraces applications in which robotic vehicles act and move autonomously in semi-structured and non-structured environments. Their field of application is immense, and in fact very many systems have been designed, both for outdoor use (agriculture, forestry engineering, etc.) and for indoor use (facilities inspection, care assistance, etc.). Because of the growing sophistication of these applications, more sophisticated languages and platforms are needed to express, validate and implement them.

There are a number of authors who maintain that software development should address what the user seeks from it—i.e. should be goal-oriented—and that these goals should guide the requirements engineering process (Lamsweerde 2009). Since reactive systems may benefit from this approach, many authors have adopted the Teleo-Reactive paradigm, a goal-oriented approach for modelling systems that change their actions, outputs and status in response to stimuli from within or outside them. The Teleo-Reactive paradigm has produced highly interesting results, particularly in the domain of robotics as detailed further below. This paradigm as applied to reactive systems is both a viable and an exciting prospect in that it inherently recovers from errors and unexpected events whilst proceeding towards its goal. Teleo-Reactive programs are written in a production-rule-like language. The Teleo-Reactive programming model is a high-level approach to implementing systems that react dynamically to changes in their environment. Because of that, they are very suitable for describing reactive systems from a goal-oriented perspective. The changes in the environment will be detected by the reactive system through sensors used to perceive its own state and the state of the environment.

In his first work Nilsson discussed what the most interesting contributions to the field of Teleo-Reactive programming might be Nilsson (1994). Principal among these are:

- To derive tools and techniques for verifying Teleo-Reactive programs. For example, in some environments under particular conditions, agents could never achieve their goals.
- To devise methods for implementing and interpreting TR-programs and the real-time properties of such implementations.
- To devise methods for modifying Teleo-Reactive programs by automatic planning and machine learning.
- To allow simultaneous and asynchronous execution of multiple actions.
- To include an explicit reference to time in actions. For instance, an action should not be initiated until after some time t_1 or should cease some time after t_2.
- To devise methods for goal definition.

This helps us to identify the areas of particular interest for purposes of the present systematic review. To that end it will be useful to classify each of the research articles found in the scientific literature in terms of their contribution to one of the categories listed in Table 1.

Thus, the purpose of this article, following an introduction to the Teleo-Reactive paradigm, is to provide an overview of the advances achieved, both through the Teleo-Reactive approach to the development of reactive systems and through contributions aimed at improving that approach. The most relevant information was gathered on each article, then a comparative and systematic analysis of the different solutions was conducted. To our knowledge this is the first study to provide a systematic review of the Teleo-Reactive paradigm. This study helps to identify and summarize the work done so far and to guide future research into the

Table 1 List of categories considered in this review

Category	Description
C1	Teleo-Reactive program definition and formalization
C2	Platforms for Teleo-Reactive program validation/emulation
C3	Development of reactive systems
C4	Methodological aspects of Teleo-Reactive program inception
C5	Advances in artificial intelligence
C6	The Teleo-Reactive paradigm as reference for other paradigms

Teleo-Reactive paradigm. The outcomes of the study are therefore an important step towards expanding the store of knowledge in the field of Teleo-Reactive system development.

The article is organized as follows: Sect. 2 summarizes the Teleo-Reactive approach with a basic case study. Section 3 describes the review process. Section 4 summarizes the results of the study. Section 5 presents a discussion of those results and details the principal challenges and difficulties posed by use of the Teleo-Reactive approach. Finally, Sect. 6 presents the conclusions.

2 The Teleo-Reactive paradigm

As Nilsson stated in (Nilsson 1994), a Teleo-Reactive program (hereafter TR-program) is a mid-level agent control program that robustly directs an agent towards a goal in a manner that continuously takes into account the system-changing perceptions of a dynamic environment. In other words, TR-programs are a set of reactive rules that sense the environment continuously and trigger actions whose continuous execution eventually leads the system to satisfy a goal. The main advantage of TR-programs is their ability to react robustly to changes in their environment due to the continuous computation of the parameters and conditions. TR-programs offer engineers an intuitive approach within which to write goal-directed programs.

A TR-program is given by a set of prioritized condition/action rules. A TR-program implementation should constantly re-evaluate the triggering conditions set for each rule, and should execute the action corresponding to the highest-priority rule with a satisfied pre-condition. Thus, a TR-program can be denoted by Nilsson (1994):

$$k_1 \rightarrow a_1; k_2 \rightarrow a_2; \ldots k_i \rightarrow a_i \ldots k_m \rightarrow a_m$$

where k_i are conditions on sensory inputs and on a model of the world, and a_i are actions on the world or which change the model of the world. The list of rules is evaluated from the left for the first rule whose condition is *true*, and its corresponding action is executed. An action a_i may consist of a single action, or may itself be a TR-program.

Nilsson states that if a TR-program is *complete* (where k_1 and k_2 and … k_m is a tautology) and respects the *regression* property (i.e., each condition K_i is a regression of some higher condition through an action a_i) then the system implementing the TR-program will always achieve its goal. Nilsson generalizes the notion of a TR-program by permitting:

- the rules to contain free variables (that are bound when the sequence is called),
- hierarchical sequences in which the actions in the rules are themselves TR-programs, and recursive definition of actions.

In a general TR-program, conditions, actions and goals may have free variables that are bound when the TR-program is called. It is important to highlight that unlike in conventional programming, where a called subroutine assumes control of the execution until it is completed, a TR-program continues to constantly evaluate its set of conditions even when a subgoal has been called. Consequently, if any higher-priority rule condition becomes true, the subgoal is terminated by its parent. At the same time, it is important to note that TR-programs differ substantively from conventional production systems in that actions can be durative—rather than discrete. A durative action is one that once initiated can be executed continuously, for as long as the condition holds true (for example, move the robot, as opposed to a discrete action such as move the robot ten meters). The effects of an action on the environment are a consequence of system interaction; they are not modelled in the TR-program but are present in the changes detected in the sensor inputs.

As an example of a TR-program, we take a robot that has to reach a goal. To achieve this, its movement has to be guided to the left or the right depending on where the goal is located. When it detects uncertainties in the environment (obstacles or other hazards), it switches to not-safe mode then performs the above operations in reverse: i.e., when it detects an obstacle to its left it turns right, etc. So long as the goal 'Robot' is satisfied, the robot is inactive ('nil' represents the null action). Whenever the goal (for changes in the environment) is not satisfied, the robot becomes active and continues until it reaches the goal. The equivalent TR-program is the following:

```
Robot:

     at_goal → nil
     safe → ToGoal
     true → ToSafe

ToGoal:

     see_goal_ahead → forward
     see_goal_right → turn_right
     true → turn_left

ToSafe:

     see_obstacle_left → turn_right
     see_obstacle_right → turn_left
     true → forward
```

Note that if the robot detects obstacles to both left and right at the same time, this could trigger an unwanted effect, to counter which some of the extensions described in this review may be appropriate. Nilsson also considers the graphical representation of TR-programs where nodes are labelled by conditions and arcs by actions. In order to interpret the graphical version of a TR-program, the action labelling the arc leading out from the shallowest true node should be executed. When there are two or more actions that can achieve a condition, a TR-tree is obtained instead of a single-path graph.

3 Systematic review method

As stated in Kitchenham (2004), "a systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question,

Table 2 Research questions

RQ#	Description
RQ1	What advances have been derived from extensions to the original formalism proposed by Nilsson in 1994?
RQ2	What broad areas of study has this approach addressed?
RQ3	What fields have benefited most from the approach and what have been the outcomes there?
RQ4	What are the main challenges and difficulties that have to be overcome for successful adoption of the approach? What areas of research around the approach should be encouraged?

or topic area, or phenomenon of interest". There are several possible reasons for undertaking a systematic review of a subject of interest. The most common reasons are:

- To summarize the existing evidence relating to a piece of knowledge or a particular technology.
- To provide a base on which to define new lines of action on a particular subject.
- To identify the gaps in an area of interest with a view to suggesting specific areas of research.

In our case the main purpose of undertaking such a systematic review was to answer the questions listed in Table 2.

With respect to RQ1, we propose to look at all works published since the first Teleo-Reactive formalism. There are many other formalisms proposed for prescribing sensory-directed, real-time activity in dynamic environments. Some of these are closely related to the Teleo-Reactive formalism. However, we felt it reasonable to exclude these since objective RQ1 is strictly confined to Nilsson's Teleo-Reactive paradigm. The aim is thus to determine the most significant publications in the field that have made a particular contribution to achieving immediate utility. In the case of RQ2 the idea is to present the directions followed by researchers in relation to this approach. This part identifies studies ranging from the demonstration of properties to their application in the field of artificial intelligence. To address RQ3, we have identified and analysed the various fields in which the approach has been applied, with particular attention to mobile robotics and artificial intelligence in view of the orientation announced by the very definition. In the case of RQ4 the aim is to highlight the lines of research that show most promise of results in the light of the most recent advances (basically in the last five years).

A systematic review must be, and seen to be, fair in order to produce results and conclusions that are of interest and can be extrapolated. In other words, a systematic review should only be undertaken on the basis of a previously-defined information search procedure or strategy. This study was carried out following the guidelines defined by Kitchenham (2004) for the conduct of systematic reviews.

3.1 Search process

The search for case studies was performed manually, analysing proceedings and journal papers since 1992. As stated in Brereton et al. (2007), we need to search many different electronic sources as no single source identifies all primary studies. For that purpose the following electronic databases have been used: IEEE Xplore, Scopus, ISI Web of Science,

ACM Digital Libray, SpringerLink, CiteSeer and Google Scholar. The following four groups of search terms were used in each one:

(1) "Teleo-Reactive",
(2) "TR-programs"
(3) "TeleoReactive"
(4) "Teleo" AND "Reactive"

Our experience supports the suggestion made by Kitchenham (2004): "the simpler search string might have been just as effective". Each journal and conference proceedings was reviewed by the authors, and any papers that addressed literature surveys of any type were identified as potentially relevant. The three researchers later agreed on what works were representative following the pre-defined inclusion/exclusion criteria.

3.2 Inclusion and exclusion criteria

Study selection criteria are intended to identify primary studies that provide direct evidence on the subject of the research. Peer-reviewed articles on the topic of interest, published between 1994 and 2011, were included. Articles were discarded if they presented one or more of the following characteristics:

- In the case of demonstrations of the use of the approach in implementing reactive systems, where the article did not demonstrate the deployment of real systems. To so demonstrate it had to give results of the implementation and include detailed pictures showing the design of the architecture or the physical device (i.e. a robot).
- How much emphasis, if any, the article placed on the use of the Teleo-Reactive approach.
- Repeat articles on the same study. Where reports of a study had been published in several different journals, the most complete version of the study was included in the review.

3.3 Data collection and analysis

The data extracted from each study were:

- The source (journal, conference proceedings, etc.) and full reference.
- The organization that carried out the work (universities, research centres, etc.).
- The type of contribution (research article, conference paper, etc.).
- Technical aspects concerning the results (reactive system that was implemented, tool implemented, extensions to the formalism, etc.).

The researchers did the data extracting and checked the resulting data. Sánchez coordinated the data extraction and the checking, which involved all the authors of this article. In the event of disagreements, the data were discussed in detail until agreement was reached. The intention in principle was to exclude any studies that had not been published in first-rate journals or proceedings, in any of the fields related to reactive system development, but eventually it was decided not to exclude any study on that basis since some of the most significant contributions fell outside that category and to exclude them would be to ignore findings of interest.

4 Systematic review results

Tables 3 and 4 show the results of the search procedure taking into account the category classification given in Table 4. It lists all the works identified in the survey published since

Table 3 List of conference proceedings and journals identified by the review

Publication title	Type	Year	Category
Journal of Artificial Intelligence Research	J	1994	C1
National Conference on Artificial Intelligence	C	1994	C1
Machine Intelligence Workshop	W	1995	C1
Canadian Conference on Electrical and Computer Engineering	C	1995	C1, C3
IEEE Intl. Symposium on Computational Intelligence in Robotics and Automation	C	1997	C1
IEEE International Conference on Systems, Man, and Cybernetics	C	1998	C4
Robotics Laboratory. Computer Science Department. Stanford University	T	2000	C5
European Conference on Artificial Intelligence	C	2000	C3, C5
Electronic Transactions on Artificial Intelligence	J	2001	C1
Conference on Innovative Applications of Artificial Intelligence	C	2002	C5
Stanford University	B	2002	C6
European Conference on Genetic Programming	C	2003	C5, C6
International Joint Conference on Autonomous Agents & Multiagent Systems	C	2003	C5
Advances in Artificial Intelligence	C	2003	C3
Imperial College London	T	2003	C4
German Conference on Artificial Intelligence	C	2004	C3, C4, C5
German Conference on Multiagent System Technologies	C	2004	C5
Artificial Intelligence and Soft Computing	C	2004	C3
Computer Systems: Science & Engineering	J	2005	C5
International Symposium Abstraction, Reformulation and Approximation	C	2005	C5
AAAI Fall Symposium on Agents and the Semantic Web	C	2005	C5, C6
International Conference on Inductive Logic Programming	C	2005	C5, C6
International Workshop on Multi-Agent Robotic Systems	C	2006	C5
Journal of Machine Learning Research	J	2006	C5, C6
International Symposium on Skill Science	C	2007	C5, C6
Workshop on Planning and Plan Execution for Real-World Systems	W	2007	C6
International Workshop on Software Engineering for Resilient Systems	W	2008	C1, C6
Intl. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems	C	2008	C1, C2, C3
International Conference on Intelligent Robots and Systems	C	2008	C5
Joint International Workshop on Software Engineering for Resilent Systems	W	2008	C1
International Conference on Robotic and Autonomation	C	2008	C6
IEEE International Conference on Mechatronics	C	2009	C5
International Conference in Inductive Logic Programming	C	2009	C5, C6
European Conference on Ambient Intelligence	C	2009	C3
Instituto Nacional de Astrofísica, Óptica y Electrónica	P	2009	C5
Intl. Conference on Autonomic Computing and Communications Systems	C	2009	C2, C6
The Computer Journal	J	2010	C3, C4, C5
The University of Queensland	T	2010	C1
International Conference on Network and Service Management	C	2010	C6
International Conference on Pervasive Computing and Communications	W	2010	C3
IEEE Intl. Symposium on Policies for Distributed Systems and Networks	C	2010	C6
IEEE/OES Autonomous Underwater Vehicles	C	2010	C3

Table 3 continued

Publication title	Type	Year	Category
Sea Technology Journal	J	2010	C3
International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing	C	2010	C4
Imperial College London	T	2011	C5, C6
International Conference on Autonomic and Autonomous Systems	C	2011	C4
Annals of Mathematics and Artificial Intelligence	J	2011	C2
Newcastle University	T	2011	C6
Universidad Politécnica de Cartagena	T	2011	C3
Universidad Politécnica de Cartagena	T	2011	C2

Type: *J* journal, *B* book chapter, *P* PhD Thesis, *C* conference, *W* workshop, *T* technical report

1994, the first in the table being the landmark work of Nilsson. In the course of the search we found a Technical Report by Nilsson (1992) predating the article that he published in this journal two years later. In the event we decided not to include this preliminary work in the review findings, as explained in Sect. 5. The identification of relevant literature produced 316 results, of which only 126 articles were found to be potentially useful. However, after filtering based on article titles, abstracts and in some cases a preliminary reading of contents, only 53 were finally selected as relevant to the review. In some cases an article was discarded because it was based on different approaches for prescribing sensory-directed real-time activity in dynamic environments because although closely related to the Teleo-Reactive approach, they fell outside the scope of this review. Of the 53 works selected, thirty-one articles were published in proceedings of international congresses and workshops and seven in journals of acknowledged prestige. We additionally identified a doctoral thesis, a book published by an international publisher and seven Technical Reports, all particularly relevant in terms of content.

There are considerable differences in the proportions of the categories analysed in the selected articles. In 23 out of the 53 the Teleo-Reactive formalism was used to develop artificial intelligence-related systems (category C5), the largest of all the categories at 43.4 % of cases. Following at a distance (14 out of 53) were articles in which the Teleo-Reactive paradigm was used as a reference for other paradigms (C6), accounting for 26.41 % of cases. Next (12 out of 53) were articles the original approach or some extension of which was used to develop reactive systems (C3)—22.64 % of cases. Not far behind (10 out of 53) are those articles where Nilsson's original formalism is extended (C1), accounting for 18.87 % of cases. The remaining categories (C4 and C2) account for smaller percentages (16.98 and 9.43 %, respectively). Obviously, the percentages add up to more than 100 because there are several articles that fit into more than one category.

To assess the quality of the data reported, it was decided to e-mail the first two authors of those publications where there were doubts about the classification, asking them to check or complete the information gathered. We received replies to six out of ten such requests. In some cases the reply furnished further details of the results. In general terms, there were no significant discrepancies between the data gathered from the publications and the information received by e-mail. Figure 1 shows the distribution of the works considered into the categories defined.

Table 4 List of categories identified by the review

RQ#	Description	References [first-authorYY]
C1	Teleo-Reactive program definition and formalization	Benson and Nilsson (1995), Hayes (2008), Dongol et al. (2010), Gubisch et al. (2008), Katz (1997), Lee and Durfee (1994), Mousavi and Broda (2003), Nilsson (1994, 2001), Zelek (1995)
C2	Platforms for Teleo-Reactive program validation/emulation	Gubisch et al. (2008), Hawthorne and Anthony (2009), Weiglhofer (2007), Russell et al. (2001), Soto et al. (2011)
C3	Development of reactive systems	Broda (2000), Broda and Hogger (2004a, 2010), Broda et al. (2009), De Paola et al. (2004), Di Fatta et al. (2003), Gubisch et al. (2008), Marinovic et al. (2010a), Rajan et al. (2010), Saigol et al. (2010), Zelek (1995), Sánchez et al. (2011)
C4	Methodological aspects of Teleo-Reactive programs inception	Broda and Hogger (2004a, 2010), Hawthorne and Anthony (2010), Katz (1998), Kochenderfer (2002, 2003), Mousavi and Broda (2003), Srinivasan (2002)
C5	Advances in artificial intelligence	Ali et al. (2009), Broda (2000), Broda and Hogger (2004a,b, 2005a,b, 2010), Choi and Langley (2005), Coffey and Clark (2006), Gordon and Logan (2003), Kochenderfer (2002, 2003), Kowalski and Sadri (2011), Könik et al. (2009), Langley and Choi (2006), Li et al. (2007), Nilsson (2000), Parmar (2002), Vargas (2008), Vargas (2009), Vargas and Morales (2009), Salomaki et al. (2005), Srinivasan (2002)
C6	Teleo-Reactive paradigm as reference for other paradigms	Ali et al. (2009), Choi and Langley (2005), Gamble and Riddle (2011), Hawthorne and Anthony (2009), Kowalski and Sadri (2011), Könik et al. (2009), Langley and Choi (2006), Li et al. (2007), Marinovic et al. (2010b), McGann et al. (2007, 2008), Payne (2008), Salomaki et al. (2005), Twidle et al. (2010)

5 Discussion

This section discusses the replies received to the questions that were initially put. To answer these questions the study analyses the works in each of the categories. Hence, RQ1 is answered by analysing categories C1, C2 and C4; RQ2 is answered by analysing category C5; RQ3 by categories C3 and C6; and finally, question RQ4 rounds up all the categories.

5.1 What advances have been derived from extensions to the original formalism proposed by Nilsson in 1994?

Since it was first defined by Nilsson, the formalism has been further enriched by various extensions endowing it with greater expressive and semantic power. As noted earlier, the

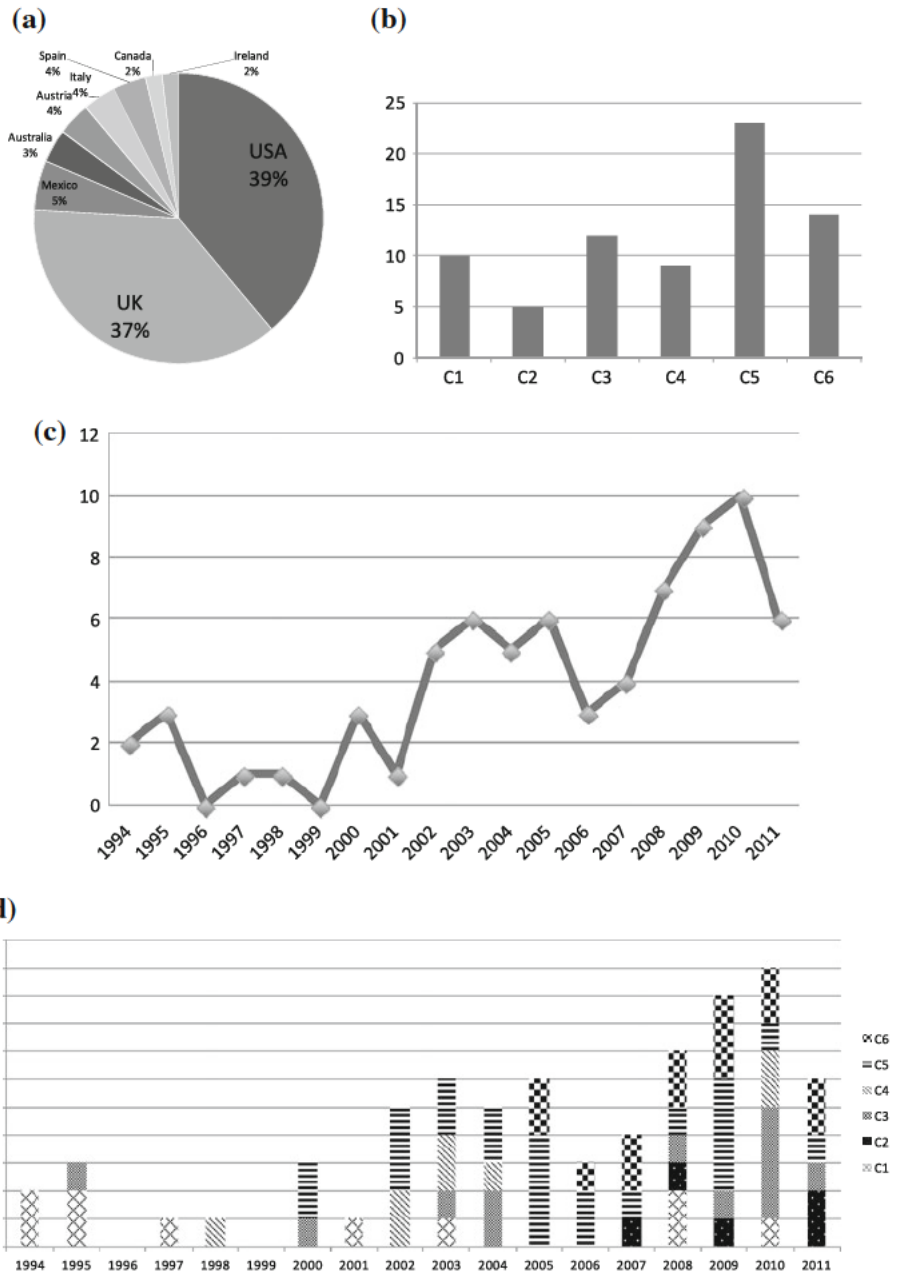


Fig. 1 Distribution of the works considered into the categories defined. **a** Distribution of works by country. **b** Distribution of works by category. **c** Number of works per publication year (2011 will be greater than showed). **d** Works by category per year

same author published a technical report in 1992 (Nilsson 1992) where the Teleo-Reactive formalism is really mentioned for the first time, albeit it was presented as "teleo-reactive trees" (the TR-program concept was not coined until the 1994 article). In that work

Nilsson explains what TR-trees are and even proposes a grammar for a TR programming language. The report details the equivalence between what it calls "teleo-reactive sequences" and electrical circuits and paves the way to a definition of these sequences using hierarchies and considering the invocation of parallel actions.

The most immediate field of application that is presented is Robotics and Artificial Intelligence (using the example of the "B-not-last" or BNL-problem where three blocks must be placed at several locations). In addition, it stresses the need to import many concepts from Control Theory into the discipline of "Computer Science". In general terms, both works constitute one and the same contribution, and therefore we decided to continue using the 1994 work published in an international journal (Nilsson 1994) as the benchmark for the approach and we did not include it in the set of papers selected.

The first work referencing the above two that was identified by the systematic review is Lee and Durfee (1994), from a national conference on Artificial Intelligence. This proposes a new circuit semantics (entitled *Structured Circuit Semantics*, SCS), which extends Nilsson's formalism and, according to its authors, addresses the following shortcomings of TR-programs:

- Frequency of the perception-cognition-action cycle: in formalisms based on the notion of circuit semantics (including TR-programs), ideally conditions are assessed and actions executed continuously. However, in real circuits the cycle frequency imposes a lag between the three phases, which does not entirely solve the problem, although the effect is minimized with high-speed frequencies. According to Lee and Durfee (1994), the problem with TR-programs lies in whether or not actions are atomically executed and the race conditions that may arise between the world model update and the updating resulting from execution of the perception phase. Therefore, in SCS conditions are checked only between atomic actions rather than continuously, providing developers with a clear semantics for the execution and feedback cycles.
- Impossibility of choosing actions non-deterministically: the order of the rules set in the design imposes rigidities that can affect the system's performance. This is one of the major contributions of the SCS formalism to TR-programs. For example, of two actions that can potentially be executed, it is the one with the higher priority that will be executed even although in terms of performance, in a particular context, it would be more appropriate or more useful to execute the other action. To deal with this, Lee and Durfee (1994) proposes a more flexible arrangement that uses a *decision layer* above the *circuit layer* where the system can choose which action is more appropriate and a choice can be made non-deterministically (by introducing the *do best*, *do first* and *do any* constructors).
- It would be useful to allow actions to feed back information about success or failure: in a TR-program, if an action fails without changing any of the rule conditions, the same action will be triggered until the action eventually succeeds. If the actions could feed back information on success or failure, the program could react to that information, for example following a standard exception-handling format.

The first study to produce tangible results on the original definition of the TR formalism was (Zelek 1995) in 1995. This work proposed what it called *TR+ formalism*, extending the basic syntax of TR-programs while furnishing an interpreter implemented on a PVM (Parallel Virtual Machine). With this implementation, it can be run in parallel and in real time on distributed and heterogeneous computers, as the authors demonstrated with several case studies of mobile robotic systems. The environment further provides a graphic user interface with capacity to develop TR+ programs and monitor execution of the program in real time. The syntactic extensions are basically: allowing logical expressions in the conditions, execution

of actions in parallel (using asynchronousness) or in sequence, and programmer-adjustable frequency rate of computation of the rule condition.

In the same year, Benson and Nilsson (1995) presented a study which constituted a significant extension of the original model. There, they presented an architecture and integrated subsystems providing facilities for the development of robust and flexible systems in dynamic environments based on autonomous agents. These facilities included selection from multiple competing goals and planning and learning of new TR sequences. In this context, according to the authors the TR-formalism has proven highly suitable because of the facility it provides for replanning during execution. As to the management of multiple competing goals, the architecture furnishes a "Plan Library" and a "Planning and Learning System" which are used to create new TR-programs and to modify existing ones. An "Arbitrator" decides which of the actions should be selected for execution using a method based on *rewards*. In addition to the obvious contribution that this study made to the field of Artificial Intelligence, the extension to the TR formalism offers the possibility of allowing subprograms to be executed in arbitrary order depending on circumstances at execution time. To do this, "conjunctive nodes" and "AND nodes" are defined. Each AND node is the root of a TR-subtree that achieves its condition without interacting with the other conditions in the conjunction. Thus, these TR-subtrees can be executed in whatever order environment conditions dictate. Moreover, since continuous computation by processing the entire TR-tree becomes impractical in many situations, they develop a heuristic method of action selection which usually produces the equivalent result.

Katz (1997) introduced a TR extension, called Fuzzy-Teleo-Reactive, where fuzzy logic is substituted for Boolean logic. Since the TR formalism is intended to embody a continuous monitoring process rather than a state transition model, a continuous-valued logic (more specifically, the Zadeh logic) is considered as the extension of the work. Here predicates have continuous value truth strengths and the argument for actions is the truth strength value of the corresponding condition. Moreover, they propose to eliminate the first true condition constraint since it allows several rules to be active, each to a different degree (as in fuzzy expert rule-based systems). The proposal allows for a more parallel evaluation capability and potentially more robust functionality while preserving the original continuous orientation of TR formalism.

In 2001, Nilsson published the next representative article for the TR-formalism (Nilsson 2001), where he introduced an architecture for linking perception and action in robots. The use of TR-programs is allocated to an "action power", perceptual rules are used in a "reception tower", and the predicates are kept faithful to the current environmental situation in a "model tower". Moreover, a Java-based program is provided to implement this triple-tower. While the paper is not actually an extension to the original formalism, it clearly contributes in the sense of consolidating it in both architectural and methodological terms, to the point where it serves as a reference for other authors.

The article published by Mousavi and Broda (2003) presents an algorithm for simplifying TR-programs which provides programs that are smaller but semantically equal to the given ones. This algorithm removes redundant rules (never executed) and redundant literals (whose removal does not affect the output of the TR-program) from the given TR-program. The purpose of this simplification is to obtain a smaller but semantically equal TR-program. The resulting TR-program will require less storage and will be more readable, and hence it can be processed faster and will give faster responses to stimuli. The article validates the application of the algorithm to various significant case studies, for all of which it indicates the percentage of simplification, with highly positive results.

One significant contribution to the TR-formalism is the possibility of carrying out concurrent actions, presented by Gubisch et al. (2008). Thanks to the grammar they have developed, it is possible to associate a single condition with several actions which are executed concurrently when the associated condition is fulfilled. The possibilities of this contribution can be verified with the compiler they have developed, available in Weiglhofer (2007).

The most recent contribution in this category comes from Dongol et al. (2010). The authors introduce a temporal logic over continuous intervals (called "durative temporal logic", a combination of duration calculus with linear temporal logic) which is used to formalise the semantics of TR-programs. In the article, the authors present a theory on reasoning about TR-programs, define durative temporal logic and provide syntax and semantics for TR-programs. The proposal provides a means of proving TR-program progress and correctness, which are judged on the basis of behaviour with respect to the environment in which the reactive system is operating. Earlier, in Hayes (2008) the same author presented a semantics and an advanced formal notation for TRPs in real time, which is the basis for Dongol et al. (2010).

Much less has been published on the creation or use of TR-program validation or simulation platforms. Nilsson used LISP in Nilsson (1992) to simulate TR-programs, but very few publications mention platforms of this kind. Among all the references considered there is a TR-program compiler for the C++ programming language. Gubisch et al. (2008) at the University of Grazhas have proposed a grammar for what they call *Extended Teleo-Reactive Programs* and implemented a compiler with which to translate programs following that grammar into C++-executable programs. The outstanding feature of this grammar is that several actions can be executed simultaneously through activation of one of the rules. The implementation of that compiler can be found in Weiglhofer (2007). Russell et al. (2001) describes the implementation of a multi-agent system using a special framework for agents called *Agent Factory*, which includes a specific kit for TR-agent development called *AF-TeleoReactive*. This provides support for the creation of agents based on the AF-TR agent-oriented programming language. Hawthorne and Anthony (2009) demonstrates a Java framework in which the programmer has to inherit Condition and Action classes to create conditions and associated actions.

In the field of educational applications (Soto et al. 2011) presents a novel hw/sw platform that has proven useful for students to both implement and validate the design of reactive systems using the Teleo-Reactive approach. The platform explores the possibility of implementing TR-programs through electrical circuits using reconfigurable hardware such as Field Programmable Gate Arrays (FPGA). Trials conducted with students show that the tool they have developed helps the students to understand the basics of reactive system development, allowing them to focus on robotics concepts and not just on programming and platform-dependent matters.

There are several studies reviewing methodological aspects for the definition of TR-programs. Particularly significant are those carried out by Kochenderfer and Srinivasan at Stanford University, where they used genetic algorithms to create TR-programs. Thus, Srinivasan (2002) produced TR-programs using stacking techniques. The proposal is an interesting one, but it has a drawback in that it does not preserve satisfaction of the regression property. Kochenderfer (2002) added the use of *indexicals*; these are context-dependent expressions with which much better results than (Srinivasan 2002) can be achieved—that is within the context of block stacking. Kochenderfer (2003) proposed a generalization of the solution of the Blocksworld problem to cover all hierarchical TR-programs, with better results than initially proposed by a human programmer. The formal framework for mobile agents developed by Krysia Broda at Imperial College (London) has also proven very effective in

the development of TR policies for mobile agents. Using a graph where the intersections are actions possible in a given state, which is represented by the graph nodes, it is possible to define such policies by evaluating these graphs using discounted reward techniques (Broda and Hogger 2004a), and even to evaluate the different policies applicable to an agent (Broda and Hogger 2010). In 1998 Katz proposed an algorithm to simplify the semantic circuits in both Nilsson's TR-programs and *Fuzzy-Teleo-Reactive programs* which he himself had introduced in Katz (1997). Mousavi and Broda (2003) presented an algorithm to simplify TR-programs by eliminating redundant rules or literals and formally demonstrated its effectiveness. Finally, Hawthorne and Anthony (2010); Hawthorne et al. (2011) contains a number of basic Software Engineering guidelines and practices for writing TR-programs.

5.2 What broad areas of study has this approach addressed?

Where the Teleo-Reactive formalism has contributed most is in the field of Artificial Intelligence (AI). Computational learning in particular is undoubtedly the field that has most benefited from the use of the formalism.

Nilsson himself noted in (Nilsson 2000) that the Teleo-Reactive formalism could be appropriate to represent control programs for robotic systems, both purpose-written and learned automatically. P. Srinivasan and M. Kochenderfer have used genetic algorithms to create new Teleo-Reactive Programs in relation to BlocksWorld. Kochenderfer (2002) improved on the results of Srinivasan (2002) by using expressions whose meaning is context-dependent, which the authors call *indexicals*. Kochenderfer (2003) generalized the results from BlocksWorld to all recursive and hierarchical TR-programs. Also important in this respect are the works of B. Vargas in the field of learning. The learning process proposed in Vargas (2008) and Vargas and Morales (2009) begins with a user-guided execution. Taking the low-level traces generated in the execution, ALEPH is used to learn basic TR-programs. These TR-programs are the basis for constructing hierarchical TR-programs using induction algorithms like FOSeq. Vargas's doctoral thesis (Vargas 2009) focused on learning of TR-programs for mobile robotics, building on and broadly contextualizing the findings in Vargas (2008) and Vargas and Morales (2009).

Articles relating to Teleoreactive logic programs (TLP) are of particular interest in the field of learning. TLPs are programs whose syntax is very similar to the Horn clauses in Prolog. They are goal-driven programs which make it possible to react to changes in the environment. They use two knowledge bases: *Concepts* (equivalent to the conditions in the rules of traditional TR-programs) and *Skills* (similar to TR-programs where certain actions are executed when certain primitive conditions or concepts are fulfilled). They are, then, a logical adaptation of traditional TR-programs. They first appeared in Choi and Langley (2005), who used them as a learning goal based on *Problem Solving*. The article presents a problem solver that links up primitive skills to deal with new tasks and an associated learning method that compiles these solutions in executable programs. Langley and Choi (2006) proposed a modification making it possible to learn recursive TLPs. Salomaki et al. (2005) proposed a method for learning TLPs through observation: an expert solves a problem, leaving traces of the primitive skills used, and more complex skills can be learned from these traces. Li et al. (2007) added priorities to the goals of the TLPs. Könik et al. (2009) proposed a new algorithm for learning by observation that combines inductive and analytical learning. Lastly, Ali et al. (2009) used TLPs to demonstrate that in multi-agent environments where training examples are very costly, purely theoretical knowledge should be reviewed with specific knowledge of the field using a training agent.

Leaving aside the field of learning, we should draw attention to the work of K. Broda's team at Imperial College (London). Taking as base the formal framework for synthesizing Teleo-Reactive robot control programs presented in Broda (2000), they have carried out a series of studies for the design and assessment of Teleo-Reactive policies in multi-agent environments based on analysis and manipulation of situation graphs. The intersections in these graphs represent the possible actions that may be taken from a given node, which in turn represents the environment and the agent's perception of the environment. For instance, Broda and Hogger (2004a) presents a method for designing individual agents by means of discounted-reward evaluation of certain sub-graphs produced by applying certain policies to these situation graphs. Broda and Hogger (2004b) and Broda and Hogger (2005a) showed the utility of situation graphs to predict near-optimal policies for groups of cloned robots by observing the individual behaviour of each one in response to events caused by the others. Broda and Hogger (2005b) described a method for constructing and assessing TR policies for one or more agents based on discounted reward evaluation of restricted sub-graphs from complete situation graphs. The plethora of combinations that could emerge from an evaluation of all the possible sub-graphs is attenuated by using abstractions. Broda and Hogger (2010) adds the possibility of the agents carrying out cooperative tasks by sharing perceptions and offers a means of evaluating the possible policies associated with an agent.

Finally, there are a number of papers dealing with other aspects of artificial intelligence. Kowalski and Sadri (2011) shows how TR-programs can be embedded in a high-level framework for ALPAs (abductive logic programming agents) to extend the traditional logical programming so that some predicates can be left undefined. Parmar (2002) proposes a logical measurement of progress towards the goal in heuristic planners, from which it might be possible to derive TR-programs indirectly. Gordon and Logan (2003) used TR-programs to create an architecture that would enable games agents to define new goals in the light of the state of play. And lastly, Coffey and Clark (2006) presented a hybrid architecture that combines TR-programs and BDI (beliefs, desires, intentions) with a view to graduating the transition from the cognitive to the behavioural layer.

5.3 What fields have benefited most from the approach and what have been the outcomes there?

Although the idea of the Teleo-Reactive formalism is in principle intended for application to reactive systems where the factors determining the system's behaviour are sensorization of the environment and decisions based on perceived changes, from the outset the fact that Nilsson himself focused chiefly on mobile robotic systems has meant that most of the work on this approach has been to do with the development of such systems.

Among the articles considered we have found several examples of the use of the TR paradigm in the development of reactive mobile systems. For instance, Gubisch et al. (2008) used the C++ TR-program compiler developed at the University of Graz to implement the architecture described in their own paper on RoboCup middle-size soccer robots. However, the authors do no more than report promising results with robots following implementation. Rajan et al. (2010) and Saigol et al. (2010) used the T-REX architecture presented in McGann et al. (2007) for implementation of the control system for autonomous underwater vehicles (AUV). This system enables the vehicle to survey areas in more detail if biogeochemical markers indicate the presence of a target feature, and even to follow dynamic ocean phenomena such as fronts. Zelek (1995) used an extension of TR-programs called TR+ to implement behaviour of mobile robots exploring a given environment. The authors stated that the robot navigated successfully around newly discovered obstacles to different goal

locations within the experimental environment. The work of Krysia Broda's Group at Imperial College London (Broda 2000; Broda and Hogger 2004a, 2010) has proven highly useful for the development of individual or collaborative multiagent (and not necessarily mobile) systems, offering frameworks for policy creation, means of assessing these and simulations of the systems they have designed. One of their most interesting products in this field is the SAGE (Sense, Abduce, Gather, Execute) environment control and monitoring system presented in Broda et al. (2009), which provides a flexible, distributed, open and component-based approach to environmental monitoring and control. Its computational processes reflect natural, multi-stage, collaborative human reasoning: when events are detected it forms and tests hypotheses about these before reacting appropriately, using a TR policy. Its multi-agent and multi-threaded architecture allows it to form and act upon different hypotheses concurrently. At the time of writing this paper work on SAGE is at the specification stage, and no further information about the implementation has been found. Obviously the Teleo-Reactive paradigm makes it possible to create reactive systems, but not necessarily mobile ones. Examples of these systems can be found in De Paola et al. (2004) and Di Fatta et al. (2003), who have used TR agents in a multiagent environment for network management tasks, or in Marinovic et al. (2010a), who propose the use of TR-programs in the field of medical workflows. In this sphere, the intervention of human agents (doctors, nurses, patients, etc.) generally produces unpredictable reactions. TR-programs offer flexible means of dealing with such unforeseen changes not found in traditional methods of workflow specification, such as Petri nets or structured languages like BPEL.

From a more general viewpoint, Sánchez et al. (2011) presents a systematic method for deriving architectural models with structural and behaviour descriptions using TR-programs, combining two approaches: on the one hand the Teleo-Reactive approach, oriented towards describing the system in terms of goals and the state of the environment; and on the other hand an architectural approach oriented towards component-based design. This method is part of an environment in which code can be generated by means of model-model transformations, paving the way for an integrated approach to the development of reactive systems. The article also reports an experiment carried out with students which demonstrates the simplicity and efficiency of the method.

The Teleo-Reactive paradigm has served as inspiration in environments other than the ones for which it was originally conceived. In Payne (2008), for example, we find the bases of a specification language for dynamic reconfiguration policies in in component-based systems. This language has a syntax very like that of TR-programs. In Marinovic et al. (2010b) it is asserted that the flexibility provided by the TR paradigm can be very useful in specifying policies for human agents given the unpredictability of their behaviour. Twidle et al. (2010) implemented these policies using Ponder2, the policy environment developed at Imperial College (London). Gamble and Riddle (2011) analysed the characteristics that a good policy definition language ought to have, taking as example, among others, the TR-program definition language. Hawthorne and Anthony (2009) proposed using the TR paradigm as a basis for the creation of self-correcting applications. Based on the properties of TRPs and the way in which they deal with unexpected situations, they assert that designing applications following this paradigm can greatly facilitate the correction of errors, including errors not foreseen at the time of designing. And we should not forget the work done on T-REX at the Monterey Bay Aquarium Research Institute (McGann et al. 2007, 2008) which consists of an architecture that makes it possible to create object-oriented agents with embedded automatic planning and adaptive execution. Although it does not apply the TR paradigm in the strictest sense, the paradigm has obviously had a profound influence on this architecture. Likewise, neither the TLPs (TeleoReactive logic programs) used in Choi and Langley (2005), Salomaki et al.

(2005), Langley and Choi (2006), Li et al. (2007), Könik et al. (2009), Ali et al. (2009)) nor the ALP (Abductive Logic Programming) agents mentioned in Kowalski and Sadri (2011) can be considered direct applications of the TR paradigm, but rather adaptations to environments that are more oriented towards logical formalisms.

5.4 What are the main challenges and difficulties that have to be overcome for successful adoption of the approach? What areas of research around the approach should be encouraged?

The works referenced here clearly highlight the significant progress that has been achieved in the fields of AI and Robotics. One token of this is the abundance of works in these fields and the large growth in numbers of publications in the last four years. The foregoing survey suggests a number of challenges or fields of study that we believe need to be resolved or promoted to usefully extend the Teleo-Reactive approach:

1. *Code generation.* More proposals are needed to achieve generation of executable code that can be reused as part of systems development. Most of the solutions we found furnish a TR-program interpreter which in no case can be reused in a more general software development context. In the same vein, it would be desirable to have TR-program validation tools to provide designers with a rapid means of emulating the conditions of the environment and the simulation results. The work that comes closest in this sense is the one published in Soto et al. (2011).
2. *TR-program catalogue.* TR-program catalogues should be supplied that can be reused by applying the subgoal concept. A modular breakdown of TR-goals would facilitate the development of TR-programs through reuse, so as to be able to take advantage of prior experience in TR-program validation.
3. *Synergies with Software Engineering.* More progress is needed in integrating the TR-approach into the field of Software Engineering. A systematic approach where architectural models with structural descriptions and behaviour could be derived from TR-programs would be particularly useful. In Sánchez et al. (2011) the authors demonstrate that the development of reactive systems can derive very significant benefits if the Teleo-Reactive approach could be made compatible with the architectural approach, which is oriented towards the design of component-based software. Integration of this work into a development environment that allows code to be generated via model-to-model transformations would open up new possibilities in the development of this type of systems.

In short, the focus should be on the adoption of Software Engineering tools, techniques and methods to improve the development of reactive systems using the Teleo-Reactive approach. Within this focus the most promising area may be synergies with model driven software development (MDSD) (Selic 2003) and with component based software development (CBSD) (Szyperski 2002). The first of these furnishes techniques for modelling and code generation from TR-programs. The second provides architectural mechanisms to promote reuse and integration with existing frameworks for the development of real-time systems. Consequently, it will favour "design for reuse" since it is possible to guide the cataloguing of components by using the specification provided in the TR-programs; and it will also favour "design from reuse" since it is more direct for the developer to use the catalogue to identify which components can be adapted to the desired behaviour.

6 Conclusions

The studies reviewed in this survey show that the Teleo-Reactive approach is an important technological breakthrough for developing reactive systems. From this survey it is fair to conclude that the Teleo-Reactive approach can be tackled from different points of view given the number of areas of research in which it can be used, from the more formal oriented towards the demonstration of properties, to the more practical applied to the implementation of robotic systems. With the systematic approach we have been able to assess and aggregate research outcomes to achieve an objective summary of research results for the Teleo-Reactive approach. The results indicate that nowadays Teleo-Reactive implementations for reactive system development are still a highly promising subject of study. Although research on the systematic use of this formalism is still in its infancy, it points to several exciting challenges which require further interdisciplinary collaboration, particularly with the Software Engineering area. As a well-defined and methodological way of summarizing evidence concerning the Teleo-Reactive paradigm, this literature review offers comprehensive background that will allow readers to identify new research challenges.

Acknowledgments The authors wish to thank the Fundación Séneca of the Murcia Region (research project MISSION with ref. 15374/PI/10) and the Ministry of Education and Science (research project EXPLORE with ref. TIN2009-08572), Spain, for partially supporting this work. They also wish to thank N. Dulay (Imperial College, London) for his explanations about the use of TR-programs in implementing workflow-based and human-centric pervasive systems.

References

Ali K, Leung K, Konik T, Choi D, Shapiro D (2009) Knowledge-directed theory revision. In: Proceedings of ILP-09

Benson S, Nilsson NJ (1995) Reacting, planning, and learning in an autonomous agent. In: Furukawa K, Michie D, Muggleton S (eds) Machine intelligence, vol 14. The Claredon Press, Oxford

Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M (2007) Lessons from applying the systematic literature review process within the software engineering domain. J Syst Softw 80:571–583

Broda K (2000) Constructing Teleo-Reactive robot programs. In: Proceedings of ECAI-00

Broda K, Hogger CH (2004a) Designing and simulating individual Teleo-Reactive agents. In: Proceedings of KI-04

Broda K, Hogger CH (2004b) Policies for cloned teleo-reactive robots. In: Proceedings of MATES-04

Broda K, Hogger CH (2005a) Determining and verifying good policies for cloned Teleo-Reactive agents. Int J Comput Syst Sci Eng 20:249–258

Broda K, Hogger CH (2005b) Abstract policy evaluation for reactive agents. In: Proceedings of SARA-05

Broda K, Hogger CJ (2010) Designing effective policies for minimal agents. Comput J 53:1184–1209

Broda K, Clark K, Miller, R, Russo A (2009) SAGE: a logical agent-based environment monitoring and control system. In: Proceedings of AmI-09

Choi D, Langley P (2005) Learning teleoreactive logic programs from problem solving. In: Proceedings of ILP-05

Coffey S, Clark K (2006) A hybrid, Teleo-Reactive architecture for robot control. In: Proceedings of MARS-06

De Paola A, Fiduccia S, Gatani L, Pizzitola A, Storniolo P (2004) Introducing automated reasoning in network management. In: Proceedings of ICAISC-04

Di Fatta G, Gaglio S, Presti G, Re G, Selvaggio I (2003) Distributed intelligent management of active networks. In: Proceedings of AI*AI-03

Dongol B, Hayes IJ, Robinson PJ (2010) Reasoning about real-time Teleo-Reactive programs. Technical Report SSE-2010-01, Division of Systems and Software Engineering Research, The University of Queensland

Gamble C, Riddle S (2011) Dependability explicit metadata: extended report on properties, policies and exemplary application to case studies. Technical Report CS-TR-1248, The Newcastle University

Gordon E, Logan B (2003) A goal processing architecture for game agents. In: Proceedings of AAMAS-03

Gubisch G, Steinbauer G, Weiglhofer M, Wotawa F (2008) A Teleo-Reactive architecture for fast, reactive and robust control of mobile robots. In: Proceedings of IEA/AIE-08
Hawthorne J, Anthony R (2009) Using a Teleo-Reactive programming style to develop self-healing application. In: Proceedings of ICST-09
Hawthorne J, Anthony R (2010) A methodology for the use of the Teleo-Reactive programming technique in autonomic computing. In: Proceedings of SND/ACIS-10
Hawthorne J, Anthony R, Petridis M (2011) Improving the development process for Teleo-Reactive programming through advanced composition. In: Proceedings of ICAS-11
Hayes IJ (2008) Towards reasoning about Teleo-Reactive programs for robust real-time systems. In: Proceedings on SERENE08
Katz EP (1997) Extending the Teleo-Reactive paradigm for robotic agent task control using Zadehan (Fuzzy) logic. In: Proceedings of CIRA-97
Katz EP (1998) A simplifying diagrammatic representation of crisp and fuzzy Teleo-Reactive semantic circuitry for application in robotic agent task control. In: Proceedings of the SMC-98
Kitchenham BA (2004) Procedures for undertaking systematic reviews. Joint Technical Report TR/SE-0401, Computer Science Department, Keele University and National ICT Australia Ltd
Kochenderfer M (2002) Evolving Teleo-Reactive programs for block stacking using indexicals through genetic programming. Book chapter: genetic algorithms and genetic programming at Stanford Bookstore
Kochenderfer M (2003) Evolving hierarchical and recursive teleo-reactive programs through genetic programming. In: Proceedings of EuroGP-03
Könik T, Negin N, Ugur K (2009) Inductive generalization of analytically learned goal hierarchies. In: Proceedings of ILP-09
Kowalski R, Sadri F (2011) Teleo-Reactive abductive logic programs. Technical Report. The Imperial College London
Lamsweerde A (2009) Requirements engineering: from goals to UML models to software specifications. Wiley, England
Langley P, Choi D (2006) Learning recursive control programs from problem solving. J Mach Learn Res 7:493–518
Lee J, Durfee EH (1994) Structured circuit semantics for reactive plan execution systems. In: Proceedings of AAAI-94, 1232-1237
Li N, Choi D, Langley P (2007) Adding goal priorities to Teleoreactive logic programs. In: Proceedings of the international symposium on skill science
Marinovic S, Twidle K, Dulay N (2010a) Teleo-Reactive workflows for pervasive healthcare. In: Proceedings of PerCom-10
Marinovic S, Twidle K, Dulay N, Sloman M (2010b) Teleo-Reactive policies for managing human-centric pervasive services. In: Proceedings of CNSM-10
McGann C, Py F, Rajan K, Thomas H, Henthorn R, McEwen R (2007) T-REX: a model-based architecture for AUV control. In: Proceedings of ICAPS-07
McGann C, Py F, Rajan K, Thomas H, Henthorn R, McEwen R (2008) A deliberative architecture for AUV control. In: Proceedings of ICRA-08
Mousavi SR, Broda K (2003) Simplification Of Teleo-Reactive sequences. Technical Report, Imperial College London
Nilsson NJ (1992) Toward agent programs with circuit semantics. Tech. Report STAN-CS-92-1412, Dept- of Computer Science, Stanford University
Nilsson NJ (1994) Teleo-Reactive programs for agent control. J Artif Intell Res 1:139–158
Nilsson NJ (2000) Learning strategies for mid-level robot control: some preliminary considerations and experiments. Tech. Report, Stanford University
Nilsson NJ (2001) Teleo-Reactive programs and the triple-tower architecture. Electron Trans Artif Intell 5:99–110
Parmar A (2002) A logical measure of progress for planning. In: Proceedings of AAAI-02, 498-505
Payne RJ (2008) RPL: a policy language For dynamic reconfiguration. In: Proceedings of SERENE-08
Rajan K, Py F, McGann C (2010) Adaptive control of AUVs using onboard planning and execution. Sea Technology Magazine
Russell SE, Carr D, Dragone M, O'Hare GM, Collier RW (2011) From bogtrotting to herding: a UCD perspective. Ann Math Artif Intell 61:349–368
Saigol Z, Py F, Rajan K, McGann C, Wyatt J, Dearden R (2010) Randomized testing for robotic plan execution for autonomous systems. In: Proceedings of IEEE/OES-10
Salomaki B, Choi D, Nejati N, Langley P (2005) Learning Teleoreactive logic programs by observation. In: Proceedings of AAAI-05

Sánchez P, Alonso D, Morales JM, Navarro PJ (2011) From Teleo-Reactive specifications to architectural components: a model-driven approach, Technical Report ref. DT-EXPLORE-05.rev0, Universidad Politécnica de Cartagena

Selic B (2003) The pragmatics of model-driven development. IEEE Trans Softw Eng 20:19–25

Soto F, Sánchez P, Mateo A, Alonso D, Navarro PJ (2011) An educational tool for implementing reactive systems following a goal-driven approach. Technical Report ref. DT-EXPLORE-06.rev0, Universidad Politécnica de Cartagena

Srinivasan P (2002) Development of block-stacking Teleo-Reactive programs using genetic programming. Book chapter: genetic algorithms and genetic programming at Stanford Bookstore

Szyperski C (2002) Component software: beyond object-oriented programming. Addison-Wesley, Reading

Twidle K, Marinovic S, Dulay N (2010) Teleo-Reactive policies in Ponder2. In: Proceedings of POLICY-10

Vargas B (2008) Solving navigation tasks with learned Teleo-Reactive programs. In: Proceedings of IROS-08

Vargas B (2009) Aprendizaje de Programas Teleo-Reactivos para Robótica Móvil. PhD Thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica

Vargas B, Morales EF (2009) Learning navigation Teleo-Reactive programs using behavioural cloning. In: Proceedings of ICM-09

Weiglhofer M (2007) Extended Teleo-Reactive compiler. Online available at http://www.ist.tugraz.at/staff/weiglhofer/projects/trcompiler/index.html

Zelek JS (1995) Teleo-Reactive autonomous mobile navigation. In: Proceedings of CCECE-95

A CONTROLLED EXPERIMENT TO EVALUATE THE UNDERSTANDABILITY OF KAOS AND I* FOR MODELING TELEO-REACTIVE SYSTEMS

RESUMEN

Las especificaciones Teleo-Reactivas permiten a los ingenieros definir el comportamiento de sistemas reactivos teniendo además en cuenta sus objetivos y los cambios que se producen en el estado del entorno. Este artículo evalúa dos notaciones de ingeniería de requisitos orientada a objetivos, i* y KAOS, para determinar su nivel de comprensibilidad a la hora de especificar sistemas Teleo-Reactivos. Para ello se llevó a cabo un experimento controlado en el que participaron dos grupos de estudiantes de grado. Cada uno de esos grupos analizó en primer lugar el modelo de requisitos de un robot móvil especificado mediante uno de los lenguajes a evaluar, rellenando a continuación un cuestionario para evaluar su comprensibilidad. Después, cada grupo procedió de forma análoga con el modelo de otro sistema especificado con el otro lenguaje evaluado. El análisis estadístico de los datos obtenidos por medio de este experimento mostró que la comprensibilidad de i* es mayor que la de KAOS cuando se modelan sistemas Teleo-Reactivos. El estudio demuestra que ambos lenguajes pueden ser usados para especificar sistemas Teleo-Reactivos, aunque los resultados sugieren que las notaciones deberían ser especializadas para maximizar la comprensibilidad de las especificaciones. i* supera a KAOS en términos de comprensibilidad debido a dos razones principales:

1. Los modelos i* representan dependencias entre agentes y objetivos o tareas.
2. Las diferencias notacionales entre tareas y objetivos en i* son más evidentes que las que hay entre objetivos y requisitos en KAOS.

The Journal of Systems and Software 100 (2015) 1–14

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A controlled experiment to evaluate the understandability of KAOS and *i** for modeling Teleo-Reactive systems

José Miguel Morales[a], Elena Navarro[b,1], Pedro Sánchez[a,*], Diego Alonso[a]

[a] Systems and Electronic Engineering Division (DSIE), Universidad Politécnica de Cartagena, Campus Muralla del Mar s/n, Cartagena, Spain
[b] LoUISE Research Group, Computing Systems Department, University of Castilla-La Mancha, Avda. España s/n, 02071 Albacete, Spain

ARTICLE INFO

Article history:
Received 24 March 2014
Received in revised form 2 October 2014
Accepted 4 October 2014
Available online 14 October 2014

Keywords:
Teleo-Reactive
Requirements engineering
Controlled experiment

ABSTRACT

Context: Teleo-Reactive (TR) specifications allow engineers to define the behavior of reactive systems while taking into account goals and changes in the state of the environment.
Objective: This article evaluates two different Goal Oriented Requirements Engineering notations, *i** and KAOS, to determine their understandability level for specifying TR systems.
Method: A controlled experiment was performed by two groups of Bachelor students. Each group first analyzed a requirements model of a mobile robotic system, specified using one of the evaluated languages, and then they filled in a questionnaire to evaluate its understandability. Afterwards, each group proceeded similarly with the model of another system specified with the second language.
Results: The statistical analysis of the data obtained by means of the experiment showed that the understandability of *i** is higher than that of KAOS when modeling TR systems.
Conclusion: Both languages are suitable for specifying TR systems although their notations should be specialized to maximize the understandability attribute. *i** surpasses KAOS due to two main reasons: *i** models represent dependencies between agents and goals or tasks; and notational differences between tasks and goals in *i** are more evident than those between goals and requirements in KAOS.
© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Reactive systems, as event-driven systems that react to external/internal stimuli, possess specific characteristics that make them particularly sensitive to the architectural decisions that are made during the course of their construction. Issues such as task planning, concurrency management, fault tolerance, and distributed communication need to be addressed as far as possible through the solutions that the literature has shown to be most suited to systems of this kind.

Some authors (Lamsweerde, 2001; Yu and Mylopoulos, 1998) argue that software development for this kind of systems should benefit from an approach driven by what the user wishes to obtain from it, i.e. its goals, rather than how it is constructed. In other words, goals should drive the development process. We opted for the Teleo-Reactive (TR) paradigm (Nilsson, 1993), a goal-oriented approach for modeling systems that manages their actions, outputs and status in response to stimuli coming from either inside or outside the system. The TR approach thus offers engineers a high-level goal-oriented formalism for the development of reactive systems that allows them to define the system behavior while taking into account goals and changes in the state of the environment. As detailed below in Section 3, a TR specification is defined as a set of prioritized condition/action rules. Conditions are a set of inputs either sensory or coming from a model of the world, while actions change the world or the model of the world. A TR specification interpreter should constantly re-evaluate the triggering condition set for each rule, and should execute the action corresponding to the highest-priority rule with a satisfied condition. Actions may also be TR specifications themselves, thus allowing the definition of hierarchical TR programs. This kind of nested TR specifications can be considered as system subgoals. The root of the hierarchy of a TR program is called the goal of the *system-to-be*.

One representative instance of reactive systems is the mobile robotics domain, where the TR paradigm has provided very interesting results (Ranjan et al., 2010; Gubisch et al., 2008; Broda et al., 2000), since TR-based systems inherently recover from errors and unexpected events whilst proceeding toward their goals. This domain embraces applications in which robotic vehicles work

* Corresponding author.
E-mail addresses: josemiguel.morales@upct.es (J.M. Morales), Elena.Navarro@uclm.es (E. Navarro), pedro.sanchez@upct.es (P. Sánchez), diego.alonso@upct.es (D. Alonso).
[1] Tel.: +34 967 599 200x2624; fax: +34 967 599 343.

http://dx.doi.org/10.1016/j.jss.2014.10.010
0164-1212/© 2014 Elsevier Inc. All rights reserved.

autonomously in non-structured environments. The reuse of existing solutions in current robotic systems is mostly focused on algorithms, and has not usually taken place at a higher abstraction level. The growing sophistication of robotic applications has led to the need for more sophisticated languages and platforms with which to specify, implement and validate them. To overcome this situation, we introduced a Model-Driven Software Development approach (Atkinson and Kühne, 2003; Bézivin, 2005) that separated the component-based description of real-time applications from their possible implementations on different platforms. This separation of concerns was supported by the automatic integration of the code obtained from the input architectural models (basically, UML component diagrams) into executable object-oriented frameworks. More recently, we introduced a systematic approach (Sánchez et al., 2012) that made it possible to automatically derive architectural models (including both structural and behavioral descriptions) from TR specifications. Thus, the development of reactive systems benefited significantly from the integration of the TR and architectural approaches for the design of component-based software. This integration was demonstrated with a development environment that supports the generation of code via model transformations, opening up new possibilities in the development of reactive systems.

In spite of the suitability of the TR formalism for specifying reactive systems, and more specifically robotic systems, developers perceive the system mainly as a set of prioritized condition/action rules. The development of these rules is an error prone task because it relies on mostly the expertise of the developers and their understanding of the requirements. When constructing TR specifications, the main causes of errors are the following: (1) minimal changes in the specification of the priorities produce huge execution errors, (2) it is not straightforward to ensure that each condition of a rule is a regression of some higher (priority) condition, that will ultimately lead the system to achieve its goal, and (3) it is difficult to observe the system encapsulating details or subgoals because the textual representation of the specification cannot be easily fragmented. Other inherent limitations of the TR formalism is that reuse cannot be easily achieved, since the paradigm does not offer clear resources for supporting both 'design for reuse' (for example, by means of catalogs of goals defined at different levels of abstraction), and 'design from reuse' (by using the previous catalog to identify which goals can be adapted to the behavior of the software-to-be), closer to the problem domain. All these limitations lead us to conclude that an approach that facilitates a Requirements Engineering (RE) approach for eliciting, specifying, and analyzing goals for TR specifications, i.e., that provides mechanisms for reasoning about the specifications, is needed. It is also needed for facilitating the process of evaluating alternatives of the system-to-be (Navarro et al., 2007; Chung et al., 2000).

Considering the definition of the TR formalism, it is evident that the most adequate RE technique for modeling TR systems is the Goal-Oriented approach (Hawthorne et al., 2011). Goal-oriented approaches mainly focus on the question "why" of the system-to-be. They do not only consider the deterministic aspects of the system-to-be, but also the non-deterministic aspects, such as non-functional requirements or evaluation of different alternatives for achieving a goal. In this article, we focus on the deterministic part of GORE approaches, leaving the non-deterministic part for future work.

Different modeling languages and methodologies have been defined for eliciting, specifying, and analyzing goals (see Kavakli and Loucopoulos, 2005; Pohl, 2010 for exhaustive surveys), although they all share concepts such as *goal*, *agent* and many refinement relationships (AND, OR, XOR, etc.). KAOS (Lamsweerde, 2001) and *i** (Yu, 1997) are the two most well known proposals in the field. KAOS comprises six complementary submodels (goal, obstacle, object, agent, operation, and behavior model), interrelated via traceability links. *i** defines a software development process that guides developers throughout several phases from early requirements to detailed design.

The objective of this work is to evaluate which of the considered requirements languages, KAOS or *i**, has a better understandability level when modeling requirements for TR-based systems. This evaluation has been performed by means of a family of experiments conducted according to Kitchenham et al.'s (2002) guidelines (Kitchenham et al., 2002), as a controlled experiment with a 2×2 factorial design, involving students of Industrial Engineering and Telecommunications Engineering from the Technical University of Cartagena (Spain). The students were asked to fill in two understandability questionnaires just after analyzing the requirements models of two mobile robotic systems, similarly to the approach followed by Teruel et al. (2012, 2011a). Both requirements models were specified using KAOS and *i**.

Understandability has been the criteria analyzed in the controlled experiment because several international standards have recognized its importance in the software development process. The International Organization for Standardization (ISO) defines understandability as "the capability of the software product to enable the user understand whether the software is suitable, and how it can be used for particular tasks and conditions of use" (ISO/IEC, 2001). Besides, the IEEE830-1998 standard (IEEE Computer Society, 1998) considers an understandable specification as one of the means to decrease later redesign, recoding, retesting, etc., being one of the five quality characteristics that a requirements model must have. Therefore, in the context of TRs, we are evaluating which language, KAOS or *i**, offers higher capability to enable engineers understand (i) which inputs (either sensory or coming from a model of the world) are inputs for the TR system, and (ii) which actions are to be executed to achieve the goals of the TR system. Consequently, a TR specification that is easily understandable by all stakeholders will help in developing the right software and the software right. As a side effect of the experiment we have identified some additional concepts that should be considered for extending both KAOS and *i** in order to better specify TR programs.

This article is structured as follows. Section 2 presents several works related to TR systems development and Software Engineering techniques. Next, Section 3 introduces the TR paradigm along with an illustrative example. In Section 4, KAOS and *i** are briefly introduced and some directives for using them to specify TR systems are given. Section 5 describes how the controlled experiment has been carried out as well as the main results, while the threats to the validity of the experiment are analyzed in Section 6. Finally, our main conclusions and future work are described in Section 7.

2. Related work

The Teleo-Reactive paradigm has produced highly interesting results, particularly in the domain of robotics. Teleo means *to bring to an end* or *to achieve a goal*. Since the TR formalism was first defined by Nilsson (Nilsson, 1993, 1992), it has been further enriched by various extensions endowing it with greater expressive and semantic power. In Morales et al. (2012), the authors review several works related to this paradigm in order to evaluate the progress and the outlook for further research on the development of reactive systems.

The most immediate field of application of TR paradigm is Robotics and Artificial Intelligence. Lee and Durfee wrote the first work referencing the Nilsson formalism (Lee and Durfee, 1994), from a conference on Artificial Intelligence, where a new circuit semantics (entitled "Structured Circuit Semantics") was introduced. The first study to produce tangible results on the original definition of the TR formalism was presented by Zelek (Zelek,

1995) in 1995. This work (called "TR+ formalism") extended the basic syntax of TR programs while providing an interpreter implemented on a parallel virtual machine. As the author demonstrated in several case studies of mobile robotic systems, by means of this implementation TR+ programs could be run on distributed and heterogeneous computers.

In 2001, Nilsson published the next representative article for the TR formalism (Nilsson, 2001), where he introduced an architecture for linking perception and action in robots. The use of TR programs is allocated to an "action tower", perceptual rules are used in a "perception tower", and the predicates are kept faithful to the current environmental situation in a "model tower". A Java-based program was also provided to implement this triple-tower.

Mousavi and Broda (2003) presented an algorithm for simplifying TR programs providing smaller but semantically equal programs. The resulting TR program requires less storage, is more readable, and, hence, it can be processed faster. This work validated the approach with various significant case studies, for all of which it indicated the percentage of simplification, showing highly positive results. Another significant contribution to the TR-formalism is the possibility of carrying out concurrent actions, as described in Gubisch et al. (2008). Thanks to the grammar they developed, it is possible to associate a single condition with several actions, which are executed concurrently when the associated condition is fulfilled. A more recent contribution in this field (Dongol et al., 2010) is a temporal logic over continuous intervals called "durative temporal logic" introduced to formalize the semantics of TR programs.

Not much work has been published on the creation or use of TR program validation or simulation platforms. Nilsson used LISP to simulate TR programs (Nilsson, 1992), but very few publications mention platforms of this kind. Among all the references considered, Gubisch et al. presented in Gubisch et al. (2008) a TR program compiler (available from Weighofer, 2007) for the C++ programming language. In the field of educational applications Soto et al. (2011) presented a novel HW/SW platform useful for students to both implement and validate the design of reactive systems using the TR approach. This platform explores the possibility of implementing TR programs through electrical circuits using reconfigurable hardware, mainly Field Programmable Gate Arrays (FPGAs).

There are several studies reviewing methodological aspects for the definition of TR programs. Particularly significant is that presented in Kochenderfer (2003), where genetic algorithms are used to create TR programs. The formal framework for mobile agents developed by Broda and Hogger (2004) has also proven to be very effective in the development of TR policies for mobile agents. Using a graph (where nodes represent states and the node intersections are actions possible in a given state) it is possible to define such policies by evaluating these graphs using discounted reward techniques.

With regards to Software Engineering solutions, some works, such as (Hawthorne et al., 2011; Hawthorne and Anthony, 2010), contain a number of basic guidelines and practices for writing TR programs. Besides, Sánchez et al. (2012) present a systematic method for deriving architectural models with structural and behavior descriptions using TR-programs. This method is part of a tool chain in which code can be generated by means of model-to-model transformations, paving the way for an integrated approach to the development of reactive systems.

As a summary, it can be concluded that the TR formalism has mostly contributed in the field of Artificial Intelligence, being computational learning undoubtedly the field most benefited from its use (e.g., Choi and Langley, 2005; Langley and Choi, 2006). It can also be concluded from the large growth of number of publications during the last years, that the TR formalism is an active research area in the software community. Despite these achievements, a number of challenges still remain to be solved in order to extend the TR approach such as code generation or TR program cataloging, as well as identifying synergies with the Software Engineering field.

With regards to code generation, most of the solutions (Gubisch et al., 2008; Nilsson, 1992; Weighofer, 2007) provide a TR program interpreter, which in no case can be reused in a more general software development context. Thus, further proposals are needed to achieve generation of executable code that could be reused as part of systems development. As regards design by reuse, TR program catalogs should be supplied. A modular breakdown of TR goals would facilitate the development of TR programs through reuse, so as to be able to take advantage of prior experience in TR program validation. Furthermore, additional progress is needed in order to get a better integration of the TR approach into the field of Software Engineering. The integration of the TR paradigm into a development environment that allows code to be generated via model-to-model transformations opens up new possibilities in the development of reactive systems. The state of the art indicates that nowadays the definition of methods and techniques for the specification of TR systems are a highly promising subject of study. Literature points to several exciting challenges which require further collaboration with other research areas, particularly with Software Engineering, and more specifically with Requirements Engineering.

Sommerville et al. (1998) defines Requirements Engineering (RE) as the process of establishing the services that customers want software systems to offer and the constraints for the development and operation of these systems. An RE approach that favors "design for reuse" (by guiding the cataloging of components) and "design from reuse" (by using catalogs to identify reusable goals) can definitely bring benefits to the Teleo-Reactive field.

There are many approaches to RE, such as Goal-Oriented RE (GORE) (Lamsweerde, 2009), Viewpoint-Oriented RE (VORE) (Sommerville et al., 1998), and Aspect-Oriented RE (AORE) (Chernak, 2009). Among them, GORE approach has proven its usefulness for the RE process (Lamsweerde, 2001). GORE is straightforward for developing TR systems since they share the concept of *goal*. GORE proposals (see Kavakli and Loucopoulos, 2005 for an exhaustive introduction) focus on the *why* of the system-to-be by specifying the motivation and rationale that justify the requirements specification. They also share the concepts of *goal*, *agent* and the *refinement* relationship. A Goal-Oriented (GO) model can be specified using different techniques but for all of them, the GO model is built using directed graphs by iterative refinement of goals.

GORE has proven successful for different RE phases (Loucopoulos, 2005), such as eliciting, elaborating, specifying, analyzing, documenting, etc. For instance, it has been used during the elicitation phase to describe the current organizational behavior (e.g., Goal-based Workflow (Ellis and Wainer, 1994), i^* (Pohl, 2010), among others). GORE has also been used to describe how organizational changes can be implemented by relating business goals to functional and non-functional system specifications (e.g., KAOS (Lamsweerde, 2001), the NFR framework (Mylopoulos et al., 1992), and the Goal-scenario coupling framework (Rolland et al., 1998) among others). Moreover, different goal analysis techniques have been also described for requirements validation (e.g., GSN (Wilson et al., 1995) and GQM (Basili and Rombach, 1988)).

As evidenced by the previous research works, GORE has been widely used for different purposes. Among all the proposals, i^* and KAOS are the most referenced ones in the literature. Each one has its own strengths and weaknesses, providing support for representing different aspects related to the RE process (Teruel et al., 2011b). For illustrative purposes, i^* does not provide support to specify when a task is carried out by several roles; i^* has more modeling elements for the sake of expressiveness; KAOS provides better support to traceability than i^*; neither KAOS nor i^* are able to represent the

priority of requirements (Teruel et al., 2012); KAOS is better for the hierarchical representation; etc.

In this article, the authors limit the study of the understandability for the specification of Teleo-Reactive systems to two of the most well-known GO approaches, i.e., i^* and KAOS, leaving other methods and notations for further research.

3. An introduction to the Teleo-Reactive paradigm

A TR program can be informally defined as an agent that robustly controls a system toward its goal by continuously taking into account the observed perceptions when executing in a dynamic environment (Nilsson, 1993). TR programs facilitate goal-oriented descriptions of systems following a very intuitive approach. TR programs are usually specified as a set of rules that continuously sense the environment and fire the execution of actions whose continuous execution eventually leads the system to satisfy its goal. Table 1 shows the basic syntax of a TR program. For each rule, K_i is the condition (formulated over a belief store that includes both sensory inputs and computed beliefs using inference rules), and a_i is the corresponding action, which will eventually change the state of the environment.

The rules of a TR procedure are continuously evaluated from the top for the first rule whose condition is true (higher priority), and then its corresponding action is then executed. The effects of an action on the environment occur as a consequence of the interaction with the system. Two kinds of primitive actions are considered: *durative* and *discrete* (aka *ballistic*). Durative actions continue indefinitely executing as long as its condition is true and they can be prematurely terminated. For example *pick up* action for a robot building a block tower, or *move* 5 m forward. Discrete actions terminate after a short time, for example *open* for grippers of a robot, or *beep* in a horn, and they cannot be prematurely aborted once started. As an example, Table 2 shows the TR program of a robot that *moves forward* until an obstacle is detected (sensor input) and then starts to execute *rotate* until the obstacle is not in front of it (further examples can be seen in Sánchez et al., 2012, with a more detailed definition of TR programs including hierarchies of goals).

Following the definition of a TR program, rule #1 has higher priority than rule #2. Thus, when an obstacle is detected the execution of the durative action *move_forward* is interrupted. Once the condition of rule #1 becomes false (because the robot does not detect the obstacle anymore) rule #2 gains again the control and *move_forward* is re-started.

The possibility of including hierarchies for the definition of TR programs is also very interesting because it favors modularity, fosters reuse, eases refinement and tests, etc. In a hierarchical TR program, actions can also be basic TR programs. Each TR program that is referenced in a rule is called a subgoal. The root of the hierarchy of a TR program is called the goal of the system-to-be.

Table 1
TR program structure.

Priority	Rule (condition → action)
The highest	$K_1 \rightarrow a_1$ $K_2 \rightarrow a_2$ ...
The lowest	$K_m \rightarrow a_n$

Table 2
Example of a TR program.

id	Rule
rule #1:	obstacle_detected → rotate
rule #2:	True → move_forward

Table 3
Example of TR program for data packets sending.

Condition		Action
AllPacketsSent	→	Nil
EveryOneHundredPackagesSent	→	adjustPacketSize
EveryTenPackagesSent	→	adjustCommunicationSpeed
ConnectionOpened	→	sendNewPackage
True	→	waitNewConnection

Table 4
Example of a hierarchical TR program for data packets sending.

Server:		
AllPacketsSent	→	nil
ConnectionOpened	→	Send
True	→	waitNewConnection
Send:		
EveryOneHundredPackagesSent	→	adjustPacketSize
EveryTenPackagesSent	→	adjustCommunicationSpeed
True	→	sendNewPackage

A TR program continues to constantly evaluate its set of conditions even when a subgoal has been called. Consequently, if another rule condition that has higher priority becomes true, the subgoal is immediately terminated.

As an example, we consider a modified and enriched version of a TR program for sending data packets from the server perspective (shown in Table 3). It simulates a file download from a FTP-like service. The server sends a file in packets once the connection has been established. The server dynamically adjusts both the communication speed and packet size accordingly to the communication robustness and network latency. This adjustment is performed every ten and one hundred sent packets, respectively.

The service is started when a client opens the connection. If both *ConnectionOpened* and *EveryTenPackagesSent* are true, then the action *adjustCommunicationsSpeed* will be executed since *EveryTenPackagesSent* is located higher in the TR program hierarchy. When a multiple of one hundred packages has been sent, the system adjusts packet size. The program has been conceived to show how the TR-based system can recover from unexpected events. For instance, if the connection gets lost the system will wait until a new connection is established, resetting the initial values relative to packet size and communication speed. The equivalence between hierarchical and plain specification of a TR program is straightforward. A possible hierarchical version of the TR program for the file server is shown in Table 4.

4. Goal-oriented requirements engineering and Teleo-Reactive systems

The GORE approach (Langley and Choi, 2006) has proven to be useful in eliciting and defining requirements, as it does not only establish the features (i.e., activities and entities) that a system will support, but also why the system is being constructed. Although different proposals, such as KAOS and i^* (see Kavakli and Loucopoulos, 2005 for an exhaustive survey) have been presented in the literature, all of them have some commonalities. One of them is the specification of a Goal Model as a directed graph that is used to refine the system goals. These goals are further refined until they can be assigned to the agent that is responsible for the achievement of that refined goal. Other concepts shared by these proposals are:

– *Goal*: it describes *why* the stakeholders want to develop a system. Goals may be functional goals (expected services), or soft-goals that determine non-functional goals such as those related to Quality of Service (QoS) issues or constraints on the design.

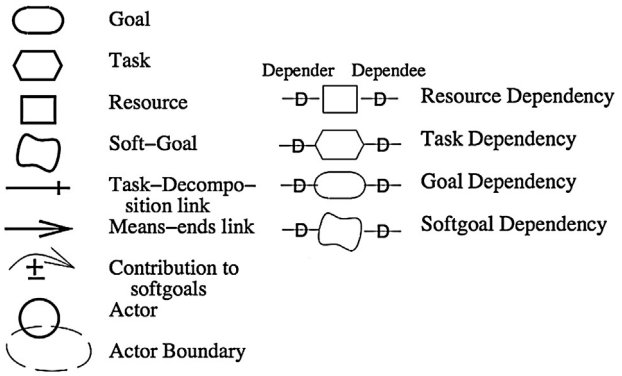


Fig. 1. Elements of the *i** framework model. Taken from Yu (1997).

- *Agent*: it is an active component whose cooperation is needed to achieve the goals of the system. These components can be either from the system itself or from the environment.
- *Refinement relationships*: goals in the model are refined as a directed graph using AND/OR/XOR relationships. A goal is refined into some other goals using AND relationships when all refined goals are needed to achieve the parent goal. OR relationships are used for optional goals where at least one of them is needed to achieve the parent goal. If a goal is refined using a XOR relationship, exactly one of the refined goals is required to achieve the parent goal. Refined goals are also refined until they can be assigned to a specific agent.

One of the main advantages of the GORE approach is that it allows reasoning about the system specification, facilitating the process of evaluating designs or alternative specifications of the system-to-be (Navarro et al., 2007). As mentioned above, two of the most widely GORE proposals are KAOS and *i**. Both of them offer graphical representations for goal modeling that can be easily used for specifying TR systems, as will be shown in sections 4.1 and 4.2. Given the deterministic nature of TR programs, in those sections we propose a mapping of the deterministic part of both languages to TR programs. The non-deterministic part of KAOS and *i** does not have a direct mapping although it can be of interest for its use to choose among different options to operationalize the requirements or to specify the non-functional requirements of the system-to-be.

4.1. *i** for TR system requirements specifications

The *i** framework guides the stakeholders throughout the different phases of the software development process, namely from early requirements analysis until detailed design. The graphical representations of the main *i** modeling elements are shown in Fig. 1.

Most of them have been already described in the previous section, such as goal or *softgoal*. However, others need further explanation to facilitate the understanding of the following work (refer to Yu, 1997 for a detailed explanation):

- An *actor* is a person or a system that is related to the system-to-be. An actor has a *boundary* to indicate which elements it explicitly desires.
- *Task* specifies what an actor wants to do, typically by means of a number of steps (or sub-tasks).
- *Resource* is an entity, either physical or informational, needed by an actor to achieve a goal or execute a task.
- *Dependency* is a relationship specifying that a depender (an actor) depends on a dependee (an actor) because of a dependum (a goal/softgoal to achieve, a task to execute or a resource to use).
- *Means-ends link* is a relationship from a task (mean) to a goal (end) that it attains.
- *Task decomposition link* relates a task to its different subcomponents, which can be sub-goals, sub-tasks, resources, or softgoals.
- *Contribution links* can be used to link an element (goal, task or softgoal) of the model to a softgoal to specify how it positively or negatively contributes to the satisfaction of that softgoal.

As the main aim of this work is to provide a way for specifying the requirements of robotic systems that facilitates their automatic transformation into TR programs, it is necessary to establish the mappings from *i** to TR concepts. Taking into account this need, the following mappings have been identified:

In order to build TR programs capable of achieving the objectives represented by an *i** Goal Model, the following construction rules have been defined:

- The main *i** agent is transformed into the TR system-to-be. The main *i** agent is the one that has the goal that the final system wants to achieve in its boundary.
- The rest of *i** agents are transformed into sensors of the system-to-be.
- *i** goals become TR goals.
- *i** tasks are specified as TR atomic actions.
- *i** resources are transformed into percepts provided by the sensors.
- Provided that a TR rule can be defined as *condition* → *goal/action*, every *i** resource whose depender is an agent and having a task or goal as a dependee is translated into a TR rule having as condition the resource and as action the task or goal. An *i** task or goal not depending on any resource is directly translated into a rule of the form *True* → *goal/action*.
- Since a TR goal can be defined as a set of prioritized TR rules, an *i** goal being refined into goals and tasks through task-decomposition links is transformed into a TR goal containing as many rules as *i** tasks and goals are refining the original *i** goal. It is worth noting that the advices of the *i**-based *GORE Language*, part of the *ITU Recommendation* named *User Requirements Notation*, have been followed. This recommendation allows, for the sake of simplicity, goal decomposition using task-decomposition links (ITU-T, 2008).
- *i** does not provide a way to represent priority among the tasks or subgoals needed to achieve a higher-level goal. Therefore, we decided that priority would be specified in the Goal Model by using the relative position on the diagram of the tasks or goals refining the original goal. Thus, the tasks or goals situated on the right end of the Goal Model will be translated into the highest priority rules in the TR program. This alternative allows us to analyze *i** without modifying its notation, although as future work an alternative representation for task priority shall be proposed.

The first four mappings shown in Table 5 are obvious since the definitions of those elements are similar in both *i** and Teleo-Reactive. Regarding the last two, accepting that a sensor is represented as an agent, we have three *i** elements that could be mapped to TR percepts: (1) resources, if the interest is in the new data received through the percept; (2) tasks, when the focus is on how the percept is obtained; and (3) goals, when the focus is on why the percept is obtained. Since the Teleo-Reactive approach is only concerned about the incoming data, the first option is the most adequate (resources mapped to percepts). Besides, a condition relates changes in the environment (that is, percepts) with the

Table 5
Mapping concepts between TR and *i**.

*i**	TR
Main agent	System-to-be
Agent	Sensor
Goal	Goal
Task	Action
Resource	Percept
Resource dependency	Condition

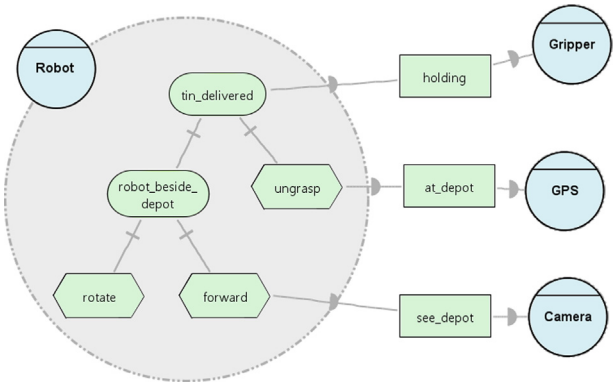


Fig. 2. Tin collector specification using *i**.

TR actions or goals. As a consequence of the previous mapping, *i** resource dependencies seem to be the most reasonable element to be mapped to TR conditions in order to state which percept is involved in the activation of which task (or subgoal). The rest of the *i** elements shown in Fig. 1 do not have a direct mapping to TR programs as they are not needed to specify a deterministic TR program, and thus they have not been considered in this work, but they can nevertheless be used in other stages of the RE process.

The following example shows how the previous guidelines have been applied. It is an excerpt of one of the examples used in the controlled experiment (as detailed in Section 5.2). Fig. 2 shows the *i** specification of a robot which delivers tins into a depot. Table 6 shows the TR program implementing such specification.

- According to the guidelines, every *i** goal in Fig. 2 is mapped to a TR goal (highlighted in bold text in Table 6, as for instance, *tin_delivered* or *robot_beside_depot)*.
- Every *i** agent (except for the main agent) is mapped to a sensor or device, such as *camera* or *gripper*.
- Every *i** resource that has a dependency relationship with an agent is mapped to a condition monitored by the corresponding sensor, such as *holding* monitored by the *gripper* or *at_depot* monitored by the *GPS*. It can be noticed that *i** goals or tasks that have no dependency relationship, such as *robot_beside_depot* or *rotate*, are mapped to TR goals and actions with a True condition.
- Every *i** task, such as *ungrasp*, is mapped to an action.
- *i** tasks and goals linked by a task-decomposition link to an *i** goal are mapped to rules of the same TR goal. For instance, the goal *tin_delivered* is decomposed into the goal *robot_beside_depot* and the task *ungrasp*. Then, a TR goal named *tin_delivered* is created with two rules whose respective actions are *robot_beside_depot* and *ungrasp*, as can be seen in Table 6.
- Finally, as can be observed, the priority established in the *i** specification by using the location of the elements is also kept in the TR specification. For instance, as *ungrasp* has a higher priority than *robot_beside_depot*, it has been specified on the right hand of the diagram (see Fig. 2). As can be seen in Table 6, the rule with action *ungrasp* is over the rule with action *robot_beside_depot*.

Table 6
Tin collector TR program.

robot:		
holding	→	tin_delivered
tin_delivered:		
at_depot	→	ungrasp
True	→	robot_beside_depot
robot_beside_depot:		
see_depot	→	forward
True	→	rotate

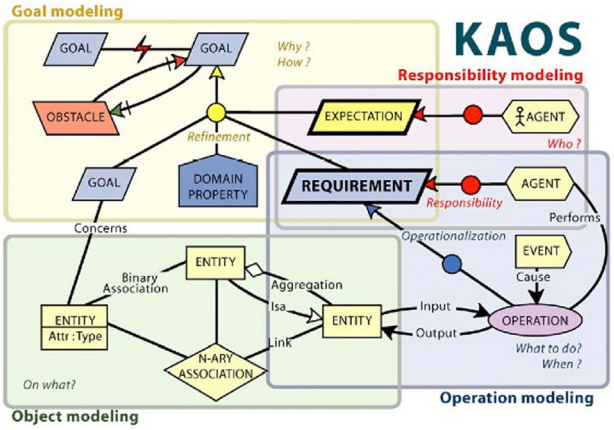


Fig. 3. Basic elements of KAOS.
Taken from Respect – IT (2007).

4.2. KAOS for TR Systems requirements specifications

KAOS framework (Lamsweerde, 2001) has been defined for eliciting, specifying, and analyzing goals and requirements. A KAOS model is made up by six sub-models, namely *Goal model*, *Obstacle model*, *Object model*, *Agent model*, *Operation model* and *Behavior model*, which are related among them via traceability links. Fig. 3 illustrates the basic elements provided by the KAOS framework.

KAOS has the following main elements (see Respect – IT, 2007 for a detailed explanation):

- A *Goal* describes an objective to be achieved by the system-to-be.
- An *Agent* is an active component of the system-to-be, which can be a human agent, a device or a software component, that acts to satisfy a goal.
- A *requirement* is a low-level type of goal that has an agent responsible for its achievement.
- An *operation* is a binary relation over system states. This is the way an agent may achieve its goals.
- An *event* is a conceptual item referenced in a goal specification whose instances may exist in a single state only.
- An *AND/OR-decomposition* is a relationship from a set of subgoals to a goal that specifies that all/at least one sub-goal must be satisfied for the goal to be satisfied.
- A *responsibility assignment* is a relationship from an agent to the goal it is responsible for.
- *A concerns link* is a relationship used to link a goal to the objects that it needs to be satisfied. Identifying those objects shall further restrict the space of solutions that can be proposed by the system-to-be.

Table 7 summarizes the mappings proposed between KAOS and TR programs concepts to model reactive systems.

In order to guide the definition of KAOS models, the following guidelines have been defined by using the previously defined mappings:

Table 7
Mapping concepts between KAOS and TR.

KAOS	TR
Goal OR requirement	Goal
Operation	Action
Event	Condition

- Every KAOS goal is mapped to a TR goal with as many rules as subgoals or requirements the original goal has. For every subgoal refining a KAOS goal, the corresponding TR rule has a TR goal as its action. In case the item refining the goal is a requirement, the TR rule has an atomic action given by the operation that operationalizes the requirement.
- As TR rules are mandatory to achieve a goal and there are no optional rules, AND-decomposition links are always used when refining goals.
- A KAOS event with a *concerns link* to a goal or requirement is translated into the condition of the rule obtained from the mapping of that goal or requirement. A KAOS goal or requirement not linked to any event is translated into a rule with a True condition.
- Like *i**, KAOS does not provide a way to represent priorities among the tasks or subgoals needed to achieve a higher-level goal. A similar decision to that already proposed for *i** is adopted in this case. Priorities given by the order of the rules in TR programs are being specified in KAOS by using the order in which the subgoals or requirements refining the original goal are presented. Therefore, the subgoals or requirements situated on the right end of the Goal Model are translated into the highest rule in the TR program. As a future work, an alternative representation for task priority shall be proposed.

The first two mappings of Table 7 are evident according to the definitions of the KAOS and Teleo-Reactive elements. Regarding the last one, since we have accepted the mapping of actions to operations it is obvious that events should be mapped to conditions because they are responsible for triggering operations.

Fig. 4 shows the KAOS specification for the tin collector example that leads to the TR program in Table 6.

The description of how the previous guidelines are being applied is the following:

- Every KAOS goal (such as *tin_delivered*) is mapped to a TR goal.
- Every KAOS goal refined into several subgoals or requirements is mapped to a TR goal with a rule per subgoal or requirement of the input KAOS model. For instance, *tin_delivered* is decomposed by using an AND decomposition link into the goal *robot_beside_depot*, and the requirement *tin_dropped*, which in turn is operationalized by *ungrasp*. Table 6 shows the TR goal *robot_beside_depot* with two rules.

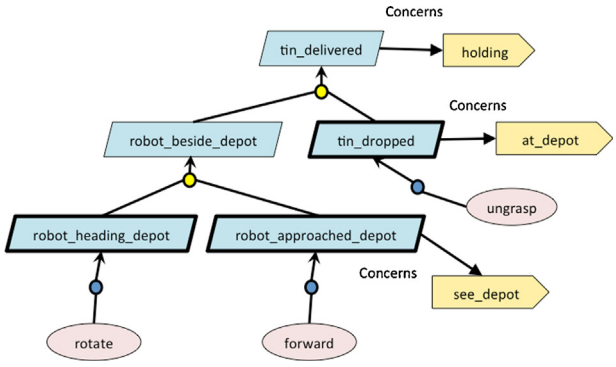


Fig. 4. Tin collector specification using KAOS.

- KAOS events are represented as TR conditions. For instance, the goal *tin_delivered*, related by means of a *concerns* relationship to the event *holding* is mapped to the TR rule *holding* → *tin_delivered*.
- KAOS operations are mapped to TR atomic actions. For instance, the KAOS operation *ungrasp*, which operationalizes the requirement *tin_dropped* and has a *concerns* relationship with the KAOS event *at_depot*, is translated into the rule *at_depot* → *ungrasp*.
- Finally, the priority established in the KAOS specification by using the location of the elements is also kept in the TR program. For instance, as *ungrasp* has a higher priority than *robot_beside_depot*, it is placed on the right hand of the diagram (see Fig. 4). As can be seen in Table 6, the rule whose action is *ungrasp* is over the rule whose action is *robot_beside_depot*.

Next section describes the experiment performed in order to identify which of these languages has better understandability for this purpose.

5. A controlled experiment

In order to assess the *understandability* of KAOS and *i** to model the software requirements of TR systems, a controlled experiment to compare both of them has been performed based on the guidelines described by Kitchenham et al. (2002). In this section we describe the context, the design and how the experiment was conducted. At the end of the section the results are deeply analyzed.

5.1. Experimental context

The Goal Question Metric template (Basili et al., 1994) has been used to define the main goal of the controlled experiment as follows: *analyze i** and KAOS requirements specifications for TR systems with the *purpose of* evaluating their understandability *from the viewpoint of* requirements engineering researchers and *in the context of* undergraduate students. To address this goal the following null hypothesis has been defined:

- H_0: *i** has the same average score for understandability than KAOS when modeling TR requirements specifications. $H_1: \neg H_0$

It is worth remembering that the objective of this work is to evaluate which one, KAOS or *i**, has a better understandability when modeling requirements for TR-based systems. We have adopted each of the languages without altering their original syntax/semantics. When evaluating the understandability of KAOS/*i** for representing Teleo-Reactive concepts we are implicitly evaluating the acceptance of the defined mapping since a wrong mapping between KAOS/*i** and Teleo-Reactive concepts would have given a non-acceptable understandability of the representations obtained in KAOS/*i** for the domain with much worse results in the survey.

5.2. Experimental design

The variable *understandability* (*Und*) of *i** requirements models related to KAOS models is used in this study. For this aim, the ability of the undergraduate students to understand the experimental material correctly is evaluated by means of True/False questionnaires about two TR systems specified with both *i** and KAOS (see Appendix 1).

The subjects in this experiment were 38 university students, being 16 students from the third year of the Bachelor in Telecommunication Systems, and the remaining 22 students from the third year of the Bachelor in Industrial Electronics and Automation Engineering. Both groups already had some background about reactive systems, as they were enrolled in different subjects related to this

Table 8
Experiment main characteristics.

Null-hypotheses	H_{0A}: i^* has the same understandability average score than KAOS when specifying TR requirements. H_{1A}: $\neg H_{0A}$ H_{0B}: The understandability average score is the same regardless the system used in the experiment. H_{1B}: $\neg H_{0B}$ H_{0AB}: i^* has the same understandability average score than KAOS when specifying TR systems requirements, regardless the system used in the experiment and vice versa. H_{1AB}: $\neg H_{0AB}$
Location	Technical University of Cartagena (Cartagena, Spain)
Date	June 6th 2013 (Group 1); June 10th 2013 (Group 2)
Subjects	Group 1: 16 students from 3rd year of Bachelor in Telecommunication Systems Group 2: 22 students from 3rd year of Bachelor in Industrial Electronics and Automation Engineering
Dependent variable	Understandability of requirements modeling notations, measured by *Und*
Independent variable	The system (football player or tin collector) to which the diagrams were related to and the language (i^* or KAOS) used for specifying them

topic. Moreover, because the subjects were not required to carry out tasks with high levels of industrial experience, their expertise is considered as appropriate for this experiment as other authors suggest (Höst et al., 2000; Basili et al., 1999). The main characteristics of this controlled experiment are illustrated in Table 8.

This experiment studied the understandability level of the requirements specification of two different TR systems specified by using both i^* and KAOS. The first system was a robot developed to clear tins from the floor by moving them to a depot. The robot is able to rotate, to scan the environment looking for tins or a depot, to move forward, and to open/close its gripper. The robot includes sensors that evaluate whether the robot is holding, seeing, and touching a tin. For example, if the robot sees a tin, then it moves forward while it can see the tin. Provided that the environment does not move the tin, the robot will eventually touch it. Using its current location, the robot is able to see the depot and to know whether it is at the depot.

The second system was another robot developed for playing football in defensive positions. The robot can detect the ball and is able to tell if the ball is being controlled by itself or by an opponent. In addition, it can recognize other robots of its own team. Its main goal is to keep the ball under control of any of its teammates. To achieve this, the robot is able to rotate, move forward, dribble and kick the ball. This system is inspired by TRSoccerbots (Kochenderfer et al., 2004), a free educational program that uses TR programming to expose the fundamental ideas involved in the creation of autonomous agents to high school and college students.

In order to assign the requirements specifications of the two systems to the students, they were distributed into two different groups, G1 and G2. It was decided that G1 would perform the experiment using the i^* specification for the football player and the KAOS specification for the tin collector, while G2 will use the KAOS specification for the football player and the i^* diagram for the tin collector. These decisions are summarized in Table 9.

The distribution of the subjects into 4 different treatments, carried out by combining the two dependent variables (System and Language), is a *2 × 2 factorial design* with confounded interaction (Winer et al., 1991). Therefore, within a system, the variable language changes along with the group of subjects canceling out the *learning effect*.

It was also decided to filter out those students that satisfied any of the following criteria:

- The student was at least five years older than the class mean age.
- The student had a RE background.
- The student had previous experience with i^*, KAOS or other Goal-Oriented techniques.

Finally, it was also decided that those students who wanted to leave the experiment would be interviewed recording this fact in the experiment results, as well as any other possible interruption. No time restrictions were imposed. Questions were answered personally and recorded in order to know if any other subject had the same doubt. All questions were recorded for further analysis.

5.3. Conducting the experiment

The experiment was carried out in two different sessions, the first with G1 and the second with G2. The first session was held at the *Escuela Técnica Superior de Ingeniería en Telecomunicación* of the Technical University of Cartagena, and the second one at the *Escuela Técnica Superior de Ingeniería Industrial* of the same university. Both sessions were performed in a similar way. First, the subjects were given an introductory session to TR-Systems, KAOS and i^*. Then, considerations about representing TR systems requirements in both notations were also explained. Afterwards, both examples, the tin collector and the football player, where shown. All these explanations took about 15 min in each session.

Before providing the students with the specifications, the following information about them was collected:

- Gender (Male/Female)
- Age
- Qualification
- Average score
- Have you had any previous experience of working with goal-oriented requirements engineering?
- Have you had any previous experience of working with any other requirements engineering technique?

In the same way, the subjects were told to write the exact start and end time in the spaces reserved for that purpose. In order to have the same time reference for all the subjects, an online clock was projected on a screen. Table 10 summarizes the results of the experiment by showing the maximum time (max), minimum time (min) and average time (avg) spent by each group.

5.4. Results analysis

Once the experiment concluded, all the data were analyzed using the SPSS tool. First of all, it was determined that 6 observations of the group G2 had to be discarded as these subjects satisfied some of the criteria stated in Section 5.2. Then, a descriptive statistic analysis of the experiment was made. As Table 11 shows, the mean understandability of the i^* specifications for both the tin collector and the football player is better than the one for KAOS. Therefore,

Table 9
Experiment design.

		System	
		Tin collector	Football player
Language	KAOS	Group 1	Group 2
	i^*	Group 2	Group 1

Table 10
Time (mm:ss) spent on the experiment.

Group-language	*N*	Max	Min	Avg
G1-KAOS	16	14:10	07:14	09:59
G2-i^*	16	15:19	08:24	11:40
G1-i^*	16	12:05	07:35	09:32
G2-KAOS	16	09:06	05:50	07:04

Table 11
Descriptive statistic for dependent variable understandability.

Language	System	Mean	Std. Deviation	N
i^*	collector	0.706250	0.1181454	16
	football	0.700000	0.1211060	16
	Total	0.703125	0.1177323	32
KAOS	collector	0.668750	0.1302242	16
	football	0.637500	0.1087811	16
	Total	0.653125	0.1190944	32
Global	collector	0.687500	0.1237844	32
	football	0.668750	0.1176038	32
	Total	0.678125	0.1201438	64

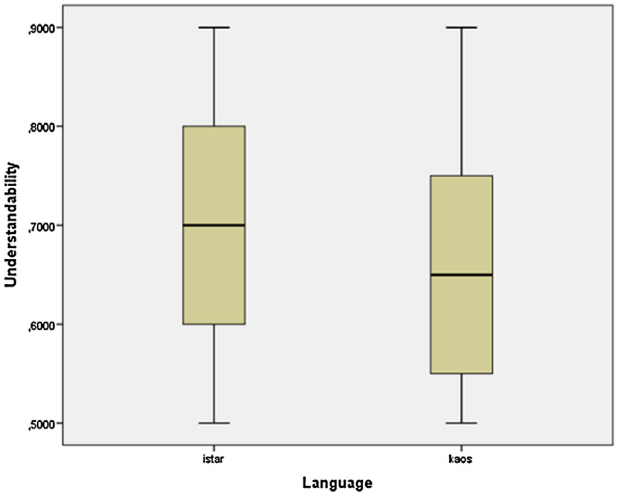


Fig. 5. Summary of results.

this initial analysis could indicate that the used system did not have any kind of influence on the results. When we analyzed the results grouped by the modeling language (see rows *Total* in Table 11 for KAOS and i^*), i^* exhibits positive differences with regard to KAOS. This difference between both languages is also shown in Fig. 5.

After filtering out the subjects that satisfied the criteria stated in Section 5.2, an ANOVA test (Table 12) was carried out. The result of the test was a p-value of 0.100 for the independent variable "Language". Then, the null hypothesis H_{0A} can be rejected with $\alpha = 0.10$, therefore a statistically significant difference exists between the results of KAOS and i^*. This difference between both results is graphically illustrated in Fig. 5. Although this result is acceptable, we would have expected a higher power of the test ($\alpha \leq 0, 05$) so that the results were more conclusive. Moreover, the obtained p-values were much higher than α for "System" and "Language * System", thus null hypotheses H_{0B} and H_{0AB} cannot be rejected. This means that the system had no effect on the results of the experiment and there was no interaction between language and system

Table 12
ANOVA results (dependent variable: understandability a. R-squared = 0.053) (adjusted R-squared = 0.006).

Source	Type III sum of squares	df	Mean square	F	Sig.
Corrected Model	0.048	3	0.016	1.118	0.349
Intercept	29.431	1	29.431	2050.319	0
Language	0.040	1	0.040	2.787	0.100
System	0.006	1	0.006	0.392	0.534
Language*System	0.003	1	0.003	0.174	0.678
Error	0.861	60	0.014		
Total	30.340	64			
Corrected Total	1.473	75			

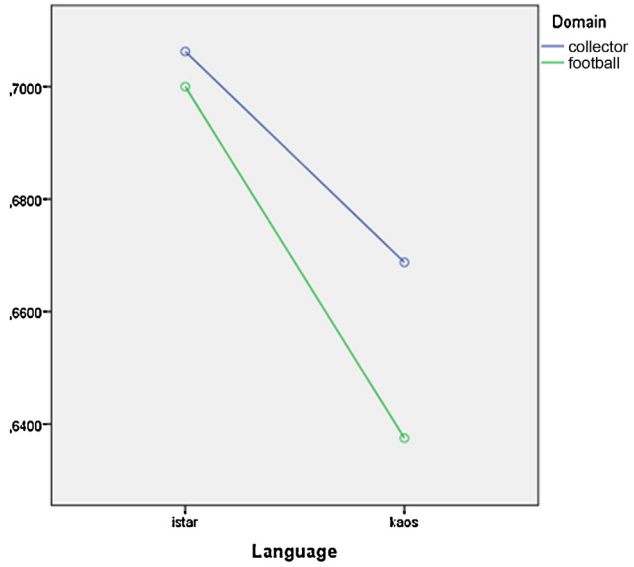


Fig. 6. Estimated marginal means for Understandability.

as Fig. 6 shows. Therefore, if we observe the descriptive statistic shown in Table 11, we can state that the Understandability of i^* is higher than KAOS when modeling TR systems. This difference is graphically illustrated in Fig. 5.

5.4.1. Questions analysis

Next, it is analyzed, question by question, when KAOS obtains better scores than i^* and vice versa, in order to determine which factors contribute to these differences. Fig. 7 shows graphically the average scores of understandability grouped by question for each system.

When observing the results for each system, several differences can be noticed. Firstly, it is remarkable that i^* obtains higher or similar scores for most of the questions of both systems. This is an expected result regarding the descriptive statistic shown in Table 11. This means that i^* satisfies most of the expressive needs for this kind of systems. However, it does not only exhibit some problems when compared to KAOS in some questions but also because of the low scores it gets. In the following these questions are analyzed:

- Despite questions 5 and 6 of the Tin Collector get very high scores for both languages, it is slightly higher for KAOS. The only explanation was the location of the conditions in the i^* and KAOS models because, when the subjects were interviewed, they expressed that they had thought the priority was also denoted by the location of the conditions. This problem is also applicable to questions 4 and 5 of the football player.
- Question 9 of the Tin Collector gets a very high score for both KAOS and i^*. This question is related to a condition that must be satisfied for a goal to be achieved. Only two subjects failed this question when they analyzed the i^* specification and one when analyzed the KAOS specification.
- Question 10 of the Tin Collector obtains a very low score, no matter whether the KAOS or i^* specification is used. This question is related to priority, a critical issue in TR systems specifications. Other two questions are also related to priority: question 3 does not get a high score either but question 6 does get it. What differentiates question 6 from 3 and 10 is that no information about conditions needed to be analyzed to answer it. Subjects had to pay attention to two different concepts, what could be misleading especially because the priority is not specified by any graphical notation but by using the location of the elements in the diagram.

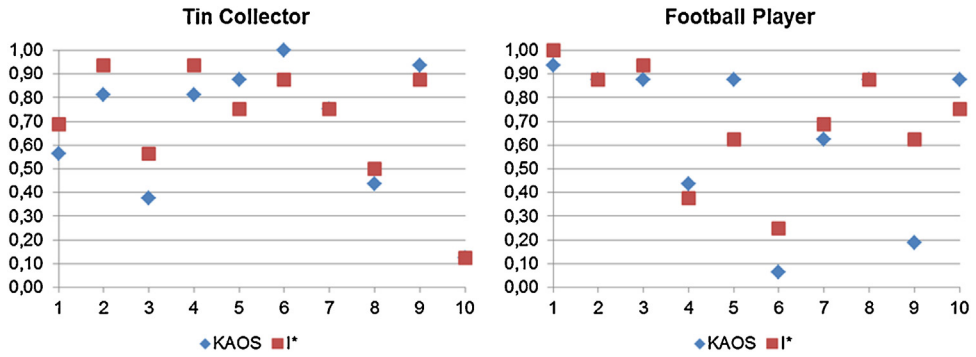


Fig. 7. Summary of results with regard to the systems and the modeling languages.

This conclusion is also applicable to question 6 of the football player. Therefore, it is a clear indication that an alternative solution must be defined to specify the priority.

– Question 10 for the football player obtains a slightly higher score for KAOS. It is worth noting that tasks in i^* cannot be duplicated and their location states the priority. These restrictions may result in diagrams hard to understand because a task refining two different goals can be very distant from one of those goals. In this case, task *rotate* is far away from goal *friend_found*. This limitation should be solved by proposing alternative notations too.

The students asked a number of questions during the performance of the experiment that may point out why i^* got better scores than KAOS. Some of these questions where related to the differences between goals and requirements in KAOS. The graphical representation of both concepts is very similar and may be a source of errors. One of the students even asked if the priority of a requirement was higher than that of a goal or vice versa. On the other hand, nobody asked questions about tasks and goals in i^*, so we can deduce that these concepts were clearer in i^*.

There were also some other questions with regards to the conditions that may fire the reactions of the system. Most of these kinds of questions were asked about the KAOS models. This fact leads us to the conclusion that the lack of agent representations in the KAOS models does not help to understand the specification. The presence of the agents in the i^* models and their dependency relationships with goals and tasks through the conditions makes these models easier to understand.

6. Threats to the validity of the experiment

In the following some issues that could have threatened the validity of the experiment are analyzed following the recommendations presented by Wohlin et al. (2000).

6.1. Conclusion validity

In the experiment, although the statistical power was not as high as we expected, it is high enough to reject the null hypothesis. Moreover, we were able to avoid the "fishing for the result" effect. The test was not performed to demonstrate that i^* was better than KAOS, but to identify really which language provided better support for the specification of TR systems.

6.2. Internal validity

The number of subjects that participated in the experiment was large enough, according to the central limit theory (Grinstead and

Table 13
Levene's test for language.

		Levene's test for equality of variances		*t*-Test for equality of means						
		F	Sig.	*t*	df	Sig. (2-tailed)	Mean difference	Std. error difference	95% confidence interval of the difference	
									Lower	Upper
Understandability	Equal variances assumed	0.008	0.929	−1.352	66	0.181	−0.04118	0.03045	−0.10196	0.01961
	Equal variances not assumed			−1.352	65.943	0.181	−0.04118	0.03045	−0.10196	0.01961

Table 14
Levene's test for system.

		Levene's test for equality of variances		*t*-Test for equality of means						
		F	Sig.	*t*	df	Sig. (2-tailed)	Mean difference	Std. error difference	95% confidence interval of the difference	
									Lower	Upper
Understandability	Equal variances assumed	0.191	0.664	0.252	66	0.801	0.00779	0.03086	−0.05383	0.06941
	Equal variances not assumed			0.253	65.975	0.801	0.00779	0.03083	−0.05375	0.06934

Snell, 2006). This number was slightly higher than 30 (actually, it was 32 once the subjects that met any of the criterion stated in Section 5.2 were filtered out). Moreover, ANOVA needs homogeneity of variances for each combination of the groups of the two independent variables, language and systems. This assumption was checked by using a Levene's test. From its results, shown in Tables 13 and 14, we can conclude that the null hypothesis can be rejected and, thus, the assumption is satisfied.

Finally, the learning and fatigue effects were canceled out because a 2×2 factorial design was carried out, changing both language and system, and having a break between the two sessions carried out by each group.

6.3. Construct validity

As construct validity is mainly related to the method used to evaluate the outcome of the task of the experiment, the threats were avoided by using a questionnaire, similarly to other studies such as (Teruel et al., 2012, 2011a). The understandability of the specifications was measured dividing the number of correct answers by the total number of answers. Furthermore, the specification of the systems and the questionnaires used in the experiment were reviewed by several external experts in KAOS, i^* and/or TR systems in order to avoid a possible source of bias.

6.4. External validity

According to Höst et al. (2000), the subjects can be considered appropriate because the tasks they had to carry out did not require a high level of industrial experience. It allowed us to avoid the interaction between subject selection and treatment. Moreover, the students had carried out several courses on reactive systems, so that they were mature enough to participate in the experiment.

Finally, the material used in the experiment was realist due to the fact that it was part of several industrial studies developed by the Systems and Electronic Engineering Division (DSIE) at the Technical University of Cartagena.

7. Conclusions and further works

We have introduced two alternative methods to model the software requirements for implementing goal-based reactive systems, one of them using i^* and the other using KAOS. A controlled experiment to compare the *understandability* of both languages has been performed. The results show that the understandability of i^* is higher than KAOS when modeling reactive systems, although the statistical power is not as high as we had expected.

The representation in i^* of the dependency relationships between goals and the agents, responsible for the conditions which cause the reactions of the system, increases the understandability of the models. Besides, i^* tasks and goals are much easier to distinguish than KAOS goals and requirements. Moreover, during the results analysis of the experiment two main weaknesses were identified in i^* for modeling software requirements for TR systems. One of them is related to the graphical representation of the priority of the requirements. The other has to do with the restriction of duplicating tasks or goals in order to improve the readability of the diagrams.

Finally, the non-deterministic aspects of i^*, such as non-functional requirements, should be taken into account although they do not have a direct mapping to TR programs. For instance, they may be useful for the specification and analysis of the system-to-be. However, all these aspects are left out of this work, as they do not have a direct mapping to TR rules, but to other stages of the development process of TR systems.

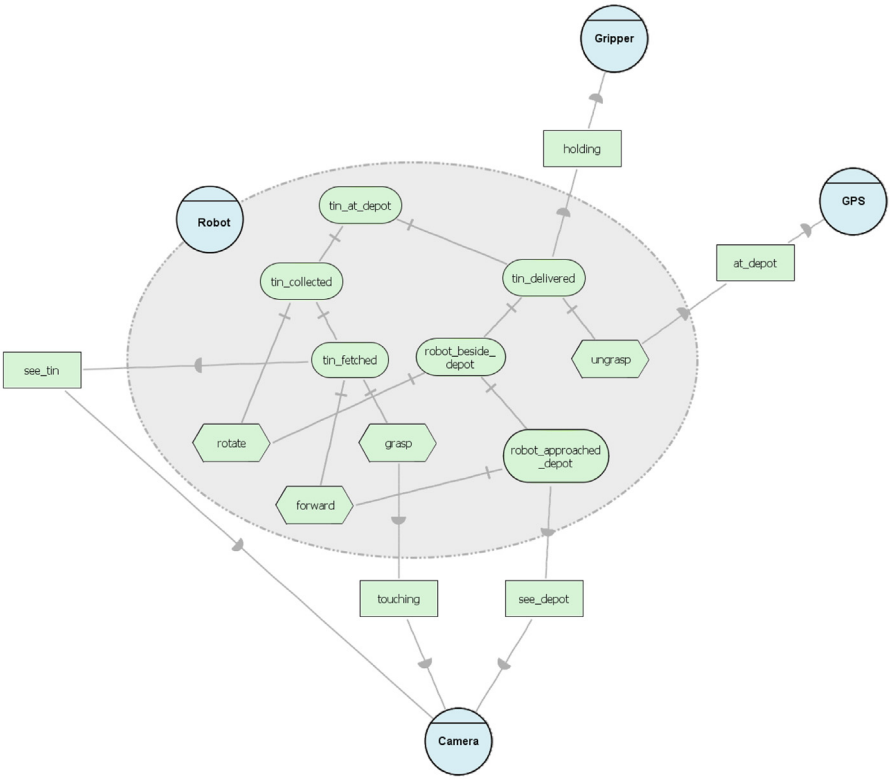
Further research is needed to compare other quality attributes of both languages apart from understandability such as suitability, accuracy, learnability or operability, as shown in Bertoa and Vallecillo (2010). In addition, we are working on new i^* extensions that can help to mitigate the weaknesses found during the experiment and all other problems that may appear. Further empirical studies will be done to provide valuable answers in terms of understandability. An experiment making a comparison between the original i^* specification and the extended version shall be performed in order to determine whether the proposed extensions are useful or not.

Acknowledgements

This research has been funded by the Spanish Ministry of Economy and Competitiveness and by the FEDER funds of the EU under the project Grant insPIre (ref. TIN2012-34003), cDrone (ref. TIN2013-45920-R), and ViSelTR (ref. TIN2012-39279) from the Spanish Government.

Appendix 1. Experimental material – an example of an understanding task (test for group 2)

Gender (Male/Female):
Age:
Qualification:
Average score:
Have you had any previous experience of working with goal-oriented requirements engineering?:
Have you had any previous experience of working with any other requirements engineering technique?:

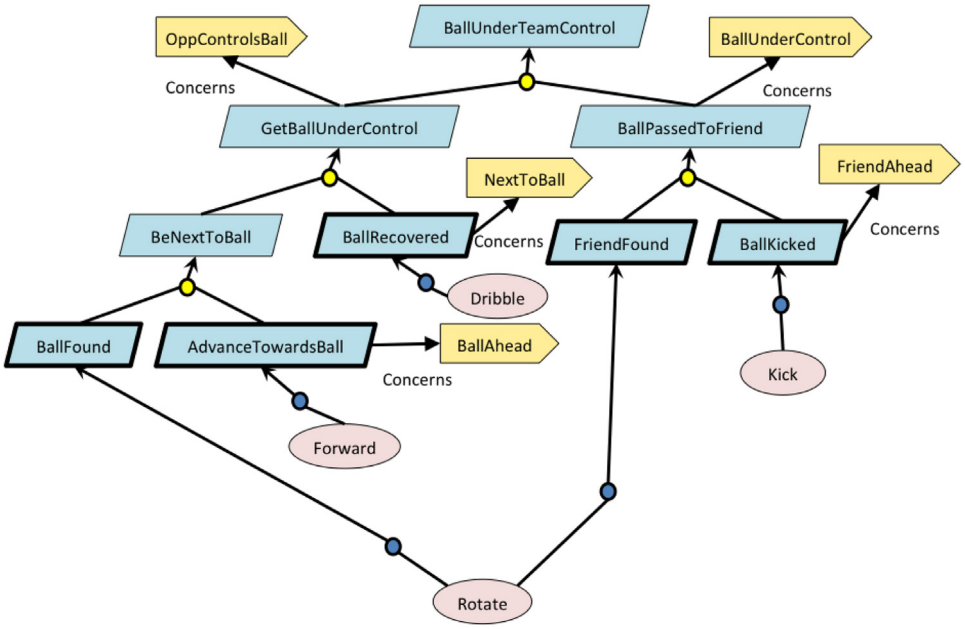


START TIME:

T F

1. If the robot is taking a tin to the depot and drops it, the robot continues on its way to the depot.
2. While the robot does not hold any tin and there isn't any tin in sight, the robot continues rotating.
3. Whenever the robot sees a tin, it will go for it.
4. While the robot holds a tin it rotates until it sees the depot.
5. If the robot is holding a tin and drops it in the depot, it starts rotating looking for another tin.
6. Provided that the robot cannot find more tins, it stops.
7. If the robot is looking for a tin and we put one in its grip, the robot rotates looking for the depot.
8. While the robot is holding a tin, it will not drop it until it sees the depot.
9. If the robot sees a tin and is not holding one, it will go for the tin.
10. Whenever the robot reaches the depot it drops a tin.

END TIME:



START TIME:

T F

1. At each time the robot tries to control the ball.
2. If the robot is next to the ball, it will be able to control it.
3. Whenever the robot starts up, it goes forward.
4. Whenever the robot sees a friend ahead, it kicks the ball.
5. While an opponent controls the ball, the robot will go for it.
6. Whenever the robot sees the ball ahead, it goes forward.
7. If a friend controls the ball, the robot makes nothing.
8. If an opponent controls the ball, the robot tries to control it.
9. If the robot is next to the ball and sees a friend ahead, it will kick the ball.
10. If the robot has the ball, it will rotate until it sees a friend ahead.

END TIME:

References

Atkinson, C., Kühne, T., 2003. Model-driven development: a metamodelling foundation. IEEE Softw. 20 (5), 14–18.

Basili, V.R., Rombach, H.D., 1988. The TAME Project: towards improvement-oriented software environments. IEEE Trans. Softw. Eng. 14 (6), 758–773.

Basili, V.R., Caldiera, G., Rombach, H.D., 1994. The goal question metric approach. Encyclopedia of Software Engineering, vol. 2. Wiley, pp. 528–532, Retrieved from: http://wwwagse-old.informatik.uni-kl.de/pubs/repository/basili94b/encyclo.gqm.pdf

Basili, V., Shull, F., Lanubile, F., 1999. Building knowledge through families of experiments. IEEE Trans. Softw. Eng. 25 (4), 456–473.

Bertoa, M.F., Vallecillo, A., 2010. Quality attributes for software metamodels. In: Proceedings of the Workshop on Quantitative Approaches to Object-Oriented Software Engineering (QAOOSE 2010), Malaga, Spain, July 2, 2010.

Bézivin, J., 2005. On the unification power of models. Softw. Syst. Model. 4 (2), 171–188, http://dx.doi.org/10.1007/s10270-005-0079-0.

Broda, K., Hogger, C.H., 2004. Designing and simulating individual Teleo-Reactive agents. In: Proceedings of KI-04.

Broda, K., Hogger, C.J., Watson, S., 2000. Constructing Teleo-Reactive robot programs. In: ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20–25, 2000. IOS Press.

Chernak, Y., 2009. Building Foundation for Structured Requirements. Aspect-Oriented Requirements Engineering Explained – Part 1. Requirements Networking Group (RQNG).

Choi, D., Langley, P., 2005. Learning teleoreactive logic programs from problem solving. In: Proceedings of ILP-05.

Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J., 2000. Non-Functional Requirements in Software Engineering. International Series in Software Engineering, vol. 5. Springer, Heidelberg.

Dongol, B., Hayes, I.J., Robinson, P.J., Technical Report SSE-2010-01 2010. Reasoning About Real-Time Teleo-Reactive Programs. Division of Systems and Software Engineering Research, The University of Queensland.

Ellis, C.A., Wainer, J., 1994. Goal-based models of collaboration. Collab. Comput. 1, 61–86.

Grinstead, C.M., Snell, J.L., 2006. Introduction to Probability. American Mathematical Society.

Gubisch, G., Steinbauer, G., Weiglhofer, M., Wotawa, F., 2008. A Teleo-Reactive architecture for fast, reactive and robust control of mobile robots. In: IEA/AIE'08 Proceedings of the 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: New Frontiers in Applied Artificial Intelligence.

Hawthorne, J., Anthony, R., 2010. A methodology for the use of the Teleo-Reactive programming technique in autonomic computing. In: Proceedings of SND/ACIS-10.

Hawthorne, J., Anthony, R.J., Petridis, M., 2011. Improving the development process for Teleo-Reactive programming through advanced composition. In: 7th International Conference on Autonomic and Autonomous Systems (ICAS 2011), pp. 75–80.

Höst, M., Regnell, B., Wohlin, C., 2000. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. Emp. Softw. Eng. 5 (3), 201–214.

IEEE Computer Society, 1998. IEEE Std 830-1998 – IEEE Recommended Practice for Software Requirements Specifications.

ISO/IEC, 2001. ISO/IEC 9126: Software Engineering – Product Quality.

ITU-T, 2008. Recommendation Z.151 (09/08): User Requirements Notation (URN) – Language Definition, Geneva, Switzerland, 206 pp.

Kavakli, E., Loucopoulos, P., 2005. Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods. Information Modeling Methods and Methodologies, Idea Group.

Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. 28 (8), 721–734, http://dx.doi.org/10.1109/TSE.2002.1027796.

Kochenderfer, M., 2003. Evolving hierarchical and recursive Teleo-Reactive programs through genetic programming. In: Proceedings of EuroGP-03.

Kochenderfer, M., Sego, M., Shimano, K., Tansuwan, J., 2004. TRSoccerbots, Available from: http://trsoccerbots.software.informer.com
Lamsweerde, A.,2001. Goal-oriented requirements engineering: a guided tour. In: 5th IEEE International Symposium on Requirements Engineering (RE'01). IEEE Computer Society, Washington, DC, USA, pp. 249–262, http://dx.doi.org/10.1109/ISRE.2001.948567.
Lamsweerde, A., 2009. Requirements Engineering: From Goals to UML Models to Software Specifications. John Wiley & Sons Ltd, England.
Langley, P., Choi, D., 2006. Learning recursive control programs from problem solving. J. Mach. Learn. Res. 7, 493–518.
Lee, J., Durfee, E.H., 1994. Structured circuit semantics for reactive plan execution systems. In: Proceedings of AAAI-94, pp. 1232–1237.
Loucopoulos, P., 2005. Requirements engineering. In: Clarkson, J., Eckert, C. (Eds.), Design Process Improvement. Springer, London, pp. 116–139.
Morales, J., Sánchez, P., Alonso, D., 2012. A Systematic Literature Review of the Teleo-Reactive Paradigm. Artificial Intelligence Review. Springer, Netherlands, pp. 1–20.
Mousavi, S.R., Broda, K., Technical Report 2003. Simplification of Teleo-Reactive Sequences. Imperial College London.
Mylopoulos, J., Chung, L., Nixon, B., 1992. Representing and using nonfunctional requirements: a process-oriented approach. IEEE Trans. Softw. Eng. SE-18 (6), 483–497.
Navarro, E., Letelier, P., Reolid, D., Ramos, I., 2007. Configurable satisfiability propagation for goal models using dynamic compilation techniques. In: Maygar, G., Knapp, G., Wojtkowski, W., Wojtkowski, G., Zupancic, J. (Eds.), Advances in Information Systems Development. Springer, pp. 167–179.
Nilsson, N.J., Tech. Report STAN-CS-92-1412 1992. Toward Agent Programs with Circuit Semantics. Department of Computer Science, Stanford University.
Nilsson, N.J., 1993. Teleo-Reactive programs for agent control. J. Artif. Intell. Res. 1 (1), 139–158, Retrieved from: http://dl.acm.org/citation.cfm?id=1618595.1618602
Nilsson, N.J., 2001. Teleo-Reactive programs and the triple-tower architecture. Electron. Trans. Artif. Intell. 5, 99–110.
Pohl, K., 2010. Requirements Engineering: Fundamentals, Principles, and Techniques, 1st ed. Springer, pp. 830.
Rajan, K., Py, F., McGann, C., 2010. Adaptive control of AUVs using onboard planning and execution. Sea Technol. April, 51–55.
Respect – IT, 2007. A KAOS Tutorial, Available from: http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf
Rolland, C., Souveyet, C., Ben Achour, C., 1998. Guiding goal modeling using scenarios. IEEE Trans. Softw. Eng. 24 (12), 1055–1071.
Sánchez, P., Alonso, D., Morales, J.M., Navarro, P.J., 2012. From Teleo-Reactive specifications to architectural components: a model-driven approach. J. Syst. Softw. 85 (11), 2504–2518.
Sommerville, I., Sawyer, P., Viller, S., 1998. Viewpoints for requirements elicitation: a practical approach. In: IEEE International Conference on Requirements Engineering (ICRE'98), Colorado Springs, CO, p. 1998.
Soto, F., Sánchez, P., Mateo, A., Alonso, D., Navarro, P.J., 2011. An educational tool for implementing reactive systems following a goal-driven approach. Comp. Appl. Eng. Educ., http://dx.doi.org/10.1002/cae.21568.
Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., Jaen, J., González, P., 2011a. Assessing the understandability of collaborative systems requirements notations: an empirical study. In: Proceedings of the International Workshop on Empirical Requirements Engineering (EmpiRE 2011), Trento, Italy, August 30, 2011. IEEE.
Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., González, P., 2011b. CSRML: a goal-oriented approach to model requirements for collaborative systems. Conceptual Modeling – ER 2011. Lect. Notes Comp. Sci. 6998, 33–46.
Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., Jaen, J., González, P., 2012. Analyzing the understandability of requirements engineering languages for CSCW systems: a family of experiments. Inf. Softw. Technol. 54 (11), 1215–1228, http://dx.doi.org/10.1016/j.infsof.2012.06.001.
Weiglhofer, M., 2007. Extended Teleo-Reactive Compiler, Available from: http://www.ist.tugraz.at/staff/weiglhofer/projects/trcompiler/index.html
Wilson, S.P., Kelly, T.P., McDermid, J.A., 1995. Safety case development: current practice, future prospects. Paper presented at the 1st ENCRESS/5th CSR Workshop, Bruges, Belgium.
Winer, B.J., Brown, D.R., Michels, K.M., 1991. Statistical Principles in Experimental Design, 3rd ed. McGraw-Hill Humanities/Social Sciences/Languages, pp. 928.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction, first ed. Kluwer Academic Publishers, Norwell, USA.
Yu, E., 1997. Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), Washington, DC, USA, January 6–8, 1997, pp. 226–235.
Yu, E., Mylopoulos, J., 1998. Why goal-oriented requirements engineering. In: Proceedings of the 4th International Workshop on Requirements Engineering: Foundation for Software Quality (RESFQ 1998), Presses Universitaires de Namur, Namur.
Zelek, J.S., 1995. Teleo-Reactive autonomous mobile navigation. In: Proceedings of CCECE-95.

José Miguel Morales is an Assistant Professor and a Ph.D. student in computer science at the Universidad Politécnica de Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include real time systems, and specification and design of reactive systems. Morales has a master's in information technology engineering from the Universidad de Murcia.

Elena Navarro is an Associate Professor of Computer Science at the University of Castilla-La Mancha (Spain). Prior to this position, she worked as a researcher at the Informatics Laboratory of the Agricultural University of Athens (Greece) and as a staff member of the Regional Government of Murcia, at the Instituto Murciano de Investigación y Desarrollo Agroalimentario. She got her bachelor degree and Ph.D. at the University of Castilla-La Mancha, and her master degree at the University of Murcia (Spain). She is currently an active collaborator of the LoUISE group of the University of Castilla-La Mancha. Her current research interests are Requirements Engineering, Software Architecture, Model-Driven Development, and Architectural Knowledge.

Pedro Sánchez received his Ph.D. degree in computer science from the Technical University of Valencia, Spain, in 2000. Since 1996, he has participated in different projects focused on software engineering and conceptual modeling applied to the development of reactive systems. In 2000, he joined the Systems and Electronic Engineering Division (DSIE) at the Technical University of Cartagena. He is currently an Associate Professor at the Technical University of Cartagena in the field of computer science. His current research interests include software engineering for implementing teleo-reactive systems.

Diego Alonso is an associate professor of Computer Science at the Universidad Politécnica de Cartagena and a member of the DSIE (Division of Systems and Electronic Engineering) research group. His research interests focus on the application of the model-driven engineering approach to the development of component-based reactive systems with real-time constraints. He has a Ph.D. with "Doctor Europaeus" Mention form the Universidad Politécnica de Cartagena.

A FAMILY OF EXPERIMENTS TO EVALUATE THE UNDERSTANDABILITY OF TRISTAR AND I* FOR MODELING TELEO-REACTIVE SYSTEMS

RESUMEN

El enfoque Teleo-Reactivo facilita el desarrollo de sistemas reactivos sin perder de vista los objetivos del sistema. Este artículo presenta TRiStar, una extensión a la notación i* para especificar sistemas Teleo-Reactivos y trata de evaluar si esta extensión supone una mejora en términos de eficacia y eficiencia sobre la notación original cuando se usa para especificar sistemas Teleo-Reactivos. Con tal fin se llevó a cabo una familia de experimentos con estudiantes de último año de ingeniería y con desarrolladores de software experimentados en la que los participantes tenían que rellenar un formulario especialmente diseñado al efecto. Tanto el análisis estadístico de cada uno de los experimentos de la familia por separado como el meta-análisis de la familia de experimentos como un todo permiten determinar que TRiStar es más efectivo y más eficiente que i* como lenguaje de especificación de requisitos para modelar sistemas Teleo-Reactivos.

The Journal of Systems and Software 114 (2016) 82–100

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A family of experiments to evaluate the understandability of TRiStar and *i** for modeling teleo-reactive systems

José Miguel Morales[a,*], Elena Navarro[b], Pedro Sánchez[a], Diego Alonso[a]

[a] Systems and Electronic Engineering Division (DSIE), Universidad Politécnica de Cartagena, Campus Muralla del Mar s/n, Cartagena, Spain
[b] LoUISE Research Group, Computing Systems Department, University of Castilla- La Mancha, Avda. España s/n, 02071 Albacete, Spain

ARTICLE INFO

Article history:
Received 21 July 2015
Revised 1 December 2015
Accepted 31 December 2015
Available online 8 January 2016

Keywords:
Teleo-reactive
*i**
TRiStar
Requirements engineering

ABSTRACT

The teleo-reactive approach facilitates reactive system development without losing sight of *the system goals.*

Objective: To introduce TRiStar as an extension of *i** notation to specify teleo-reactive systems. To evaluate whether the notational extension is an improvement in terms of effectiveness and efficiency over the original language when it is used to specify teleo-reactive systems.

Method: A family of experiments was carried out with final-year engineering students and experienced software development professionals in which the participants were asked to fill in a form designed to evaluate the efficiency and effectiveness of each of the languages.

Results: Both the statistical results of the experiments, analyzed separately, and the meta-analysis of the experiments as a whole, allow us to conclude that TRiStar notation is more effective and efficient than *i** as a requirements specification language for modeling teleo-reactive systems.

Conclusion: The extensions made on *i** have led to TRiStar definition, a more effective and efficient goal-oriented notation than the original *i** language.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

The teleo-reactive paradigm (TR) (Nilsson, 1993) is a goal-oriented approach for modeling systems in which actions, outputs and states are computed as a response to a stimulus received from the system's surroundings and from the system itself. Teleo means "to reach a goal". "Reactive" means "highly sensible to perceptions". As a consequence, the TR approach offers engineers a formal high-level goal-oriented way to develop reactive systems, allowing developers to define behavior without losing sight of the goals and state changes occurring in the environment.

A TR specification can be defined as a set of prioritized condition/action rules. The conditions are defined by inputs from sensors or from a model of the world created by the system. The actions change the world in some way from a physical or logical point of view (the model of the world). The condition of the rule with the highest priority represents the main goal of the system-to-be. At the same time, actions can be TR specifications, thus allowing the creation of hierarchical decompositions of the goals. The subgoals are therefore those objectives to be reached in each of the sub-specifications and are needed to fulfill the main goal. For more details on this topic, see Morales et al. (2012), which gives a systematic review of works published between 1994 and 2011, as well as the extensions provided by Keith Clark with TeleoR (Clark and Robinson, 2014).

Although the TR paradigm has proved useful when it comes to specifying reactive systems (Rajan et al., 2010, Gubisch et al., 2008), it is nonetheless true that developing TR systems is a hard and error-prone task. The main challenges involved have been identified in Morales et al. (2015) and can be summarized as follows:

1. *Rule priorities*: a small change in priorities or order in the rules may lead to a very different system behavior.
2. *Regression property*: a sound TR specification must guarantee that acomplishing a subgoal takes the system closer to reaching a higher priority goal, which in turn takes the system closer to the main goal. The demonstration of this property for a given system is not a trivial issue.
3. *Modularity and encapsulation*: in spite of the fact that the paradigm considers the use of subgoals (allowing a certain degree of encapsulation) the textual representation makes the understandability of the behavior of the system particularly difficult at a single glance.

* Corresponding author. Tel.: +34 967 599 200x2624; fax: +34 967 599 343.
E-mail addresses: josemiguel.morales@upct.es (J.M. Morales), Elena.Navarro@uclm.es (E. Navarro), pedro.sanchez@upct.es (P. Sánchez), diego.alonso@upct.es (D. Alonso).

http://dx.doi.org/10.1016/j.jss.2015.12.056
0164-1212/© 2016 Elsevier Inc. All rights reserved.

4. *Reuse*: as a result of the above, the creation of reusable components has not been a key issue in the evolution of the TR paradigm. The most remarkable exception can be found in Sánchez et al. (2012), in which the authors propose a model-driven approach to obtain architectural components starting from a TR specification.

With the aim of overcoming these difficulties, it would be useful to find a Software Engineering approach to specify the requirements of TR systems. Morales et al. (2015) argue that the most suitable Requirements Engineering technique for modeling TR systems is the Goal-Oriented approach, as both systems share the same foundations (*goals*). In the study cited, two techniques are proposed that use goal-oriented requirements languages to demonstrate that *i** (Yu, 1997) gives better results in terms of understandability. Starting from these results and going deeper into the study of the technique based on *i**, we detected a sort of weaknesses that, if fixed, would improve the understandability, efficiency and effectiveness of *i** as a specification language for TR systems. For this reason, and following the path used in other approaches, such as (Teruel et al., 2011), we propose here an *i** extension that overcomes the limitations mentioned above. This extension, named TRiStar, was first presented in Morales et al. (2015). In the present paper we delve deeper into the definition of TRiStar and analyze the results by means of a family of experiments carried out to compare the efficiency and effectiveness of the original notation using *i** with the TRiStar extension. It is important to clarify that TRiStar extends the *i** notation but does not limit it in any way. Thus, all the expressiveness of the original language is available to deal with topics from the early stages of requirements engineering, such as uncertainty, conflicts among multiple agents or alternative ways of achieving the same goal. All these topics may be very useful when specifying complex TR systems in which several agents collaborate or compete with each other to achieve the goals in an application (see (Clark and Robinson, 2014) for examples of such systems).

The Oxford English Dictionary defines the word "understandable" as "that can be understood; intelligible" (understandable, adj, 2015). The understandability of a given notation is therefore something inherently subjective and linked to the modeler's capacity to understand such notation. In this vein, many studies, besides measuring what can be called "subjective understandability", have looked for other more objective ways of evaluating understandability by means of performance-based measures. For instance Genero et al. (2008) define the concepts used throughout this document as follows:

- *Understandability Time* (UT): The time needed to understand a TR diagram (expressed in minutes).
- *Understandability Effectiveness* (UEffec): The number of correct answers reflects how well the participants performed the required understandability tasks.
- *Understandability Efficiency* (UEffic): The number of correct answers divided by UT relates the understanding performance of the participants to their effort (in terms of time spent).

In this paper we introduce a family of experiments in which the above concepts have been evaluated for each of the notations introduced: *i** and TRiStar. The rest of the paper is organized as follows: Section 2 gives an overview of related works on the development of TR systems and goal-oriented requirements engineering techniques needed to understand the contents of this paper. Section 3 gives a brief introduction to *i** and its use for defining TR systems. Section 4 describes in detail the TRiStar extension, starting from the limitations detected in *i** notation. Section 5 details the family of experiments carried out, while Section 6 describes possible threats to the validity of the experiments. Finally, Section 7 summarizes our conclusions and some worthwhile future lines of research.

2. Related work

The TR paradigm has obtained many important results in distinct fields of research, perhaps with the most valuable outcomes in the Robotics and Artificial Intelligence domain. In Morales et al. (2012) a detailed summary of the existing literature on the TR paradigm is given, including several contributions to TR formalism, platforms for TR program simulation and validation purposes, as well as methodologic and engineering concerns for creating TR programs or generating executable code.

Among the existing Requirements Engineering approaches (Sommerville et al., 1998, Chernak, 2009), Goal Oriented Requirements Engineering (GORE) has been shown to be particularly helpful in many stages of the system development process (Lamsweerde, 2009). In addition, Yu and Mylopoulos state in (Yu and Mylopoulos, 1998, June) that "*some researchers have considered goals to be an important construct in a number of different areas of RE.*". Those areas include, among others, requirements acquisition, clarifying requirements or driving design, which are very useful in the latter stages of requirements specification in TR systems. Morales et al. (2015) state that GORE is the most straightforward choice for developing TR systems, as both paradigms share the same fundamental concept: 'goal'. The choice of the GORE paradigm to specify TR systems is not only based on this coincidence. The search for a graphical notation to help stakeholders to understand the specification of a TR system and avoid wrong interpretations was motivated by the desire to increase the abstraction level. TR systems need a notation which allows the concept of 'goal' to be represented in the most natural possible way and at the same time specifies the rules with the appropriate level of detail. GORE offers both these advantages. Other approaches, such as the rule-based approach (Tsalgatidou et al., 1990), are not suitable as they stay at the same abstraction level as that of the TR program. In addition, the mapping between TRiStar and TR programs means that the corresponding code can be obtained directly, which obviously makes the work of the developers easier. The study in Morales et al. (2015) compares the most common GORE languages (*i** (Yu, 1997) with KAOS (Lamsweerde, 2001)) and concludes that *i** is the best GORE language to specify TR systems. In spite of the advantages of using *i**, the notation has some weaknesses when it comes to specifying TR systems, and this is why we decided to create an extension that would overcome these limitations.

There are many examples in the literature of extensions to well known languages with the aim of adapting them to specific domains. In this context, CSRML (Teruel et al., 2011) (Collaborative Systems Requirements Modeling Language) is a representative extension for *i**, targeting collaborative systems to create the well-known Computer Supported Cooperative Work (CSCW). These systems allow users to do collaborative tasks, communication and coordination, besides other tasks, on common software applications. However, the specification of such systems using traditional Requirements Engineering techniques is rather complicated, while an *i** extension provides the expressiveness needed to specify CSCW more simply.

In Ayala et al. (2005), the authors make a comparative analysis of the original *i** language with its two most widespread variants: Goal-oriented Requirement Language (GRL) (Amyot and Mussbacher, 2003) and the language used in the TROPOS methodology (Bresciani et al., 2004). It also analyzes the following three *i** extensions:

- The REDEPEND tool (Lockerbie and Maiden, 2008), which extends *i** and allows new types of Means-End relationships

using satisfaction arguments, Contribution relationships, and other minor differences. It provides systems engineers with *i** modeling and analysis functions, coupled with additional functionality and the reliability of Microsoft Visio. It provides a graphical palette from which systems engineers can drag-and-drop *i** concepts to develop Strategic Dependency and Rationale models.

- The Formal TROPOS Language. Formal Tropos adds to *i** temporal specification primitives (Fuxman et al., 2004). It allows specifying cardinality constraints in the dependencies among intentional elements and also allows a new dependency type (prior-to) to be defined to specify temporal order between intentional elements.
- In Sutcliffe and Minocha (1999) the authors propose new types of dependencies among actors and intentional elements: responsibility dependencies between an agent and a goal or a task; authority dependencies between two agents; audit dependencies between an agent and a goal or a task; and capability dependencies of an agent with respect to a goal or task.

On the other hand, controlled experiments to determine the understandability of a given notation or language is a widely accepted practice. Jamison and Teng (1993) carried out an experiment to determine the perceived ease of use of several types of textual and graphical database representations. They concluded that graphical notations were more easily and efficiently accessed and the participants declared that graphical representations were much easier to understand.

Lee and Choi (1998) compared a set of conceptual data-modeling languages to determine which gave more accurate and understandable models in the shortest time. The best results were obtained by the Extended Entity-Relationship Model (ERM) and the Object Modeling Technique. Bajaj (2004) studied the influence of the number of metamodel concepts on the readability of schemes created using such metamodels. They carried out an experiment using many variants of the original ERM, each one with a different number of concepts in order to evaluate efficiency, effectiveness and learnability (defined as an improvement in efficiency and effectiveness over time). The results led the authors to conclude that the variants with most concepts allowed higher precision in the domain conceptualization and at the same time were easier to learn, although the time needed to process the schemes was increased significantly.

Many other approaches are based on ERM: in Genero et al. (2008) a set of objective metrics were defined on ER diagrams and an experiment was performed to determine whether these metrics had any correlation with the "subjective understandability", efficiency and effectiveness of ER diagrams. Three of the proposed metrics (number of entity attributes, number of 1:1 relationships, and number of 1:N relationships) were significantly correlated with scheme understandability: the more attributes and relationships a diagram had, the less understandable it turned out to be.

A family of experiments was carried out in Teruel et al. (2012) to compare the understandability of *i** and CSRML when specifying Collaborative Systems in which the users could perform collaborative, communication and coordination tasks. Similarly to the system used in the present study, they used two replicas in which the subjects answered a set of questions related to the understandability of the two notations. The statistical analysis showed that the specifications made by CSRML scored higher than *i**, especially in collaboration aspects. The study concluded that in terms of understandability CSRML outdid *i** as a specification language for collaborative systems.

More recently, a controlled experiment was performed in Morales et al. (2015) to determine the understandability of *i** versus KAOS as a language for specifying TR systems. The results showed that both languages obtained similar scores in terms of understandability, although *i** notation stood out slightly. The statistical analysis of the results led to the conclusion that *i** notation was more understandable than KAOS as a specification language for TR systems.

Following the strategy defined in Morales et al. (2015), the aim of the present study is to statistically validate whether or not the notational extensions are an improvement of the original *i** notation by means of a family of experiments

Table 1
Mapping concepts between *i** and TR.

*i**	*TR*
Main Agent	System-to-be
Agent	Sensor
Goal	Goal
Task	Action
Resource	Percept
Resource Dependency	Condition

3. Previous background: *i** for TR system requirements specifications

The *i** framework guides the stakeholders through the different phases of the software development process, namely from the early requirements analysis up to the detailed design. As already mentioned, *i** can also be employed to specify the requirements of TR systems.

The work by Morales et al. (2015) introduces the language and gives a detailed description of the technique developed for specifying TR systems. Table 1 briefly summarizes the mapping from i* concepts to TR concepts, which constitutes the kernel of the technique described in the work.

Fig. 1 shows the application of this approach by using a simplified version of the *i** specification of one of the examples used in the family of experiments described in Section 5.2. The example consists of a drone that always goes back to its origin, no matter where it has taken off from.

The details of the application of the technique described in Morales et al. (2015) to the *i** specification shown in Fig. 1 are given below. The resulting TR program is shown in Code 1:

- Every *i** goal in Fig. 1 becomes a TR goal (in bold text in Code 1, as for instance, *Land* or *MaintainHeightOK).*
- Every *i** agent becomes a sensor or device, such as *GPS* or *altimeter*.
- Every *i** resource that has a dependency relationship with an agent becomes a condition monitored by the homologous sensor, such as *ground* with the *altimeter. i** goals or tasks that lack

Code 1
TR program for Fig. 1

```
DronAtOrigin:
overOrigin → Land
NOT(height > hMax) AND NOT(height < hMin) → followGPS
height > hMax OR height < hMin → MaintainHeightOK
MaintainHeightOK:
height < hMin → go_up
height > hMax → ReduceHeight
ReduceHeight:
True → go_down
Land:
ground → nil
True → go_down
```

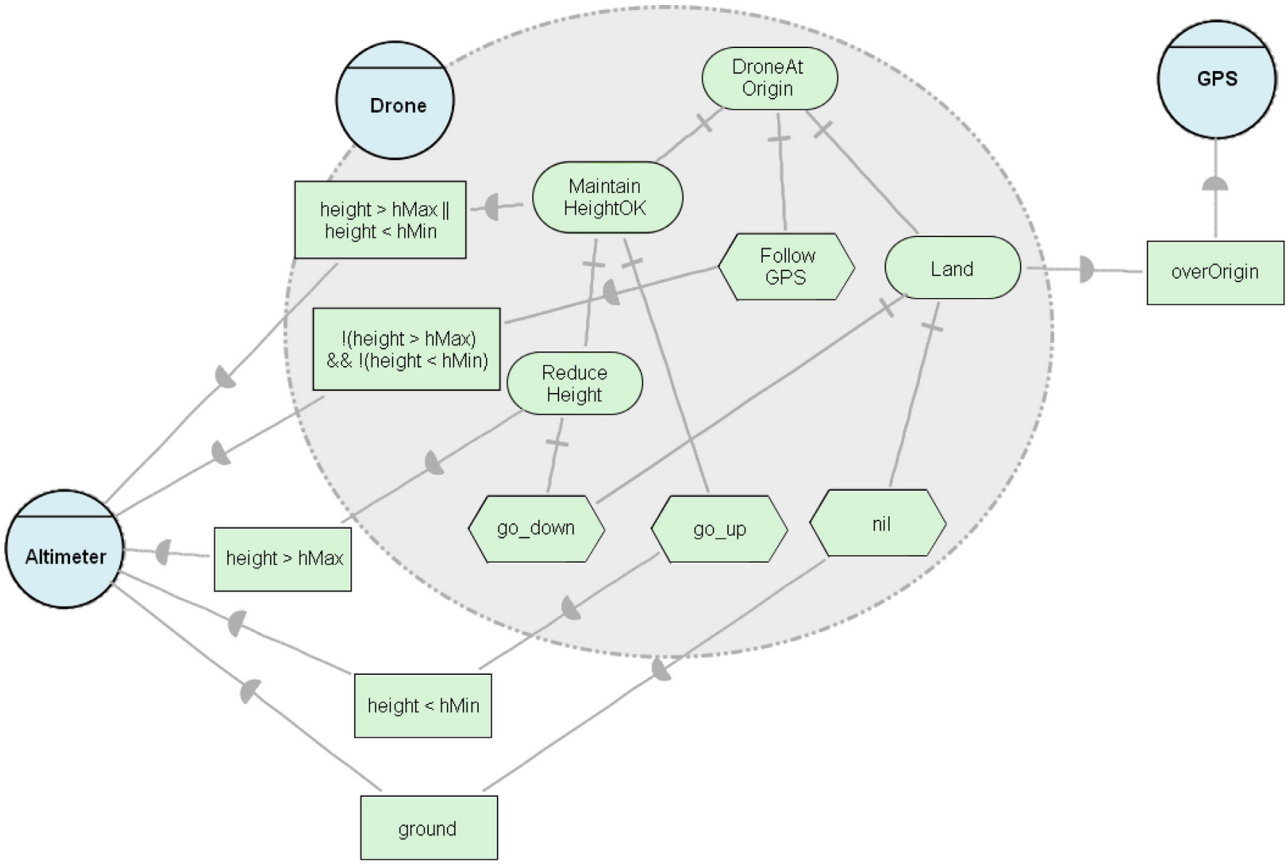


Fig. 1. Drone application specification using $i*$.

dependency relationships, such as *go_down*, are mapped to TR rules whose condition is always true (*True* → *goal/action*).

- Every i^* task, such as *followGPS*, becomes an action.
- i^* tasks and goals linked by a task-decomposition link to an i^* goal become rules of the same TR goal. For example, the *Land* goal is decomposed into the tasks *go_down* and *nil*. Therefore, two rules are created for the TR goal named *Land*: one whose action is *go_down* and another whose action is *nil*, as shown in Code 1.
- The relative position of the items in the i^* specification states the priority of the rules in the TR program. For example, *Land* is drawn farther to the right than *followGPS* (see Fig. 1). As can be seen in Code 1, the rule whose action is *Land* is over the rule whose action is *followGPS* because it has higher priority.

3.1. Shortcomings of i* for TR systems

As shown in the previous section, it is possible to specify the requirements of a TR system using i^*. Although the validity of the proposed mapping between i^* and TR programs has been established in previous works (Morales et al., 2015, Morales et al., 2015), in this last paper the authors pointed out some limitations found in applying such a technique and briefly presented an extension to i^* named TRiStar, which aims to overcome them. We firstly summarize these limitations by an illustrative example, while the following section describes the enhancements provided by TRiStar for specifying TR systems employing i^*.

- **S1**. Setting the priority by using the order in which tasks or goals refining a goal are positioned in the diagram constrains the likely position of subtasks or subgoals in it. Occasionally, this may result in messy diagrams hard to interpret. In addition, it is difficult to automatically process a diagram in which the relative position of two items has an important meaning. The example shown in Fig. 2, a variation of Fig. 1, will help us to explain this shortcoming. Task *go_down* must be placed far away from the *Land* goal because it needs to be on the left of *go_up,* as the priority of *go_down* when refining *MaintainHeightOK* is lower than that of *go_up*.
- **S2**. If the same task is involved in two or more goals, it may then depend on different resources when refining one of the goals. This may cause ambiguity when obtaining the conditions of the associated TR rule. Considering the i^* specification of the drone shown in Fig. 2, it can be seen that it is very similar to that of Fig. 1. In this case we introduced a goal named *ReduceHeight* to avoid the ambiguity around the *go_down* task. If this artificial goal is not used (note that there is no goal *IncreaseHeight*, as there is no possible ambiguity with the *go_up* task), it is not possible to say whether *go_down* depends on *height* > *hMax* when refining *MaintainHeightOK* or when refining *Land*, or in both cases. Code 2 shows a TR program that fits this specification. Note the condition of the lowest rule in the subgoal *Land*; a drone programmed with Code 2 would crash when it flew over its origin. When *overOrigin* became true, the subgoal

Code 2
A TR program fitting the specification in Fig. 2

```
DronAtOrigin:
overOrigin → Land
NOT(height > hMax) AND NOT(height < hMin) → followGPS
height > hMax OR height < hMin → MaintainHeightOK
MaintainHeightOK:
height < hMin → go_up
height > hMax → go_down
Land:
ground → nil
height > hMax → go_down
```

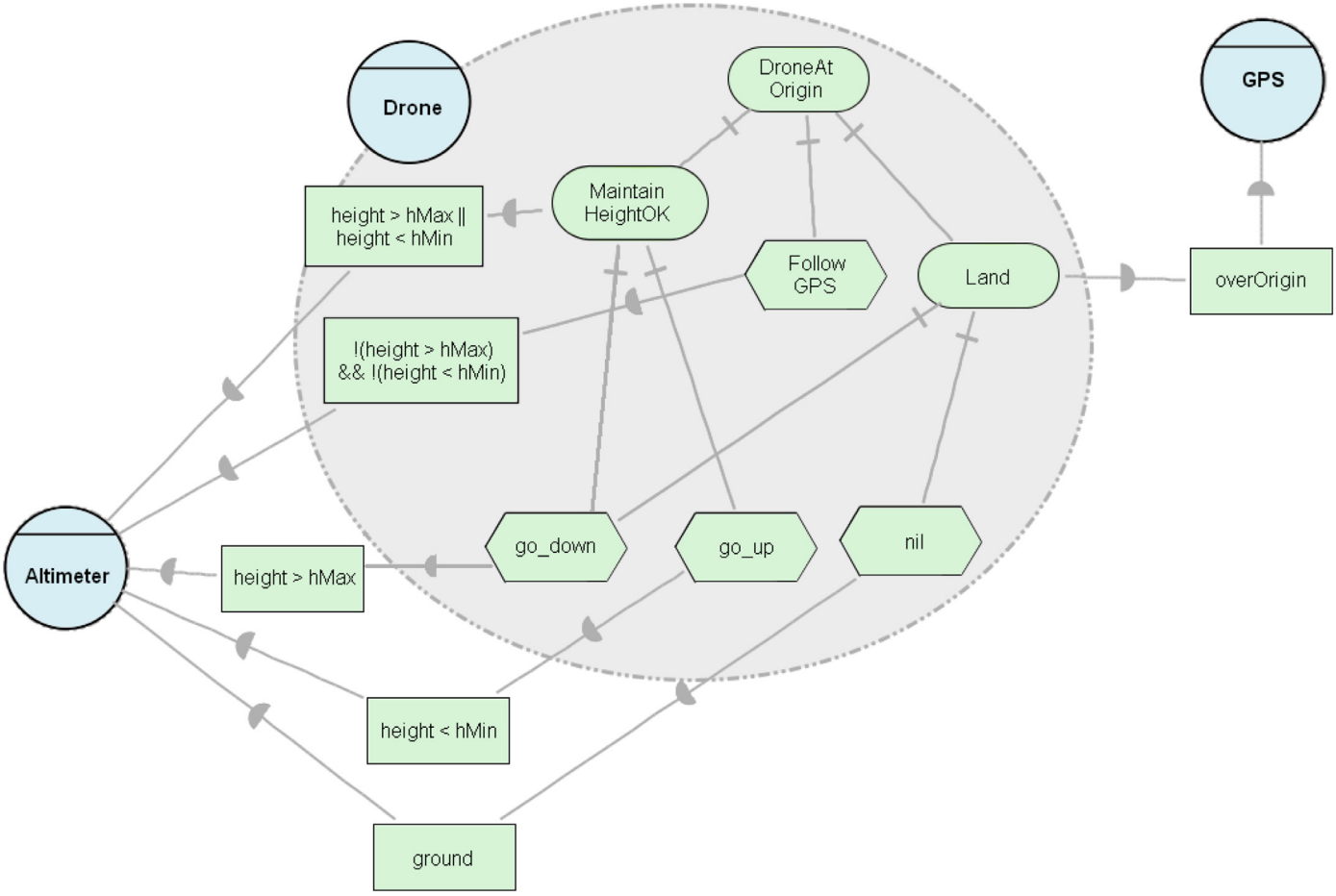


Fig. 2. Ambiguous i^* specification.

Land would take control, but as none of the conditions of the rules in *Land* are actually true, the drone would do nothing and thus would fall to the ground.

We got around this problem in the *i** version shown in Fig. 1 by using the additional subgoal *ReduceHeight*. The resource *height* > *hMax* depends on this new subgoal, which is decomposed into only one task, freeing this task (*go_down*) from dependencies. With this alternative specification a correct TR program can be generated, but the extra item needed reduces the diagram's readability.

- **S3**. The conditions of TR rules are usually composed of logical combinations of percepts given by the sensors. In *i** there is no way to graphically represent a Boolean combination of some of the percepts provided by sensors. Retaking the example shown in Fig. 1, in *i** there is no symbol to represent a dependency on *height* > *hMax* OR *height* < *hMin*, for instance. We got around this limitation by adding resources that are labeled with the Boolean expression we wanted to represent inside the boundary of the system. These expressions may become difficult to read in systems with a certain degree of complexity.

4. TRiStar enhancements

To overcome the limitations identified in the previous section, an extension to *i** is proposed. The following three main new features compose this extension, named TRiStar:

- E1: Prioritized decomposition links.
- E2: Dependent decomposition links.
- E3: Logical resources.

In this section these new features are described in depth.

- **E1**: **Prioritized decomposition links**. To avoid relying on the relative position of the diagram elements when information about their priority is needed (shortcoming S1), a new decomposition link has been defined. This new type of decomposition link provides the priority of the rule whose subgoal or task is at the end of the link by changing its own representation. The standard *i** decomposition link has a short perpendicular line at the end that is closer to the goal being decomposed. The new decomposition links have as many of these short lines as needed to show the priority of the subgoal or subtask. One line means the lowest priority. The more lines a decomposition link has, the higher priority its related subgoal or subtask has. Now, the position of these lines on the diagram does not have any intended meaning. Fig. 3 shows a diagram in which these new decomposition links can be seen. Note that although the *highest_priority_subgoal* is in the middle of the subgoals, it has the highest priority as its decomposition link has three short perpendicular lines. In the TR program corresponding to this specification, the rule containing *highest_priority_subgoal* would be the uppermost rule in *Goal*, then *medium_priority_subgoal* would be next, and finally *lowest_priority_subgoal*.

  In order to avoid scalability problems when a goal is refined into many tasks or subgoals, the short perpendicular lines can be substituted by a circle with the priority specified in its interior, with '1' being the lowest priority. Although this notation facilitates the insertion of new subtasks or subgoals and avoids the excessive cluttering that can be generated by the addition of many perpendicular lines, we recommend the use of short lines to maintain the similarity with the original *i** notation.

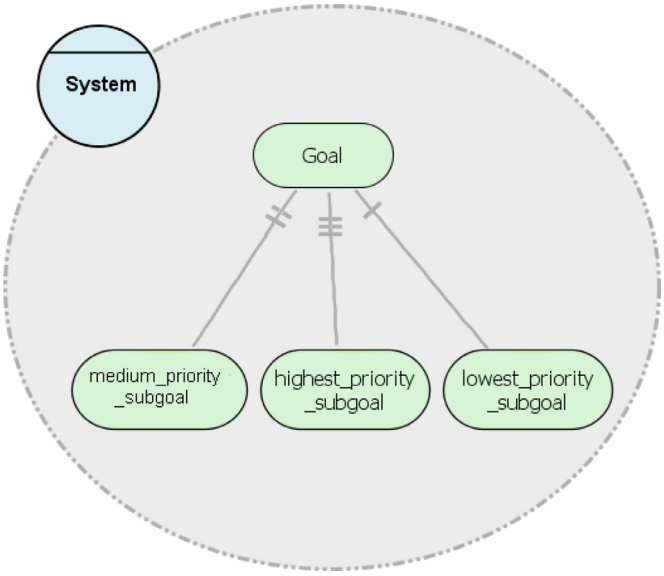


Fig. 3. Prioritized decomposition links.

- **E2**. Dependent decomposition links have been introduced to avoid linking dependencies directly to subgoals or tasks (limitation S2). It is worth remembering that the condition of a rule in a TR program cannot be generated from a dependency on a task or subgoal alone, but the relationship between the task and the goal it refines is also needed. This relationship is obviously represented by the decomposition link that connects them and explains why a dependency link between the decomposition link and the resource has been introduced. For example, as Fig. 4 shows, the decomposition link between *Goal* and *Subgoal* depends on *resource1*, which is in *Sensor1*'s boundary. Similarly, the decomposition link between *Goal* and *Task* depends on *resource2*, which is in *Sensor2*'s boundary.
  Note that the new prioritized decomposition link has been used in the example. As the link between *Goal* and *Subgoal* has a higher priority than the other, the first rule in the TR program is the one whose condition is *resource2*. Code 3 shows the TR program generated from this specification.
- **E3**. To overcome limitation S3, we introduced a specialization of *i** resources to represent the logical combinations of percepts. These specialized resources are related to all the percepts they involve by using directed dependency links. In addition, the *logical resource* is given a name, which acts as an alias for such combinations of percepts. A table is provided to link every name with its logical expression. Fig. 5 shows an example of this new kind of resource.
  The decomposition between *Goal* and *Subgoal* depends on a logical resource which is the result of an OR operation between *resource1* and *resource2*. As the logical resource uses the percepts *resource1* and *resource2* as operands, dependency links are established from the logical resource to its two operands. The expression represented by *LogicalResource* can be seen in the table just under *System*'s boundary. Code 4 shows the TR program that corresponds to this specification:
  Lastly, although it cannot be considered an extension to *i**, the dependencies between a percept and the sensor that generates it are represented by inserting the resource inside the agent's boundary, as already shown in Figs. 4 and 5. In this way, only a dependency link is needed to represent the condition of a rule, unlike plain *i** specifications, which require two such links.

Code 3
TR program for Fig. 4

```
Goal:
resource1 → Subgoal
resource2 → task
```

Code 4
TR program for Fig. 5

```
Goal:
resource1 OR resource2 → Subgoal
True → task
```

The mapping from a TRiStar specification to a TR program is very similar to that of *i**. In fact, Table 1 still remains valid. There are however some differences:

1. The main TRiStar agent is transformed into the TR system-to-be. The main TRiStar agent is the one that has the goal that the final system wants to achieve in its boundary.
2. TRiStar goals become TR goals.
3. TRiStar tasks are specified as TR atomic actions.
4. TRiStar resources (except logical resources) become percepts generated by sensors.
5. A logical resource will be translated into the expressions found in the table associated to its alias.
6. Considering that a TR rule is defined as *condition* → *goal/action*, every TRiStar resource having a decomposition link as a dependee is transformed into a TR rule whose condition is that resource and its action is the task or goal that is at the end of the decomposition link. A decomposition link not depending on any resource is transformed into a rule of the form *True* → *goal/action*.
7. Since a TR goal is defined as a set of prioritized TR rules, a TRiStar goal being refined into goals and tasks through task-decomposition links is transformed into a TR goal formed by as many rules as TRiStar tasks or goals refine the original *i** goal.

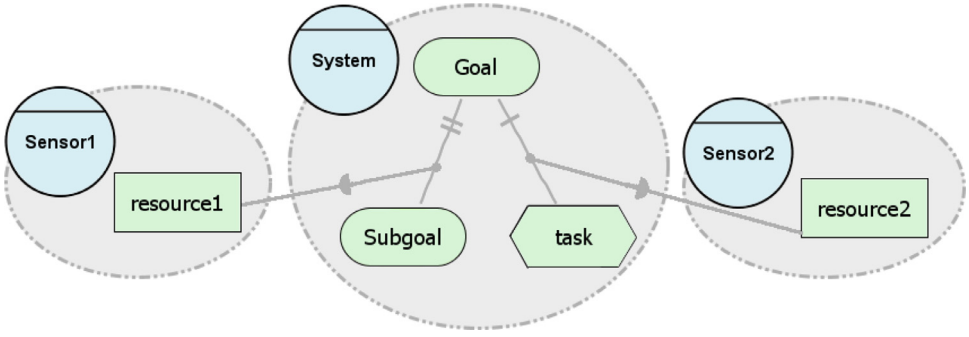


Fig. 4. Dependent decomposition links.

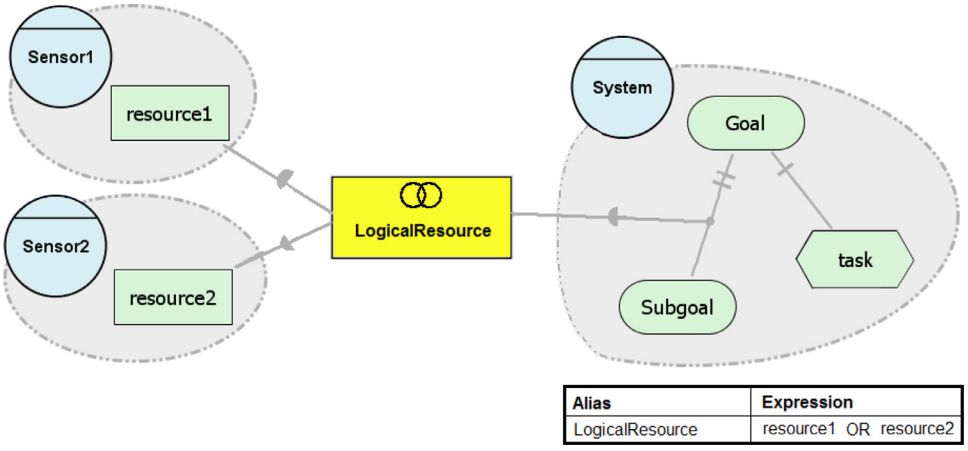


Fig. 5. Use of logical resources.

8. Rule priority, given by the order of the rules in TR programs, is specified in TriStar diagrams by using prioritized decomposition links. So, the tasks or goals placed at the end of the highest priority decomposition link will be translated into the action of the highest priority rule in the TR program. The resource on which the highest priority decomposition link depends will be transformed into the condition of that rule.

Fig. 6 shows the specification of the same drone as that in Fig. 1 but using TRiStar.

All the proposed extensions have been employed in this example:

- Prioritized decomposition links allow positioning *go_down* near both *MaintainHeightOK* and *Land*, which helps keep the diagram organized and uncluttered, with no crossing lines.
- Dependent decomposition links enable the artificially created subgoal *ReduceHeight* to be removed. The resource *height* > *hMax* depends on the link between *MaintainHeightOK* and *go_down* and not on the link from *Land*, so that the ambiguity of the rule condition is eliminated.
- Two logical resources have been introduced: *HeightOK* and *HeightKO,* whose expressions can be found in the table in Fig. 6. The aliases make it easier to understand the conditions that apply to the rules involved.

Code 5 shows the TR program obtained by applying the mapping rules described in Section 4 to the TRiStar specification depicted in Fig. 6:

- Every TRiStar goal in Fig. 6 becomes a TR goal (in bold text in Code 5, as for instance, *Land* or *MaintainHeightOK).*
- Every TRiStar agent becomes a sensor or device, such as *GPS* or *altimeter*. The resources within their boundary are mapped to the percepts provided by each of them. See *ground* inside *Altimeter*'s boundary, for example.

Code 5
TR program for Fig. 6

```
DronAtOrigin:
atOrigin → Land
NOT(height > hMax) AND NOT(height < hMin) → followGPS
height > hMax OR height < hMin → MaintainHeightOK
MaintainHeightOK:
height < hMin → go_up
height > hMax → go_down
Land:
ground → nil
True → go_down
```

- A decomposition link depending on a resource, logical or not, becomes a rule whose condition is the percept represented by the resource and its action is the goal or task at the end of the decomposition link. See for example in Fig. 6 the link between *DronAtOrigin* and *Land*, which depends on *atOrigin*. It is mapped to the first rule in goal *DronAtOrigin* as can be seen in Code 5.
- Logical resources are mapped to the conditions corresponding to their aliases in the table. For instance, *HeightKO* is mapped to height > hMax OR height < hMin.
- Decomposition links that lack dependency relationships, such as that between *Land* and *go_down*, are mapped to TR rules whose condition is always true (*True* → *goal/action*).
- Just as in the *i** case, every TRiStar task, such as *followGPS*, becomes an action.
- TRiStar tasks and goals linked by a task-decomposition link to a TRiStar goal become rules of the same TR goal. For example, the goal *Land* is decomposed into the tasks *go_down* and *nil*. Then, a TR goal named *Land* appears with two rules: one whose action is *go_down* and another whose action is *nil*, as shown in Code 5.
- The number of short perpendicular lines in the decomposition links states the priority of the rules in the TR program. For example, the decomposition link from *DronAtOrigin* to *Land* has three of these perpendicular lines, while the decomposition link between DronAtOrigin and *followGPS* has only two (see Fig. 6). As can be seen in Code 5, the rule whose action is *Land* appears before the rule whose action is *followGPS*.

5. The family of experiments

In order to assess the *understandability* of both the newly created TRiStar extension and *i** when modeling the software requirements of TR systems, a family of experiments (see Fig. 7) performed to compare both of them based on the guidelines described by Kitchenham et al. (2002). In this section we will describe the context, the design and how the experiments were conducted. All the three members of the family were designed in a similar way, so that only one description is given.

5.1. Experimental context

The main goal of this family of experiments was to study the requirements specifications of TR systems using both *i** and TRiStar and evaluate their effectiveness and efficiency from the perspective of requirements engineering researchers, using undergraduate B. Sc. students and experimented software developers as subjects.

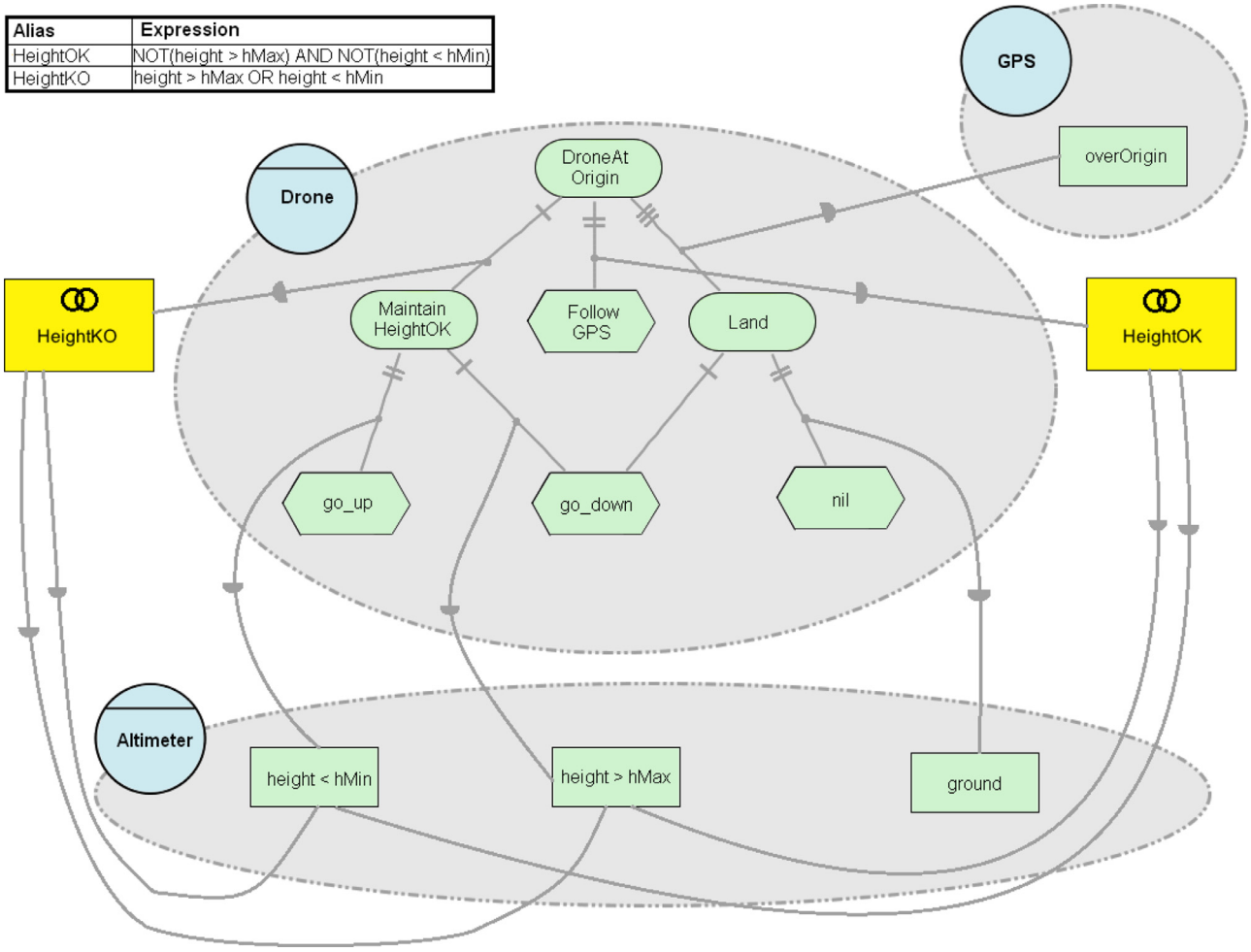


Fig. 6. Drone specification using TRiStar.

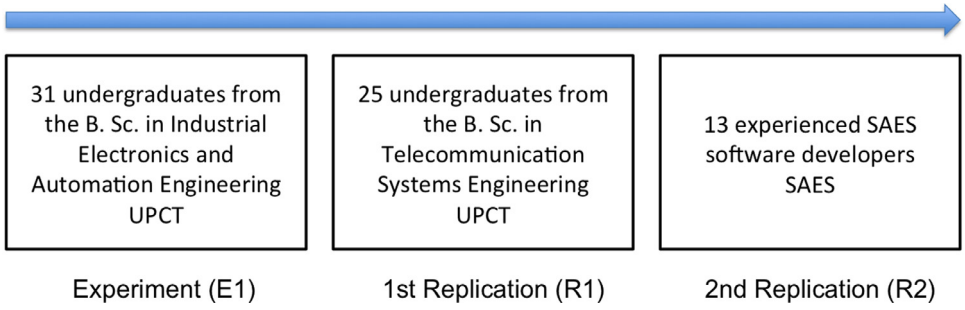


Fig. 7. Chronology of the family of experiments.

To achieve this goal, the null hypotheses shown in Table 2 were defined using the Goal Question Metric template (Basili et al., 1994).

As Table 2 shows, the subjects in the experiments were engineering students and software development professionals. All were familiar with requirements engineering but none had previously used either *i** or any other GORE language and none had any previous experience of TR systems.

The *Sociedad Anónima de Electrónica Submarina* (SAES) collaborated in this study and allowed almost all their software engineers to be subjects for the second replication. SAES is a Spanish company specializing in underwater acoustics and develops undersea security and environmental protection systems. The company has more than 25 years of experience in developing advanced technology in the fields of Sonar, Acoustic Signal Processing, Underwater Signature Measurement and Management, Simulation and Training. Highly skilled and experienced engineers and scientists in various disciplines make SAES an innovative and competitive company in both national and international markets.

5.2. Experimental design

All the experiments in this family were aimed at evaluating the understandability of the requirements specification of two different TR systems specified by both *i** and TRiStar. The first system consisted of a drone which was able of deliver a package to a destination and go back to its origin, always keeping at a safe height. GPS informs the drone when it is flying over its origin, over the destination and gives it directions to reach both places. Weight is

Table 2
Main features of the family of experiments.

Null-Hypotheses	$H_{UEffec0A}$: *i** has the same average score for understandability effectiveness as TRiStar when specifying TR requirements. $H_{UEffec1A}$: $H_{UEffec0A}$ $H_{UEffec0B}$: The understandability effectiveness average score is the same regardless of the domain used in the experiment. $H_{UEffec1B}$: $H_{UEffec0B}$ $H_{UEffec0AB}$: *i** has the same understandability effectiveness average score as TRiStar when specifying TR systems requirements, regardless of the domain used in the experiment and viceversa. $H_{UEffec1AB}$: $H_{UEffec0AB}$ $H_{UEffic0A}$: *i** has the same average score for understandability efficiency as TRiStar when specifying TR requirements. $H_{UEffic1A}$: $H_{UEffic0A}$ $H_{UEffic0B}$: The understandability efficiency average score is the same regardless of the domain used in the experiment. $H_{UEffic1B}$: $H_{UEffic0B}$ $H_{UEffic0AB}$: *i** has the same understandability efficiency average score as TRiStar when specifying TR systems requirements, regardless of the domain used in the experiment and viceversa. $H_{UEffic1AB}$: $H_{UEffic0AB}$		
Dependent variables	Understandability effectiveness of requirements modeling languages, measured by UEffec Understandability efficiency of requirements modeling languages, measured by UEffic		
Independent variables	The *system* the models specify and the *language* used to specify these models		
Location	ETSII at UPCT (Cartagena, Spain)	ETSIT at UPCT (Cartagena, Spain)	SAES Facilities (Cartagena, Spain)
Date	February 2015	February 2015	February 2015
Subjects	31 undergraduates of the B.Sc. in Industrial Electronics and Automation Engineering (16 Group 1; 15 Group 2)	25 undergraduates of the B.Sc. in Telecommunication Systems Engineering (13 Group 1; 12 Group 2)	13 experienced software development professionals (6 Group 1; 7 Group 2)

Table 3
Experimental design.

		System	
		Drone	Football player
Language	**TRiStar**	Group 1	Group 2
	i*	Group 2	Group 1

monitored so that the drone knows whether it is loaded or not and an altimeter is in charge of updating height information. The actions the drone is able to carry out are limited to going up, going down, following GPS directions and releasing the load.

The second system was a variation of one of the systems used in Morales et al. (2015), which was a soccer robot which plays in defensive positions. When the robot considers the danger is over, it goes back to its own goal. The robot can find the ball and knows who is controlling the ball: i.e. himself, an opponent or a teammate. The robot can identify other members of its own team, in fact, its main goal is to keep the ball in his team's possession. To do this, the robot can turn, move forward, dribble and kick the ball.

The subjects in all the tests were divided into two groups, Group 1 and Group 2, each group using one of the languages. Table 3 summarizes these decisions:

Dividing the subjects into 4 different groups starting from the combination of the two independent variables makes up a 2×2 factorial design with confounded interaction (Winer et al., 1991) and thanks to this combination system-language among the groups, the *learning effect* is cancelled. Every subject answered a brief questionnaire on both models. The questionnaire (see Appendix) consisted of some TR program fragments from the presented models using the appropriate mapping. In every fragment there was an element missing and the subject was asked to fill in the blanks. They were also asked to record the time they need to answer the questions. With this information, effectivenes (UEffec) was calculated as the number of correct answers divided by the total number of questions. Efficiency (UEffic) was calculated as UEffec divided by the number of minutes required to fill in the questionnaire. In the final question the subjects were also asked which language they thought was most understandable in specifying TR systems.

Since all the participants had previous experience in requirements engineering but not in GORE or TR systems some filtering criteria were laid down to eliminate any subjects whose previous experience would give them an advantage that could adulterate the results. Those that matched any of the following criteria were discarded:

- Those more than 5 years older than the group's average age.
- Previous experience in GORE languages.
- NO previous experience in requirements engineering
- Previous experience in TR systems.

Finally, each subject was interviewed on his opinion of the questions and the answers were recorded for subsequent analysis.

5.3. Test procedure

The tests were carried out in three different sessions: one for the original test and two more for the replicas. The first session took place in the Industrial Engineering School of the University of Cartagena and the second in the Telecommunications Engineering Faculty of the same university. The third session took place in the SAES facility in Cartagena.

The same procedure was used for all three sessions. An instructor initially briefed the subjects on TR systems, *i** and TRiStar, and how to represent TR systems requirements in both notations. The examples used in the experiment, the drone and the football player, were also described. The time needed for the complete briefing was about 20 min. Before giving the models to the subjects, the following information was obtained:

- For subjects in groups G1 and G2:
  - Gender (Male/Female)
  - Age
  - Qualifications
  - Average score
  - Have you had any previous experience of working with goal-oriented requirements engineering?
  - Have you had any previous experience of working with any other requirements engineering technique?
  - Have you had any previous experience of working with teleo-reactive systems?
- For subjects in group G3:
  - Gender (Male/Female)
  - Age
  - Years of experience in software development
  - Have you had any previous experience of working with goal-oriented requirements engineering?
  - Have you had any previous experience of working with any other requirements engineering technique?
  - Have you had any previous experience of working with teleo-reactive systems?

The subjects were asked to record their exact start and end times from an online clock projected on a screen.

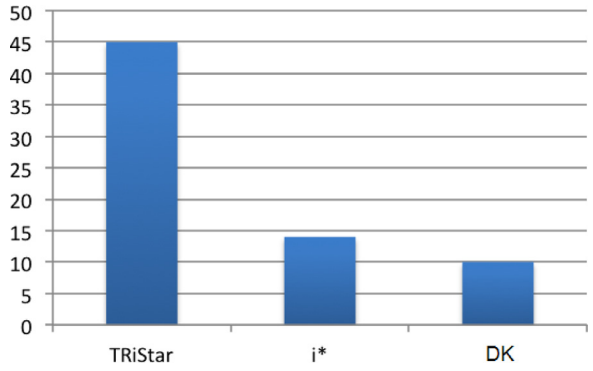


Fig. 8. Subjective Understandability.

Table 4
Levene's test for UEffec in E1.

F	df1	df2	Sig.
0.814	3	58	0.493

5.4. Analysis of the results

Fig. 8 shows the participants' subjective preference in the form of the combined answers for the three experiments to the question "In your opinion, which language is more understandable?".

As can be seen in Fig. 8, the answers show that TRiStar is more understandable than *i**. 45 subjects declared that they found TRiStar more understandable, vs less than 15 who preferred *i** or the 10 people who did not give a clear answer (Don't Know). As regards the effictiveness and efficiency aspects; the factorial design of the experiments in this family makes them particularly appropriate for a two way ANOVA test in order to analyze the results. The three main assumptions for this test are the following:

- Independence of observations.
- The distribution of the residuals must be normal.
- Homocedasticity: homogeneity of variances.

In the following subsections the original experiment and its replications will be analyzed to check firstly whether these assumptions are achieved or not. In those cases in which the assumptions are achieved, the results of the ANOVA tests will be presented and anlyzed. The way in which the data was obtained guarantees the independence of the observations, so that only normal distribution and homocedasticity need be proven. The results were analyzed by IBM SPS Statistics v. 22.

5.4.1. Original experiment (E1)

The answers of 5 participants in this test were discarded from the sample either because they did not comply with one of the criteria in Section 5.2 or they had not completed the questionnaires. The total number of remaining subjects in the sample was 31. According to the central limit theory (Grinstead and Snell, 2006), the normality of the sample may be assumed.

Table 6
Levene's test for UEffic in E1.

F	df1	df2	Sig.
0.238	3	58	0.869

UEffec: as can be seen in column "Sig." in Table 4, Levene's test (Levene, 1960) for homogeneity of variances provides a *p*-value of 0.493, allowing us to assume the homocedasticity of the sample. This test was designed to fit the two-way ANOVA test to be performed: language + system + language * system, each of these elements corresponding to one of the three null hypotheses to be evaluated ($H_{UEffec0A}$, $H_{UEffec0B}$ and $H_{UEffec0AB}$).

The results provided by the ANOVA test are shown in Table 5.

As the *p*-value obtained for language is 0.047 (see column "Sig.") and therefore less than $\alpha = 0.05$ $H_{UEffee0A}$ can be rejected and it can be concluded that there is a statistically significant difference between the UEffec results obtained from *i** and those obtained from TRiStar. On the other hand, as the *p*-values for system and language*system are much bigger than α, neither $H_{UEffee0B}$ nor $H_{UEffee0AB}$ can be rejected. We can thus be sure that language influences UEffec, but neither the system nor the combination of language and system does so.

To calculate the confidence interval of the mean differences between *i** and TRiStar: [−0.15986, −0.00168], as all the values in the interval are less than 0, we can say with a 95% confidence level that the effectiveness of TRiStar is higher than that of *i** when modeling TR systems.

Table 6 shows the homocedasticity of the sample for UEffic as it provides a *p*-value of 0.869.

Table 7 shows the results of the ANOVA test. As with UEffec, $H_{UEffie0A}$ may be rejected but $H_{UEffie0B}$ or $H_{UEffie0AB}$ may not, given the *p*-values obtained for language (0.029), system (0.309) and language*system (0.091). Then, as in the case of effectiveness, we can conclude that the language used does affect the efficiency, but the system or the combination of language and system does not.

Once we know that language does affect UEffic, we will obtain the confidence interval of the mean differences between *i** and TRiStar in order to determine which language obtains the best results. The calculated interval is [−0.09175, −0.00373] and we can conclude at a 95% confidence level that TRiStar is more efficient than *i** when specifying TR systems requirements.

5.4.2. First replication (R1)

For the first replication of the experiment, after discarding 6 students that did not comply with the criteria in Section 5.2, we had a sample of 25 subjects, whose normality had to be shown as the sample size was less than 30. As the ANOVA test is robust before moderated normality deviations, a graphical proof of the

Table 5
ANOVA results for UEffec in E1.

Source	Type III sum of squares	df	Mean square	*F*	Sig.
Model	33.907[a]	4	8.477	415.250	0.000
Language	0.085	1	0.085	4.154	0.047
System	0.016	1	0.016	0.797	0.376
Language* system	0.012	1	0.012	0.573	0.453
Error	0.980	58	0.020		
Total	34.887	62			

[a] *R* Squared = 0.972 (Adjusted *R* Squared = 0.970).

Table 7
ANOVA results for UEffic in E1.

Source	Type III sum of squares	df	Mean square	*F*	Sig.
Model	1.388[a]	4	0.347	47.809	0.000
Language	0.036	1	0.036	5.008	0.029
System	0.008	1	0.008	1.052	0.309
Language * system	0.021	1	0.021	2.953	0.091
Error	0.421	58	0.007		
Total	1.809	62			

[a] *R* Squared = 0.767 (Adjusted *R* Squared = 0.751).

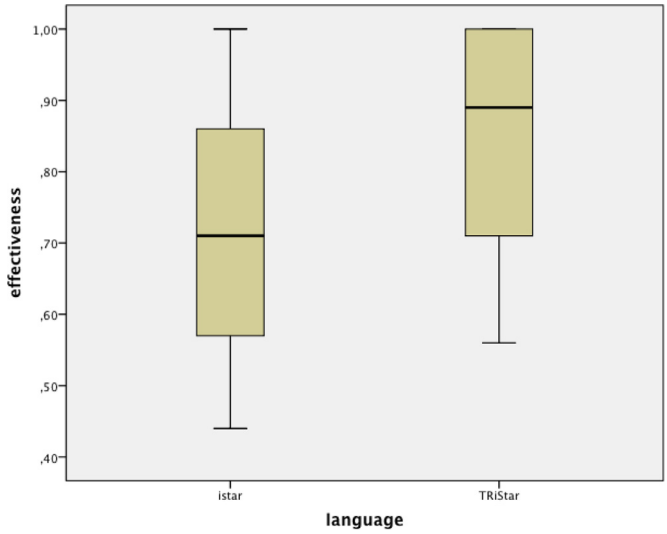


Fig. 9. Normal distribution of UEffec in R1.

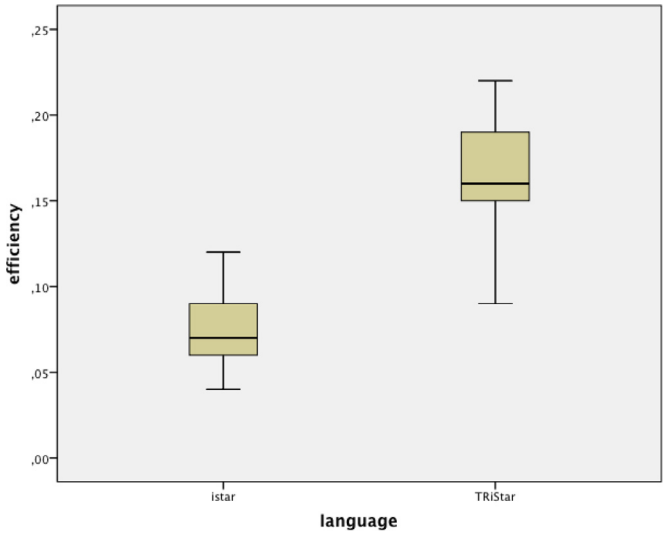


Fig. 10. Normal distribution of UEffic in R1.

Table 8
Levene's test for UEffec in R1.

F	df1	df2	Sig.
0.16	3	46	0.922

Table 10
Levene test for UEffic in R1.

F	df1	df2	Sig.
2.336	3	46	0.086

distribution was enough. Fig. 9 contains a box graph showing the normality of the UEffec distribution:

As in the case of E1, a Levene's test was performed to check the homocedasticity of the samples. This test is also designed to ensure the homogeneity of variances for language, system and the combination of both (language * system). The results are shown in Table 8 in which a *p*-value of 0.922 can be seen to prove the homogeneity of the error variances.

After checking the assumptions, the ANOVA test was performed and the results are shown in Table 9. In this case, the *p*-values displayed in column "Sig." for language (0.017), system (0.437) and language*system (0.101) support the same conclusions as in E1: only language affects the effectiveness of the specification.

To show that TRiStar provided a better UEffec value, the confidence interval of the mean differences between *i** and TRiStar was calculated: [−0.2152, −0.02]. As all the values in the interval were lower than 0, TRiStar obtained the best language effectiveness values. In other words, TRiStar is more effective when specifying TR systems requirements.

Fig. 10 shows the normality of the UEffic samples:

Homocedasticity was proven again using Levene's test (see Table 10). The small p-value (0.086) obtained was still higher than 0.05 and therefore the null hypothesis of the homogeneity of variances could be assumed.

Table 11 summarizes the results of the ANOVA test to analyze UEffic in this experiment:

Table 9
ANOVA results for UEffec in R1.

Source	Type III sum of squares	df	Mean square	*F*	Sig.
Model	29.151[a]	4	7.288	254.738	0.000
Language	0.177	1	0.177	6.188	0.017
System	0.018	1	0.018	0.616	0.437
Language * system	0.080	1	0.080	2.801	0.101
Error	1.316	46	0.029		
Total	30.467	50			

[a] *R* Squared = 0.957 (Adjusted *R* Squared = 0.953).

Table 11
ANOVA results for UEffic in R1.

Source	Type III sum of squares	df	Mean Square	*F*	Sig.
Model	0.789[a]	4	0.197	230.751	0.000
Language	0.084	1	0.084	98.500	5.17×10^{-13}
System	0.000	1	0.000	0.198	0.658
Language * system	0.002	1	0.002	1.933	0.171
Error	0.039	46	0.001		
Total	0.829	50			

[a] *R* Squared = 0.953 (Adjusted *R* Squared = 0.948).

Table 12
Results for UEffec in R2.

	Levene's test	Kruskal–Wallis
Language	0.079	0.046
System	0.359	0.217

Table 13
Levene's test for UEffic in R2.

F	df1	df2	Sig.
1.616	3	22	0.214

The *p*-value for language is 5.17×10^{-13} so that $H_{UEffie0A}$ can be rejected. The p-values for system (0.658) and language*system (0.171) do not allow us to reject $H_{UEffie0B}$ or $H_{UEffie0AB}$ thus reaching the same conclusion as in E1: language affects the efficiency of the specifications but system or the combination of both do not.

As in the previous cases, the confidence interval of the mean differences was calculated, giving [−0.09878, −0.06522]. As the whole interval was formed by negative values, we could conclude that TRiStar was more efficient than *i** in specifying TR systems.

5.4.3. Second replication (R2)

The small sample of the second replication (13 subjects) forced us to check the normality of the distribution. This was not possible for effictiveness because the sample hugely deviated from normality, so we could not use an ANOVA test. As the use of non-parametric tests is recommended for this type of sample, we chose the Kruskal–Wallis test to check the equality of the distributions among the categories of the samples. As this test only allows one factor to be checked at a time, two tests were necessary: one for language and one for system.

Kruskal–Wallis does not need the normality assumption but it does need the homocedasticity condition. To prove this, we performed a Levene's test for language that provided a *p*-value of 0.079 and another for system, giving a *p*-value of 0.359. These results are summarized in Table 12.

The Kruskal–Wallis test provided a *p*-value for language of 0.046 with a significance level of 0.05, which indicated that language did affect the UEffec distribution. We obtained a p-value of 0.217 for system, which prevented us from concluding that UEffec was affected by the system. Therefore, if only language affects the UEffec distribution and taking into account the distribution shown in Fig. 11, we can state that TRiStar is more effective at specifying TR systems.

Fig. 12 shows the normality of the UEffec distribution. Homocedasticity was checked by Levene's test and the result is shown in Table 13. As its *p*-value is 0.214, the homogeneity of variances can be assumed.

After checking all the conditions, the two-way ANOVA test was performed. The results are shown in Table 14:

Language obtained a *p*-value of 0.009 so we could reject $H_{UEffic0A}$. System and language*system obtained *p*-values well over

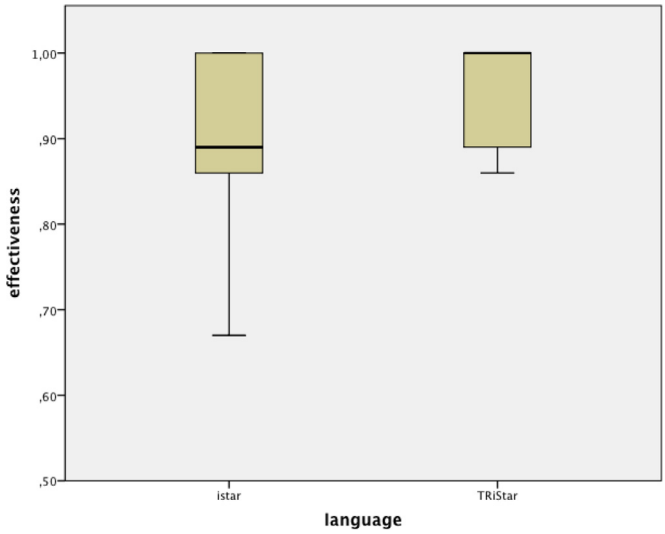


Fig. 11. Distribution of UEffec in R2.

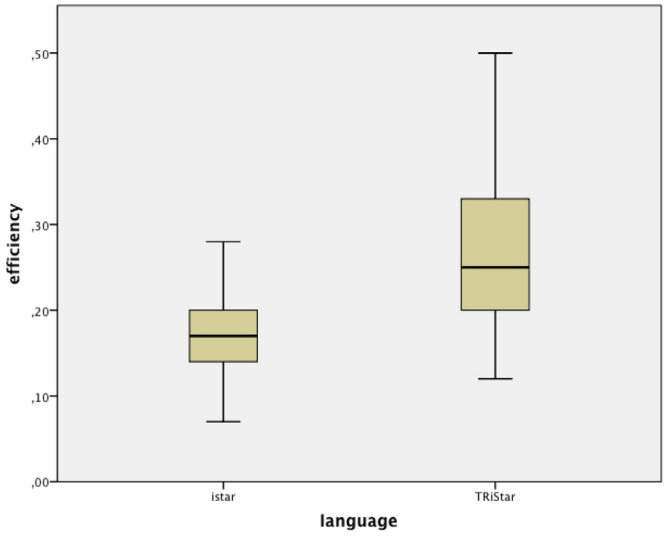


Fig. 12. Normal distribution of UEffic in R2.

0.05, preventing us from rejecting $H_{UEffie0B}$ or $H_{UEffie0AB}$. From these results ir can be concluded that, as in the previous cases, language does affect efficiency when specifying TR systems but the selected system or the combination of language and system do not.

In order to determine the language which obtains the best results in terms of UEffic, the confidence interval at 95% of the mean differences was calculated and the result was [−0.17945, −0.03132]. As all the values in the interval were less than 0, we could assume that TRiStar is more efficient at specifying TR systems.

Table 14
ANOVA results for UEffic in R2.

Source	Type III sum of squares	df	Mean square	*F*	Sig.
Model	1.398[a]	4	0.349	39.589	0.000
Language	0.072	1	0.072	8.181	0.009
System	0.000	1	0.000	0.013	0.909
Language * system	0.000	1	0.000	0.017	0.899
Error	0.194	22	0.009		
Total	1.592	26			

[a] *R* Squared = 0.878 (Adjusted *R* Squared = 0.856).

Table 15
Levene's test for global UEffec.

F	df1	df2	Sig.
0.088	3	134	0.966

Table 16
Levene's test for global UEffic.

F	df1	df2	Sig.
0.838	3	134	0.475

5.5. Meta-analysis

After analyzing the isolated results of every experiment in the family, we performed a global analysis of all the experiments. First, we performed a similar study to those performed for every isolated experiment but using all the data from the original experiment and the two replications. This meant performing a two-way ANOVA test both for UEffec and UEffic, keeping the null hypotheses given in Table 2.

The sample size (31 + 25 + 13 = 69) was big enough to satisfy the normality assumption. A similar Levene's test to those described in the previous section (language + system + language * system) was applied to the data to prove homocedasticity. Table 15 shows the results of the test for UEffec and Table 16 for UEffic.

In both cases homogeneity of variances could be assumed, as the calculated *p*-values were well over 0.05.

After checking the assumptions, a two-way ANOVA test for all the samples used was performed. Table 17 summarizes the results for UEffec and Table 18 for UEffic:

With these results $H_{UEffce0A}$ could be rejected, thanks to the calculated *p*-value of 4.61×10^{-4} for language. However, $H_{UEffee0B}$ and $H_{UEffee0AB}$ could not be rejected as the obtained p-values for system and language*system were much higher than 0.05. The calculated confidence interval was [−0.14303, −0.04102], which proved that there was enough statistical evidence to affirm that TRiStar is more effective than *i** when specifying the requirements of TR systems.

The results for efficiency were similar to those for effectiveness: $H_{UEffie0A}$ could be rejected but $H_{UEffie0B}$ and $H_{UEffce0AB}$ must be accepted. The *p*-value for language was 8.73×10^{-7} but those of systems and language*system were well over 0.05. The confidence interval for the mean differences between *i** and TRiStar was [−0.09844, −0.04359]. Therefore, taking into account the aggregate results for the family of experiments, we had enough statistical evidence to state that TRiStar is more efficient than *i** when specifying requirements for TR systems.

We used BioStat's Comprehensive Meta-Analysis (Biostat Inc, Comprehensive Meta-Analysis 2006) for the meta-analysis. We first obtained the Global Effect Size of the family of experiments and then used it to decide the specific meta-analysis method to use. The Global Effect Sizes for UEffec and UEffic are shown in Table 19 and Table 20, respectively.

With these values and following Dieste's directions (Dieste et al., 2011) Weighted Mean Difference (WMD) method was chosen, as it gets the best score in reliability and statistical power for both UEffec and UEffic. Figs. 13 and 14 summarize the WMD results for both variables.

Calculated *p*-values (0.00016 for UEffec and $< 1 \times 10^{-5}$ for UEffic) allow us to reject the null hypothesis and say that both effectiveness and efficiency of TRiStar and *i** are different. In addition, the cell in the "Overall" row and "Std. diff. in means" column of both tables show the WMD values for effectiveness (−0.66455) and efficiency (−1.18097). As both values are less than 0 we can state that TRiStar provides better effectiveness and efficiency when specifying TR systems requirements.

5.6. Observational findings

This section deals with the conclusions extracted from the observations made during the experiments. Most questions asked by the participants were related to the representation of priority in *i**. They could all remember how priority was represented in TRiStar but some had forgotten how thus was done in *i**, although both techniques had been explained at the same time. This suggests that the participants found the prioritized decomposition links in TRiStar more intuitive and when they saw an *i** diagram with no prioritized decomposition links they could not figure out a way of representing priority without them.

The results obtained in the second replication, with a sample of experienced software developers, were better for effectiveness and efficiency than those obtained from the students. However, the relationship between both languages is similar: effectiveness

Table 17
ANOVA results for global UEffec.

Source	Type III sum of squares	df	Mean square	*F*	Sig.
Model	90.900[a]	4	22.725	991.645	0.000
Language	0.296	1	0.296	12.896	4.61×10^{-4}
Domain	0.047	1	0.047	2.044	0.155
Language * domain	0.003	1	0.003	0.151	0.699
Error	3.071	134	0.023		
Total	93.971	138			

[a] *R* Squared = 0.967 (Adjusted *R* Squared = 0.966).

Table 18
ANOVA results for global UEffic.

Source	Type III sum of squares	df	Mean square	*F*	Sig.
Model	3.350[a]	4	0.838	127.574	0.000
Language	0.175	1	0.175	26.617	8.73×10^{-7}
Domain	0.004	1	0.004	0.656	0.419
Language * domain	0.019	1	0.019	2.834	0.095
Error	0.880	134	0.007		
Total	4.230	138			

[a] *R* Squared = 0.792 (Adjusted *R* Squared = 0.786).

Table 19
Global effect size for UEffec.

		*i**			TRiStar					
Study	System	Mean	SD	*N*	Mean	SD	*N*	Hedges' g	Std. Err.	Effect size
E1	Drone	0.7653	0.10895	15	0.8075	0.14201	16	−0.3232	0.3524	Small
E1	Football	0.7575	0.13685	16	0.8767	0.13162	15	−0.8640	0.3668	Medium
R1	Drone	0.64	0.16657	12	0.8392	0.17168	13	−1.1381	0.4192	Large
R1	Football	0.7577	0.15996	13	0.7967	0.17839	12	−0.2231	0.3884	Small
R2	Drone	0.8586	0.1224	7	0.945	0.06025	6	−0.8109	0.5414	Medium
R2	Football	0.93	0.07668	6	0.98	0.05292	7	−0.7176	0.5363	Medium
Global effect size								−0.6411	0.1697	Medium

Table 20
Global effect size for UEffic.

		*i**			TRiStar					
Study	System	Mean	SD	*N*	Mean	SD	*N*	Hedges' g	Std. Err.	Effect size
E1	Drone	0.13	0.08341	15	0.1413	0.07429	16	−0.1396	0.3504	Small
E1	Football	0.115	0.07607	16	0.2007	0.10491	15	−0.9157	0.3688	Medium
R1	Drone	0.0817	0.01642	12	0.1523	0.03898	13	−2.2488	0.5010	Large
R1	Football	0.0738	0.02256	13	0.1675	0.03306	12	−3.2273	0.5984	Large
R2	Drone	0.1729	0.06473	7	0.2833	0.10053	6	−1.2382	0.5716	Large
R2	Football	0.1733	0.06055	6	0.2743	0.12921	7	−0.9052	0.5471	Medium
Global effect size								−1.1389	0.1867	Large

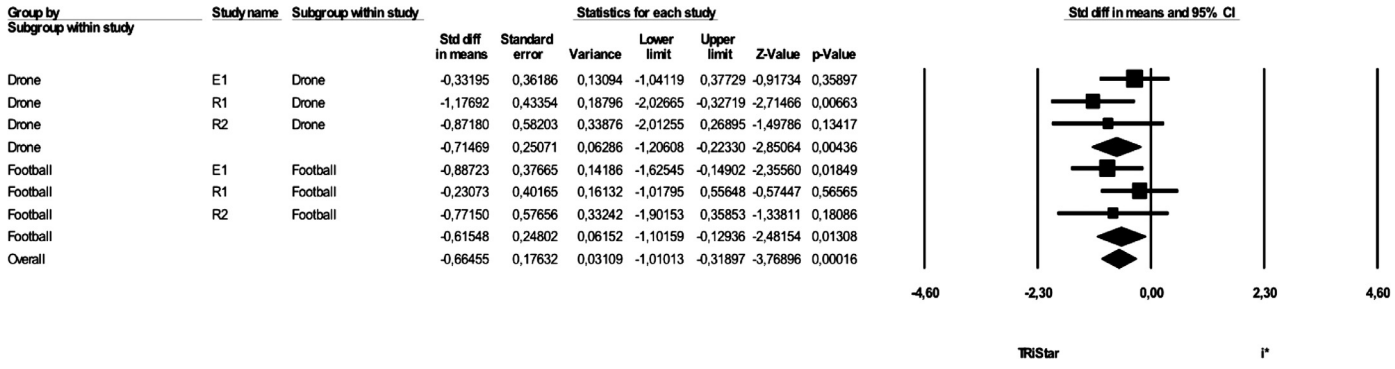


Fig. 13. UEffec WMD meta-analysis.

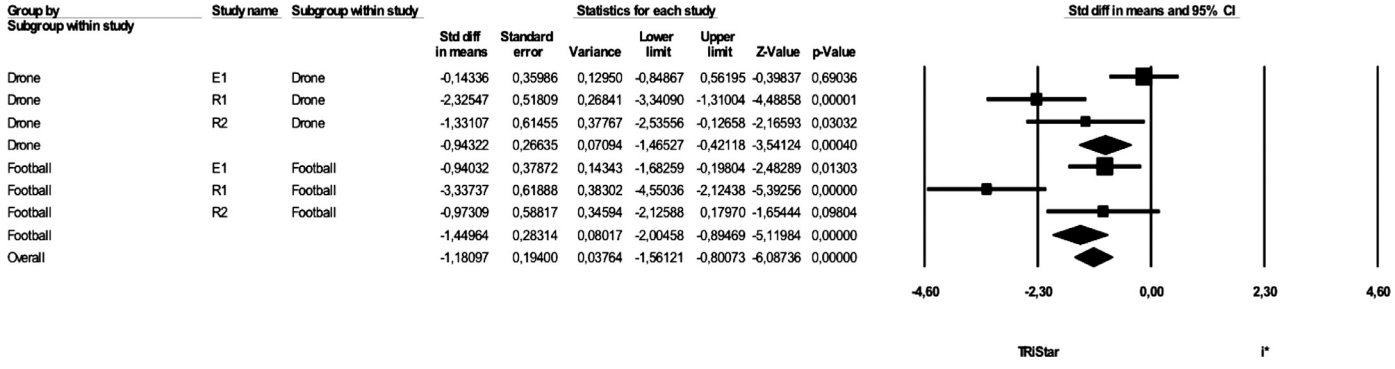


Fig. 14. UEffic WMD meta-analysis.

and efficiency are better in TRiStar. The experience of the software developers probably helped them to learn new notations. In addition, SAES developers are used to dealing with much more complex problems than those given in the experiment. Most of them stated that they preferred using graphical notations instead of directly reading TR program rules.

TRiStar obtained better results in effectiveness but the efficiency results were much better than those for *i**. This suggests that although *i** is still an appropriate language for representing TR systems, TRiStar does the job better and faster.

The question in the questionnaire which obtained most incorrect answers in every experiment was one included in the drone example. In fact, none of the subjects who specified the drone with *i** answered this question correctly. Those who specified the drone with TRiStar had better results, but there were still a lot of wrong answers. This question was related to representing rules whose condition is always true. These results suggest that dependent decompostion links help to link conditions to rules, even though the representation of unconditioned rules in TRiStar could be improved.

6. Threats to the validity of the family of experiments

In order to reduce research and publication bias, as recommended in Jørgensen et al. (2015), the raw experimental data can be consulted in http://xurl.es/RawData. This section deals with some issues that could have threatened the validity of the experiment, in line with the recommendations of Wohlin et al. (2000).

6.1. Validity of the conclusions

The statistical indicators obtained from both the individual experiments and the meta-analysis are well above a 95% confidence level, which allows us to reject the initial null hypotheses.

6.2. Internal validity

As detailed in the previous section, we showed that all the results of the individual experiments satisfied the requirements of the selected statistical methods (ANOVA and Kruskal–Wallis tests). The questionnaires were reviewed by several experts in the development of TR systems and the use of *i** to minimize the risk of incorrect questions.

None of the experiments lasted more than one hour, including the initial briefing by the instructor, to avoid the subjects becoming fatigued. Besides, the students that participated in the experiments were given an extra half point towards their final exam, while in the second replication, the professionalism of the subjects ensured their motivation.

6.3. Construct validity

The method employed to obtain the data from the experiments was a questionnaire similar to those used in other studies, e.g. (Morales et al., 2015) and (Teruel et al., 2012), which reduced the threats to the construct validity. Understandability efficiency and effectiveness were also measured in a similar way to the above-cited studies: efficiency was obtained by dividing the number of correct answers by the total number of answers, while effectiveness was calculated as efficiency divided by the time in minutes taken by each participant to complete the questionnaire, as described in ISO/IEC 25000:2014.

6.4. External validity

According to Höst et al. (2000), the differences between final-year students and software professionals when performing relatively small judgement tasks are minor. Since the questions in the questionnaire for both students and software professionals were not excessively complex, the mixture of students and professionals in the experiment did not involve a threat to it. This view is supported by the the good results obtained for the efficiency parameter, as well as the few questions raised by the participants on the experiments.

Regarding the nature of the proposed problems, we can affirm that the examples employed in the experiments were realistic, since both are part of already existing systems.

7. Conclusions and further work

In Morales et al. (2015) we showed that the understandability of *i** notation was better than that of KAOS for specifying the requirements of teleo-reactive systems. From these results we developed TRiStar, an extension designed to overcome some shortcomings we identified in *i**, which is briefly introduced in Morales et al. (2015) and fully described in the present paper. With the aim of validating the proposal, we conducted a family of experiments to compare the efficiency and effectiveness of the understandability of *i** versus TRiStar for specifying the requirements of teleo-reactive systems.

Subjectively, the vast majority of the participants stated that they found the TRiStar specifications more understandable than those of *i**. Regarding efficiency and effectiveness, the statistical results are conclusive; on one hand, the results of the analysis of the original experiment and the two replicas, and on the other, the results of the meta-analysis of the aggregate data considered as a single experiment, provide enough statistical certainty to reach the following conclusion: both the efficiency and effectiveness of TRiStar are higher than that of *i** diagrams for specifying the requirements of teleo-reactive systems.

In future research work we plan to extend TRiStar in order to cope with the new extensions proposed by Prof. Keith Clark in TeleoR (Clark and Robinson, 2014). We would also like to complete the requirements specification process for teleo-reactive systems by defining a method of guiding the process, starting from natural language specifications.

In the sequel to this research, we intend to make a study of the advantages of TRiStar for the requirements specification of TR systems as compared with a direct approach to TR programs. Starting from a textual description of a reactive system, the results obtained with TRiStar will be compared to those obtained by writing the TR programs directly. Among other objectives, this study will focus on detecting coupling problems among agents, detecting cohesion problems among goals, implementation effectiveness and early error detection.

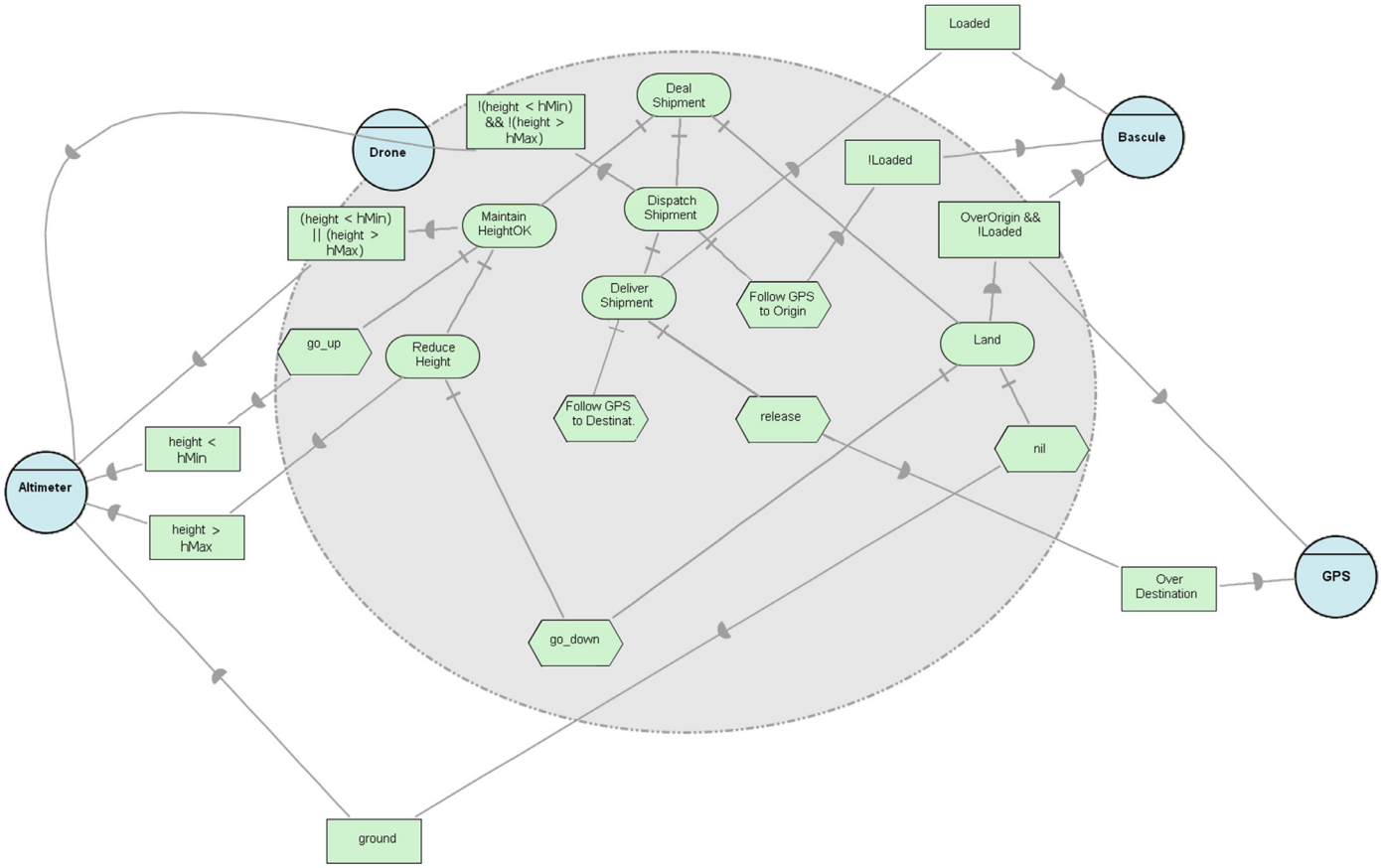
Lastly, we would also like to develop a graphical tool to help developers depict TRiStar diagrams. This tool would include functionalities such as subgoal expand/collapse, which would be helpful in improving the scalability of the models. This tool will also allow the generation of the TR program which corresponds to the specified diagram.

Acknowledgments

This work was partially supported by the insPIre (ref. TIN2012-34003), cDrone (ref. TIN2013-45920-R) and ViSelTR (ref. TIN2012-39279) projects of the Spanish Government. Diego Alonso wishes to thank the Spanish Ministerio de Educación, Cultura y Deporte, Subprograma Estatal de Movilidad, Plan Estatal de Investigación Científica y Técnica y de Innovación *2013-2016* for grant CAS14/00238. We specially acknowledge the support provided by the SAES Company and its employees, as well as by the students of the Politechnic University of Cartagena who participated in the study. Lastly, we wish to thank Esther Corbalán for her support and valuable contribution in carrying out the statistical analysis.

Appendix A. Experimental material - an example of an understanding task (Test for group 2)

Gender (Male/Female)
Age
Qualification
Average score
Have you had any previous experience of working with goal-oriented requirements engineering?
Have you had any previous experience of working with any other requirements engineering technique?
Have you had any previous experience of working with teleo-reactive systems?
[FILL IN AT THE END] In your opinion, which notation has better understandability?



STARTING TIME:
Fill in the blanks so that the obtained TR program agrees with the one that would be obtained from the previous specification.

1. - DealShipment:

_____ → Land
_____ → DispatchShipment
_____ → MaintainHeightOK

2. - Land:

Ground → _____
_____ → go_down

3. - Choose the correct (a) or (b):

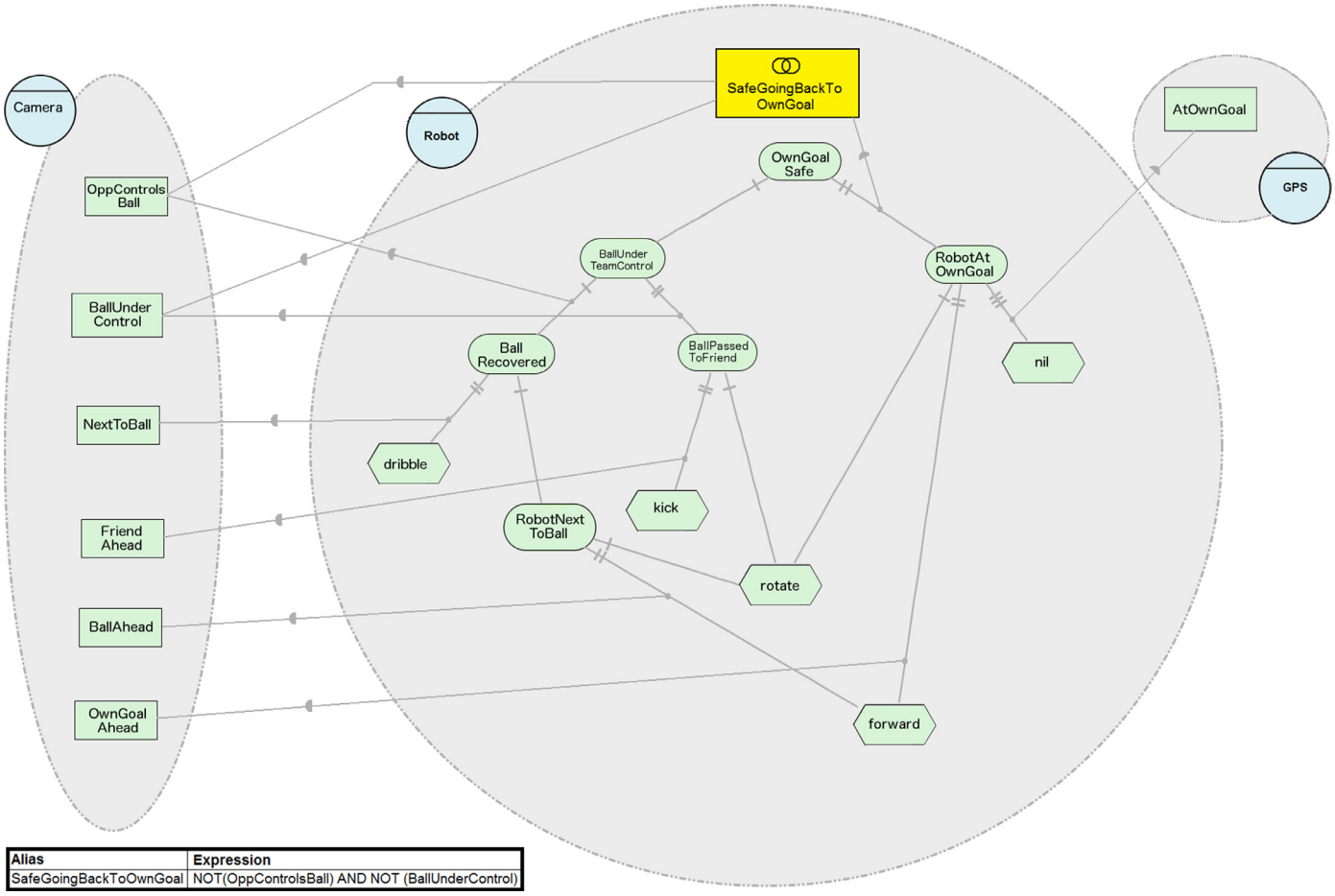
(a) DispatchShipment:
Loaded → DeliverShipment
NOT(Loaded) → followGPSToOrigin
(b) - DispatchShipment:
NOT(Loaded) → followGPSToOrigin
Loaded → DeliverShipment

4. - DeliverShipment:

_____ → release
true → followGPSToDestination

5. - MaintainHeightOK:

height > hMax -> _____
_____ -> go_up
ENDING TIME:



Alias	Expression
SafeGoingBackToOwnGoal	NOT(OppControlsBall) AND NOT (BallUnderControl)

STARTING TIME:
Fill in the blanks so that the obtained TR program agrees with the one that would be obtained from the previous specification.

1. - RobotNextToBall:

BallAhead → _____
True → rotate

2. - OwnGoalSafe:

_____ → RobotAtOwnGoal
True → _____

3. - RobotAtOwnGoal:

_____ → nil
OwnGoalAhead → _____
True → rotate

4. - BallPassedToFriend:

FriendAhead → kick
_____ → rotate

5. - Choose the correct (a) or (b):

(a) - BallUnderTeamControl:
BallUnderControl → BallPassedToFriend
OppControlsBall → BallRecovered
(a) - BallUnderTeamControl:
OppControlsBall → BallRecovered
BallUnderControl → BallPassedToFriend

ENDING TIME:

References

Amyot, D., Mussbacher, G., 2003. URN: towards a new standard for the visual description of requirements. Telecommunications and beyond: The Broader Applicability of SDL and MSC. Springer, Berlin Heidelberg, pp. 21–37.

Ayala, C.P., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., ..., Quer, C., 2005. A comparative analysis of i*-based agent-oriented modeling languages. SEKE, Vol. 5, pp. 43–50.

Bajaj, A., 2004. The effect of the number of concepts on the readability of schemas: an empirical study with data models. Requir. Eng. 9, 261–270.

Basili, V.R., Caldiera, G., Rombach, H.D., 1994. The goal question metric approach. Encyclopedia of Software Engineering, vol. 2. Wiley, pp. 528–532 Retrieved from http://wwwagse-old.informatik.uni-kl.de/pubs/repository/basili94b/encyclo.gqm.pdf.

Biostat Inc, Comprehensive Meta-Analysis, 2006. <http://www.meta-analysis.com> (accessed June 2015).

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J., 2004. Tropos: an agent-oriented software development methodology. Auton. Agents Multi-Agent Syst. 8 (3), 203–236.

Chernak, Y. (2009). Building Foundation for Structured Requirements. Aspect-oriented Requirements Engineering Explained- Part 1, in: Requirements Networking Group (RQNG), 2009.

Clark, K. L., & Robinson, P. J. Multi-tasking Robotic Agent Programming in TeleoR. 2014, Research report downloaded from http://www.doc.ic.ac.uk/~klc/TeleoRMT2.pdf

Dieste, O., Fernández, E., García Martínez, R., Juristo, N., 2011. Comparative analysis of meta-analysis methods: when to use which? In: Proceedings of the 15th International Conference on Evaluation & Assessment in Software Engineering (EASE'11). Durham, UK. IET, pp. 36–45.

Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P., 2004. Specifying and analyzing early requirements in Tropos. Requir. Eng. 9 (2), 132–150.

Genero, M., et al., 2008. Defining and validating metrics for assessing the understandability of entity–relationship diagrams. Data Knowl. Eng. 64 (3), 534–557 March 2008.

Grinstead, C.M., Snell, J.L., 2006. Introduction to Probability. American Mathematical Society.

Gubisch, G., Steinbauer, G., Weiglhofer, M., Wotawa, F., 2008. A teleo-reactive architecture for fast, reactive and robust control of mobile robots. In: IEA/AIE '08 Proceedings of the 21st international conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: New Frontiers in Applied Artificial Intelligence.

Höst, M., Regnell, B., Wohlin, C., 2000. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. Empir. Softw. Eng. 5 (3), 201–214.

Jamison, W., Teng, J.T.C., 1993. Effects of graphical versus textual representation of database structure on query performance. J. Database Manag. 4 (1), 16–23.

Jørgensen, M., 2015. Incorrect results in software engineering experiments: how to improve research practices. J. Syst. Softw. http://dx.doi.org/10.1016/j.jss.2015.03.065.

Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. 28 (8), 721–734. doi:10.1109/TSE.2002.1027796.

Lamsweerde, A., 2001. Goal-oriented requirements engineering: a guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE'01). Washington DC, USA, pp. 249–262. doi:10.1109/ISRE.2001.948567.

Lamsweerde, A., 2009. Requirements Engineering: from goals to UML models to software specifications. John Wiley & Sons Ltd, England.

Lee, H., Choi, B.G., 1998. A comparative study of conceptual data modeling techniques. J. Database Manag. 9 (2), 26–35.

Levene, Howard, 1960. Ingram Olkin, Harold Hotelling. In: et alia (Ed.), Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling. Stanford University Press, pp. 278–292.

Lockerbie, J., Maiden, N.A., 2008. REDEPEND: tool support for i* modelling in large-scale industrial projects. CAiSE Forum, Vol. 344, pp. 69–72.

Morales, J.M., Navarro, E., Sánchez, P., Alonso, D., 2015. TRiStar: an i* extension for teleo-reactive systems requirements specifications. In: Proceedings of the ACM Symposium on Applied Computing. April 13-17, 2015, Salamanca, Spain http://dx.doi.org/10.1145/2695664.2695703.

Morales, J., Sánchez, P., Alonso, D., 2012. A systematic literature review of the teleo-reactive paradigm. Artificial Intelligence Review. Springer, Netherlands, pp. 1–20 2012.

Morales, J.M., Navarro, E., Sánchez, P., Alonso, D., 2015. A controlled experiment to evaluate the understandability of KAOS and i* for modeling Teleo-Reactive systems. J. Syst. Softw. 100, 1–14.

Nilsson, N.J., 1993. Teleo-reactive programs for agent control. J. Artif. Intell. Res. 1 (1), 139–158 Retrieved from http://dl.acm.org/citation.cfm?id = 1618595.1618602.

Rajan, K., Py, F., McGann, C., 2010. Adaptive control of AUVs using onboard planning and execution. Sea Technol. 51–55 April 2010.

Sánchez, P., Alonso, D., Morales, J.M., Navarro, P.J., 2012. From teleo-reactive specifications to architectural components: a model-driven approach. J. Syst. Softw. 85 (11), 2504–2518.

Sommerville, I., Sawyer, P., Viller, S., 1998. Viewpoints for requirements elicitation: a practical approach. In: IEEE International Conference on Requirements Engineering (ICRE'98). Colorado Springs, Colorado 1998.

Sutcliffe, A., Minocha, S., 1999. Linking business modelling to sociotechnical system design. In: Proceedings of the 11th International Conference on Advanced Information Systems Engineering, CAiSE'99. Heidelberg, Germany, June 14-18, pp. 73–87.

Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., González, P., 2011. CSRML: a goal-oriented approach to model requirements for collaborative systems. In: Conceptual Modeling – ER 2011, Lecture Notes in Computer Science Volume 6998, 2011, pp. 33–46.

Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., Jaen, J., González, P., 2012. Analyzing the understandability of requirements engineering languages for cscw systems: a family of experiments. Inf. Softw. Technol. 54 (11), 1215–1228. doi:10.1016/j.infsof.2012.06.001.

Tsalgatidou, A., Karakostas, V., Loucopoulos, P., 1990. Rule-based requirements specification and validation. Proceedings of the Advanced Information Systems Engineering. Springer, Berlin Heidelberg, pp. 251–263 January.

"under·standable, adj." OED Online. Oxford University Press, March 2015. Web. 20 March 2015.

Winer, B.J., Brown, D.R., Michels, K.M., 1991. Statistical Principles in Experimental Design, 3rd ed.. McGraw-Hill Humanities/Social Sciences/Languages, p. 928.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A., 2000. Experimentation in Software Engineering: An Introduction, first ed Kluwer Academic Publishers, Norwell, USA 2000.

Yu, E., 1997. Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97). Washington D.C., USA, pp. 226–235. Jan. 6-8, 1997.

Yu, E., Mylopoulos, J., 1998, June. Why goal-oriented requirements engineering. In: Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, Vol. 15.

José Miguel Morales is an Assistant Professor and a Ph.D. student in computer science at the Universidad Politécnica de Cartagena and a member of the university's DSIE (Division of Systems and Electronic Engineering) research group. His research interests include real time systems, and specification and design of teleo-reactive systems. Morales has a master's in information technology engineering from the Universidad de Murcia.

Elena Navarro is an Associate Professor of Computer Science at the University of Castilla-La Mancha (Spain). Prior to this position, she worked as a researcher at the Informatics Laboratory of the Agricultural University of Athens (Greece) and as a staff member of the Regional Government of Murcia, at the Instituto Murciano de Investigación y Desarrollo Agroalimentario. She got her bachelor degree and Ph.D. at the University of Castilla-La Mancha, and her master degree at the University of Murcia (Spain). She is currently an active collaborator of the LoUISE group of the University of Castilla-La Mancha. Her current research interests are Requirements Engineering, Software Architecture, Model-Driven Development, and Architectural Knowledge.

Pedro Sánchez received his Ph.D. degree in computer science from the Technical University of Valencia, Spain, in 2000. Since 1996, he has participated in different projects focused on software engineering and conceptual modeling applied to the development of reactive systems. In 2000, he joined the Systems and Electronic Engineering Division (DSIE) at the Technical University of Cartagena. He is currently an

Associate Professor at the Technical University of Cartagena in the field of computer science. His current research interests include software engineering for implementing teleo-reactive systems.

Diego Alonso is currently an Associate Professor of Computer Science at the Universidad Politécnica de Cartagena (Spain) and a member of the DSIE (Division of Systems and Electronic Engineering) research group. He received a M. Sc. Degree in Industrial Engineering from the Universidad Politécnica de Valencia (Spain), and a Ph.D. with "Doctor Europaeus" Mention form the Universidad Politécnica de Cartagena. His research interests focus on the application of the model-driven engineering approach to the development of component-based reactive systems with real-time constraints, mainly in the field of robotics.

CONCLUSIONES

Uno de los principales objetivos de este trabajo es extender los recursos disponibles para el desarrollo de sistemas reactivos siguiendo el enfoque Teleo-Reactivo. Este objetivo global abarca sub-objetivos que tienen que ver con aspectos de ingeniería de requisitos, metodológicos, de diseño orientado a objetivos y de implementación en plataformas ejecutables utilizando tecnología de componentes.

Con esa intención realizamos un experimento controlado para comparar la comprensibilidad de KAOS e i*, los dos lenguajes GORE más extendidos en la actualidad. La descripción y los resultados obtenidos con este experimento están cuidadosamente detallados en el artículo "*A controlled experiment to evaluate the understandability of KAOS and i* for modeling Teleo-Reactive systems*" que forma parte de este compendio. Tras la realización de este estudio decidimos usar i* como base de nuestro lenguaje ya que obtuvo unos resultados ligeramente mejores a los de KAOS. No obstante, i* también demostró tener algunas debilidades a la hora de especificar sistemas Teleo-Reactivos.

Para intentar mitigar las debilidades detectadas en i* propusimos una serie de extensiones al lenguaje que denominamos TRiStar. La validación de la propuesta vino gracias a la familia de experimentos que realizamos para comparar nuestra propuesta con el lenguaje original y que detallamos en profundidad en el artículo "*A family of experiments to evaluate the understandability of TRiStar and i* for modeling Teleo-Reactive systems*", que también forma parte de este compendio. Los resultados demostraron holgadamente que TRiStar mejoraba la eficacia y la eficiencia de TRiStar cuando eran usados para especificar sistemas Teleo-Reactivos.

Un diagrama TRiStar puede ser transformado sistemáticamente a un programa Teleo-Reactivo gracias al *mapping* descrito en el artículo en el que se presentaba el lenguaje. A partir de ese programa, es posible obtener la arquitectura de componentes de un sistema que lo implemente. Además, también puede obtenerse el comportamiento de dichos componentes en forma de máquinas de estados. Esta transformación se describe en el artículo "*From Teleo-Reactive specifications to architectural components: A model-driven approach*", que cronológicamente fue el primero en publicarse de este compendio. Estos componentes pueden integrarse en el framework de desarrollo creado por el grupo de investigación DSIE [Iborra09].

A todo lo anterior le añadimos las posibilidades abiertas por TeleoR [Clark14] y TRiStar+ [Sánchez16] para introducir restricciones temporales en la especificación de requisitos de sistemas Teleo-Reactivos. Aunque TeleoR se ha mostrado capaz de soportar el tipo de restricciones temporales necesarias para desarrollar sistemas en tiempo real, también es cierto que existen algunas limitaciones que hacen que sea necesario seguir trabajando en el lenguaje y en su implementación.

Gracias a esto, disponemos de un mecanismo que nos permite partir de un diagrama en TRiStar+ y obtener una arquitectura de componentes con un comportamiento que se corresponde con la especificación del diagrama inicial y que además pueden integrarse en un framework de desarrollo ampliamente probado.

Entre los trabajos futuros queda pendiente proponer una metodología que permita obtener lo más sistemáticamente posible un diagrama TRiStar+ a partir de una descripción textual de un sistema Teleo-Reactivo y un caso de estudio que cubra todo el ciclo de desarrollo. Asimismo, tal y como hemos mencionado más arriba, queda pendiente la profundización en el lenguaje TeleoR y en su implementación para poder superar las limitaciones identificadas.

REFERENCIAS

[Atkinson03] Atkinson, C.; Kühne, T.: "Model-driven development: a metamodelling foundation.", IEEE Software, 2003.

[Brugali09] Brugali, D., Scandurra, P., 2009. "Component-based robotic engineering. Part I: Reusable building blocks." IEEE Robotics and Automation Magazine 16 (4),84–96.

[Brugali10] Brugali, D., Scandurra, P., 2010. "Component-based robotic engineering. Part II: Systems and models." IEEE Robotics and Automation Magazine 17 (1), 100–112.

[Clark14] Clark, K. L., Robinson, P. J., 2014. "Robotic Agent Programming in TeleoR." Research Report. Available online at http://www.doc.ic.ac.uk/~klc/ShortTeleoRIntro.pdf

[Iborra09] Iborra, A., Alonso, D., Ortiz, F., Pastor, J.A., Sánchez, P., Álvarez, B., 2009. "Experiences using software engineering in the design of service robots." IEEE Robotics and Automation Magazine 16 (1), 24–33.

[Kavakli05] Kavakli, E., Loucopoulos, P., 2005. "Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods." Information Modeling Methods and Methodologies, Idea Group.

[Lamsweerde01] Lamsweerde, Axel van: "Goal-Oriented Requirements Engineering: A Guided Tour.", International Symposium on Requirements Engineering, Toronto, 2001.

[Lau07] Lau, K; Wang, Z.: "Software Component Models.", IEEE Transactions on Software Engineering, 2007.

[Morales15] J.M. Morales, E. Navarro, P. Sánchez, D. Alonso. "TRiStar: an i* extension for Teleo-Reactive systems requirements specifications.", Proceedings of the 30th Annual ACM Symposium on Applied Computing, 13-17 April, Salamanca, pp. 283-288.

[Sánchez16] P. Sánchez, B. Álvarez, J. M. Morales, D. Alonso, A. Iborra. "An Approach to Modeling and Developing Teleo-Reactive Systems considering Timing Constraints", Journal of Systems and Software, 117, 317-333 ISSN 0164-1212, http://dx.doi.org/10.1016/j.jss.2016.03.064

[Selic03] Selic, B.: "The pragmatics of model-driven development.", IEEE Transactions on Software Engineering, 2003.

[Szypersky02] Szypersky, C.: "Component Software: Beyond Object-Oriented Programming.", Addison- Wesley, 2002.

ISI Web of Knowledge℠

Journal Citation Reports®

WELCOME HELP RETURN TO JOURNAL

2012 JCR Science Edition

Rank in Category: JOURNAL OF SYSTEMS AND SOFTWARE

Journal Ranking

For **2012**, the journal **JOURNAL OF SYSTEMS AND SOFTWARE** has an Impact Factor of **1.135**.

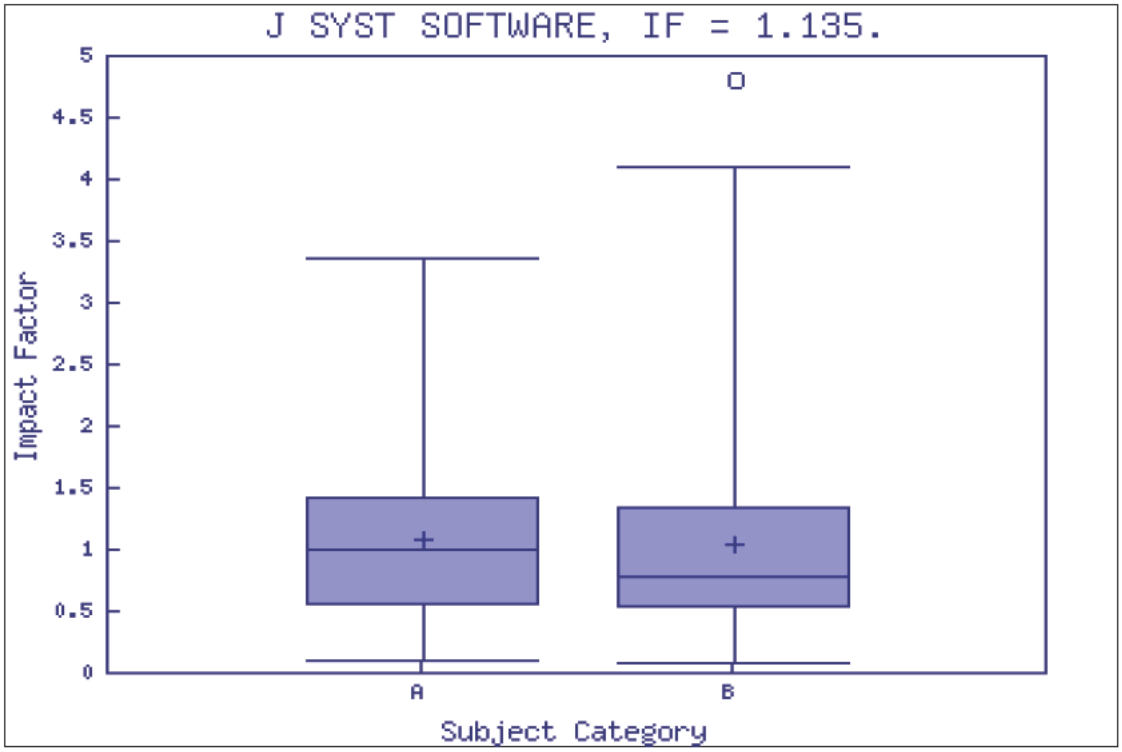
This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, SOFTWARE ENGINEERING	105	41	Q2
COMPUTER SCIENCE, THEORY & METHODS	100	30	Q2

Category Box Plot

For **2012**, the journal **JOURNAL OF SYSTEMS AND SOFTWARE** has an Impact Factor of **1.135**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.



Key

A - COMPUTER SCIENCE, SOFTWARE ENGINEERING

B - COMPUTER SCIENCE, THEORY & METHODS

Acceptable Use Policy
Copyright © 2016 Thomson Reuters.

THOMSON REUTERS
Published by Thomson Reuters

Rank in Category: ARTIFICIAL INTELLIGENCE REVIEW

Journal Ranking

For **2014**, the journal **ARTIFICIAL INTELLIGENCE REVIEW** has an Impact Factor of **2.111**.

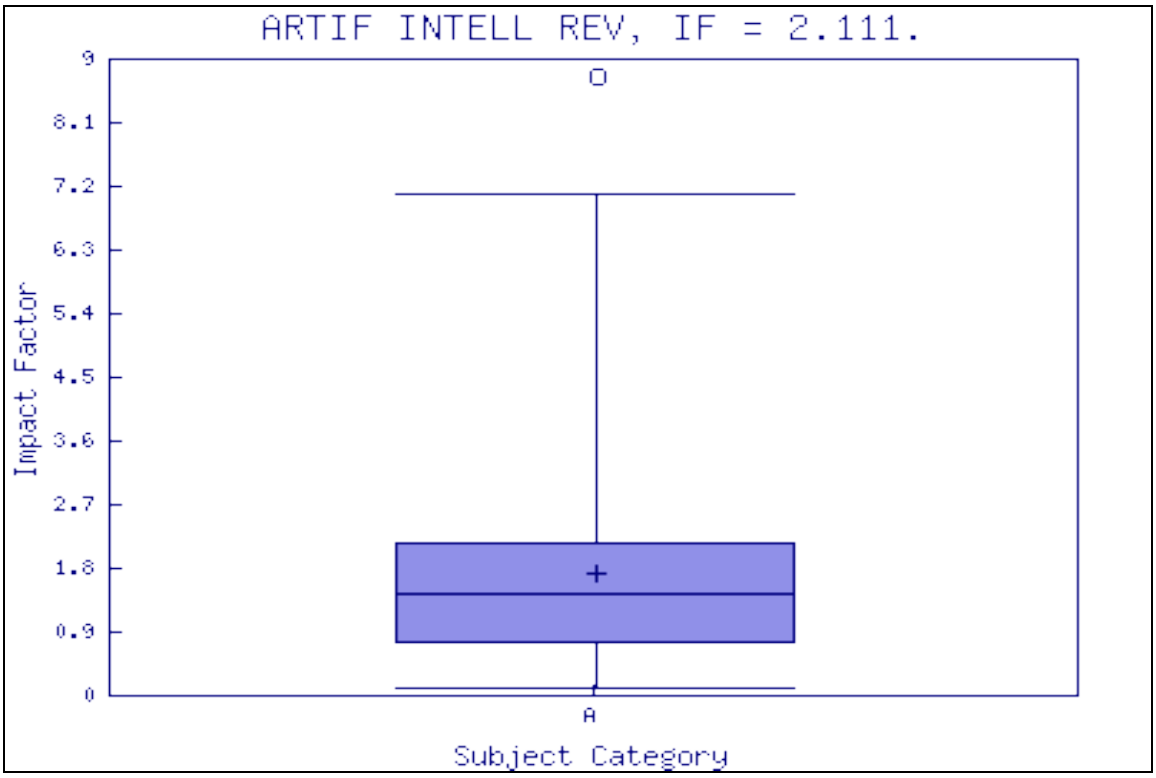
This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE	123	35	Q2

Category Box Plot

For **2014**, the journal **ARTIFICIAL INTELLIGENCE REVIEW** has an Impact Factor of **2.111**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.



Key

A - COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE

Acceptable Use Policy
Copyright © 2016 Thomson Reuters.

THOMSON REUTERS

Published by Thomson Reuters

ISI Web of Knowledge℠

Journal Citation Reports®

WELCOME HELP RETURN TO JOURNAL

2014 JCR Science Edition

Rank in Category: JOURNAL OF SYSTEMS AND SOFTWARE

Journal Ranking

For **2014**, the journal **JOURNAL OF SYSTEMS AND SOFTWARE** has an Impact Factor of **1.352**.

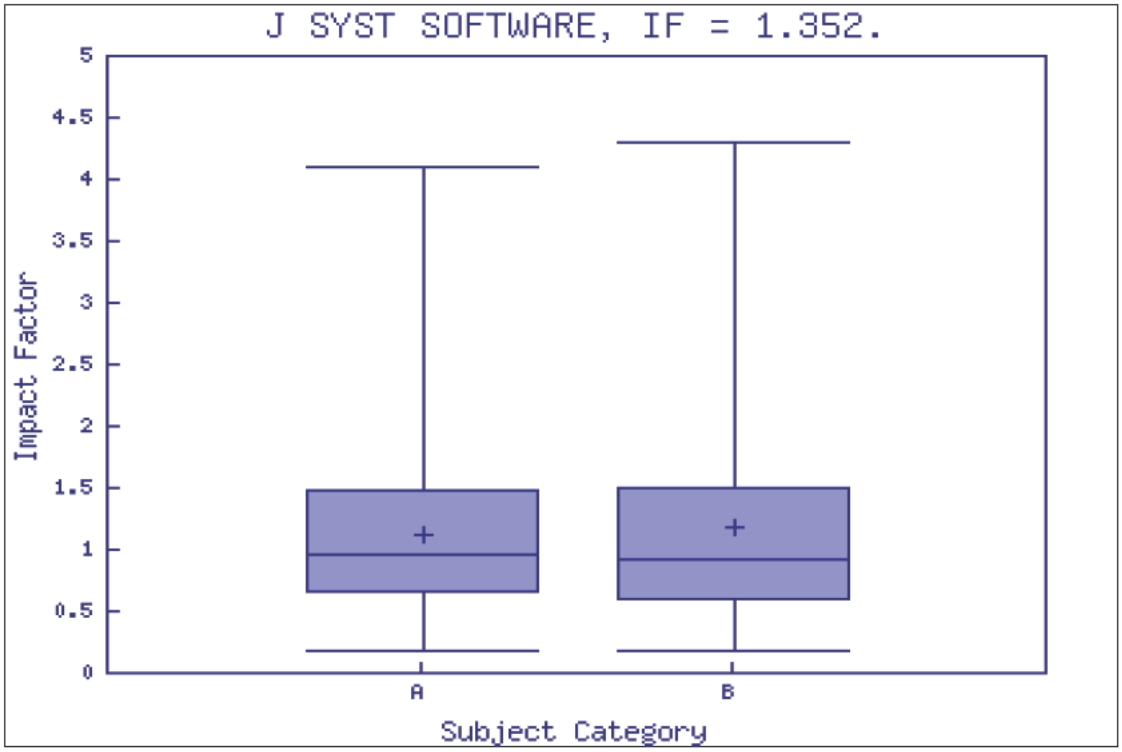
This table shows the ranking of this journal in its subject categories based on Impact Factor.

Category Name	Total Journals in Category	Journal Rank in Category	Quartile in Category
COMPUTER SCIENCE, SOFTWARE ENGINEERING	104	33	Q2
COMPUTER SCIENCE, THEORY & METHODS	102	31	Q2

Category Box Plot

For **2014**, the journal **JOURNAL OF SYSTEMS AND SOFTWARE** has an Impact Factor of **1.352**.

This is a box plot of the subject category or categories to which the journal has been assigned. It provides information about the distribution of journals based on Impact Factor values. It shows median, 25th and 75th percentiles, and the extreme values of the distribution.



Key

A - COMPUTER SCIENCE, SOFTWARE ENGINEERING

B - COMPUTER SCIENCE, THEORY & METHODS

Acceptable Use Policy
Copyright © 2016 Thomson Reuters.

THOMSON REUTERS
Published by Thomson Reuters