



ESCUELA POLITÉCNICA



UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO DE INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL  
SOFTWARE

TRABAJO FIN DE GRADO

**PiLHaR: Herramienta para facilitar la definición, composición y  
ejecución de las tareas de un robot educativo**



ESCUELA POLITÉCNICA



UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO DE INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL  
SOFTWARE

TRABAJO FIN DE GRADO

**PiLHaR: Herramienta para facilitar la definición, composición y  
ejecución de las tareas de un robot educativo**

**Autor: Gloria Díaz González**

**Tutor: Cristina Vicente Chicote**

**Co-Tutor: José Ramón Lozano Pinilla**

# Agradecimientos

En primer lugar, me gustaría agradecer a mis tutores, Cristina y José Ramón, todo el esfuerzo y el tiempo empleado en ayudarme en todo momento en el desarrollo de este Proyecto, aún estando muy ocupados en su trabajo.

Por otro lado, dar las gracias a la profesora Pilar Bachiller, del grupo de investigación RoboLab de la Universidad de Extremadura, que nos ha prestado el robot Cozmo empleado en el desarrollo del Proyecto. Sin él no habríamos podido llevarlo a buen puerto.

También me gustaría agradecer a Sol y a Rocío, del Taller de los Sueños, el tiempo que nos dedicaron cuando fuimos a enseñarles la herramienta desarrollada en el Proyecto y el feedback tan positivo que nos dieron.

Gracias a mi familia por el apoyo incondicional que siempre me han dado y aún más en el periodo empleado en el Trabajo Fin de Grado. Tampoco me quiero olvidar de esos amigos que llevan ahí casi desde los inicios del Grado y que me han ofrecido su ayuda en cualquier etapa de este Trabajo. Ahora estoy a punto de finalizar una de las etapas de la vida y nunca podré olvidar los buenos momentos que nos han llevado a este punto.

Por último, hacer especial mención a José Ramón, mi amigo y co-tutor, por confiar en mi y darme tantos consejos como su experiencia le permite. También agradezco que quisiera acompañarme en este final de etapa y conseguir con creces enseñarme todo lo que está en sus manos.

# Resumen

En los últimos años, el uso de robots se ha extendido a muchos de los ámbitos de nuestra vida cotidiana ya que son capaces de realizar tareas repetitivas, complejas o incluso peligrosas para las personas, a un coste cada vez más reducido. En campos como el de la salud o la educación, la incorporación de robots ha demostrado tener numerosas ventajas y resulta cada vez más habitual. Por ejemplo, la incorporación de robots en terapias de apoyo a personas que padecen un trastorno del espectro autista (TEA) o un trastorno de déficit de atención e hiperactividad (TDAH), ha demostrado obtener muy buenos resultados. Sin embargo, programar estos robots para adaptar las terapias a las peculiaridades de cada paciente resulta muy complejo, sobre todo para los terapeutas que diseñan y aplican estas terapias ya que, en general, no suelen ser especialistas en robótica o programación.

En esta línea, como parte del Trabajo de Fin de Grado (TFG) que aquí se presenta, se ha desarrollado una herramienta dirigida a facilitar la programación de robots educativos, para su incorporación en terapias personalizadas para niños y niñas con TEA o TDAH. Esta herramienta, denominada *PiLHaR* (Pinta Lo que Hace tu Robot), permite a los terapeutas: (1) seleccionar de un catálogo las tareas que quieren que lleve a cabo el robot, personalizándolas (configurando sus parámetros) cuando sea necesario; (2) diseñar gráficamente sus terapias, secuenciando las tareas seleccionadas como estimen oportuno; (3) ejecutar las terapias diseñadas en el robot educativo

---

Cozmo; y (4) guardar cualquiera de sus diseños en el catálogo para poder reutilizarlo posteriormente, ya sea tal cual, personalizándolo de otro modo o integrándolo como parte de otra terapia, más compleja.

Conviene señalar que la herramienta desarrollada ha sido presentada a las terapeutas del Taller de los Sueños, centro especializado en autismo, con una larga trayectoria en la provincia de Cáceres, habiendo mostrado un gran interés por la utilidad que esta podría tener de cara a plantearse poder incorporar robots en algunas de sus terapias.

**Palabras clave:** Ingeniería del Software Dirigida por Modelos (ISDM), Modelado Gráfico de Tareas y Flujos de Tareas, Generación Automática de Software, Programación de Robots Educativos.

# Abstract

In recent years, the use of robots has spread to many of the areas of our daily life, since they can perform repetitive, complex and even dangerous tasks, at an increasingly reduced cost. In fields such as health or education, the use of robots has proven to have numerous advantages and is becoming more and more frequent. For example, the inclusion of robots in therapies for people suffering from Autism Spectrum Disorder (ASD) or Attention Deficit Hyperactivity Disorder (ADHD) has shown to obtain very good results. However, programming these robots so that the therapies they are used into can be customized according to the peculiarities of each patient is very complex, especially for the therapists who design and apply these therapies since, in general, they are not specialists in robotics or programming.

In this line, the Final Degree Project (TFG) presented here has developed a tool aimed at easing the programming of educational robots for their integration into customized therapies for children suffering ASD or ADHD. This tool, called *PiLHaR* (Paint What Your Robot Does), allows therapists to: (1) select from a catalog the tasks they want the robot to perform, customizing them (setting their parameters) as and when needed; (2) graphically design their therapies, sequencing the selected tasks as appropriate; (3) execute the desired therapies in the Cozmo educational robot; and (4) store their designs in the catalog so that they can be later reused, either as is, customized in a different way, or as part of other (more complex) therapies.

---

It is worth noting that the tool developed as part of this Project has been presented to the therapists working at Taller de los Sueños, a center specializing in autism, with a long experience in Cáceres, who have shown great interest in its potential to help them include robots into some of their therapies.

**Keywords:** Model-Driven Software Engineering (MDSE), Graphical modeling of Tasks and Workflows, Automatic Software Generation, Educational Robot Programming.

# Índice general

<b>1. Introducción</b>	<b>2</b>
1.1. Objetivos . . . . .	4
1.2. Metodología . . . . .	5
1.3. Planificación . . . . .	6
1.4. Estructura del documento . . . . .	7
<b>2. Estado de la Técnica</b>	<b>9</b>
2.1. Background . . . . .	9
2.2. Herramientas . . . . .	11
2.2.1. Herramientas de modelado . . . . .	12
2.2.2. Lenguajes y <i>frameworks</i> para la programación de robots . . . . .	15
2.2.3. Librerías Python para la programación del robot Cozmo . . . . .	17
2.3. Otros robots educativos . . . . .	19
2.3.1. NAO . . . . .	19
2.3.2. Robot Atent@ . . . . .	20
2.4. Trabajos relacionados . . . . .	21
<b>3. Análisis y diseño</b>	<b>23</b>
3.1. Análisis . . . . .	23
3.1.1. Requisitos funcionales y no funcionales . . . . .	23
3.1.2. Descripción de los actores . . . . .	25
3.1.3. Descripción y Diagramas de Casos de Uso . . . . .	25

## ÍNDICE GENERAL

---

3.2. Diseño . . . . .	31
3.2.1. Arquitectura . . . . .	31
3.2.2. Sintaxis abstracta del lenguaje de modelado de <i>PiLHaR</i> . . . . .	34
3.2.3. Sintaxis concreta . . . . .	46
<b>4. Implementación y desarrollo</b>	<b>49</b>
4.1. Librerías utilizadas . . . . .	49
4.2. Ampliación de la librería . . . . .	50
4.2.1. Métodos sin parámetros . . . . .	50
4.2.2. Métodos con parámetros . . . . .	51
4.2.3. Tareas desarrolladas . . . . .	52
4.3. Generador de código . . . . .	62
4.4. Despliegue en Cozmo . . . . .	66
4.5. Eclipse Application . . . . .	68
4.6. Interfaz de interacción . . . . .	70
4.7. Ajustes de Cozmo . . . . .	71
4.7.1. Conexión con Cozmo . . . . .	71
4.7.2. Actualización de Cozmo . . . . .	72
<b>5. Ejemplos de funcionamiento</b>	<b>75</b>
5.1. Ejemplo 1: Presentarse . . . . .	75
5.2. Ejemplo 2: Mostrar Alegría . . . . .	77
5.3. Ejemplo 3: Flujo Complejo . . . . .	79
<b>6. Conclusiones y trabajos futuros</b>	<b>85</b>
6.1. Principales dificultades . . . . .	85
6.2. Conclusiones . . . . .	87
6.3. Trabajos futuros . . . . .	88
<b>Bibliografía</b>	<b>89</b>
<b>Anexos</b>	<b>96</b>



## **ÍNDICE GENERAL**

---

<b>A. Anexos</b>	<b>96</b>
A.1. Ficheros java del editor . . . . .	96
A.2. Ficheros del editor gráfico . . . . .	104
A.3. Manual de implementación del editor gráfico . . . . .	110
A.4. Ficheros de generación de código con Acceleo . . . . .	122
A.5. Manual de usuario . . . . .	134
A.6. Manual rápido de usuario . . . . .	142
A.7. Manual del desarrollador . . . . .	147

# Índice de tablas

2.1. Comparativa herramientas de modelado . . . . .	15
2.2. Comparativa lenguajes de programación . . . . .	17
2.3. Comparativa de librerías del robot . . . . .	19
3.1. Requisitos funcionales . . . . .	24
3.2. Requisitos no funcionales . . . . .	24
3.3. Casos de Uso de los desarrolladores. . . . .	26
3.4. Casos de Uso de los terapeutas. . . . .	26
3.5. Descripción de caso de uso 03 . . . . .	28
3.6. Descripción de caso de uso 05 . . . . .	29
3.7. Descripción de caso de uso 06 . . . . .	30
3.8. Restricciones del meta-modelo del catálogo . . . . .	43
3.9. Restricciones del meta-modelo del flujo de actividades . . . . .	44
3.10. Restricciones OCL del meta-modelo del catálogo versión 2 . . . . .	45
3.11. Restricciones del meta-modelo del flujo de actividades versión 2 . . . . .	45
3.12. Leyenda del editor gráfico de catálogo . . . . .	47
3.13. Leyenda del editor gráfico del flujo de actividades . . . . .	47
4.1. Descripción de la tarea Hablar . . . . .	53
4.2. Descripción de la tarea MoverBrazos . . . . .	54
4.3. Descripción de la tarea MoverRuedas . . . . .	55
4.4. Descripción de la tarea MoverCabeza . . . . .	56



## ÍNDICE DE TABLAS

---

4.5. Descripción de la tarea MostrarSentimiento . . . . .	58
4.6. Descripción de la tarea HacerFoto . . . . .	59
4.7. Descripción de la tarea MostrarImagen . . . . .	60
4.8. Descripción de la tarea CambiarColorLuz . . . . .	61

# Índice de figuras

1.1.	Diagrama de Gantt . . . . .	8
2.1.	Robot Cozmo . . . . .	10
3.1.	Diagrama de Casos de Uso. . . . .	27
3.2.	Arquitectura del sistema . . . . .	31
3.3.	Arquitectura de la librería pyCozmo . . . . .	33
3.4.	Protocolo de Conexión Cozmo . . . . .	34
3.5.	Meta-modelo Catalogue . . . . .	37
3.6.	Meta-modelo Workflow . . . . .	37
3.7.	Ilustración abstracta de dos tareas simples y su código . . . . .	40
3.8.	Ilustración abstracta de un ejemplo de flujo de actividades . . . . .	40
3.9.	Ilustración de una tarea compleja vinculada a su flujo de actividades. .	41
3.10.	Ilustración abstracta de un ejemplo de flujo de actividades . . . . .	42
3.11.	Ilustración abstracta de una tarea compleja y un parámetro . . . . .	42
3.12.	Carpeta metamodelos del proyecto principal . . . . .	48
3.13.	Carpeta código fuente del proyecto principal . . . . .	48
4.1.	Creación de nuevo método sin parámetros . . . . .	50
4.2.	Ejemplo real de la nomenclatura . . . . .	51
4.3.	Creación de nuevo método con parámetros . . . . .	51
4.4.	Ejemplo real de la creación de parámetros . . . . .	52
4.5.	Parámetros del parámetro destino . . . . .	52

## ÍNDICE DE FIGURAS

---

4.6. Contenido del fichero “ <i>EjecutarCozmo.bat</i> ” . . . . .	62
4.7. Esquema del contenido del fichero que contiene el código del programa . . . . .	63
4.8. Esquema del <i>template</i> “ <i>GenetareTarea</i> ” de <i>Activity</i> . . . . .	65
4.9. Esquema del <i>template</i> “ <i>GenetareTarea</i> ” de <i>Question</i> . . . . .	66
4.10. Esquema del <i>template</i> “ <i>GenetareTarea</i> ” de <i>Loop</i> . . . . .	67
4.11. Ejemplo real de la creación de parámetros . . . . .	68
4.12. Interfaz del editor de Cozmo . . . . .	69
4.13. Paleta del catálogo . . . . .	69
4.14. Paleta del flujo de actividades . . . . .	69
4.15. Interfaz con las opciones de respuestas . . . . .	70
4.16. Interfaz que pregunta si repite o no el flujo . . . . .	71
4.17. Pantalla de información de Cozmo . . . . .	72
4.18. Pantalla de error en la ejecución del programa . . . . .	73
4.19. Pantalla con la actualización de Cozmo cargando . . . . .	73
4.20. Pantalla con la conexión de Cozmo completa . . . . .	74
5.1. Ejemplo 1: Presentarse . . . . .	76
5.2. Tarea compleja Presentarse . . . . .	77
5.3. Ejemplo 2: Mostrar Alegria . . . . .	78
5.4. Tarea compleja MostrarAlegria . . . . .	79
5.5. Diagrama de flujo con la tarea pregunta . . . . .	80
5.6. Código Ejemplo 3: Flujo Complejo (1) . . . . .	81
5.7. Código Ejemplo 3: Flujo Complejo (2) . . . . .	82
5.8. Código Ejemplo 3: Flujo Complejo (3) . . . . .	83
5.9. Código Ejemplo 3: Flujo Complejo (4) . . . . .	84
A.1. Meta-modelo del catálogo . . . . .	112
A.2. Meta-modelo del flujo de actividades . . . . .	113
A.3. Propiedades File Exetension modificadas . . . . .	115
A.4. Representación del modelo con el fichero gmfmap . . . . .	120

## ÍNDICE DE FIGURAS

---

A.5. Propiedades del nodo <i>Activity</i> del fichero gmfmap . . . . .	121
A.6. Ejemplo de flujo de actividades sin acabar . . . . .	136
A.7. Actividades con parámetros por definir . . . . .	137
A.8. Ejemplo de valores en las actividades . . . . .	137
A.9. Ejemplo flujo de actividad . . . . .	138
A.10. Generación de código . . . . .	139
A.11. Creación tarea simple . . . . .	140
A.12. Propiedades de la tarea compleja . . . . .	141
A.13. Propiedades del parámetro de la tarea compleja . . . . .	142
A.14. Definición de "Activity" como tarea compleja . . . . .	143
A.15. Resultado de la asignación de definición de tareas . . . . .	143
A.16. Ejemplo de valores en las actividades . . . . .	143
A.17. Flujo de actividad con tarea compleja . . . . .	144
A.18. Expresión de Cozmo con sus ojos . . . . .	146
A.19. Estructura de la creación de carpetas . . . . .	148
A.20. Estructura final . . . . .	149
A.21. Creación tarea simple . . . . .	150
A.22. Relación entre parámetros . . . . .	150
A.23. Catálogo completo . . . . .	151

# **Capítulo 1**

## **Introducción**

Durante la reciente pandemia de COVID-19, uno de los colectivos más afectado ha sido probablemente el de los niños. El confinamiento, la cancelación de las clases presenciales o el no poder coincidir ni jugar con sus amigos, supuso un cambio enorme en sus vidas para el que, sobre todo los más pequeños, no entendían el porqué. Entre las consecuencias de la pandemia, algunos informes han reportado un aumento considerable de los trastornos mentales en menores. De hecho, según la ONG Save The Children [1], los trastornos mentales afectan hoy al 4 % de niños y adolescentes (entre 4 y 14 años), mientras que en 2017 afectaban solo al 1,1 %. Estos trastornos incluyen, además de la depresión o la ansiedad, síntomas como el aumento de la irritabilidad o la frustración. Los trastornos del comportamiento también han aumentado en el mismo rango de edad del 2,5 % al 6,9 %. Estos últimos presentan síntomas como el déficit de atención o la hiperactividad.

Si la pandemia afectó en general a los niños, su impacto en aquellos que padecían, por ejemplo, un Trastorno del Espectro Autista (TEA), fue mayor aún si cabe, ya que sus necesarias rutinas se vieron completamente alteradas de repente. Actualmente, se estima que el porcentaje de niños y niñas con TEA es inferior al 1 % de la población [2] [3] [4]. Entre otras dificultades, las personas con TEA suelen (con distinto grado de afectación) sufrir aislamiento social, presentar patrones de conducta estereo-

---

tipados y tener dificultades para reconocer los estados emocionales ajenos, para comunicarse e interactuar con otras personas y para adaptarse a cambios en su entorno o en sus rutinas.

Estas dificultades crean barreras para el acceso, la participación y el aprendizaje de los niños que sufren de TEA. Para ayudarles a superar estas dificultades, existen diversos programas de apoyo tanto en los centros educativos como en diversos centros externos especializados. Estos últimos cuentan servicios para ayudar tanto a los niños como a sus familias, haciéndoles más capaces de aprender nuevas tareas, de comunicarse, de disfrutar de distintos tipos de actividades, participar en juegos y afrontar situaciones difíciles. En Cáceres, existen varios centros especializados en TEA como el *Taller de los Sueños*, AFTEA (Asociación de Familias de Personas con TEA) o DIVERTEA (Asociación para personas con TEA y sus Familias).

La mayoría de los centros especializados en TEA incorporan, en mayor o menor medida, tecnología en sus actividades. Resulta frecuente que apoyen sus terapias con imágenes mostradas en tablets o presentaciones de ordenador, incluso que algunas de sus actividades giren en torno a algún videojuego. Sin embargo, la mayoría de los dispositivos y aplicaciones que utilizan son de propósito general y ni están específicamente diseñados para este tipo de terapias ni son personalizables en función de las necesidades de cada niño o niña con TEA.

Entre las tecnologías que parece que está obteniendo mejores resultados en las terapias para niños con TEA, cabe destacar la robótica. Según investigaciones recientes, se ha descubierto que los niños con autismo mejoran su capacidad de entender y expresar sus emociones cuando interatúan con robots [5].

Actualmente, el mundo de la robótica avanza a pasos agigantados y los robots cubren necesidades cada vez más específicas. Entre los más extendidos, además de los robots industriales utilizados, por ejemplo, en cadenas de montaje, cabe destacar los robots educativos [6]. Estos robots suelen ser de un tamaño pequeño o mediano y sueles poder adquirirse a precios razonablemente asequibles. Los hay desde muy simples, capaces

## 1.1. OBJETIVOS

---

sólo de ejecutar un número reducido de tareas predefinidas, hasta otros más complejos, que pueden programarse y que, cada vez más, cuentan con características avanzadas como el reconocimiento de voz o de expresiones faciales.

Los robots que suelen utilizarse con niños autistas suelen tener una actitud y un aspecto amigables y suelen poder demostrar emociones. El objetivo que se persigue incorporando estos robots a las terapias es que los niños aprendan a identificar y responder adecuadamente distintas situaciones, interacciones y emociones.

### 1.1. Objetivos

Para abordar algunos de los problemas antes descritos, el principal objetivo de este Trabajo Fin de Grado es desarrollar una herramienta que facilite a los terapeutas, especialistas en TEA, la definición, composición, reutilización y ejecución de distintas actividades en el robot educativo Cozmo, a fin de que puedan incorporarlo en algunas de sus terapias. Dado que estos terapeutas, en general, no tienen conocimientos sobre robótica o programación, la herramienta que se desarrolle deberá ser muy sencilla e intuitiva de aprender y utilizar.

Este objetivo principal puede desglosarse en los siguientes sub-objetivos:

- Analizar los requisitos funcionales y no funcionales de la herramienta que se va a desarrollar.
- Estudiar qué acciones básicas debe ofrecer el robot.
- Estudiar las herramientas disponibles para programar las tareas anteriores en el robot Cozmo.
- Utilizando la herramienta seleccionada, desarrollar una librería que implemente las acciones básicas previamente seleccionadas.
- Estudiar las herramientas de modelado y generación de código disponibles y seleccionar las más adecuadas para el Proyecto.

## 1.2. METODOLOGÍA

---

- Utilizando las herramientas anteriores:
  - Diseñar e implementar un lenguaje gráfico de modelado que permita representar, de forma sencilla e intuitiva, las actividades planificadas por el terapeuta. Estas actividades se definirán a partir de tareas previamente implementadas y disponibles en un catálogo. Las tareas del catálogo podrán ser convenientemente personalizadas (configurando sus parámetros) una vez incorporadas al diseño de una nueva actividad. Los modelos gráficos deberán poder ser validados, para comprobar su corrección sintáctica y semántica.
  - Diseñar e implementar una transformación modelo-a-texto que, a partir de los modelos gráficos anteriores, genere automáticamente el código correspondiente del robot. Cuando el modelo de partida incluya alguna estructura condicional, también se generará una aplicación de consola que permitirá al terapeuta controlar la ejecución de la actividad. El código generado deberá poder enviarse y ejecutarse en el robot de forma sencilla.
  - Diseñar e implementar un lenguaje gráfico de modelado que permita representar las tareas incluidas en el catálogo. Este catálogo que, inicialmente contará sólo con las tareas simples definidas en la librería del robot, deberá poder extenderse con nuevas tareas definidas a partir de actividades previamente diseñadas por los terapeutas. Estas nuevas tareas podrán parametrizarse y reutilizarse como parte de nuevas actividades.

## 1.2. Metodología

Este Trabajo Fin de Grado se ha realizado siguiendo una metodología ágil [7] con iteraciones semanales. Al final de cada iteración se realizaba una reunión con los tutores del Proyecto, en la que se reportaban los avances realizados y las dificultades encontradas, y se planificaba el trabajo para la siguiente iteración.

### 1.3. PLANIFICACIÓN

---

Las reuniones semanales se han llevado a cabo tanto de forma presencial como telemática. En cada una de ellas, se ha redactado un acta breve recogiendo los temas tratados. Además, se ha utilizado la herramienta Click Up [8] para establecer las tareas definidas en cada iteración y registrar las horas dedicadas a cada una de ellas. Como herramienta de control de versiones para el Proyecto, se ha utilizado GitHub [9].

Conviene destacar que toda la documentación, tanto externa como interna, se ha ido realizando progresivamente a medida que se iba avanzando en el Proyecto y no al final. Esto ha permitido no sólo registrar todas las tareas realizadas, sino también documentar los cambios que ha sufrido tanto el diseño como la implementación de la herramienta.

## 1.3. Planificación

A continuación, se listan las tareas desarrolladas a lo largo del Proyecto, con una breve explicación e indicación de las horas empleadas en cada una de ellas.

- **T01. Análisis de requisitos:** en esta tarea se han definido los principales requisitos, funcionales y no funcionales, del sistema. En este caso, se han especificado las necesidades presentadas por el personal del Taller de los Sueños para identificar diversos aspectos relacionados con el sistema. En esta tarea se han empleado alrededor de 20 horas.
- **T02. Estudio de herramientas:** en esta tarea se ha realizado un estudio comparativo de las diferentes herramientas disponibles para implementar cada uno de los elementos del sistema. A esta tarea se le ha dedicado alrededor de 25 horas.
- **T03. Diseño e implementación de las funcionalidades básicas en Cozmo:** en esta tarea se han definido y desarrollado las diferentes tareas básicas que podrá realizar el robot Cozmo. En esta tarea se han empleado alrededor de 45 horas.
- **T04. Diseño del sistema:** en esta tarea se ha diseñado la arquitectura completa del sistema, incluidos todos sus componentes. A esta tarea se le ha dedicado

#### 1.4. ESTRUCTURA DEL DOCUMENTO

---

alrededor de 15 horas.

- **T05. Implementación de los lenguajes gráficos de modelado para especificar el catálogo y los workflows de actividades:** como parte de esta tarea se han desarrollado los meta-modelos que definen las sintaxis abstractas de ambos lenguajes y los editores gráficos asociados. En esta tarea se han empleado alrededor de 60 horas.
- **T06. Implementación de las transformaciones modelo-a-texto:** en esta tarea se ha implementado el generador de código que se ejecutará en el robot Cozmo y, eventualmente, el generador de la aplicación de consola que será utilizada por los terapeutas para controlar el flujo de ejecución de la actividad. En esta tarea se han empleado alrededor de 40 horas.
- **T07. Puesta en marcha y validación de la herramienta:** como parte de esta tarea se han creado varios modelos de ejemplo con los editores gráficos desarrollados y se han generado las aplicaciones correspondientes, cuyo funcionamiento se ha validado en el robot Cozmo. A esta tarea se le ha dedicado alrededor de 35 horas.
- **T08. Documentación:** como parte de esta tarea se ha creado la documentación del Proyecto, tanto interna (in-code) como externa (este documento). En esta tarea se han empleado alrededor de 70 horas.

A continuación, la Figura 1.1 muestra el Diagrama de Gantt del Proyecto, en el que se recogen las tareas anteriores y su planificación en el tiempo. Como se puede observar, se ha dedicado un total de unas 310 horas.

## 1.4. Estructura del documento

El resto del documento está organizado del siguiente modo:

## 1.4. ESTRUCTURA DEL DOCUMENTO

---

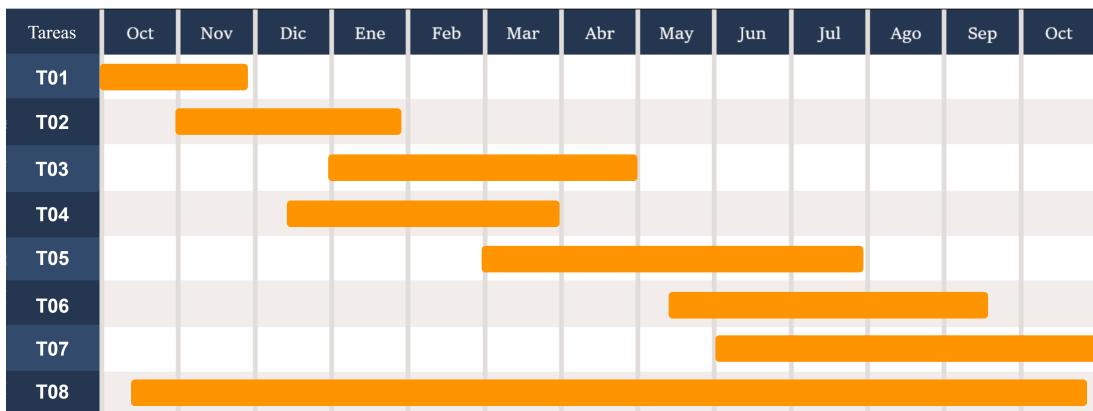


Figura 1.1: Diagrama de Gantt

- **Capítulo 2.** Analiza el estado actual de la técnica, los trabajos relacionados y las herramientas analizadas, justificando las elecciones realizadas;
- **Capítulo 3.** Describe el análisis y el diseño del sistema. Detalla los metamodelos diseñados para describir la sintaxis abstracta de los dos lenguajes de modelado definidos en el Proyecto;
- **Capítulo 4.** Detalla la implementación de los dos editores gráficos de modelos, de los generadores de código y de la librería que recoge las funcionalidades básicas implementadas para el robot Cozmo;
- **Capítulo 5.** Desarrolla varios ejemplos de uso de la herramienta *PiLHaR*, de complejidad creciente;
- **Capítulo 6.** Describe las conclusiones del Proyecto, los principales problemas encontrados y algunos posibles trabajos futuros.
- **Bibliografía.** Recoge la bibliografía referenciada en este documento.
- **Anexos.** Incluye siete anexos, algunos de los cuales recogen detalles de implementación y otros que incluyen los manuales de uso e implementación de las distintas herramientas.

# Capítulo 2

## Estado de la Técnica

Este capítulo comienza introduciendo los antecedentes (*background*) del Proyecto. A continuación, se analizan las distintas herramientas y tecnologías disponibles para abordar su implementación y se justifican las elecciones tomadas. Por último, se describen algunos trabajos relacionados y se justifica por qué no cubren adecuadamente los objetivos planteados al principio del Proyecto.

### 2.1. Background

En esta sección se describen brevemente las principales características del robot *Cozmo*, utilizado en el Proyecto y de su variante *Vector*, y se introducen los conceptos básicos relacionados con la Ingeniería del Software Dirigida por Modelos y la programación de robots.

#### *El robot Cozmo (y su variante Vector)*

El robot **Cozmo** [10] (ver Figura 2.1), fabricado por la empresa Digital Dream Labs, está enfocado a un segmento de la población con una edad inferior a los 14 años. Este robot, cuyo precio ronda los 400€, puede mover sus brazos, desplazarse y girar, mostrar en su pantalla distintas expresiones faciales (sólo con los ojos) e imágenes, repro-

## 2.1. BACKGROUND

ducir sonidos o encender sus luces led en distintos colores. Existen distintas librerías para programar su funcionamiento, disponibles en distintos lenguajes de programación, aunque probablemente las más utilizadas sean las programadas en Python (ver sección 2.2.3).



Figura 2.1: Robot Cozmo

Una alternativa a Cozmo, fabricado por la misma empresa, es el robot **Vector** [11]. Este robot, posterior a Cozmo, es más completo y está enfocado a un segmento de la población un poco más mayor (de 15 años en adelante). Su precio ronda los 450€y, además de las funcionalidades incluidas en Cozmo, Vector puede, entre otras cosas, ser controlado mediante comandos de voz.

Aunque existen otros muchos robots educativos (algunos se revisan más adelante, en la sección 2.3) en este Proyecto hemos trabajado con el robot Cozmo, ya que era el que teníamos disponible, gracias al préstamo de la profesora Pilar Bachiller.

### *Ingeniería del Software Dirigida por Modelos*

El término Ingeniería del Software Dirigida por Modelos (ISDM) [12] hace referencia a un paradigma de Ingeniería de Software que promueve el uso sistemático de modelos como elemento central en todas las etapas del desarrollo de software. El uso de modelos, permite a los desarrolladores abstraer detalles relativos a la implementación y centrarse en aquellos aspectos verdaderamente relevantes. De hecho, a partir de un modelo es posible obtener diversas implementaciones para distintas plataformas. Según este enfoque, todo es un modelo, incluso las herramientas utilizadas para definir

## 2.2. HERRAMIENTAS

---

los lenguajes de modelado, los editores de modelos, las transformaciones de modelo-a-modelo y de modelo-a-texto se definen utilizando modelos. La abstracción que proporcionan los modelos, junto con la posibilidad de generar código automáticamente a partir de ellos, hacen que este enfoque logre mejoras considerables, no sólo en cuanto a productividad sino también en cuanto a calidad y mantenibilidad del software. La sección 2.2.1 recoge algunas de las herramientas disponibles para dar soporte a este paradigma.

### ***Programación de robots***

Los robots son sistemas complejos que integran diversos componentes hardware (sensores, actuadores, controladores, etc.) y software. La programación de robots es una tarea compleja y, todavía hoy, bastante artesanal. El software desarrollado para lograr que un robot realice una tarea, difícilmente puede ser reutilizado en otro robot, incluso para realizar una tarea similar. Existen numerosos *frameworks* destinados a facilitar el desarrollo de software para robótica (algunos de ellos descritos más adelante, en la sección 2.2.2), si bien, en general, son bastante complejos de utilizar y suelen tener curvas de aprendizaje largas.

## **2.2. Herramientas**

Esta sección se divide en varias subsecciones, cada una de ellas centrada en uno de los componentes principales de la herramienta *PiLHaR*. En cada subsección se analizarán las herramientas y tecnologías disponibles para desarrollar los siguientes elementos: herramientas de modelado, lenguajes de programación de robots y librerías para la programación del robot Cozmo [10]. Al final de cada subsección, se realizará una comparativa de las herramientas/tecnologías analizadas y se justifica el por qué de la elección de unas frente a otras para el desarrollo del Proyecto.

## 2.2. HERRAMIENTAS

---

### 2.2.1. Herramientas de modelado

A continuación, se presentan las dos posibles herramientas que se han barajado para diseñar e implementar los distintos editores gráficos de modelos y generadores de código incluidos en el sistema.

**Eclipse** [13]: es un Entorno Integrado de Desarrollo (IDE, de sus siglas en inglés, *Integrated Development Environment*), gratuito y de código abierto. Eclipse permite, mediante un mecanismo de extensión basado en *plug-ins*, integrar diferentes aplicaciones relacionadas con el desarrollo de proyectos de software, dando soporte a todas las fases del proceso, desde el análisis y el modelado a la implementación, prueba, simulación y gestión de este tipo de proyectos. A continuación, se listan algunos de los componentes de Eclipse que se ha barajado utilizar para el desarrollo de este Proyecto:

- **Eclipse Modeling Framework (EMF)** [14]: se trata de una herramienta que facilita la especificación de la sintaxis abstracta de nuevos lenguajes de modelado mediante la definición de meta-modelos basados en Ecore: una implementación de un subconjunto del estándar de modelado MOF (*Meta-Object Facility*). EMF permite la definición de meta-modelos y la generación, a partir de ellos, de una infraestructura básica de modelado que incluye, entre otros artefactos, un editor básico de modelos en forma de árbol y una serie de facilidades para validar los modelos creados con este editor contra su correspondiente meta-modo.
- **OCLInEcore** [15]: es una herramienta de especificación de restricciones basada en el estándar OCL (*Object Constraint Language*) que permite anotar los meta-modelos EMF con restricciones sintácticas adicionales, imposibles de expresar en Ecore. Así, OCLInEcore complementa a EMF en la tarea de especificar la sintaxis abstracta de nuevos lenguajes de modelado.
- **Graphical Modeling Framework (GMF)** [16]: es una herramienta basada en EMF y GEF (*Graphical Editing Framework*) para la generación de editores gráficos de modelos a partir de meta-modelos EMF. A partir de los conceptos de

## 2.2. HERRAMIENTAS

---

modelado recogidos en un meta-modelo EMF (sintaxis abstracta), GMF permite al desarrollador asociarles una sintaxis gráfica concreta y una herramienta de creación en la paleta del editor. De este modo, con un sencillo mapeado entre concepto-representación-herramienta, GMF es capaz de generar automáticamente un editor gráfico completamente funcional con el que los diseñadores pueden crear y validar modelos gráficos conformes con al lenguaje de modelado de partida.

- **EuGENia** [17]: es una herramienta basada en EMF y GMF que facilita la especificación de la sintaxis gráfica concreta asociada a un lenguaje de modelado. EuGENia permite anotar directamente los meta-modelos EMF con una serie de etiquetas con las que el diseñador puede configurar el aspecto gráfico que tendrá cada concepto de modelado. A partir del meta-modelo anotado, EuGENia genera automáticamente los modelos GMF y, a partir de ellos, el editor gráfico correspondiente.
- **Acceleo** [18]: es un lenguaje de transformación modelo-aTexto basado en plantillas con el que resulta sencillo especificar cómo generar automáticamente código a partir de modelos definidos conforme a meta-modelos EMF. Acceleo soporta, entre otras características, sobrecarga y polimorfismo.

**EMF cloud** [19]: conjunto de componentes y tecnologías que hacen que EMF y sus beneficios estén disponibles en la web y la nube. Esto incluye nuevos *frameworks* específicos, pero también soluciones que permiten reutilizar on-line algunos de los componentes y herramientas relacionadas con Eclipse EMF. Algunos de sus componentes son:

- **Herramienta Theia Ecore** [20]: permite crear y modificar gráficamente modelos Ecore utilizando un editor de diagramas integrado en la web, basado en el IDE Eclipse Theia.
- **Ejemplo proporcionado** [21]: los desarrolladores de EMF.cloud ofrecen en su web un ejemplo completo, denominado “coffee editor”, que ilustra cómo pue-

## 2.2. HERRAMIENTAS

---

den usarse las distintas herramientas integradas en este entorno para construir editores de modelos web.

- **Otros componentes:** EMF.cloud incluye, entre otras, herramientas para (1) convertir JSON en modelos EMF, permitiendo la serialización y deserialización de recursos EMF en JSON; y (2) un framework de tipo Tree Editor, que permite crear editores centrados en datos de Eclipse Theia de manera eficiente.

**Justificación de las herramientas seleccionadas.** La Tabla 2.1 muestra una comparativa de algunas de las características consideradas a la hora de tomar la decisión sobre cuál de los dos entornos de desarrollo utilizar. Aunque contábamos con experiencia en el manejo de las herramientas de modelado ofrecidas por Eclipse (utilizadas en la asignatura de Diseño y Modelado de Sistemas Software, de tercer curso del Grado en Ingeniería Informática en Ingeniería del Software), inicialmente, se propuso el uso de EMF.cloud como entorno de desarrollo para el Proyecto, ya que permitía construir editores de modelos web, que podían ser utilizados tanto desde un ordenador como desde una tablet o incluso un teléfono móvil. Sin embargo, tras analizar los ejemplos disponibles (gratuitos y de código abierto), observamos que el uso algunas de las herramientas de EMF.cloud, necesarias para el desarrollo del Proyecto, ni eran de código abierto ni eran gratuitas. Así, finalmente, se optó por utilizar Eclipse y, dentro de Eclipse, las herramientas de modelado que ya se conocían (previamente descritas). Se barajaron algunas alternativas a GMF/EuGENia (por ejemplo, Sirius) y a Acceleo (por ejemplo, Xpand), pero la mayoría tenían importantes curvas de aprendizaje y no aportaban ninguna ventaja significativa.

## 2.2. HERRAMIENTAS

---

Característica	EMF.cloud	Eclipse
Gratis	X	✓
Código abierto	X	✓
Usado anteriormente	X	✓
Facilidad de uso	✓	✓
Ejecución en la nube	✓	X

Tabla 2.1: Comparativa herramientas de modelado

### 2.2.2. Lenguajes y *frameworks* para la programación de robots

Actualmente, la proliferación en el mercado de robots de bajo coste, asequibles para el público en general, ha dado lugar a la aparición de una gran variedad de lenguajes y *frameworks* para programarlos. Entre ellos, cabe destacar los siguientes:

- **RoboComp** [22]: es un *framework* de robótica de código abierto, que permite el desarrollo, despliegue y ejecución de aplicaciones software basadas en componentes. Las comunicaciones son controladas por el *framework* Ice [23], siguiendo el enfoque adoptado por el sistema robótico Orca2 [24]. Sus principales objetivos son la reutilización, la eficiencia y la sencillez. Como lenguajes de programación utiliza principalmente C++ y Python, si bien el uso de Ice facilita la integración con otros muchos lenguajes de programación.
- **SmartSoft** [25]: es un *framework* y también una metodología de desarrollo de software para robótica. SmartSoft se basa en los paradigmas de desarrollo de software basado en componentes y dirigido por modelos y ofrece herramientas específicas, orientadas a los distintos roles involucrados en el desarrollo de soluciones robóticas, entre ellos, los de diseñador, desarrollador e integrador. Los componentes SmartSoft pueden desarrollarse utilizando diversos lenguajes de programación, siendo C++ el más utilizado.

## 2.2. HERRAMIENTAS

---

- **Scratch** [26]: es un entorno orientado al aprendizaje de la programación de aplicaciones en general (no sólo circunscritas al ámbito de la robótica), que permite a personas sin nociones previas, crear proyectos de forma visual gracias a su editor. Esto se consigue gracias al uso de bloques predefinidos, en forma de ficha de puzzle, que encajan entre sí. De este modo, gracias a su sencilla interfaz, personas sin ninguna experiencia pueden lograr desarrollar aplicaciones sencillas, historias digitales, juegos o animaciones en poco tiempo.

Scratch está diseñado, desarrollado y moderado por una organización sin ánimo de lucro. Es gratuito y está disponible en más de 70 idiomas. Está pensado para promover el pensamiento computacional y las habilidades de resolución de problemas, la enseñanza y el aprendizaje creativos, la auto expresión y el desarrollo de soluciones de forma cooperativa.

- **Python** [27]: es un lenguaje de programación de alto nivel y de propósito general (no orientado específicamente a la programación de robots) que hace hincapié en la legibilidad del código. Se trata de un lenguaje multi-paradigma, funcional, dinámico y multi-plataforma. Es de código abierto y es considerado uno de los lenguajes de programación más populares actualmente.

El código escrito en Python es sencillo, legible, compacto y elegante. Su curva de aprendizaje se considera corta (al menos para quienes cuentan con algunas nociones sobre programación), ya que se basa el uso de reglas sencillas y cuenta con numerosas bibliotecas que ofrecen soluciones listas para ser usadas en numerosos ámbitos de aplicación. De este modo, es posible programar algoritmos complejos en pocas líneas de código de forma sencilla.

**Justificación del lenguaje seleccionado.** La Tabla 2.2 muestra una comparativa de algunas de las características consideradas a la hora de tomar la decisión sobre cuál de los *frameworks* o lenguajes de programación analizados utilizar. Como se observa en la tabla, si bien RoboComp y SmartSoft ofrecen herramientas específicamente orientadas al desarrollo de software para robóticas, su curva de aprendizaje es larga y,

## 2.2. HERRAMIENTAS

---

en general, se utilizan para programar robots considerablemente más complejos que el robot Cozmo. Entre Scratch y Python se ha optado por este último por dos razones fundamentales. Por una parte, la gran cantidad de librerías disponibles en Python que podían utilizarse directamente en el proyecto (por ejemplo, librerías para convertir texto en voz) y, por otra que, contando con experiencia en programación, el uso de un entorno de programación textual (en lugar de uno gráfico) nos resultaba más cómodo y apropiado.

Característica	Robocomp	SmartSoft	Scratch	Python
Gratis	✓	✓	✓	✓
Código abierto	✓	✓	✓	✓
Prog. basada en componentes	✓	✓	✗	✗
Amplitud de código	✓	✓	✗	✓
Facilidad de uso	✗	✗	✓	✓

Tabla 2.2: Comparativa lenguajes de programación

### 2.2.3. Librerías Python para la programación del robot Cozmo

Por último, una vez decidido que se iba a utilizar Python como lenguaje de programación, en esta sección se describen las dos librerías existentes en este lenguaje para conectarse y programar el robot Cozmo.

- **Cozmo SDK:** los creadores de Cozmo desarrollaron este SDK (*Software Development Kit*) para facilitar la programación de sus robots ofreciendo a la vez acceso a algunas de las tecnologías robóticas más avanzadas. Para usar Cozmo SDK [28] es necesario instalar la aplicación de Cozmo en un dispositivo móvil, que deberá estar conectado a un ordenador a través de un cable USB. Cozmo SDK ofrece una gran variedad de métodos para controlar todas las funcionalidades del robot, desde el manejo de sus ruedas, cabeza o brazos hasta el detector de mascotas, cantar canciones o el control de la cámara.

## 2.2. HERRAMIENTAS

---

- **PyCozmo** [29]: es una librería de Python que permite controlar los robots Cozmo directamente, sin necesidad de usar continuamente la aplicación Cozmo en un dispositivo móvil. Esta aplicación sólo será necesaria para actualizaciones periódicas del robot. PyCozmo es una librería de código abierto implementada por usuarios sin ánimo de lucro y no está afiliada con Digital Dream Labs [30], la empresa dueña del robot. PyCozmo ofrece soporte de todos los componentes hardware que incluye el robot, tales como sensores, actuadores, funciones integradas (punto de acceso Wi-Fi, Bluetooth LE y localización, entre otros) y otras funciones externas. Además, también existe una gran cantidad de herramientas externas y códigos de ejemplo para aprender las principales funcionalidades de la librería.

**Justificación de la librería seleccionada.** La Tabla 2.3 muestra una comparativa de algunas de las características consideradas a la hora de tomar la decisión sobre cuál de las dos librerías Python para programar el robot Cozmo utilizar. En este caso, la decisión fue sencilla. Cozmo SDK requería del uso de un dispositivo móvil con el que la aplicación debía estar en constante comunicación a través de USB. Además, los métodos de Cozmo SDK están implementados de una forma muy restrictiva, dificultando (cuando no impidiendo) el desarrollo de muchas de las tareas planteadas en el Proyecto. Por el contrario, PyCozmo nos facilitó el desarrollo de las todas las tareas planteadas y no requería del uso de ningún dispositivo externo adicional, permitiendo la comunicación directa con el robot a través de Wi-Fi. Por todo ello, se optó por usar PyCozmo en lugar de Cozmo SDK.

### 2.3. OTROS ROBOTS EDUCATIVOS

---

Característica	Cozmo SDK	PyCozmo
Gratis	✓	✓
Código abierto	✓	✓
Voz en español disponible	✗	✓
Facilidad de uso	✓	✓
Independencia de dispositivo móvil	✗	✓

Tabla 2.3: Comparativa de librerías del robot

## 2.3. Otros robots educativos

A continuación, se comentan brevemente algunas de las características principales de dos robots educativos programables, similares a Cozmo, y también utilizados como herramienta de soporte en terapias con niños.

### 2.3.1. NAO

Nao [31] es un robot de actitud y aspecto amigables, capaz de expresar emociones, para que los niños y niñas con autismo aprendan a reconocerlas, imitarlas y reaccionar adecuadamente a ellas. Tal y como se indica en [32], “*Con la ayuda del MIT Media Lab [33], los responsables del proyecto han incluido en este androide un sistema de aprendizaje automático y personalizado que ayuda a estimar el compromiso y el interés de los niños y niñas con autismo durante sus interacciones. Esta disposición es fundamental para alcanzar un resultado óptimo, ya que así el robot puede responder de manera adecuada al comportamiento del paciente, quien debe sentirse cómodo y seguro durante las sesiones. Esto resulta tremadamente complejo incluso para los terapeutas humanos, que no siempre son capaces de interpretar correctamente los movimientos, gestos y palabras de los niños y niñas con autismo.*”

Asimismo, tal y como se indica en [34], “*el objetivo a largo plazo no es crear robots*

### 2.3. OTROS ROBOTS EDUCATIVOS

---

*que reemplacen a los terapeutas humanos, sino facilitar su labor, proporcionándoles información clave que puedan usar para personalizar el contenido de las terapias, haciendo las interacciones con sus pacientes más atractivas y naturales.”.*

Sin embargo, a pesar de los beneficios que el uso de robots ha demostrado tener en las terapias relacionadas con el autismo y la ayuda que estos pueden suponer para los terapeutas, su manejo y programación resulta una tarea considerablemente compleja. Tal y como señala Rosalind Picard en [34], “crear un sistema de inteligencia artificial (IA) que interactúe correctamente con pacientes autistas resulta particularmente desafiante, ya que los métodos tradicionales de la IA requieren de una gran cantidad de datos similares para poder identificar y aprender los rasgos distintivos de cada categoría. Así, en el autismo, donde reina la heterogeneidad, los enfoques convencionales de la IA suelen fracasar.”.

#### 2.3.2. Robot Atent@

Atent@ [35] es un robot orientado a ayudar al alumnado con Trastorno por Déficit de Atención e Hiperactividad (TDAH). Este trastorno ocasiona una continua distracción en quienes lo sufren, haciéndoles muy difícil planificar su tiempo y sus actividades, para lo que suelen requerir supervisión constante. El robot Atent@ nació para intentar que el alumnado afectado por este trastorno lograra ser lo más autónomo posible a la hora de hacer sus deberes en casa.

Atent@ es un robot diseñado para complementar las terapias de los niños y niñas con TDAH. Esta tecnología pretende acompañarles para que logren planificar mejor sus tareas, mantener la atención durante su realización y organizar mejor su espacio de trabajo.

En el desarrollo de este robot ha participado un grupo de investigadores de la Universidad Politécnica de Madrid (UPM). Como primer paso, han diseñado objetos inteligentes que se amoldan a objetos cotidianos del hogar (como el escritorio o la silla), lo que

## 2.4. TRABAJOS RELACIONADOS

---

les permite vigilar el comportamiento de los niños y niñas con TDAH mientras hacen los deberes. Estos objetos están conectados a un robot que procesa la información e interactúa con el menor ayudándole a centrar su atención. *“Los objetos inteligentes integrados en el hogar son los que detectan y formalizan las acciones del estudiante conforme a las pautas de interés establecidas por los terapeutas. [...] Así, cuando se detecta que el estudiante pierde el foco de atención en la tarea que estaba realizando, el robot realiza pequeñas intervenciones para ayudarle que vuelva a centrarse en ella.”.*

## 2.4. Trabajos relacionados

A continuación, se describen brevemente algunos trabajos relacionados con este Proyecto, en los que también se han desarrollado herramientas para facilitar el desarrollo de software para robots.

Trapani, S. e Indri, M. en [36] presentan un trabajo de investigación en el que definen una interfaz de programación orientada a tareas y dirigida a usuarios sin conocimientos previos de programación. Esta interfaz permite especificar tanto los componentes del robot como las tareas que éstos pueden llevar a cabo. Así, crear programas complejos consiste en orquestar las distintas tareas que debe llevar a cabo cada uno de los componentes del robot.

Kerr, D. et al. [37] ofrecen un enfoque de generación de código diferente al anterior, pero también basado en la presunción de que los usuarios finales no poseen conocimientos previos de programación de robots. En este caso, el robot imita los gestos que realiza el usuario y, para cada gesto aprendido, se genera automáticamente el código asociado. De este modo, especificando la secuencia de gestos que se desea que realice el robot, es posible crear automáticamente programas arbitrariamente complejos.

Por último, Haiyang, H. et al. [38], proponen programar automáticamente robots industriales a través de instrucciones en lenguaje natural. Para ello, en primer lugar,

## 2.4. TRABAJOS RELACIONADOS

---

describen detalladamente el entorno de trabajo del robot (posición, atributos y restricciones de los elementos con los que interactúa) y, a continuación, buscan el porcentaje de coincidencia entre las instrucciones dadas en lenguaje natural y las acciones e interacciones del robot con los objetos de su entorno. En base a estos porcentajes, logran predecir qué acciones del robot se corresponden con cada instrucción y así, logran generar programas a partir de instrucciones dadas en lenguaje natural.

Conviene señalar que, si bien estas propuestas están orientadas a facilitar la programación de robots a usuarios sin la experiencia ni los conocimientos de programación necesarios para ello, ninguna de ellas utiliza un enfoque de Ingeniería del Software Dirigida por Modelos. Tampoco ofrecen un entorno gráfico de trabajo que permita especificar, de forma sencilla e intuitiva, las tareas que se desea que lleve a cabo el robot, ni ofrecen facilidades para validar la corrección de sus especificaciones y favorecer su reutilización. Conviene señalar, sin embargo, que estas propuestas y algunas otras similares, están dirigidas a facilitar la programación de robots que, en general, son mucho más complejos (tanto en cuanto a su estructura como en cuanto a su comportamiento) que el robot Cozmo, utilizado en este Proyecto. Por ello, así como probablemente no sería posible (o al menos lo más adecuado) utilizar el sistema aquí presentado para programar robots complejos, tampoco resultaría adecuado utilizar cualquiera de estas propuestas para abordar los objetivos planteados en este Proyecto.

# **Capítulo 3**

## **Análisis y diseño**

En este apartado se va a describir el análisis que se ha realizado previo al desarrollo del sistema, además del diseño que se ha definido para la arquitectura y la interfaz del Proyecto.

### **3.1. Análisis**

A continuación, se indican los distintos requisitos funcionales y no funcionales que se han definido para el sistema, así como los principales actores que van a interactuar con él y sus Casos de Uso asociados.

#### **3.1.1. Requisitos funcionales y no funcionales**

Las Tablas 3.1 y 3.2 recogen, respectivamente, los principales requisitos funcionales y no funcionales identificados para el sistema.

### 3.1. ANÁLISIS

---

Nº	Descripción
RF1	El sistema ofrecerá una librería que implementará las funcionalidades básicas del robot Cozmo. Cada una de estas funcionalidades se implementará como un método y podrá tener cualquier número de parámetros.
RF2	El sistema permitirá incorporar las funcionalidades programadas en la librería anterior a un catálogo de tareas simples para que estas puedan ser fácilmente utilizadas en el diseño de flujos de actividades. Las tareas simples del catálogo tendrán, como máximo, el mismo número de parámetros que el método correspondiente de la librería.
RF3	El sistema permitirá definir flujos de tareas a partir de las tareas disponibles en el catálogo, configurando sus parámetros cuando y como sea necesario.
RF4	El sistema permitirá añadir nuevas tareas (complejas) al catálogo a partir de cualquier flujo de tareas previamente diseñado.
RF5	El sistema ofrecerá un editor gráfico para facilitar la interacción de los usuarios con el catálogo de tareas.
RF6	El sistema ofrecerá un editor gráfico para facilitar a los usuarios el diseño de nuevos flujos de tareas.
RF7	El sistema permitirá generar automáticamente el código asociado a cada flujo de actividades para su ejecución en el robot Cozmo. Cuando sea necesario, también se generará una aplicación de consola para facilitar a los usuarios la interacción con el robot, específicamente, cuando el flujo de actividades incluya alguna estructura de control de tipo condicional.
RF8	El sistema permitirá enviar y ejecutar el código generado automáticamente en el robot Cozmo.
RF9	El sistema podrá conectarse con el robot Cozmo a través de una red Wi-Fi.

Tabla 3.1: Requisitos funcionales

Nº	Descripción
RNF1	El sistema debe ser fácil de usar y de aprender para usuarios sin conocimientos de programación.
RNF2	Los iconos incluidos en las paletas de los dos editores gráficos deben ser sencillos y deben representar de forma intuitiva los conceptos de modelado asociados.
RNF3	Las aplicaciones de consola generadas para ayudar a los usuarios a guiar la ejecución de los flujos de actividades deben ofrecer una interfaz de preguntas/respuestas muy simple y accesible.
RNF4	El sistema debe permitir al robot Cozmo hablar en español con una voz fácilmente entendible.
RNF5	El sistema debe ser escalable y flexible, permitiendo incorporar nuevas funcionalidades de forma sencilla.

Tabla 3.2: Requisitos no funcionales

### 3.1. ANÁLISIS

---

#### 3.1.2. Descripción de los actores

En el sistema existen dos tipos de actores principales, los cuales se describen a continuación:

- **Desarrolladores:** Este actor es el responsable de implementar las funcionalidades básicas del robot Cozmo y ofrecerlas a los usuarios de la herramienta PiLHaR a través de un catálogo. Cada una de las funcionalidades básicas identificadas se implementará como un método, con sus correspondientes parámetros, y se incorporará a una librería software. Una vez completada la implementación, para cada método de la librería se añadirá una tarea simple al catálogo gráfico de la herramienta con el mismo nombre y parámetros que el método correspondiente. Para poder llevar a cabo estas tareas, los desarrolladores deben contar con conocimientos avanzados de programación.
- **Terapeutas:** Este actor ofrece una atención personalizada a niños con TEA y utiliza la herramienta PiLHaR para diseñar y llevar a cabo actividades basadas en el uso del robot Cozmo. El sistema propuesto permite al terapeuta, de forma sencilla e intuitiva y sin necesidad de tener conocimientos sobre programación, diseñar y ejecutar el flujo de actividades que quiere que ejecute el robot, personalizándolo para cada niño.

Los terapeutas pueden desarrollar tantos flujos de actividades como deseen a partir del catálogo de tareas disponibles. También pueden incorporar al catálogo sus propios flujos de actividades (convenientemente parametrizados) para poder reutilizarlos y configurarlos posteriormente en otros diseños.

#### 3.1.3. Descripción y Diagramas de Casos de Uso

A continuación, se describen brevemente los Casos de Uso asociados a cada uno de los dos actores identificados en el apartado anterior.

### 3.1. ANÁLISIS

---

- **Desarrolladores:** la Tabla 3.3 recoge los Casos de Uso asociados al rol de los desarrolladores.

Nº CU	Descripción
01	Gestión de la librería vinculada a las tareas simples
02	Implementación del código correspondiente a cada una de las tareas simples
03	Inclusión de las tareas simples en el catálogo de la herramienta
04	Actualización del software de Cozmo

Tabla 3.3: Casos de Uso de los desarrolladores.

- **Terapeutas:** la Tabla 3.4 recoge los Casos de Uso asociados al rol de los terapeutas.

Nº CU	Descripción
05	Creación de flujos de actividades a partir de las tareas del catálogo y configuración de sus parámetros
06	Creación de tareas complejas en el catálogo a partir de flujos de actividades preexistentes para su reutilización
07	Generación del código que se mandará a Cozmo a partir de un flujo de actividades
08	Ejecución del código generado a partir de un flujo de actividades en Cozmo
09	Interacción con la interfaz del programa que se ejecuta en Cozmo para dirigir el flujo de tareas
10	Conexión con Cozmo a través de una red Wi-Fi
11	Actualización del software de Cozmo

Tabla 3.4: Casos de Uso de los terapeutas.

La Figura 3.1 muestra el Diagrama de Casos Uso del sistema y las siguientes tablas recogen una descripción detallada de los tres más relevantes: (1) CU03 (ver Tabla 3.5); (2) CU05 (ver Tabla 3.6); y (3) CU06 (ver Tabla 3.7).

### 3.1. ANÁLISIS

---

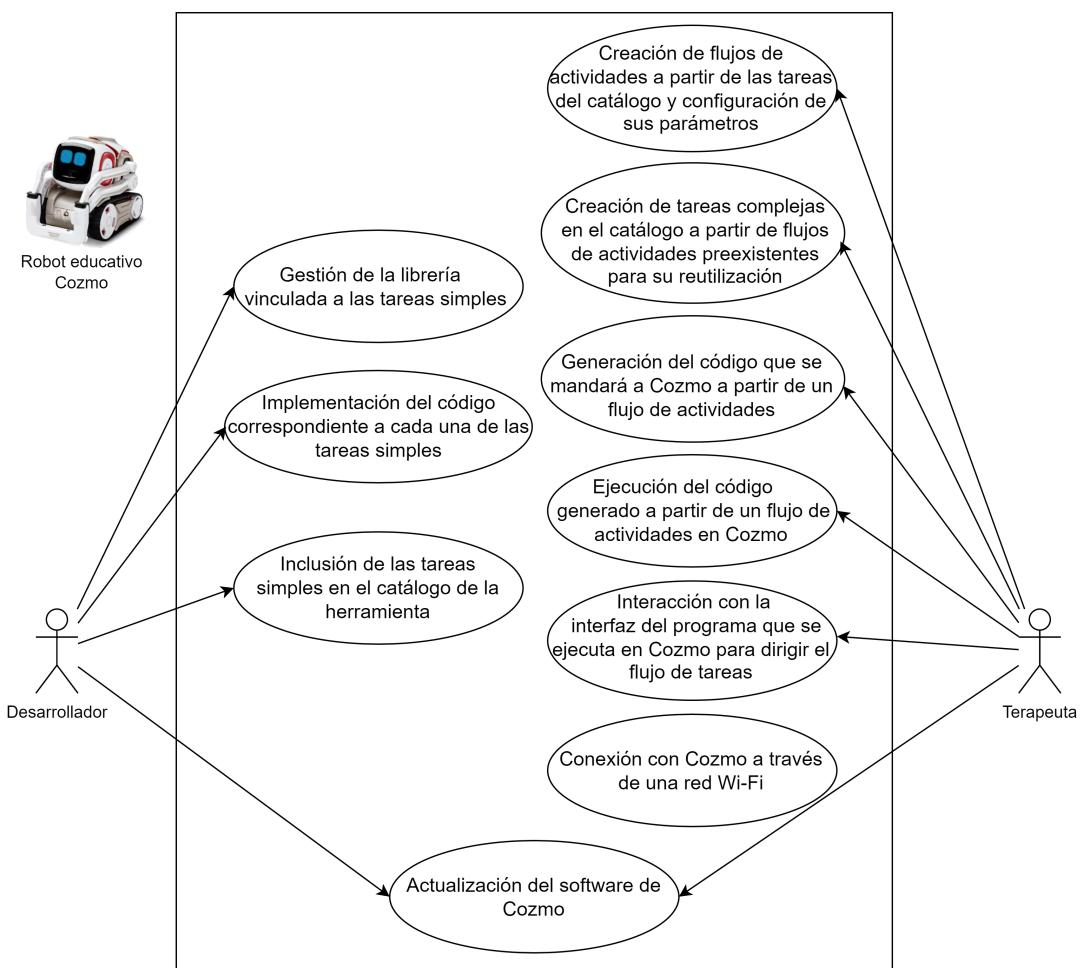


Figura 3.1: Diagrama de Casos de Uso.

### 3.1. ANÁLISIS

---

<b>CU N°03</b>	<b>Inclusión de las tareas simples en el catálogo de la herramienta</b>	
<b>Actor</b>	Desarrollador	
<b>Precondición</b>	El código asociado a la tarea simple que se desea añadir al catálogo debe estar implementado y añadido a la librería.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Abrir el catálogo.
	2	Seleccionar la tarea <i>SimpleTask</i> en la paleta.
	3	Hacer clic en la ventana de edición para colocar la tarea.
	4	Añadir como nombre de la tarea el mismo que tenga la función correspondiente en la librería.
	5	Para añadir parámetros, seleccionar <i>ParameterDefinition</i> en la paleta.
	6	Hacer clic en la tarea simple recién creada.
	7	Añadir como nombre y tipo del parámetro los mismos valores asignados al parámetro de la función correspondiente de la librería.
	8	Repetir la secuencia de pasos del 5 al 7 tantas veces como parámetros tenga el nuevo método de la librería.
	9	Guardar el fichero.
<b>Postcondición</b>	El catálogo se actualiza incluyendo una nueva tarea simple.	
<b>Comentarios</b>	Si el método que se ha añadido a la librería no tiene parámetros, la secuencia de pasos del 5 al 8 debe omitirse.	

Tabla 3.5: Descripción de caso de uso 03

### 3.1. ANÁLISIS

---

<b>CU Nº05</b>	<b>Creación de flujos de actividades a partir de las tareas del catálogo y configuración de sus parámetros</b>	
<b>Actor</b>	Terapeuta	
<b>Precondición</b>	Tener en el catálogo todas las tareas necesarias para diseñar el flujo deseado.	
<b>Secuencia Normal</b>		
Paso	Acción	
1	Seleccionar la carpeta donde se va a crear el flujo de actividades.	
2	Crear un nuevo <i>Workflow Diagram</i> con el nombre deseado.	
3	Asignar propiedades al flujo de actividades (Name).	
4	Cargar el catálogo en la ventana de edición.	
5	Añadir objetos de la paleta a la ventana de edición (Initial, Final, Activity, etc.).	
6	Cuando sea necesario, configurar las propiedades de los objetos añadidos en el paso anterior.	
7	Vincular cada tarea del flujo de actividades a una de las tareas del catálogo rellenando su propiedad <i>Task Definition</i> .	
8	Repetir la secuencia de pasos del 5 al 6 hasta añadir todos los objetos deseados.	
9	Enlazar las tareas en el orden en el que se quiera que se ejecuten en Cozmo.	
10	Validar el flujo de actividades.	
11	Guardar el fichero.	
<b>Postcondición</b>	Se crea un nuevo flujo de actividades.	

Tabla 3.6: Descripción de caso de uso 05

### 3.1. ANÁLISIS

---

<b>CU Nº06</b>	<b>Creación de tareas complejas en el catálogo a partir de flujos de actividades preexistentes para su reutilización</b>	
<b>Actor</b>	Terapeuta	
<b>Precondición</b>	Contar con un flujo de actividades previamente creado.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Abrir el catálogo.
	2	Seleccionar la tarea <i>ComplexTask</i> en la paleta.
	3	Hacer clic en la ventana de edición para colocar la tarea.
	4	Cargar el flujo de actividades en la ventana de edición.
	5	Vincular la tarea compuesta a un flujo de actividades.
	6	Para añadir parámetros, seleccionar <i>ParameterDefinition</i> en la paleta.
	7	Hacer clic en la tarea compuesta recién creada.
	8	Vincular el parámetro, mediante la propiedad <i>Bound To</i> , a uno de los parámetros de su flujo de actividades.
	9	Repetir la secuencia de pasos del 6 al 8 tantas veces como parámetros de la tarea compuesta queremos que puedan configurarse.
	10	Guardar el fichero.
<b>Postcondición</b>	El catálogo se actualiza con una nueva tarea compuesta.	
<b>Comentarios</b>	Si no se quiere que la tarea compuesta añadida al catálogo tenga parámetros configurables, la secuencia de pasos del 5 al 9 se puede omitir.	

Tabla 3.7: Descripción de caso de uso 06

## 3.2. DISEÑO

### 3.2. Diseño

En este apartado se detalla el diseño, la arquitectura y los diferentes componentes del sistema propuesto.

#### 3.2.1. Arquitectura

En la Figura 3.2 se muestra la arquitectura del sistema. A continuación, se describen los distintos componentes que la integran y cómo estos interactúan entre sí.

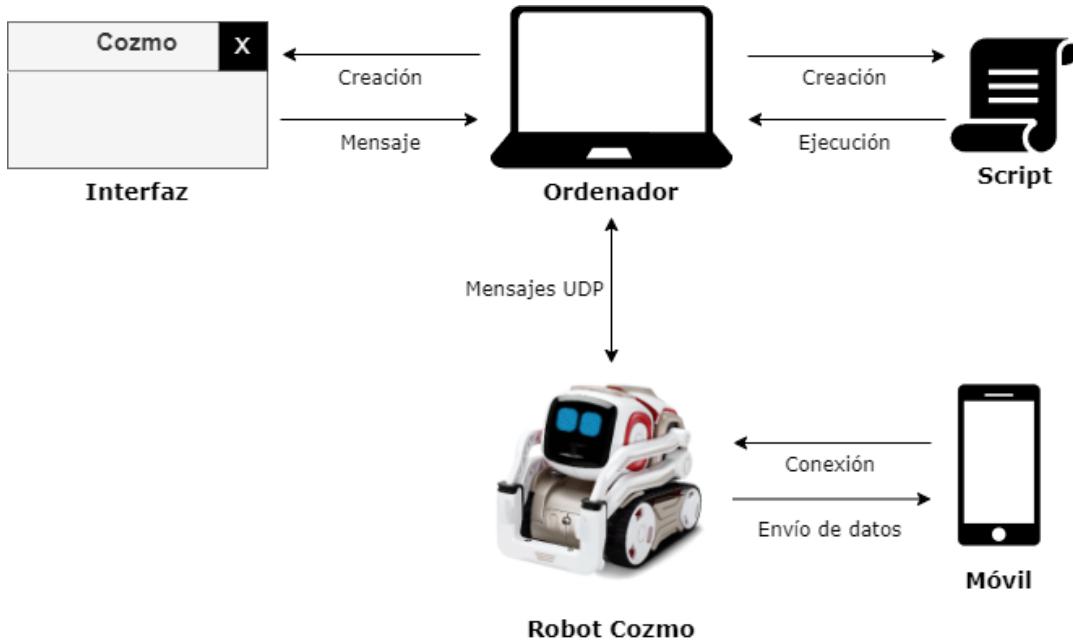


Figura 3.2: Arquitectura del sistema

En primer lugar, se encuentra el **ordenador**, encargado de interactuar con **Cozmo** mediante mensajes UDP (esta conexión se explica detalladamente al final del apartado). En el ordenador, es donde el usuario crea el flujo de actividades gracias al editor gráfico y, mediante este flujo, se genera el programa a ejecutar en Cozmo junto con un **Script** que contiene los comandos necesarios para cargar el programa en el robot. Si este programa contiene **interfaces** (por ejemplo, en condicionales) se mostrarán ven-

### 3.2. DISEÑO

---

tanazas emergentes en el ordenador y Cozmo esperará una respuesta hasta que el usuario interactúe con esa ventana y el ordenador le envíe la respuesta.

En segundo lugar, se requiere un **móvil** donde se debe tener instalada la aplicación de Cozmo, la cual será utilizada para realizar actualizaciones periódicas del robot.

#### Conexión ordenador-Cozmo

La conexión entre ordenador y robot se realiza mediante la librería “pyCozmo”, usada en el código implementado. La arquitectura de esta librería se compone de subprocesos múltiples y está organizada en tres capas donde cada capa superior depende de la inferior.

- Capa de aplicación
- Capa de cliente o SDK
- Capa de conexión de bajo nivel

La arquitectura simplificada se puede observar en la Figura 3.3 (para más destalle véase la documentación aportada en la pagina oficial [29]).

En primer lugar, la **capa de aplicación** es la encargada de implementar funciones externas de alto nivel como reacciones, comportamientos o procesamiento de imágenes, siendo denominada también como “Cerebro”. A su vez, los eventos de la capa inferior (capa de cliente, el ordenador) los convierte en reacciones y los gestiona mediante un *thread* (Hilo), desencadenando así comportamientos que envía a dicha capa mediante comandos.

En segundo lugar, la **capa de cliente (SDK)** otorga acceso a las funciones integradas de Cozmo. Estas funciones están vinculadas a la cámara, audio y animación del rostro. A su vez, permite enviar comandos y registrar el controlador de envío de paquetes y eventos. Por otro lado, dispone de un controlador de animación que sincroniza las funciones anteriormente comentadas mediante un *thread* independiente.

### 3.2. DISEÑO

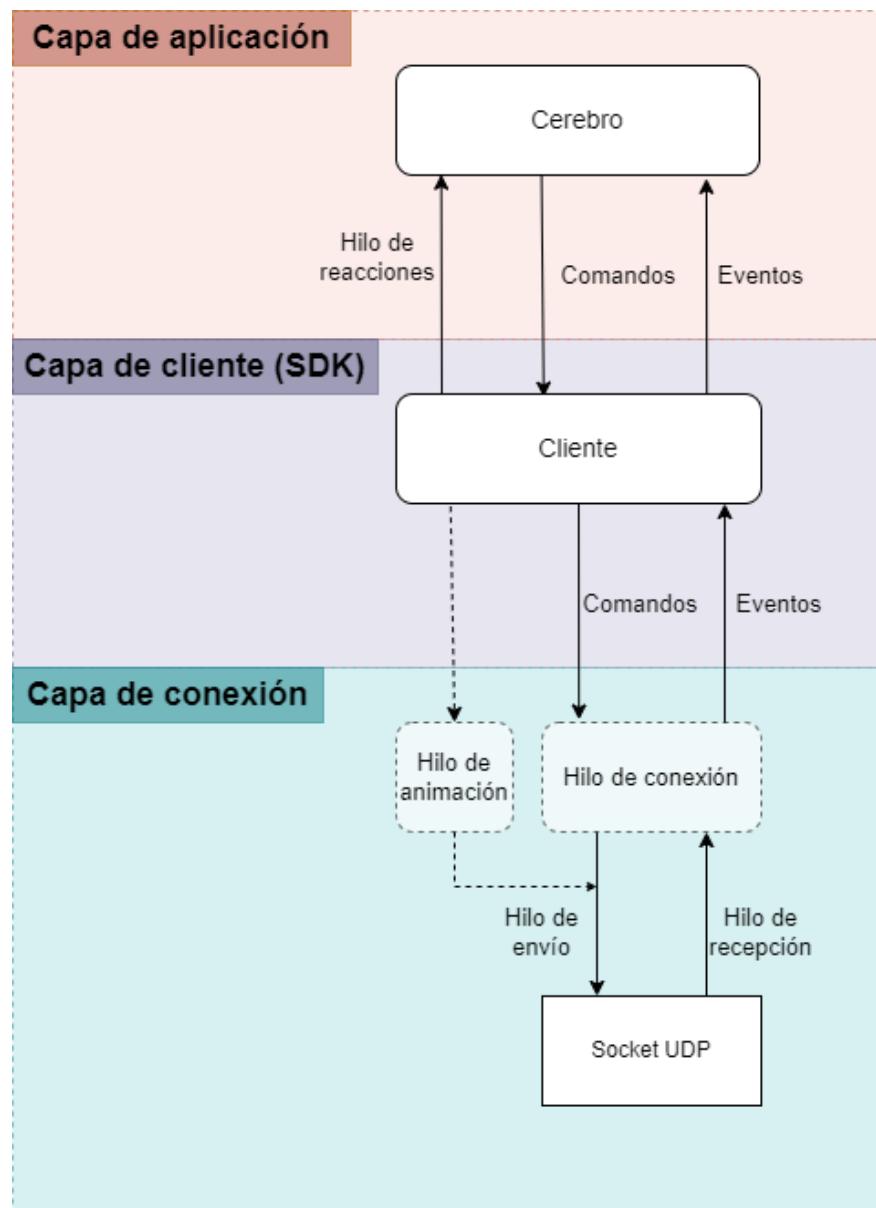


Figura 3.3: Arquitectura de la librería pyCozmo

### 3.2. DISEÑO

En tercer, y último lugar, la **capa de conexión** implementa el protocolo de comunicación de Cozmo. En esta se encuentran tres subprocessos: conexión, envío y recepción. El subprocesso de conexión lee un flujo de paquetes entrantes de la cola de mensajes entrantes y los envía a Cozmo. A su vez, envía paquetes *ping* para mantener la conexión. Los subprocessos restantes envían o reciben tramas a través del socket UDP.

En el protocolo de comunicación de Cozmo [39], el robot actúa como servidor (punto de acceso Wi-Fi) y asigna a los clientes Wi-Fi una dirección IP en el rango 172.31.1.0/24. Siempre usa la dirección IP 172.31.1.1 y esperará paquetes UDP en el puerto 5551. La aplicación iniciada en el ordenador actúa como cliente e inicia las conexiones. (Ver Figura 3.4).

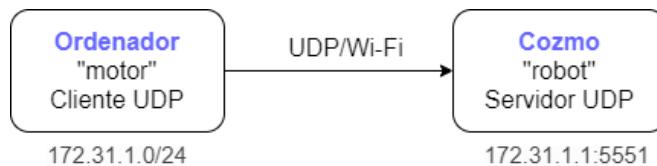


Figura 3.4: Protocolo de Conexión Cozmo

#### 3.2.2. Sintaxis abstracta del lenguaje de modelado de *PiLHaR*

La sintaxis abstracta de un lenguaje de modelado recoge los principales conceptos que se desea poder modelar, sus características esenciales y las relaciones existentes entre ellos. De este modo, se abstraen los detalles irrelevantes, permitiendo a los diseñadores centrarse en los aspectos verdaderamente importantes. Existen diversas formas de especificar la sintaxis abstracta de un lenguaje de modelado aunque, quizás la más extendida, sea utilizando un meta-modelo [40], similar a los diagramas de clases UML.

En este apartado se explica en detalle el meta-modelo desarrollado para el lenguaje de modelado en el que se basan las herramientas incluidas en *PiLHaR*, las modificaciones que ha ido sufriendo a lo largo del Proyecto y las restricciones adicionales con las que se ha enriquecido.

### 3.2. DISEÑO

---

#### Evolución del meta-monoel

En este apartado se va a describir el meta-monoel utilizado y su evolución según las diferentes necesidades que iban surgiendo en la implementación de tareas para Cozmo. Es importante mencionar que únicamente se va a mostrar gráficamente la última versión del meta-monoel para facilitar la lectura.

En **primer lugar**, el meta-monoel incluía dos elementos raíz (*root*):

- *Catalogue*: catalogo que dispondrá el usuario para la creación de los flujos de actividades. Incluye tareas simples (*SimpleTask*) predefinidas, que ejecuta el robot, y tareas complejas (*ComplexTask*), que están formada por una o más *SimpleTasks* o *ComplexTasks*, entre otras tareas; y
- *Workflow*: flujo de actividades que creará el usuario, el cual está compuesto de tareas (*Tasks*) y enlaces (*Links*). En esta primera versión, estas tareas se dividen en cuatro: *Activity*, *Initial*, *Final* y *Conditional*, siendo la característica más significativa que la tarea *Activity* está relacionada obligatoriamente con una definición de tarea (*TaskDefinition*), superclase de *SimpleTask* y *ComplexTask*.

La siguiente versión, la **segunda**, surge debido a que los tipos de tareas resultan ser demasiado ambiguos y se crean dos ampliaciones: (1) en *Catalogue*: a las *SimpleTasks* se le añade la opción de contar con varias definiciones de parámetros (*ParameterDefinitions*), cada una con un nombre (*name*) y tipo (*type*). A su vez, a la tarea *Activity*, se le añade la opción de contar con una serie de parámetros (*Parameter*), los cuáles, poseen nombre (*name*) y valor (*value*). Además, *Parameter* está enlazada con *ParameterDefinitions*, ya que una es la definición del parámetro y el otro, el valor que realmente tiene; y (2) por otro lado, se ha modificado la tarea (*Conditional*) para soportar las respuestas afirmativas y negativas (si y no).

En la **tercera versión**, se simplifica la tarea *Conditional* cambiando su nombre por *Loop* y añadiéndole los atributos de tipo y número de iteraciones (*type* y *numIterations*, respectivamente). Este *type* contempla la incorporación de *WHILE*, *DO WHILE*

### 3.2. DISEÑO

---

y *CONTER\_BASED* en el flujo de actividades. Seguidamente, a *ParameterDefinition* se le incluye un enlace a *Parameter*, llamado *boundTo*, donde considera que una *ComplexTask* ya definida, pueda cambiar su valor al incorporarla en otro flujo de actividades.

Por último, en la **cuarta versión**, se buscaba implementar dos editores gráficos para cada *root*. Desde Eclipse, la única posibilidad es creando cada elemento raíz (*root*) en diagramas independientes e insertando los enlaces entre ambos después de toda la generación de código de Eclipse. Por otra parte, se planteó la idea de realizar preguntas junto con sus respuestas en el flujo de actividades, por lo que se añadieron las tareas *Question* y *Answer* como subclases de *Final* e *Initial*, respectivamente. Esta incorporación conllevó varios problemas en el editor gráfico, por lo que finalmente se optó por implementar estas dos nuevas tareas directamente como subclases de *Task*. El metamodelo final se muestra en las Figuras 3.5 y 3.6, relacionados con el catálogo y el flujo de actividades, respectivamente. Además, en estas figuras, se muestran los enlaces que relacionan ambos meta-modelos (señaladas en azul). En concreto, las relaciones entre *Activity* y *SimplexTask*, *Parameter* y *ParameterDefinition*, *ComplexTask* y *Workflow* y *ParameterDefinition* y *Parameter*.

#### Definición del meta-modelo

Como ya se ha comentado anteriormente, el meta-modelo se compone de dos meta-modelos simples. Por un lado, se encuentra el flujo de actividades (*Workflow*) identificado con un nombre (*name*), compuesto por tareas (*task*), que pueden ser de varios tipos, y enlaces (*link*), que permiten unir varias tareas individualmente, definiendo así un flujo. Dentro de las tareas existen las siguientes subclases:

- **Inicial:** es la tarea que comienza el flujo de actividades. Es obligatoria y sólo debe existir una por flujo.
- **Final:** es la tarea que finaliza el flujo de actividades. Siempre será obligatoria y pueden existir varias.

### 3.2. DISEÑO

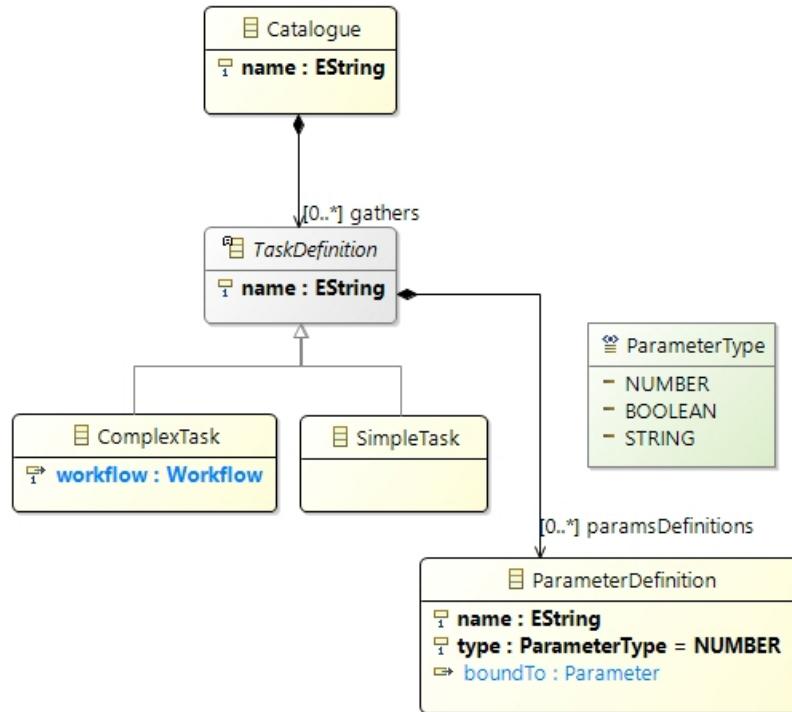


Figura 3.5: Meta-modelo Catalogue

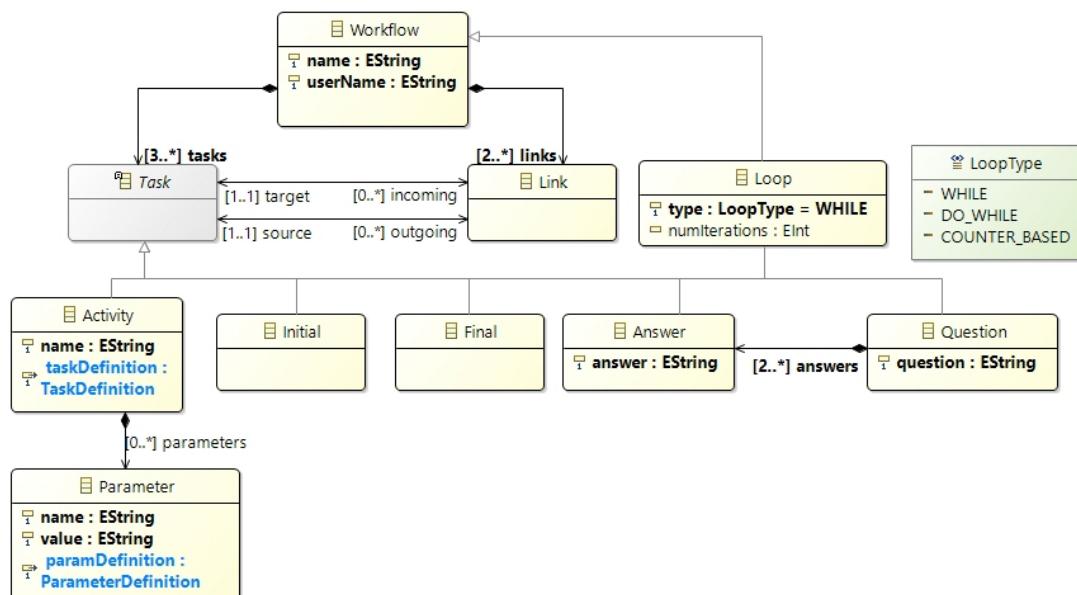


Figura 3.6: Meta-modelo Workflow

### 3.2. DISEÑO

---

- **Actividad:** se identifica con un nombre (*name*) y está vinculada a las tareas simples (*SimpleTasks*) del catálogo explicadas a continuación. Esta tarea se compone de parámetros, identificados con nombre (*name*) y valor (*value*) y a su vez, se vincula con la definición de parámetro (*paramDefinition*) también del catálogo, explicada más adelante.
- **Pregunta:** está compuesta como mínimo de dos respuestas (*Answer*). A su vez, contiene el texto que corresponde a la pregunta (*Question*). Es importante mencionar que la pregunta va a generar tantas bifurcaciones del flujo y tareas finales (*Final*) como respuestas contenga.
- **Respuesta:** se identifica con un parámetro textual que corresponde a la respuesta (*Answer*). Está contenida en una tarea de tipo Pregunta (*Question*) y de cada una de ellas, surge un flujo de tareas unidas individualmente.
- **Bucle:** se trata a su vez de un flujo de actividades. Consiste en repetir el flujo tantas veces como se le haya indicado, para ello se han establecido tres modos de iteración: realizar la acción si el usuario lo desea indicándolo a través del ordenador (*WHILE*); realizar la acción una vez y preguntar al usuario si la quiere volver a repetir (*DO WHILE*) o realizar la acción un número de veces indicado previamente (*CONTER BASED*).

Por otro lado, el catálogo (*Catalogue*), identificado por un nombre (*name*), puede tener definiciones de tareas simples o compuestas (*SimpleTasks o ComplexTasks*), también identificadas por un nombre (*name*).

Cada tarea simple (*SimpleTask*) está vinculada a uno de los métodos de la librería externa propia desarrollada para el sistema. Y, cada tarea compuesta (*ComplexTask*) se relaciona con un flujo de actividades preexistente. Ambas clases se componen de definiciones de parámetros (*ParameterDefinition*), identificados con un nombre (*name*) y tipo del valor (*type*). Como su nombre indica, la definición de un parámetro señala como tiene que ser este y por ende, no puede tener un valor específico. Este valor es el asignado a los parámetros de la tarea actividad (*Activity*), mencionado anteriormente.

### 3.2. DISEÑO

---

te. Además, el enlace *boundTo* de *ParameterDefinition*, si pertenece a *ComplexTask*, permite cambiar su valor al incorporarlo en otro flujo de actividades.

#### Descripción de algunos conceptos importantes

En este apartado se van a definir tres conceptos principales con ejemplos ilustrativos. Se comenzará por las tareas simples, ya que son el escalón más inferior del proyecto; seguidamente, se explica el flujo de actividades que se compone de tareas vinculadas a tareas simples; y para finalizar, las tareas complejas, que como ya se ha comentado, son flujos de actividades.

- **Tareas simples:** son tareas genéricas definidas en un catálogo. Como se puede observar en la Figura 3.7, para cada una de estas tareas simples (TS1 Y TS2) existe un método implementado en una librería propia desarrollada. Para su correcto funcionamiento, es necesario que el nombre de la tarea simple y del método sean iguales, actuando así como vínculo. A su vez, cada tarea simple que contenga parámetros requiere que los parámetros posean la misma nomenclatura que en el método (p1 y p2). Cada método, además, tiene un parámetro adicional que conecta con Cozmo, denominado *cli*. Estos métodos son genéricos y a través de los parámetros se consigue una modificación y personalización de su funcionamiento.
- **Flujo de actividades:** es una sucesión de tareas que como se puede observar en la Figura 3.8 se enlazan sucesivamente. Gracias a este flujo se genera un código donde se llama a sus respectivos métodos, que corresponden a las diferentes tareas de las que se compone. Este programa se carga en Cozmo para realizar, en el orden especificado, las diferentes tareas requeridas, con los parámetros definidos.
- **Tareas complejas:** están vinculadas a un flujo de actividades preexistente (ver Figura 3.9). Su funcionalidad principal es la reutilización de esta tarea en otro flujo de actividades realizado posteriormente, como se puede observar en la Fi-

### 3.2. DISEÑO

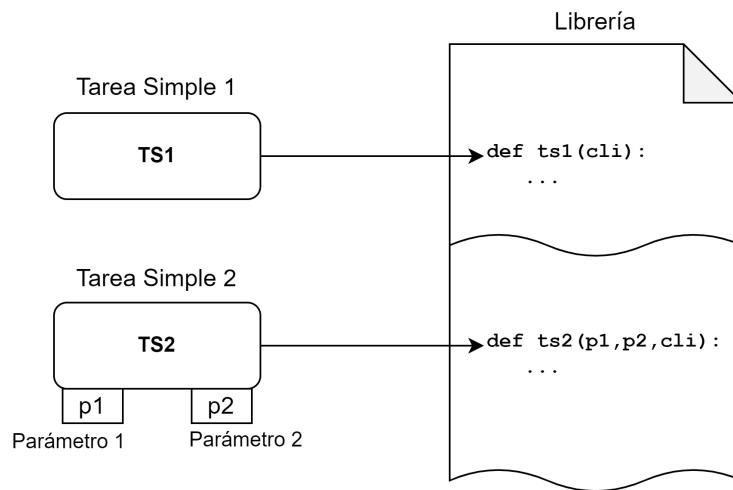


Figura 3.7: Ilustración abstracta de dos tareas simples y su código

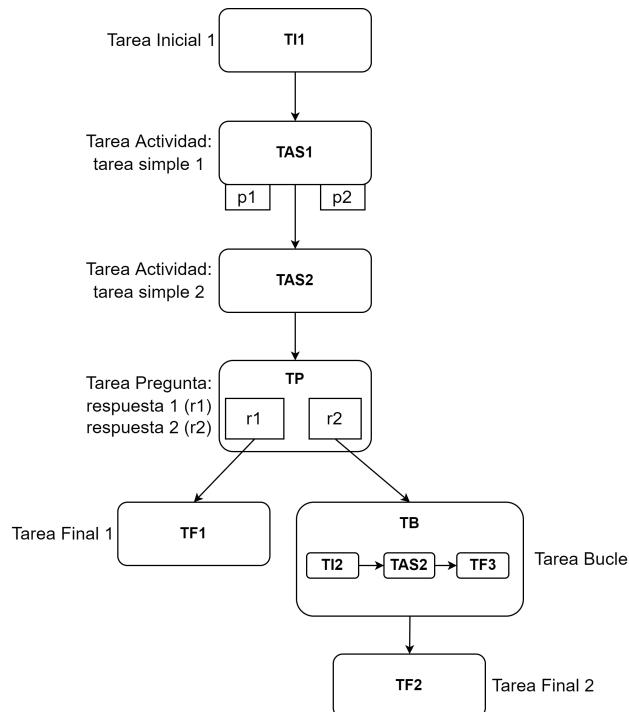


Figura 3.8: Ilustración abstracta de un ejemplo de flujo de actividades

### 3.2. DISEÑO

gura 3.10, la tarea TAC1. Es decir, el flujo de actividades 1 (FA1), a través de la vinculación a la tarea compleja 1 (TC1), puede ser utilizado en el FA2. Por lo que el comportamiento de FA2 será realizar las tareas en el siguiente orden:

1. Tarea inicial

2. Flujo de actividades 1

2.1 Tarea inicial

2.2 Tarea actividad simple 1

2.3 Tarea actividad simple 2

2.4 Tarea pregunta

2.4.1 Respuesta 1 → Tarea Final

2.4.2 Respuesta 2 → Tarea actividad simple 2 → Tarea Final

3. Tarea actividad simple 1

4. Tarea final

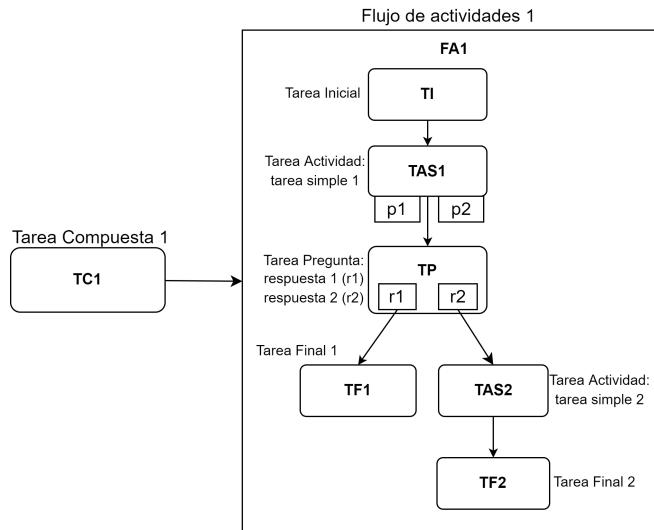


Figura 3.9: Ilustración de una tarea compleja vinculada a su flujo de actividades.

Otra funcionalidad que ofrece el sistema es que una tarea compleja puede tener vínculos a los parámetros de su flujo de actividades, como se muestra en la Figura 3.11).

### 3.2. DISEÑO

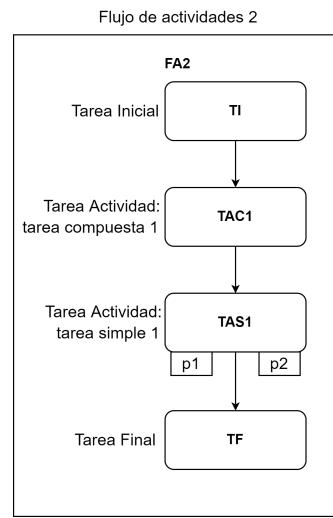


Figura 3.10: Ilustración abstracta de un ejemplo de flujo de actividades

Esta característica se utiliza para que la reutilización sea todavía más flexible y se puedan modificar los datos de ese parámetro para el nuevo flujo de actividades donde se inserte esta tarea compleja.

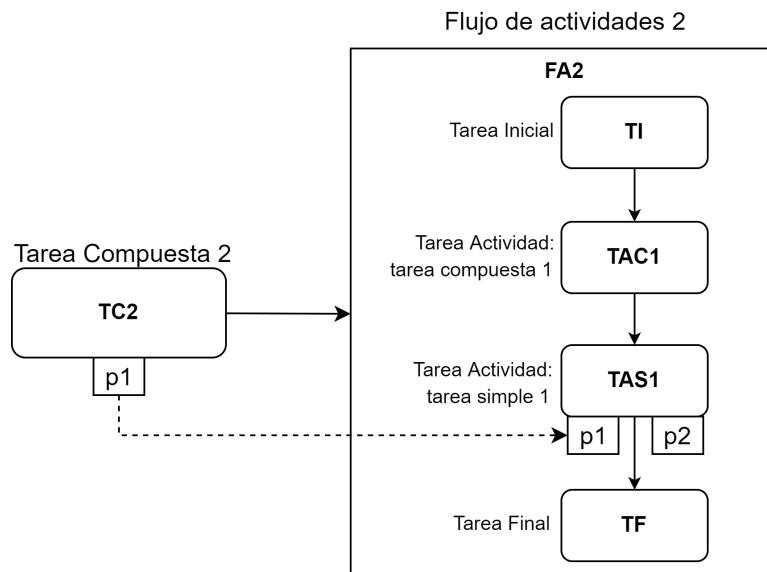


Figura 3.11: Ilustración abstracta de una tarea compleja y un parámetro

### 3.2. DISEÑO

#### Restricciones OCL

OCL (Object Constraint Language) es un lenguaje de definición de restricciones que permite completar la sintaxis abstracta de un lenguaje de modelado, descrita con un meta-modelo, añadiendo reglas que no es posible expresar directamente con EMF. En este apartado, se detallan las restricciones que se han añadido a los dos meta-modelos previamente descritos. Para cada restricción, se indicará su contexto (meta-clase en la que se ha implementado) y el código OCL correspondiente.

En primer lugar, la Tabla 3.8 recoge las restricciones OCL añadidas al meta-modelo del catálogo. Estas restricciones especifican, respectivamente, que el nombre del catalogo y el de las definiciones de los parámetro deben ser únicos.

Nº	Contexto	Restricción
1	Catalogue	self.gathers->isUnique(name);
2	TaskDefinition	self.paramsDefinitions->isUnique(name);

Tabla 3.8: Restricciones del meta-modelo del catálogo

En segundo lugar, la Tabla 3.9 recoge las restricciones añadidas al meta-modelo del flujo de actividades. La definición de cada una de las restricciones incluidas en esta tabla se corresponde con las listadas a continuación:

1. El flujo de actividades debe tener una única tarea inicial.
2. El flujo de actividades debe tener como mínimo una tarea final.
3. La tarea actividad debe tener un enlace de entrada.
4. La tarea actividad debe tener un enlace de salida.
5. La tarea inicial no debe tener ningún enlace de entrada pero si uno de salida.
6. La tarea final no debe tener ningún enlace de salida pero si uno de entrada.
7. La tarea bucle debe tener un enlace de entrada.
8. La tarea bucle debe tener un enlace de salida.

### 3.2. DISEÑO

---

9. En el caso de las tareas de bucle de tipo *COUNTER\_BASED* el atributo *numIterations* debe ser mayor que 1.
10. La tarea pregunta no debe tener ningún enlace de salida y uno de entrada.
11. La tarea respuesta debe estar contenida en una tarea pregunta.
12. La tarea respuesta no debe tener ningún enlace de entrada pero si uno de salida.

Nº	Contexto	Restricción
1	Workflow	self.tasks ->selectByType(Initial)->size()=1;
2	Workflow	self.tasks->selectByType(Final)->size()>=1;
3	Activity	self.incoming->size() = 1;
4	Activity	self.outgoing->size() = 1;
5	Initial	self.incoming->size() = 0 and self.outgoing->size() = 1;
6	Final	self.outgoing->size() = 0 and self.incoming->size() = 1;
7	Loop	self.incoming->size() = 1;
8	Loop	self.outgoing->size() = 1;
9	Loop	self.type = LoopType :: COUNTER_BASED implies numIterations>1;
10	Question	self.outgoing->size() = 0 and self.incoming->size() = 1;
11	Answer	self.oclContainer().oclIsTypeOf(Question);
12	Answer	self.incoming->size() = 0 and self.outgoing->size() = 1;

Tabla 3.9: Restricciones del meta-modelo del flujo de actividades

Por último, también ha sido necesario definir restricciones asociadas a las relaciones que unen los dos meta-modelos.

Por un lado, en la Tabla 3.10 se muestran las restricciones añadidas a las relaciones entre ambos meta-modelos contenidas en elementos del **catálogo**.

1. La definición de un parámetro de una tarea simple no debe tener enlace a un parámetro.
2. La definición de un parámetro de una tarea compleja debe tener enlace a un parámetro.
3. Comprueba que la tarea compleja que contiene el enlace *boundTo* sea igual que el workflow de la tarea compleja que contiene la definición del parámetro.

### 3.2. DISEÑO

---

Nº	Contexto	Restricción
1	ParameterDefinition	self.oclContainer().oclIsTypeOf(SimpleTask) implies self.boundTo->size()=0;
2	ParameterDefinition	self.oclContainer().oclIsTypeOf(ComplexTask) implies self.boundTo->size()=1;
3	ParameterDefinition	self.oclContainer().oclIsTypeOf(ComplexTask) implies self.boundTo.oclContainer() = self.oclContainer().oclAsType(ComplexTask).workflow;

Tabla 3.10: Restricciones OCL del meta-modelo del catálogo versión 2

Por otro lado, en la Tabla 3.11 se muestran las restricciones añadidas a las relaciones entre ambos meta-modelos contenidas en elementos del meta-modelo de **flujo de actividades**.

1. La tarea simple que contiene la definición del parámetro debe ser igual que la definición de tarea de la actividad que contiene el parámetro.
2. Comprueba que si el tipo del parámetro es un número, el valor debe ser un *Integer* (valor entero) y no decimal.
3. Comprueba que si el tipo del parámetro es un booleano, el valor en mayúsculas tiene cuatro posibles opciones: TRUE, FALSE, VERDADERO o FALSO.

Nº	Contexto	Restricción
1	Parameter	self.paramDefinition.oclContainer() = self.oclContainer().oclAsType(Activity).taskDefinition;
2	Parameter	self.paramDefinition.type=catalogue::ParameterType::NUMBER implies not self.value.toInteger().oclIsInvalid();
3	Parameter	self.paramDefinition.type=catalogue::ParameterType::BOOLEAN implies self.value.toUpperCase()="TRUE" or self.value.toUpperCase()="FALSE" or self.value.toUpperCase()="VERDADERO" or self.value.toUpperCase()="FALSO";

Tabla 3.11: Restricciones del meta-modelo del flujo de actividades versión 2

## 3.2. DISEÑO

---

### 3.2.3. Sintaxis concreta

La sintaxis concreta de un lenguaje de modelado define cómo se representan (gráfica o textualmente) cada uno de los distintos conceptos incluidos en la sintaxis abstracta. Una sintaxis abstracta puede estar asociada con varias sintaxis concretas. [41].

#### Editores gráficos

Para el desarrollo de este Proyecto sólo se ha implementado una sintaxis concreta gráfica, ya que la idea era ofrecer a los usuarios de la herramienta (no expertos en programación) un entorno de modelado sencillo e intuitivo de utilizar y se consideró que una representación gráfica se ajustaba mejor este requisito que una textual. Los dos editores gráficos desarrollados se basan en EuGENia [17] que, como ya se ha comentado anteriormente, permite anotar directamente en los meta-modelos EMF, la representación gráfica que se quiere asociar a cada uno de sus elementos.

El resultado de anotar los dos meta-modelos con EuGENia está disponible en los ficheros “catalogue.emf” y “workflow.emf”. Su contenido puede verse en el Anexo A.2. Cada fichero define la sintaxis gráfica concreta asociada a los elementos del meta-modelo correspondiente, esto es, el conjunto de figuras e iconos que representarán los elementos incluidos en cada meta-modelo. A continuación, se muestra una leyenda de los iconos utilizados en cada editor (ver tablas 3.12 y 3.13). Estos iconos son los que aparecen en la ventana de edición y en la paleta de Eclipse, a la hora de modelar.

A partir de cada uno de los meta-modelos anotados, EuGENia genera automáticamente los modelos GMF asociados *.gmfgraph*, *.gmftool* y *.gmfmap* (ver Figura 3.12) y, a partir de ellos, el código de los dos editores gráficos de modelos (ver Figura 3.13).

En este Proyecto se han editado varios de los métodos auto-generados por EuGENia/GMF para definir un comportamiento diferente al por defecto. Estos métodos modificados manualmente se han anotado con la etiqueta “@generatedNOT”. Los ficheros modificados asociados al meta-modelo *Catalogue* se encuentran en la ruta “*Robo-*

### 3.2. DISEÑO

---

Editor Catálogo		
Contexto	Ventana de edición	Paleta
SimpleTask		
ComplexTask		
ParameterDefinition		

Tabla 3.12: Leyenda del editor gráfico de catálogo

Editor Flujo de actividades		
Contexto	Ventana de edición	Paleta
Inicial		
Final		
Activity		
Question		
Answer		
Loop		
Parameter		No existe
Link		

Tabla 3.13: Leyenda del editor gráfico del flujo de actividades

### 3.2. DISEÑO

*TaskFlow/src/catalogue/impl*” y son los siguientes: “ParameterDefinitionImpl.java” y “ComplexTaskImpl.java”. Los ficheros modificados asociados al meta-modelo *Workflow* están en la ruta “*RoboTaskFlow/src/workflow/impl*” y son los siguientes: “ParameterImpl.java” y “ActivityImpl.java”. El código con las correspondientes modificaciones está disponible en el Anexo A.1.

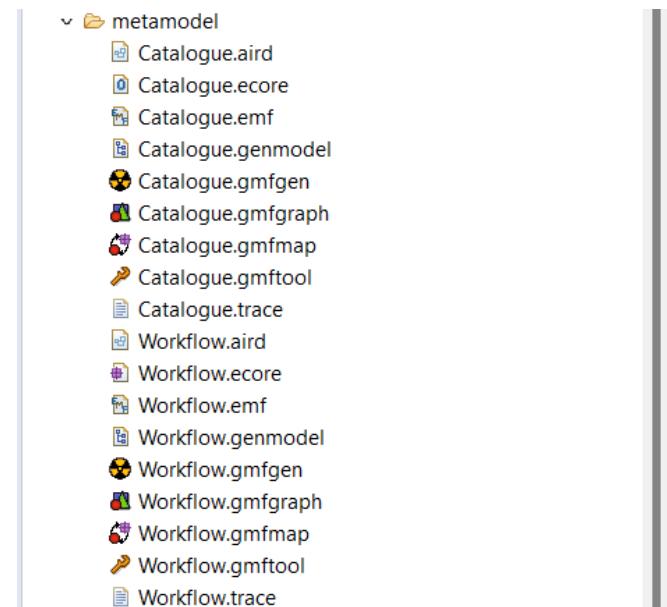


Figura 3.12: Carpeta metamodelos del proyecto principal



Figura 3.13: Carpeta código fuente del proyecto principal

# Capítulo 4

## Implementación y desarrollo

En este capítulo se describen las librerías utilizadas como base, las ampliaciones desarrolladas sobre la librería de Cozmo, el generador del código asociado a los modelos creados, el despliegue del robot Cozmo con dicho código generado junto con la interfaz desarrollada para la interacción entre los terapeutas y el sistema.

### 4.1. Librerías utilizadas

Inicialmente, todas las tareas que se van a desarrollar en este proyecto se basan en una librería, llamada **pyCozmo**[29], que contiene las funciones básicas para conectar y controlar a Cozmo. Además, son necesarias otras librerías puntuales que se usan solo en ciertos métodos y se listan a continuación:

- **Tiempo (time)** [42]: empleada en varios métodos de la librería para gestionar la espera de Cozmo para realizar sus acciones.
- **Voz (pyttsx3)** [43]: empleada en el método “Hablar” para proporcionar el audio que reproduce Cozmo.
- **Procesamiento de imágenes**: se han utilizado tres librerías diferentes: la librería de *PIL*; *numpy*; y *os*.

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

- *PIL* [44] se usa para guardar en el ordenador imágenes que Cozmo pueda capturar, para mostrar estas imágenes en el ordenador y para que Cozmo pueda modificar sus ojos según diferentes criterios.
  - *numpy* [45] es utilizado para procesamiento de imágenes, tratándolas como vectores, facilitando así diferentes tipos de operaciones sobre las mismas.
  - *os* [46] es utilizada para acceder a la ruta donde se está ejecutando el código para guardar la imagen en esa misma carpeta, interactuando con el sistema operativo del ordenador.
- **Interfaz (*tkinter*)** [47]: usada en la creación de interfaces para la interacción del usuario con Cozmo a través del ordenador.

## 4.2. Ampliación de la librería

El proceso de crear tareas simple es muy meticoloso ya que es obligatorio que siga un formato muy concreto. Inicialmente, se crea un nuevo método en el fichero “MyLibrary.py” ubicado en la carpeta “[NombreDelProyecto]/TareasComplejas/TareasGenericas”. Este, puede incluir parámetros o no, por lo que se van a explicar en diferentes subapartados.

### 4.2.1. Métodos sin parámetros

El formato de un método sin parámetros se puede observar en la Figura 4.1. Será necesario crear un método de este tipo cuando el robot no tenga valores que modificar en el código.

```
262  
263     def nuevoMetodoSinParam(cli):  
264  
265  
266
```

Figura 4.1: Creación de nuevo método sin parámetros

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

---

Seguidamente, el nombre del método será el mismo que se añadirá como tarea simple en el catálogo. Por ejemplo, si el nombre del método es “hacerFoto” el nombre que debe recibir la nueva tarea simple será “HacerFoto” (ver Figura 4.2). Obligatoriamente, el método en la librería se escribe con la primera letra en minúsculas, en cambio, en la tarea simple esto no es un requisito obligatorio.



Figura 4.2: Ejemplo real de la nomenclatura

Finalmente, se escribe el código correspondiente al método diseñado por el desarrollador. De esta forma, dicho método ya estaría preparado para poder utilizarse en el editor del catálogo.

### 4.2.2. Métodos con parámetros

El formato de este método se puede observar en la Figura 4.3. Será necesario crear un método de este tipo cuando el usuario desee modificar o personalizar el funcionamiento del mismo. La lista insertada antes del método, entre tres comillas (““”), es documentación interna que se debe añadir al manual de usuario (Ver Anexo A.5).

```

269 ...
270 ...
271 1. Opcion 1
272 2. Opcion 2
273 ...
274 def nuevoMetodoConParam(param, cli):
275
276
277

```

Figura 4.3: Creación de nuevo método con parámetros

A continuación, hay que hacer los mismos pasos que en el apartado anterior y, además, en la tarea simple creada, hay que añadir el mismo número de definiciones de parámetros que en el método (ver Figura 4.4). Estas definiciones de parámetros tienen dos

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

parámetros: el nombre, que es el mismo que tiene el parámetro en el método, y el tipo, que puede ser cadena, número o booleano (ver Figura 4.5).

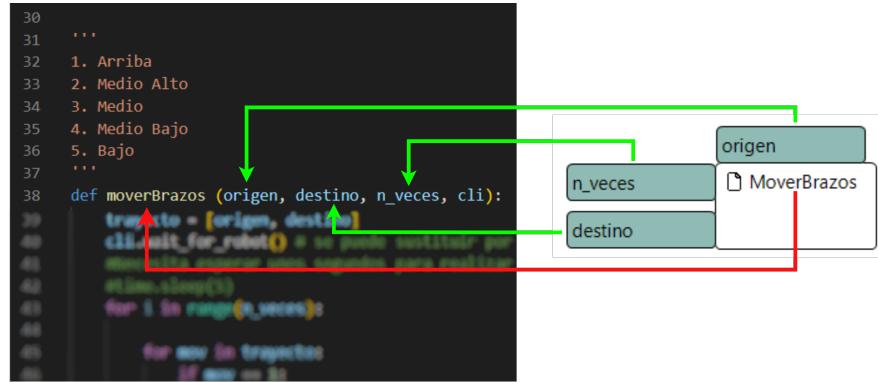


Figura 4.4: Ejemplo real de la creación de parámetros

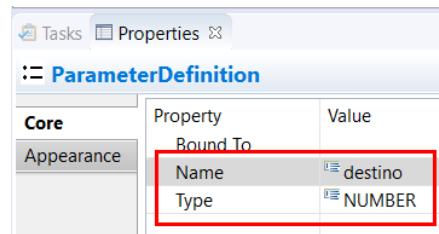


Figura 4.5: Parámetros del parámetro destino

En el ejemplo mostrado en la Figura 4.4, la documentación interna indica que los parámetros `origen` y `destino` pueden obtener los valores del 1 al 5 y Cozmo las representará como se indica en la lista. Es decir, si `origen = 5` y `destino = 1`, Cozmo moverá los brazos desde la posición más baja hasta la posición más alta el número de veces indicado en el parámetro `n_veces`.

### 4.2.3. Tareas desarrolladas

En este apartado se muestran las tareas que va a realizar Cozmo en diferentes tablas. Cada una contiene la implementación del método, la notación gráfica, el código que se auto-genera y una breve descripción. Estas tareas simples son: (1) Hablar [Tabla 4.1], (2) MoverBrazos [Tabla 4.2], (3) MoverRuedas [Tabla 4.3], (4) MoverCa-

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

beza [Tabla 4.4], (5) MostrarSentimientos [Tabla 4.5], (6) HacerFoto [Tabla 4.6], (7) MostrarImagen [Tabla 4.7] y (8) CambiarColorLuz [Tabla 4.8].

Tarea simple: <b>Hablar</b>	
Implementación	
	<pre> 1  import pyttsx3 2  import pycozmo 3 4  def hablar(cli, text): 5 6      # Initialize the Pyttsx3 engine 7      audio = pyttsx3.init() 8      #Set Rate. Defaults to 200 word per minute 9      audio.setProperty('rate', 150) 10     # We can use file extension as mp3 and wav 11     audio.save_to_file(text, 'audio.wav') 12     # Wait until above command is not finished 13     audio.runAndWait() 14 15     # Set volume to maximum. 16     cli.set_volume(65535) 17 18     # A 22 kHz, 16-bit, mono file is required. 19     cli.play_audio("audio.wav") 20 21     cli.wait_for(pycozmo.event.EvtAudioCompleted) --</pre>
Notación gráfica	Código generado
	<code>mycozmo.hablar(cli, text_Saludar)</code>
Descripción	
<p>La tarea simple Hablar tiene como parámetro de entrada una cadena de caracteres (<i>text</i>). La implementación de esta tarea transforma el texto de entrada en un fichero .wav (mediante un motor de conversión de texto a audio) y hace que el robot lo reproduzca.</p>	

Tabla 4.1: Descripción de la tarea Hablar

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

Tarea simple: MoverBrazos	
Implementación	
<pre> 23 import time 24 import pycozmo 25 26 ... 27 1. Arriba 28 2. Medio Alto 29 3. Medio 30 4. Medio Bajo 31 5. Bajo 32 ... 33 def moverBrazos (cli, origen, destino, n_veces): 34     trayecto = [origen, destino] 35     cli.wait_for_robot() 36 37     for i in range(n_veces): 38 39         for mov in trayecto: 40             if mov == 1: 41                 cli.set_lift_height(pycozmo.MAX_LIFT_HEIGHT.mm) #92mm 42                 time.sleep(0.5) 43             elif mov == 2: 44                 cli.set_lift_height(75.00) #75mm 45                 time.sleep(0.5) 46             elif mov == 3: 47                 cli.set_lift_height(pycozmo.MIN_LIFT_HEIGHT.mm*2) #64mm 48                 time.sleep(0.5) 49             elif mov == 4: 50                 cli.set_lift_height(pycozmo.LIFT_PIVOT_HEIGHT.mm) #45mm 51                 time.sleep(0.5) 52             elif mov == 5: 53                 cli.set_lift_height(pycozmo.MIN_LIFT_HEIGHT.mm) #32mm 54                 time.sleep(0.5) 55     time.sleep(1)</pre>	
Notación gráfica	
<pre> graph LR     origen[origen] --&gt; MoverBrazos[MoverBrazos]     destino[destino] --&gt; MoverBrazos     n_veces[n_veces] --&gt; MoverBrazos </pre>	
Código generado	
<pre>mycozmo.moverBrazos(cli, origen_SubirBrazos, destino_SubirBrazos, n_veces_SubirBrazos)</pre>	
Descripción	
<p>La tarea MoverBrazos toma como parámetros de entrada tres números enteros (<i>origen</i>, <i>destino</i> y <i>n_veces</i>). La funcionalidad del método consiste en que Cozmo mueve sus brazos de una posición (<i>origen</i>) a otra (<i>destino</i>) el número de veces (<i>n_veces</i>) indicado.</p>	

Tabla 4.2: Descripción de la tarea MoverBrazos

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

Tarea simple: MoverRuedas	
Implementación	
<pre> 59 import pycozmo 60 61 """ 62 1. Rápido 63 2. Medio 64 3. Lento 65 4. Marcha atrás 66 """ 67 def moverRuedas(cli, n_velocidad, tiempo, girar): 68     # Establecer velocidad 69     if n_velocidad == 1: 70         velocidad = 100.00 71 72     elif n_velocidad == 2: 73         velocidad = 50.00 74 75     elif n_velocidad == 3: 76         velocidad = 30.00 77 78     elif n_velocidad == 4: 79         velocidad = -50.00 80 81     #Establecer si ambas ruedas van en la misma dirección o no 82     if girar: 83         Rvelocidad = 0 - velocidad 84     else: 85         Rvelocidad = velocidad 86 87     cli.drive_wheels(lwheel_speed=velocidad, rwheel_speed=Rvelocidad, duration=tiempo) </pre>	
Notación gráfica	
<pre> graph TD     tiempo[tiempo] --&gt; MoverRuedas[MoverRuedas]     nVelocidad[n Velocidad] --&gt; MoverRuedas     girar[Girar] --&gt; MoverRuedas </pre>	
Código generado	
<pre>mycozmo.moverRuedas(cli, n_velocidad_Andar, tiempo_Andar, girar_Andar)</pre>	
Descripción	
<p>La tarea MoverRuedas toma como parámetros de entrada dos números enteros (<i>n_velocidad</i> y <i>tiempo</i>) y un booleano (<i>girar</i>). La implementación de la tarea consiste en que Cozmo mueve sus ruedas a la velocidad que determine “<i>n_velocidad</i>”, la cantidad de segundos indicada por “<i>tiempo</i>” y en linea recta si “<i>girar = Falso</i>” o hacia un lado si “<i>girar = Verdadero</i>”. El grado del giro va relacionado con la cantidad de tiempo indicada y la velocidad.</p>	

Tabla 4.3: Descripción de la tarea MoverRuedas

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

Tarea simple: MoverCabeza	
Implementación	
90	import time
91	import pycozmo
92	...
93	1. Arriba
94	2. Medio
95	3. Bajo
96	...
97	
98	def moverCabeza (cli, origen, destino, n_veces):
99	trayecto = [origen, destino]
100	cli.wait_for_robot()
101	
102	for i in range(n_veces):
103	
104	for mov in trayecto:
105	if mov == 1:
106	cli.set_head_angle(pycozmo.MAX_HEAD_ANGLE.radians) #44,5º
107	time.sleep(1)
108	elif mov == 2:
109	cli.set_head_angle(pycozmo.MAX_HEAD_ANGLE.radians/2) #22º
110	time.sleep(1)
111	elif mov == 3:
112	cli.set_head_angle(pycozmo.MIN_HEAD_ANGLE.radians) #-25º
113	time.sleep(1)
Notación gráfica	
	<pre> graph TD     M[MoverCabeza]     M --- destino[destino]     M --- origen[origen]     M --- n_veces[n_veces]   </pre>
Código generado	
<code>mycozmo.moverCabeza(cli, origen_SubirCabeza, destino_SubirCabeza, n_veces_SubirCabeza)</code>	
Descripción	
<p>La tarea MoverCabeza toma como parámetros de entrada tres números enteros (<i>origen</i>, <i>destino</i> y <i>n_veces</i>). La funcionalidad del método consiste en que Cozmo mueve su cabeza de una posición (<i>origen</i>) a otra (<i>destino</i>) el número de veces (<i>n_veces</i>) indicado.</p>	

Tabla 4.4: Descripción de la tarea MoverCabeza

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

Tarea simple: MostrarSentimiento	
Implementación	
144	import pycozmo
145	from PIL import Image
146	import numpy as np
147	def mostrarSentimiento(cli, n_sentimiento):
148	
149	cli.enable_procedural_face(False)
150	
151	# List of face expressions.
152	expressions = [
153	pycozmo.expressions.Neutral(),
154	pycozmo.expressions.Anger(),
155	pycozmo.expressions.Sadness(),
156	pycozmo.expressions.Happiness(),
157	pycozmo.expressions.Surprise(),
158	pycozmo.expressions.Disgust(),
159	pycozmo.expressions.Fear(),
160	pycozmo.expressions.Pleading(),
161	pycozmo.expressions.Vulnerability(),
162	pycozmo.expressions.Despair(),
163	pycozmo.expressions.Guilt(),
164	pycozmo.expressions.Disappointment(),
165	pycozmo.expressions.Embarrassment(),
166	pycozmo.expressions.Horror(),
167	pycozmo.expressions.Skepticism(),
168	pycozmo.expressions.Annoyance(),
169	pycozmo.expressions.Fury(),
170	pycozmo.expressions.Suspicion(),
171	pycozmo.expressions.Rejection(),
172	pycozmo.expressions.Boredom(),
173	pycozmo.expressions.Tiredness(),
174	pycozmo.expressions.Asleep(),
175	pycozmo.expressions.Confusion(),
176	pycozmo.expressions.Amazement(),
177	pycozmo.expressions.Excitement(),
178	]
179	
180	# Base face expression.
181	base_face = pycozmo.expressions.Neutral()
182	
183	rate = pycozmo.robot.FRAME_RATE
184	timer = pycozmo.util.FPSTimer(rate)

Sigue en la página siguiente.

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

### Implementación

```

186     # Transition from base face to expression and back.
187     for from_face, to_face in ((base_face, expressions[n_sentimiento]),
188         |   (expressions[n_sentimiento], base_face)): # Segunda parte del bucle for
189
190         if to_face != base_face:
191             print(to_face.__class__.__name__)
192
193     # Generate transition frames.
194     face_generator=pycozmo.procedural_face.interpolate(from_face,to_face,rate//3)
195     for face in face_generator:
196         im = face.render() # Render face image.
197         # The Cozmo protocol expects a 128x32 image, so take only the even lines.
198         np_im = np.array(im)
199         np_im2 = np_im[::2]
200         im2 = Image.fromarray(np_im2)
201         cli.display_image(im2) # Display face image.
202         timer.sleep() # Maintain frame rate.
203
204         for i in range(rate): # Pause for 1s.
205             timer.sleep()
206
207     cli.enable_procedural_face(True)

```

### Notación gráfica



### Código generado

```
mycozmo.mostrarSentimiento(cli, n_sentimiento_Contento)
```

### Descripción

La tarea `MostrarSentimiento` toma como parámetro de entrada un número entero (`n_sentimiento`). La implementación de la actividad consiste en que Cozmo muestra mediante su pantalla la expresión indicada en el parámetro.

Tabla 4.5: Descripción de la tarea `MostrarSentimiento`

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

Tarea simple: HacerFoto	
Implementación	
	<pre> 217  from PIL import Image 218 219  def on_camera_image(cli, image): 220      image.save("TareasGenericas/Recursos/camera.png", "PNG") 221 222  import time 223  import pycozmo 224 225  def hacerFoto(cli): 226      cli.enable_camera(enable=True, color=True) 227 228      # Wait for image to stabilize. 229      time.sleep(2.0) 230 231      cli.add_handler(pycozmo.event.EvtNewRawCameraImage, on_camera_image, one_shot=True) 232 233      # Wait for image to be captured. 234      time.sleep(1) 235      hablar("Ya he hecho la foto", cli) </pre>
Notación gráfica	Código generado
	<code>mycozmo.hacerFoto(cli)</code>
Descripción	
<p>La implementación de la tarea HacerFoto consiste en que Cozmo mediante su cámara realiza una foto y la almacena en la carpeta del proyecto “Recurso”.</p>	

Tabla 4.6: Descripción de la tarea HacerFoto

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

---

Tarea simple: MostrarImagen	
Implementación	
	<pre> 237 import os 238 import pycozmo 239 def mostrarImagen (cli, imagen): 240 241     cli.enable_procedural_face(False) 242     # Load image 243     im = Image.open(os.path.join(os.path.dirname(__file__), "Recursos", imagen)) 244 245     # Resize from 320x240 to 68x17. Larger image sometime are too big for the ro 246     im = im.resize((128, 32)) # Convert to binary image. 247 248     im = im.convert('1') 249 250     cli.display_image(im, 3) 251     cli.enable_procedural_face(True) </pre>
Notación gráfica	Código generado
	mycozmo.mostrarImagen(cli, imagen_Helloword)
Descripción	
<p>La tarea MostrarImagen toma como parámetro de entrada una cadena de caracteres (<i>imagen</i>). La implementación de la actividad consiste en que Cozmo muestra en su pantalla la imagen indicada que se debe encontrar en la carpeta “Recurso”.</p>	

Tabla 4.7: Descripción de la tarea MostrarImagen

## 4.2. AMPLIACIÓN DE LA LIBRERÍA

Tarea simple: CambiarColorLuz	
Implementación	
	<pre> 253 import time 254 import pycozmo 255 256 ... 257 0. Red 258 1. Green 259 2. Blue 260 3. White 261 4. Off 262 ... 263 def cambiarColorLuz(cli, n_luz): 264     lights = [ 265         pycozmo.lights.red_light, 266         pycozmo.lights.green_light, 267         pycozmo.lights.blue_light, 268         pycozmo.lights.white_light, 269         pycozmo.lights.off_light, 270     ] 271     cli.set_all_backpack_lights(lights[n_luz]) 272     time.sleep(0.5) </pre>
Notación gráfica	Código generado
	<code>mycozmo.cambiarColorLuz(cli, n_luz_LuzAzul)</code>
Descripción	
<p>La tarea CambiarColorLuz toma como parámetros de entrada un número entero (<i>n_luz</i>). La implementación de la actividad consiste en que Cozmo ilumina el LED de su espalda con el color indicado en el parámetro.</p>	

Tabla 4.8: Descripción de la tarea CambiarColorLuz

### 4.3. GENERADOR DE CÓDIGO

## 4.3. Generador de código

La generación del código, independientemente del lenguaje de salida, asociado al flujo de cada una de las tareas que va a realizar el robot Cozmo se lleva a cabo con Acceleo [18], de código abierto, y se encuentra integrado en el IDE de Eclipse. Cabe destacar que Acceleo ofrece grandes ventajas sobre la generación de código Modelo a Texto (*M2T, Model to Text*) como son la alta capacidad de personalización, interoperabilidad y facilidad de uso, entre otros.

El código creado en Acceleo está dividido en ficheros y estos a su vez en *templates* o plantillas. Inicialmente, se encuentra el fichero “*generate.mtl*” que contiene el *template* “*generateFiles*”, que es el encargado de crear dos ficheros distintos.

El primer fichero se llama “*EjecutarCozmo.bat*” (ver Figura 4.6) y se encarga de enviar el segundo fichero al robot, donde se encuentra el código ejecutable; y, el segundo es denominado con el mismo nombre del flujo de actividades correspondiente y contiene el código que Cozmo ejecutará, siguiendo el esquema que se muestra en la Figura 4.7.

```
EjecutarCozmo.bat
@echo off
echo Mandando [NombreFlujoAct] a cozmo
python [NombreFlujoAct].py
```

Figura 4.6: Contenido del fichero “*EjecutarCozmo.bat*”

La composición de este esquema se va a describir a continuación. Está formado por un *template* principal “*generateProgram*” que se puede encontrar en el fichero “*generateProgram.mtl*”. Además, este fichero contiene otros *templates* que son llamados por el *template* principal. Cada elemento del esquema se especifica en la siguiente lista:

### 4.3. GENERADOR DE CÓDIGO

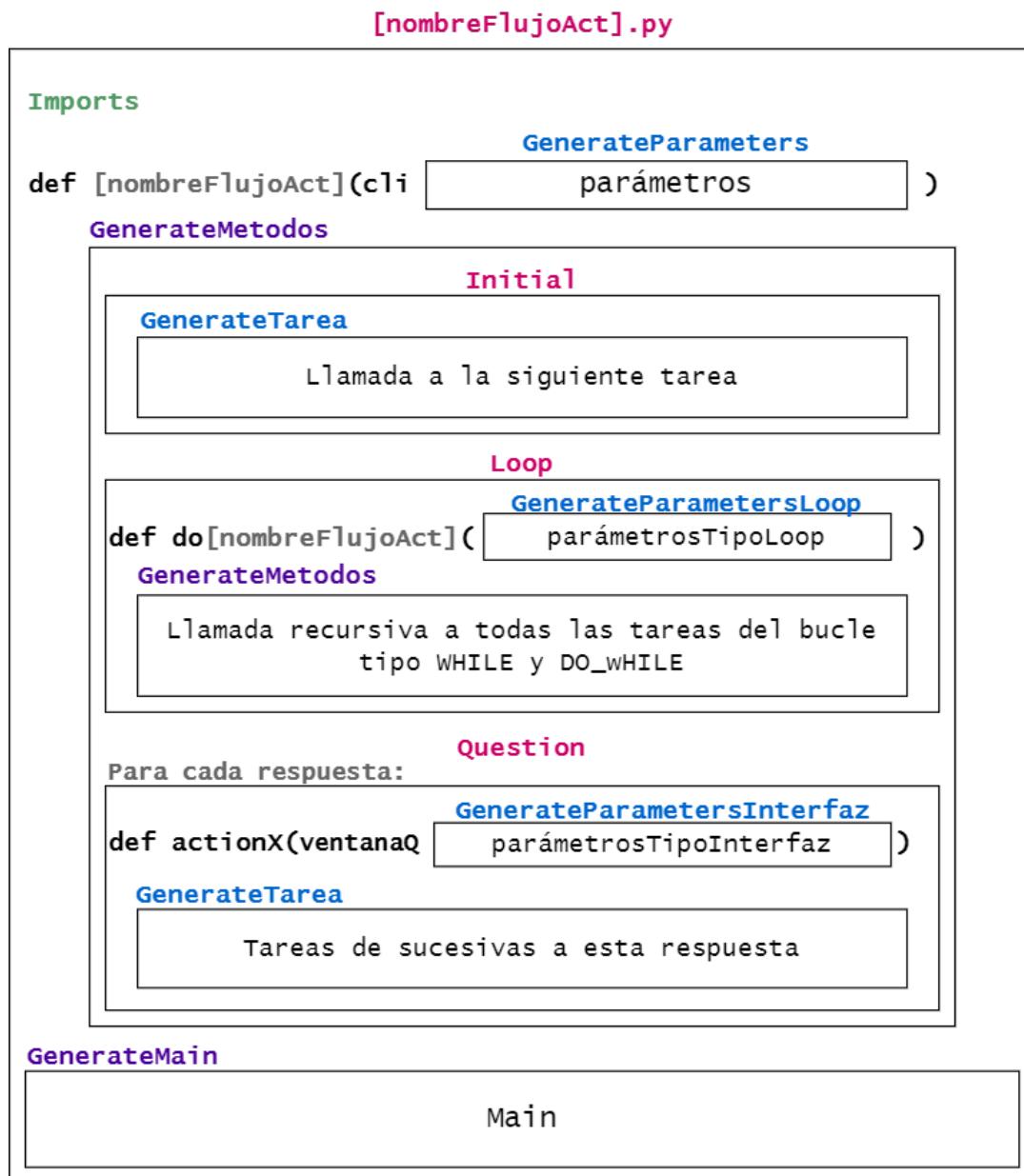


Figura 4.7: Esquema del contenido del fichero que contiene el código del programa

### 4.3. GENERADOR DE CÓDIGO

---

- *Imports*: añade las librerías necesarias según las tareas que contenga el flujo de tareas.
- Definición del método del flujo: crea el método donde se encontrarán todas las tareas indicadas en el flujo de actividades.
- “*GenerateMetodos*”: es el encargado de crear el contenido del método del flujo indistintamente del tipo de tarea que sea. Esto se consigue gracias al polimorfismo soportado por Acceleo, que en función del tipo de tarea (es decir, el parámetro de entrada) se realizará la llamada al *template* asociado a cada una de ellas. En este caso los tipos de entrada son:
  - *Initial*: una vez encuentra la tarea *Initial* del flujo de actividades, mediante “*GenerateTarea*”, genera las llamadas a todas las tareas del flujo. Este *template* se describe con detalle a continuación, debido a que es utilizado en otros tipos de tareas.
  - *Loop*: crea los métodos de los bucles de tipo “*WHILE*” y “*DO WHILE*”, necesarios para la interfaz. A continuación, llama de nuevo a “*GenerateMetodos*” para volver a encontrar la tarea *Initial* o añadir los métodos necesarios restantes.
  - *Question*: crea el método correspondiente a cada respuesta relacionada con un botón de la interfaz. Finalmente, crea las llamadas a los métodos del flujo que corresponde a esa respuesta con “*GenerateTarea*”.

Adicionalmente, se han utilizado tres *templates* más, encargados de añadir los parámetros de las tareas correspondientes al flujo. Estos son:

- “*GenerateParameters*”: encargado de añadir al método sus parámetros y asignarles un valor por defecto al parámetro.
- “*GenerateParametersLoop*”: encargado de añadir los parámetros necesarios al método, en particular, la primera llamada en los bucles. Luego, hace llamadas recursivas a “*GenerateParametersInterfaz*”.

### 4.3. GENERADOR DE CÓDIGO

---

- “*GenerateParametersInterfaz*”: encargado de añadir los parámetros necesarios al método, al igual que el anterior, pero en este caso, de forma genérica para la interfaz gráfica.
- “*GenerateMain*”: se trata del código principal (*main*) y se encarga de ejecutar el método del flujo de actividades.

Como se comentaba anteriormente, el *template* “*GenerateTarea*” es el encargado de generar las llamadas a todas las tareas del flujo, indistintamente del tipo que sea. Por lo que, como en el caso anterior, también se ha hecho uso de *templates* polimórficos. En este caso los tipos de entrada son:

- *Activity*: llama al método que se encuentra en la librería general, si es tarea simple, o en la librería específica, si es tarea compleja. Finalmente, hace una llamada recursiva a la siguiente tarea. (Ver Figura 4.8).

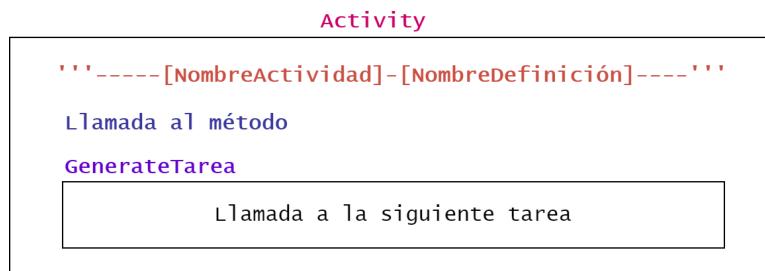


Figura 4.8: Esquema del *template* “*GenetareTarea*” de *Activity*

- *Question*: inicialmente, crea la configuración de la ventana de la interfaz. A continuación, crea un botón para cada posible respuesta, relacionada con su correspondiente acción. Esta relación, es la llamada al método si el botón es pulsado (creado en el *template* “*GenerateCodigo*”), por lo que se requiere añadir los parámetros a la llamada mediante el *template* “*GenerateParametersInterfaz*”. (Ver Figura 4.9).
- *Loop*: por un lado, si es de tipo WHILE o DO\_WHILE, se añade la asignación de una etiqueta para la ventana de la interfaz. A su vez, si es DO\_WHILE, se realiza la primera iteración de ese bucle. Por otro lado, si es de tipo COUNTER,

#### 4.4. DESPLIEGUE EN COZMO

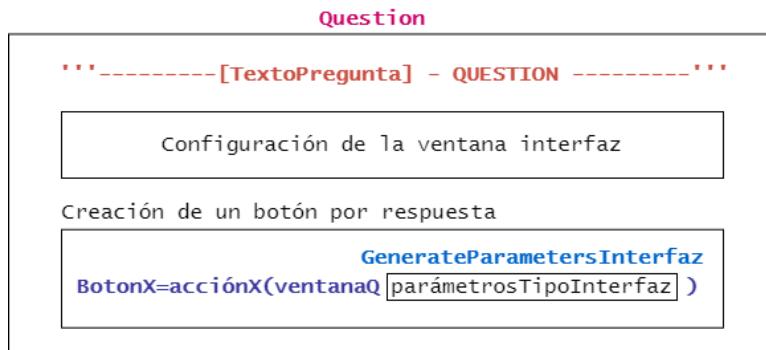


Figura 4.9: Esquema del *template* “*GenetareTarea*” de *Question*

crea un bucle *for* con el número específico de repeticiones. Este bucle contiene las llamadas a los métodos correspondientes. Y si es de tipo WHILE o DO\_WHILE, añade el resto de configuración de la interfaz y crea dos botones de respuesta. Uno está relacionado con el método que hace que se repita el bucle (creado en el *template* “*GenerateCodigo*”) y otro con la finalización del bucle. Por último, hace una llamada recursiva a la siguiente tarea. (Ver Figura 4.10).

Para más detalle del generador puede encontrar el código de Acceleo completo en el Anexo A.4.

## 4.4. Despliegue en Cozmo

Este fichero es auto generado junto con el programa que se envía a Cozmo (llamado “EjecutarCozmo.bat” y “[NombreAccion].py”, respectivamente). De esta manera, el usuario solo debe hacer doble clic sobre el primero, y este, ejecuta un comando que envía a Cozmo el segundo fichero, el cual será ejecutado en este, para lo que es necesario que el ordenador esté conectado a la red Wi-Fi de Cozmo explicado en la Sección 4.7.1.

El fichero “EjecutarCozmo.bat” contiene la llamada del programa. Un ejemplo de este fichero puede observarse en la Figura 4.11, donde la primera linea hace que no muestre la ejecución del *script* en la terminal; la segunda imprime una linea de información

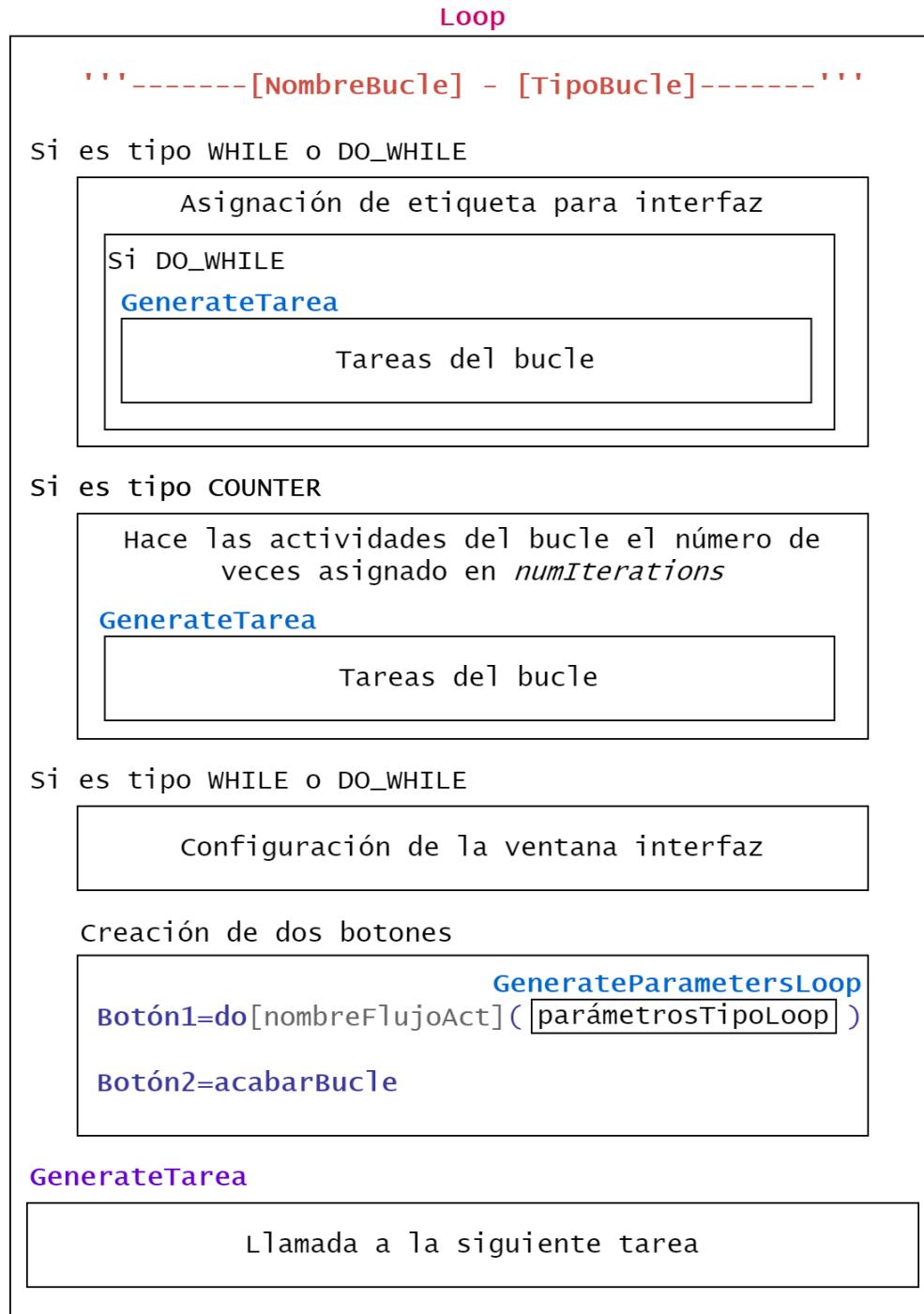


Figura 4.10: Esquema del template “*GenetareTarea*” de *Loop*

#### 4.5. ECLIPSE APPLICATION

indicando el programa que se va a ejecutar; y la tercera hace la llamada para que se ejecute el programa en Cozmo, en este caso *Contento.py*.

```
@echo off
echo Mandando Contento a Cozmo
python Contento.py
```

Figura 4.11: Ejemplo real de la creación de parámetros

## 4.5. Eclipse Application

Para facilitar el uso del editor gráfico del sistema, se ha creado una aplicación de Eclipse, la cual se compone de diferentes ventanas (ver Figura 4.12) y se indican a continuación con una breve descripción.

1. **Barra de herramientas:** contiene los ajustes principales del proyecto. Crear un proyecto, abrir un proyecto, salir del editor y guardar el proyecto.
2. **Panel conteniente del proyecto:** incluye la carpeta en la que se almacenan los diagramas del catálogo y los flujos de actividades.
3. **Ventana de edición:** en esta se abrirán los ficheros del catálogo o los flujos de actividades para realizar nuevos diagramas, modificaciones o ampliaciones.
4. **Paleta:** en el panel se encuentran los elementos que se añaden en la ventana de edición. Según el fichero que se vaya a editar se mostrarán unos elementos u otros (ver Figuras 4.13 y 4.14).
5. **Propiedades:** en este panel se encuentran las variables de los objetos añadidos a la ventana de edición. Serán modificadas por los usuarios.



#### 4.5. ECLIPSE APPLICATION

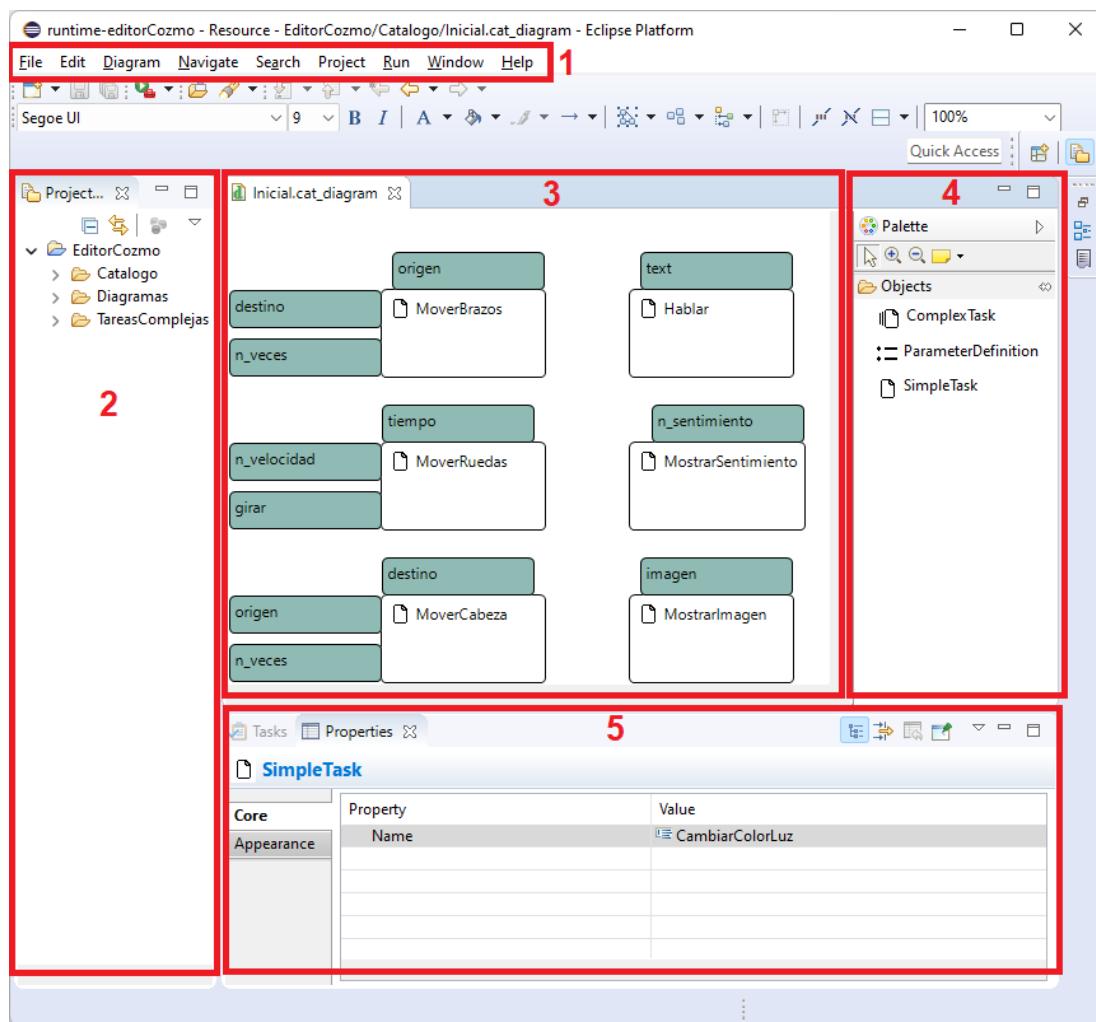


Figura 4.12: Interfaz del editor de Cozmo

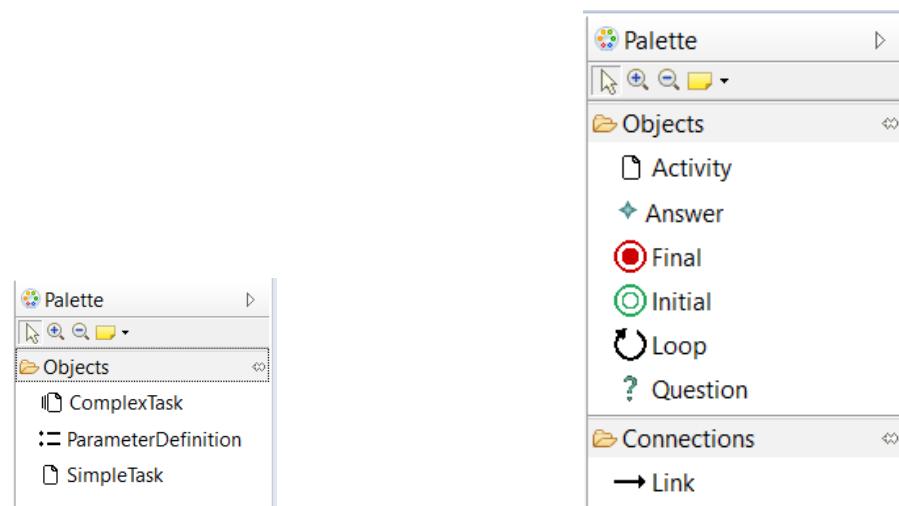


Figura 4.13: Paleta del catálogo

Figura 4.14: Paleta del flujo de actividades

## 4.6. INTERFAZ DE INTERACCIÓN

### 4.6. Interfaz de interacción

Esta interfaz ha sido desarrollada para interactuar con el usuario en tres ocasiones específicas: preguntas, tareas que se pueden repetir tantas veces como el usuario quiera y tareas que se hacen al menos una vez y luego pueden repetirse como la tarea anterior.

En el primer caso, la tarea implicada es “*Question*” (Pregunta), que se compone de al menos dos “*Answers*” (Respuestas). Estas respuestas son las mostradas en la interfaz y las posibles respuestas serán las definidas en el flujo de actividades, por ejemplo “si” o “no” (ver Figura 4.15). Dependiendo de la elección, Cozmo realizará un flujo de tarea establecido u el otro.

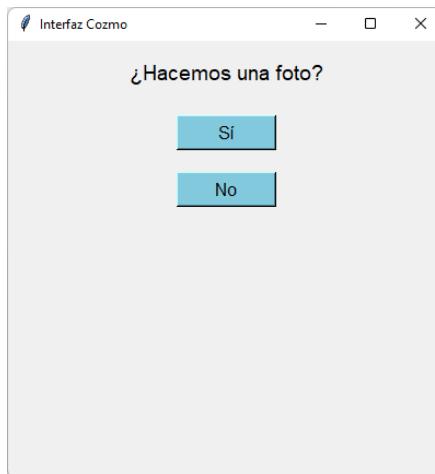


Figura 4.15: Interfaz con las opciones de respuestas

En el segundo y tercer caso, la tarea implicada es “*Loop*” (Bucle). La única diferencia es la propiedad “*LoopType*” (Tipo de bucle), en el segundo caso esta propiedad es “*WHILE*”, que puede no realizarse en ninguna ocasión; y en el tercero, “*DO WHILE*”, donde la tarea se va a realizar al menos una vez. Esta tarea contiene un flujo de actividades que va a realizar, o no, según se le indique en la interfaz (ver Figura 4.16).

## 4.7. AJUSTES DE COZMO



Figura 4.16: Interfaz que pregunta si repite o no el flujo

## 4.7. Ajustes de Cozmo

En este apartado se define cómo se realiza la conexión entre Cozmo y el ordenador y cómo se actualiza Cozmo mediante el móvil.

### 4.7.1. Conexión con Cozmo

Como se ha comentado anteriormente, la conexión de Cozmo con el dispositivo (ordenador) en el que se encuentra la secuencia de tareas que debe seguir es importante debido a que sin esta, el robot no podría cargar las tareas a realizar ni interactuar con los terapeutas.

Además, este proceso de conexión es muy sencillo y será necesario llevarlo a cabo para cada uno de los robots que se utilicen en un entorno real.

El primer paso consiste en conectar la plataforma, donde se carga el robot, a la red eléctrica. A continuación, el robot se coloca sobre la plataforma y se pulsa el botón que tiene a su espalda, el cual se debe iluminar la pantalla con nombre similar a “Cozmo\_-XXXXXX”, donde las “Xs” son una conjunto de números y letras; junto con la contraseña requerida para su conexión (ver Figura 4.17). Esta información corresponde a la red de Cozmo, la cual, es necesaria conectarse tanto en el móvil (para actualizaciones) como en el ordenador (para enviarle el programa).

#### 4.7. AJUSTES DE COZMO



Figura 4.17: Pantalla de información de Cozmo

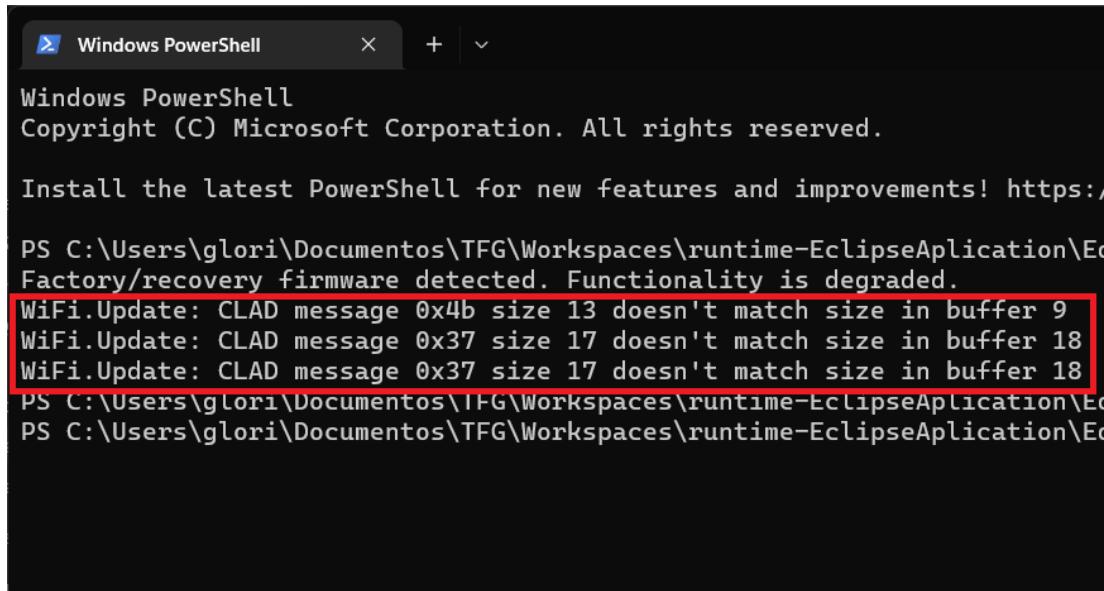
##### 4.7.2. Actualización de Cozmo

En ciertas ocasiones, el robot Cozmo puede quedarse bloqueado o no llegar a cargar el código de forma correcta, mostrando un error similar al de la Figura 4.18 que hace que no realice ninguna tarea y por tanto, no se mueva.

En este caso, la solución encontrada consiste en la instalación de la aplicación móvil de Cozmo [48], la cual se encargará de realizar el proceso de conexión y reinicio del robot. Para ello, será necesario que el dispositivo móvil se conecte a la red Wi-Fi de Cozmo y, posteriormente, se inicie dicha aplicación (ver Figura 4.19), dejando que cargue todos los recursos de la misma durante un breve periodo de tiempo.

Por último, cuando la aplicación muestre una pantalla similar a la que se muestra en la Figura 4.20, significará que el proceso de actualización se ha realizado correctamente y que ya se podrá enviar el flujo de tareas, de nuevo, al robot. Para ello, se debe cerrar la aplicación y hacer doble clic en el fichero “EjecutarCozmo.bat”, situado en el

#### 4.7. AJUSTES DE COZMO



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following text:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://go.microsoft.com/fwlink/?LinkID=832754&clcid=0x407

PS C:\Users\glori\Documentos\TFG\Workspaces\runtime-EclipseApplication\EduFactory/recovery firmware detected. Functionality is degraded.

WiFi.Update: CLAD message 0x4b size 13 doesn't match size in buffer 9
WiFi.Update: CLAD message 0x37 size 17 doesn't match size in buffer 18
WiFi.Update: CLAD message 0x37 size 17 doesn't match size in buffer 18

PS C:\Users\glori\Documentos\TFG\Workspaces\runtime-EclipseApplication\EduFactory/recovery firmware detected. Functionality is degraded.

PS C:\Users\glori\Documentos\TFG\Workspaces\runtime-EclipseApplication\EduFactory/recovery firmware detected. Functionality is degraded.
```

The last three lines of the error message are highlighted with a red rectangle.

Figura 4.18: Pantalla de error en la ejecución del programa



Figura 4.19: Pantalla con la actualización de Cozmo cargando

#### 4.7. AJUSTES DE COZMO

ordenador, de nuevo.

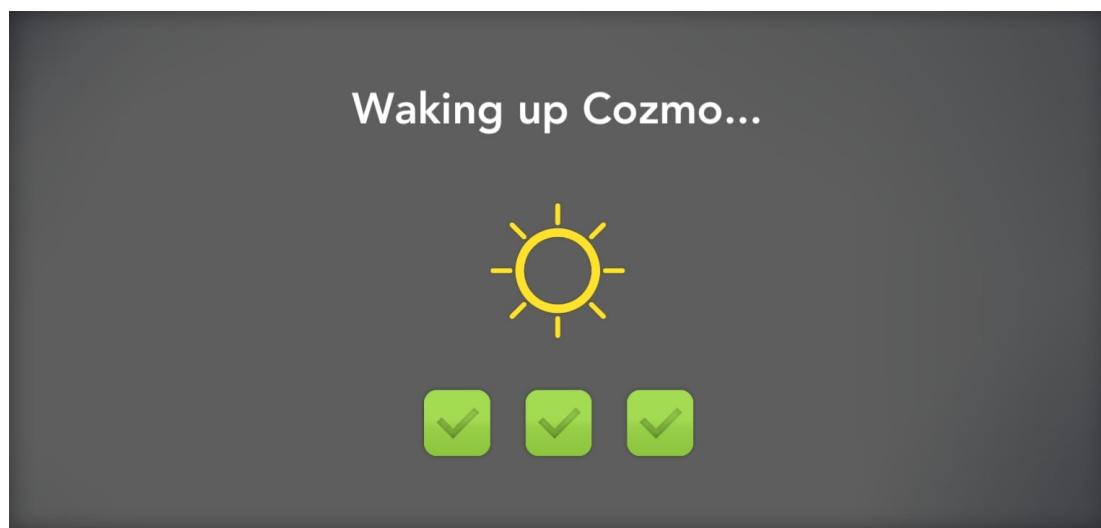


Figura 4.20: Pantalla con la conexión de Cozmo completa

# Capítulo 5

## Ejemplos de funcionamiento

En este capítulo se presentan varios flujos de actividades, de complejidad incremental, que contemplan algunas posibilidades de diseño de la herramienta *PiLHaR*.

Es importante mencionar que el código desarrollado se encuentra en un repositorio público de GitHub [49]. Para el correcto uso del entorno gráfico se han elaborado dos manuales de uso. Uno para el desarrollador, que sigue los pasos para desarrollar un editor Cozmo desde cero (ver Anexo A.7), y otro para el usuario, que explica como manejar el entorno (ver Anexo A.5). A su vez, también dispone de un manual donde indica el valor que pueden tomar los parámetros de cada método (ver Anexo A.6). Finalmente, todo el proceso de desarrollo en Eclipse se ha guardado paso a paso en el Anexo A.3.

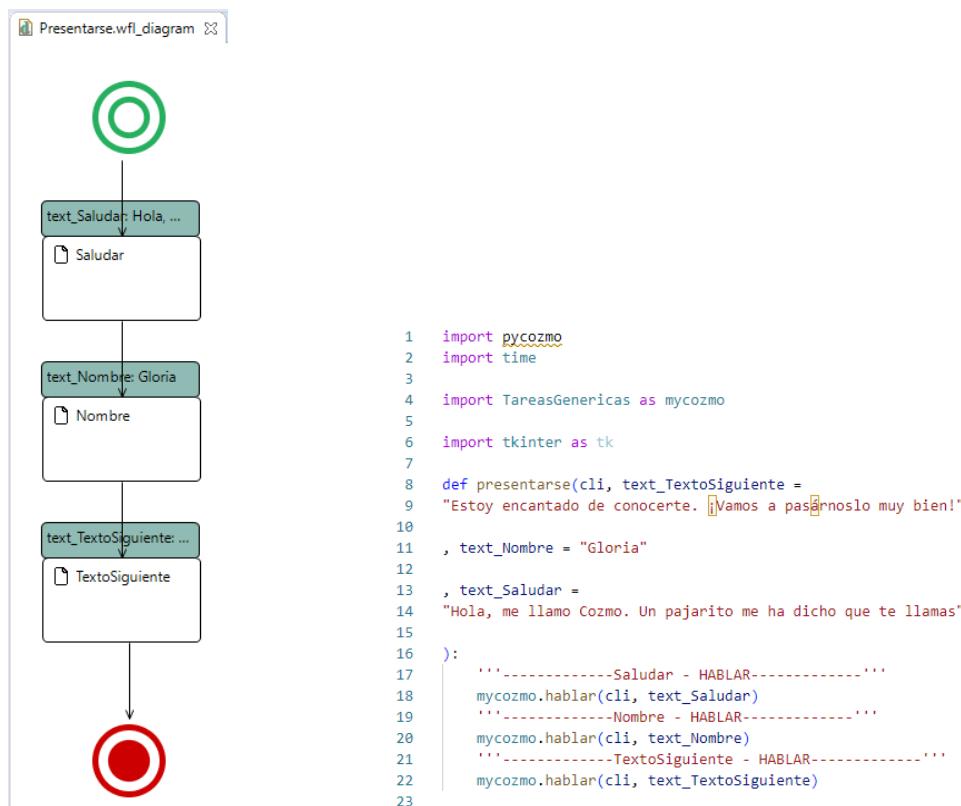
### 5.1. Ejemplo 1: Presentarse

En primer lugar, se ha realizado un flujo, *Presentarse.wfl\_diagram*, que contiene únicamente “Tareas Simples” de tipo “Hablar”. Su representación completa (a) junto con el código generado (b) se puede observar en la Figura 5.1. Este flujo de actividades hace que el robot salude al usuario encadenando tres mensajes hablados consecutivamente,

## 5.1. EJEMPLO 1: PRESENTARSE

los cuales son:

- **Saludar:** “Hola, me llamo Cozmo. Un pajarito me ha dicho que te llamas”.
- **Nombre:** “Gloria”.
- **TextoSiguiente:** “Estoy encantado de conocerte. ¡Vamos a pasárnoslo muy bien!”.



(a) Diagrama de flujo de Presentarse

(b) Código auto-generado del flujo Presentarse

Figura 5.1: Ejemplo 1: Presentarse

Adicionalmente, se ha creado una “Tarea Compleja”, llamada “Presentarse”, a partir del modelo anterior, cuya definición apuntará al flujo de actividades definido previamente. Esta “Tarea Compleja” tiene un parámetro de tipo “STRING” que se vincula con el parámetro de la segunda actividad del flujo de actividades “text\_Nombre”. De este modo, “Presentarse” será una tarea parametrizada que se podrá reutilizar para diferentes usuarios cambiando el nombre de este. En la Figura 5.2) se muestra, a la izquierda, el aspecto que presenta esta nueva “Tarea Compuesta” al Catálogo; y, a la

## 5.2. EJEMPLO 2: MOSTRAR ALEGRÍA

derecha, la representación del contenido.

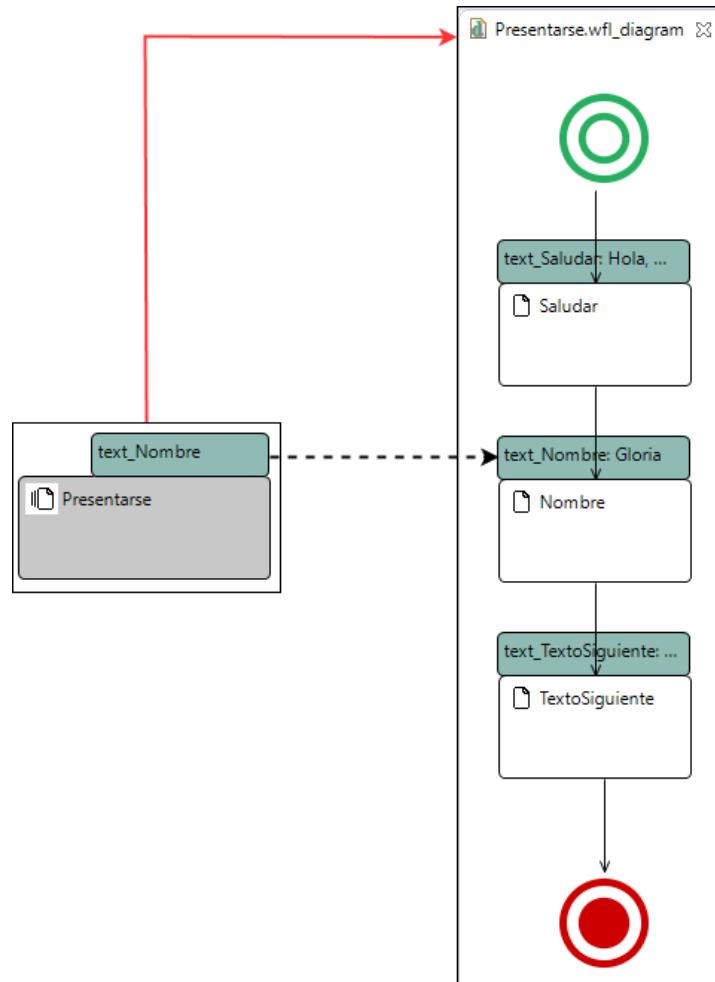


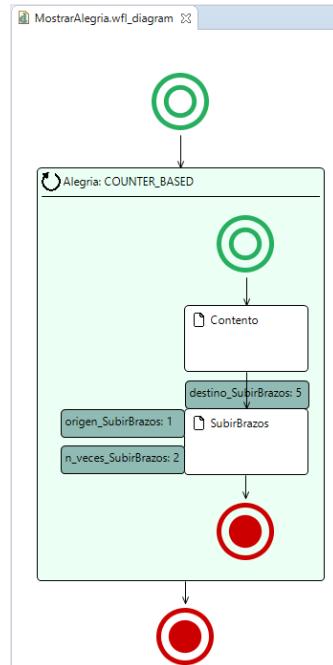
Figura 5.2: Tarea compleja Presentarse

## 5.2. Ejemplo 2: Mostrar Alegría

En segundo lugar, se ha realizado un nuevo flujo de actividades (*MostrarAlegria.wfl-diagram*) que contiene únicamente una tarea de tipo “Bucle”. Este bucle contiene, a su vez, un flujo de actividades de dos tareas: la primera es una “Tarea Compuesta” de tipo “Contento”; y, la segunda es una “Tarea Simple” de tipo “MoverBrazo”. Su representación completa (a) y el código generado (b) se puede observar en la Figura 5.3. La ejecución de este nuevo flujo hará que el robot muestre en su pantalla una cara

## 5.2. EJEMPLO 2: MOSTRAR ALEGRÍA

sonriente, posteriormente subiendo y bajando los brazos un total de 3 veces.



(a) Diagrama de flujo de MostrarAlegria

```

1  import pycozmo
2  import time
3
4  import TareasGenericas as mycozmo
5  from Contenido import *
6
7  import tkinter as tk
8
9  def mostrarAlegria(cli
10    , origen_SubirBrazos = 1
11    , destino_SubirBrazos = 5
12    , n_veces_SubirBrazos = 2
13
14  ):
15      ...
16      ...
17      ...
18      for i in range(3):
19          ...
20          ...
21          ...
22          ...
23          ...
24

```

(b) Código auto-generado del flujo Presentarse

Figura 5.3: Ejemplo 2: Mostrar Alegria

Como en el caso anterior, se va a añadir al Catálogo una nueva “Tarea Compleja”, llamada “MostraAlegria”, cuya definición apuntará al flujo de actividades definido pre-

### 5.3. EJEMPLO 3: FLUJO COMPLEJO

viamente para reutilizarlo posteriormente. En la Figura 5.4) se muestra, a la izquierda, el aspecto que presenta esta nueva “Tarea Compuesta” al Catálogo; y, a la derecha, la representación del contenido.

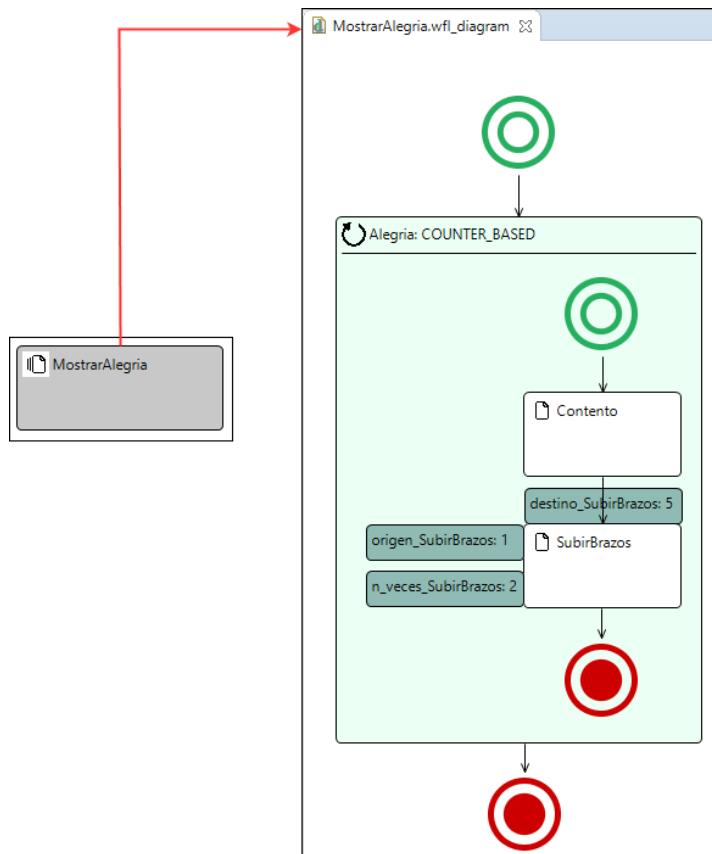


Figura 5.4: Tarea compleja MostrarAlegria

### 5.3. Ejemplo 3: Flujo Complejo

En el ejemplo 3, se ha definido un flujo de actividades (Actividad1.wfl\_diagram) que, además de incorporar varias actividades asociadas a “Tareas Simples” de tipo “Hablar”, “MoverBrazos”, “CambiarColorLuz”, reutiliza las dos “Tareas Complejas” previamente añadidas al Catálogo: “Presentarse” y “MostrarAlegria”. Además, como se muestra en la Figura 5.5, contiene dos estructuras condicionales que guiarán la ejecución del programa en función de las respuestas que proporcione el usuario a las preguntas planteadas. El código generado se muestra en la Figuras 5.6, 5.7, 5.8, y 5.9.

### 5.3. EJEMPLO 3: FLUJO COMPLEJO

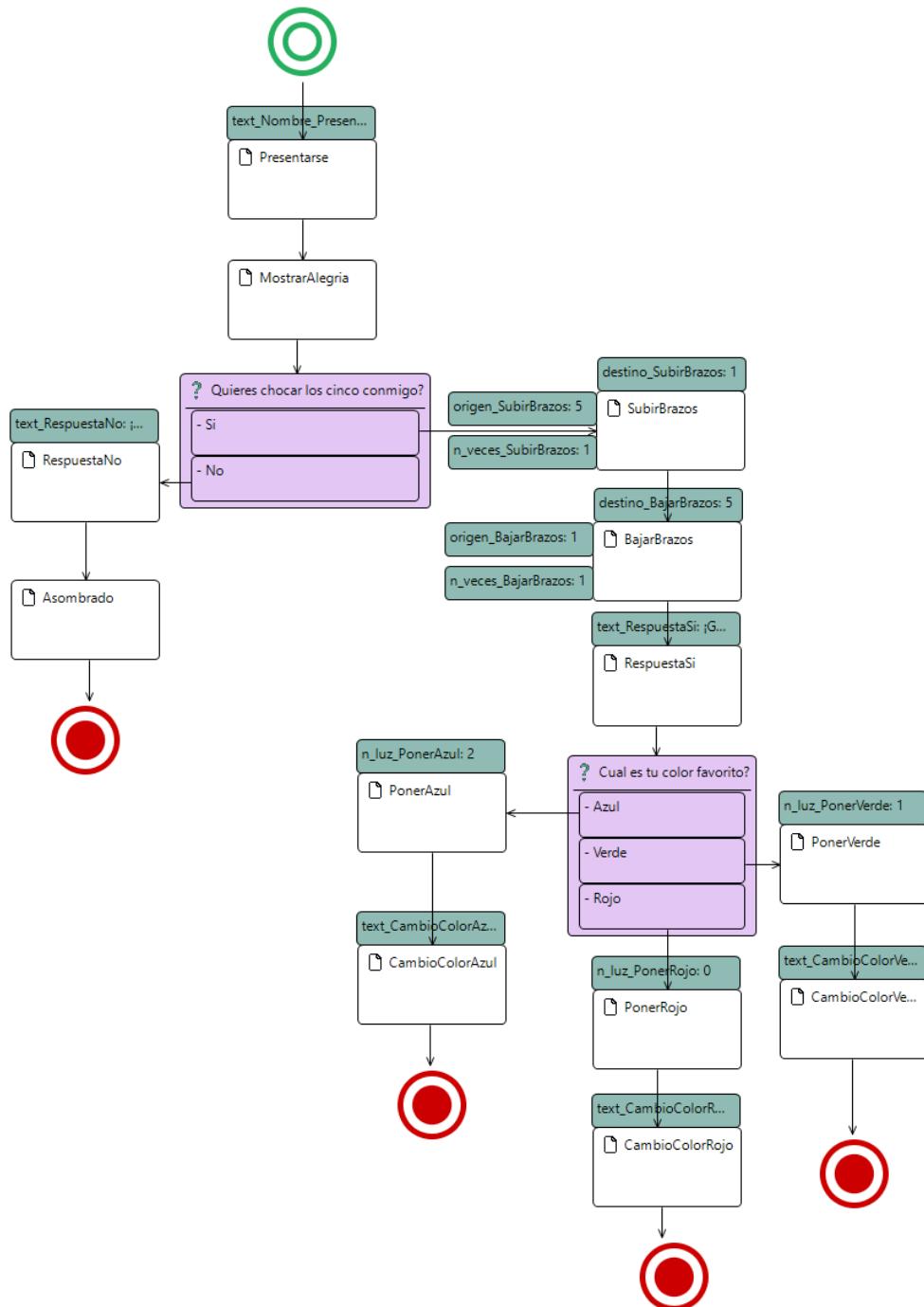


Figura 5.5: Diagrama de flujo con la tarea pregunta

### 5.3. EJEMPLO 3: FLUJO COMPLEJO

```
1  import pycozmo
2  import time
3
4  import TareasGenericas as mycozmo
5  from Presentarse import *
6  from MostrarAlegria import *
7  from Asombrado import *
8
9  import tkinter as tk
10
11 def actividad1(cli, text_Nombre_Presentarse = "Julia"
12
13
14     , text_RespuestaNo = "¡Vale! No pasa nada"
15
16
17     , origen_SubirBrazos = 5
18     , destino_SubirBrazos = 1
19     , n_veces_SubirBrazos = 1
20
21     , origen_BajarBrazos = 1
22     , destino_BajarBrazos = 5
23     , n_veces_BajarBrazos = 1
24
25     , text_RespuestaSi = "¡Genial, qué divertido!"
26
27     , n_luz_PonerAzul = 2
28
29     , n_luz_PonerVerde = 1
30
31     , n_luz_PonerRojo = 0
32
33     , text_CambioColorAzul = "¡Qué color más bonito!"
34
35     , text_CambioColorVerde = "¡Qué guay el color verde!"
36
37     , text_CambioColorRojo = "¡He puesto el color rojo!"
38
39 ):
40     """-----Presentarse - PRESENTARSE-----"""
41     presentarse(cli, text_Nombre=text_Nombre_Presentarse)
42
43     """-----MostrarAlegria - MOSTRARALEGRIA-----"""
44     mostrarAlegria(cli)
45
```

Figura 5.6: Código Ejemplo 3: Flujo Complejo (1)



### 5.3. EJEMPLO 3: FLUJO COMPLEJO

```

46   '''-----Quieres chocar los cinco conmigo? - QUESTION-----
47   text_label = "Quieres chocar los cinco conmigo?"
48
49   ventanaQ=tk.Tk()
50   #set center screen window with following coordination
51   MyLeftPos = (ventanaQ.winfo_screenwidth() - 400) / 2
52   myTopPos = (ventanaQ.winfo_screenheight() - 400) / 2
53   ventanaQ.geometry( "%dx%d+%d+%d" % (400, 400, MyLeftPos, myTopPos))
54
55   ventanaQ.title("Interfaz Cozmo")
56
57   lbl = tk.Label(ventanaQ, text=text_label, font=("Arial Bold", 14))
58   lbl.pack(pady=15)
59
60   answer1 = "Si"
61   boton1=tk.Button(ventanaQ, text=answer1, bg="#82c9dd", width = 9,
62   |           |           |           |           |           |           |
63   |           |           |           |           |           |           |           font = ("Bold"),
64   |           |           |           |           |           |           |           command=lambda: actionSi1(ventanaQ, origen_SubirBrazos,
65   |           |           |           |           |           |           |           destino_SubirBrazos, n_veces_SubirBrazos, origen_BajarBrazos,
66   |           |           |           |           |           |           |           destino_BajarBrazos, n_veces_BajarBrazos, text_RespuestaSi,
67   |           |           |           |           |           |           |           n_luz_PonerAzul, text_CambioColorAzul, n_luz_PonerVerde,
68   |           |           |           |           |           |           |           text_CambioColorVerde, n_luz_PonerRojo, text_CambioColorRojo))
69   boton1.pack(pady=10)
70   answer2 = "No"
71   boton2=tk.Button(ventanaQ, text=answer2, bg="#82c9dd", width = 9,
72   |           |           |           |           |           |           |           font = ("Bold"),
73   |           |           |           |           |           |           |           command=lambda: actionNo2(ventanaQ, text_RespuestaNo))
74   boton2.pack(pady=10)
75   ventanaQ.mainloop()
76
77   def actionSi1(ventanaQ, origen_SubirBrazos, destino_SubirBrazos,
78   n_veces_SubirBrazos, origen_BajarBrazos, destino_BajarBrazos,
79   n_veces_BajarBrazos, text_RespuestaSi, n_luz_PonerAzul,
80   text_CambioColorAzul, n_luz_PonerVerde, text_CambioColorVerde,
81   n_luz_PonerRojo, text_CambioColorRojo):
82   '''-----SubirBrazos - MOVERBRAZOS-----'''
83   mycozmo.moverBrazos(cli, origen_SubirBrazos, destino_SubirBrazos,
84   |           |           |           |           |           |           |           n_veces_SubirBrazos)
85
86   '''-----BajarBrazos - MOVERBRAZOS-----'''
87   mycozmo.moverBrazos(cli, origen_BajarBrazos, destino_BajarBrazos,
88   |           |           |           |           |           |           |           n_veces_BajarBrazos)
89
90   '''-----RespuestaSi - HABLAR-----'''
91   mycozmo.hablar(cli, text_RespuestaSi)

```

Figura 5.7: Código Ejemplo 3: Flujo Complejo (2)

### 5.3. EJEMPLO 3: FLUJO COMPLEJO

---

```

91      """-----Cual es tu color favorito? - QUESTION-----
92      text_label = "Cual es tu color favorito?"
93
94      ventanaQ=tk.Tk()
95      #set center screen window with following coordination
96      MyLeftPos = (ventanaQ.winfo_screenwidth() - 400) / 2
97      myTopPos = (ventanaQ.winfo_screenheight() - 400) / 2
98      ventanaQ.geometry( "%dx%d+%d+%d" % (400, 400, MyLeftPos, myTopPos))
99
100     ventanaQ.title("Interfaz Cozmo")
101
102     lbl = tk.Label(ventanaQ, text=text_label, font=("Arial Bold", 14))
103     lbl.pack(pady=15)
104
105     answer1 = "Azul"
106     boton1=tk.Button(ventanaQ, text=answer1, bg="#82c9dd", width = 9,
107                         | font = ("Bold"),
108                         | command=lambda: actionAzul1(ventanaQ, n_luz_PonerAzul,
109                         |                         text_CambioColorAzul))
110     boton1.pack(pady=10)
111     answer2 = "Verde"
112     boton2=tk.Button(ventanaQ, text=answer2, bg="#82c9dd", width = 9,
113                         | font = ("Bold"),
114                         | command=lambda: actionVerde2(ventanaQ, n_luz_PonerVerde,
115                         |                         text_CambioColorVerde))
116     boton2.pack(pady=10)
117     answer3 = "Rojo"
118     boton3=tk.Button(ventanaQ, text=answer3, bg="#82c9dd", width = 9,
119                         | font = ("Bold"),
120                         | command=lambda: actionRojo3(ventanaQ, n_luz_PonerRojo,
121                         |                         text_CambioColorRojo))
122     boton3.pack(pady=10)
123     ventanaQ.mainloop()
124
125     ventanaQ.quit()
126
127 def actionNo2(ventanaQ, text_RespuestaNo):
128     """-----RespuestaNo - HABLAR-----"""
129     mycozmo.hablar(cli, text_RespuestaNo)
130
131     """-----Asombrado - ASOMBRAZO-----"""
132     asombrado(cli)
133
134
135     ventanaQ.quit()

```

Figura 5.8: Código Ejemplo 3: Flujo Complejo (3)



### 5.3. EJEMPLO 3: FLUJO COMPLEJO

---

```
137 def actionAzul1(ventanaQ, n_luz_PonerAzul, text_CambioColorAzul):
138     '''-----PonerAzul - CAMBIARCOLORLUZ-----'''
139     mycozmo.cambiarColorLuz(cli, n_luz_PonerAzul)
140
141     '''-----CambioColorAzul - HABLAR-----'''
142     mycozmo.hablar(cli, text_CambioColorAzul)
143
144
145     ventanaQ.quit()
146
147 def actionVerde2(ventanaQ, n_luz_PonerVerde, text_CambioColorVerde):
148     '''-----PonerVerde - CAMBIARCOLORLUZ-----'''
149     mycozmo.cambiarColorLuz(cli, n_luz_PonerVerde)
150
151     '''-----CambioColorVerde - HABLAR-----'''
152     mycozmo.hablar(cli, text_CambioColorVerde)
153
154
155     ventanaQ.quit()
156
157 def actionRojo3(ventanaQ, n_luz_PonerRojo, text_CambioColorRojo):
158     '''-----PonerRojo - CAMBIARCOLORLUZ-----'''
159     mycozmo.cambiarColorLuz(cli, n_luz_PonerRojo)
160
161     '''-----CambioColorRojo - HABLAR-----'''
162     mycozmo.hablar(cli, text_CambioColorRojo)
163
164
165     ventanaQ.quit()
166
```

Figura 5.9: Código Ejemplo 3: Flujo Complejo (4)

# Capítulo 6

## Conclusiones y trabajos futuros

Este capítulo recoge las principales dificultades encontradas durante el desarrollo del Proyecto (junto con las soluciones adoptadas), las conclusiones extraídas tras su finalización y algunas posibles mejoras y líneas de trabajo que podrían abordarse en el futuro.

### 6.1. Principales dificultades

Entre las dificultades encontradas durante el Proyecto, cabe mencionar las siguientes:

- **Conexión con Cozmo.** A la hora de hacer pruebas para comprobar que el código desarrollado con nuestra herramienta funcionaba correctamente en Cozmo, era necesario conectarse al robot en cada prueba a través de la red Wi-Fi del ordenador. Debido a que Cozmo no es un punto de acceso con conexión a Internet, el ordenador se desconectaba de él constantemente, haciendo muy pesada la re-conexión (era necesario buscar la información de la red cada vez para volver a conectarse). Para hacer más liviano este proceso, se ha desarrollado un *script* [50] que conecta con el robot o el ordenador en función de las necesidades del usuario.

## 6.1. PRINCIPALES DIFICULTADES

---

- **Falta de documentación sobre las librerías.** Como se ha comentado en la sección 2, actualmente existen dos librerías para la interacción con el robot Cozmo pero en ambas la documentación es escasa. Cada una, contiene la información de los métodos disponibles, pero no está claro ni su funcionamiento interno, ni cómo se deben usarse.
- **Disponibilidad de voces para Cozmo.** Una de las tareas a la que más tiempo se le ha dedicado ha sido a la de “Hablar”. Inicialmente, se intentó con la librería SDK Cozmo, en la que el tono de voz es el que tiene Cozmo por defecto, como un niño. El problema es que el único idioma que soporta es el inglés, por lo que su uso en este Proyecto no era factible (los terapeutas y los niños con TEA a los que va dirigido el trabajo sólo hablan español). La solución fue usar la librería PyCozmo, que permite reproducir un audio que se le introduce por parámetro. De esta forma se genera el fichero de audio a partir de un texto mediante un TTS (*TextToSpeech*) y se introduce como parámetro.
- **Librería de audio.** Como se ha indicado anteriormente, ha sido necesario utilizar una herramienta *TTS* para generar un fichero de audio asociado a una voz en un idioma determinado. En primer lugar, se intentó usar la librería *PyAudio*, que utiliza *gTTS* (*Google Text-to-Speech*), para generar el audio a partir de un texto. El problema que se encontró en esta librería es que necesita conexión a internet y no es posible su ejecución porque el ordenador debe estar conectado a la red de Cozmo, que como se ha comentado, no ofrece acceso a Internet. Para encontrar una solución hubo que buscar otra alternativa y se encontró la librería *pyttsx3*, que realiza correctamente la transformación de texto a audio.
- **Complejidad y conexión entre los meta-modelos.** el meta-modelo definido en primera instancia se componía a su vez de dos elementos raíz (*root*). Inicialmente, esto no ocasionó ningún problema, pero a la hora de definir los editores gráficos, comprobamos que sólo era posible obtener uno de ellos individualmente. La solución fue dividir el meta-modelo en dos, lo que solucionó el problema de

## 6.2. CONCLUSIONES

---

contar con dos editores gráficos, pero complicó considerablemente el desarrollo del resto del Proyecto. La dificultad se encontraba en implementar dos editores gráficos enlazados por referencias, ya que no existe a penas documentación y se requirió de mucho tiempo para dar con una solución. Todo el desarrollo se encuentra disponible paso a paso en el Anexo A.3.

- **Desarrollo de un Eclipse RCP.** Para una mejor experiencia de usuario, se propuso desarrollar un Eclipse RCP (*Eclipse Rich Client Platform*) configurado de modo que los usuarios de la herramienta sólo tuvieran acceso a aquellas opciones y pestañas de Eclipse realmente necesarias para su funcionamiento. Se investigó bastante sobre cómo implementar aplicaciones stand-alone basadas en RCP, pero, finalmente, no dimos con una solución.

## 6.2. Conclusiones

Una vez finalizado el Proyecto, a continuación se mencionan algunas de las conclusiones extraídas del trabajo realizado, lo aprendido y la forma en la que se llevó a cabo.

Como resultado de este trabajo, se ha desarrollado *PiLHaR*: una herramienta para facilitar el modelado, la composición, la reutilización y la ejecución de tareas en robots educativos, cumpliendo con los objetivos establecidos al inicio del Proyecto.

Para ello he utilizado una gran variedad de herramientas y tecnologías, algunas de las cuales las aprendí y utilicé durante mis estudios de Grado (por ejemplo, las relacionadas con ISDM), mientras que otras las he tenido que aprender prácticamente desde cero (por ejemplo, el lenguaje de programación Python y su gestión de librerías). Así, el Proyecto me ha permitido tanto profundizar en lo que ya sabía como aprender cosas nuevas. Sin embargo, conviene señalar que de algunas de las herramientas utilizadas apenas existe documentación , lo que ha hecho que la implementación de la herramienta, en determinados momentos, haya sido más compleja de lo esperado.

### 6.3. TRABAJOS FUTUROS

---

A su vez, es muy grata la sensación de haber realizado un Proyecto tan amplio y con un fin social tan bonito. Creo que es necesario que todas las personas nos centremos un poco más en ayudar a los demás en la medida que cada uno sepamos y podamos.

Por último, como conclusión más personal me gustaría agradecer a mis tutores la ayuda y el apoyo que me han proporcionado en todo momento. El final de esta etapa ha sido muy llevadera gracias a la forma en la que hemos llevado a cabo el trabajo y como hemos afianzado la relación por el trato tan cercano que hemos tenido desde el principio del Proyecto.

## 6.3. Trabajos futuros

Para finalizar, a continuación se describen algunas posibles mejoras y líneas de trabajo futuras que se han identificado y que podría resultar interesante que alguien abordara, como continuación de este Trabajo.

- Añadir una base de datos para almacenar la información de cada estudiante junto con los flujos de actividades que le pertenecen, además de un histórico de aquellos que haya realizado, junto con algunas observaciones del terapeuta. A su vez, también sería interesante guardar datos personales del estudiante relativos a su edad, nombre, etc. para poder utilizarlos de forma más sencilla en las tareas simples.
- Investigar en mayor profundidad las funciones que ofrece la librería PyCozmo para tratar de añadir nuevas tareas interesantes al catálogo como, por ejemplo, ver cómo podrían utilizarse los sensores que tiene Cozmo.
- Añadir nuevos parámetros a algunas tareas simples, permitiendo así una mayor configurabilidad. Por ejemplo, en la tarea simple “MoverBrazos” añadir como parámetro “velocidad”.
- En el editor gráfico, desarrollar una funcionalidad que al hacer doble clic en

### 6.3. TRABAJOS FUTUROS

---

una tarea compleja, se muestre el flujo de actividades al que está enlazado, pero únicamente en modo lectura, es decir, sin que se pueda editar.

- Probar el funcionamiento del sistema en diferentes colegios y centros especializados ya que, por ahora, únicamente se ha mostrado la herramienta al Taller de los Sueños.
- Ampliar el ámbito de trabajo de PiLHaR a familias donde los que desarrollen los flujos de actividades puedan ser los padres, madres o tutores de los niños.
- Mejorar la interfaz gráfica añadiendo iconos personalizados a cada tarea simple (por ejemplo, estaría bien representar la tarea hablar con un icono de un altavoz). Y a su vez, a las tareas compuestas, poder añadirles también un icono personalizado.
- Modificar y mejorar ciertas funcionalidades y aspectos del robot como el tono de voz o la pantalla (por ejemplo, que muestre ojos y boca, en lugar de sólo los ojos, para dar más expresividad).
- Ofrecer un sistema de control de versiones de las tareas complejas, de tal forma que se pueda recuperar una tarea compleja anterior si es necesario, evitando así que se tenga que hacer desde cero o modificarla cada vez.
- Desarrollar un Eclipse RCP para facilitar el uso de la herramienta. Esto no ha sido posible en el marco de este Proyecto debido a la falta tanto de documentación como de tiempo.

# Bibliografía

- [1] Paola Nagovitch. Los trastornos mentales en niños y adolescentes se triplican con la pandemia: “Pensaba en el suicidio cada día, cada noche”. <https://cutt.ly/oB5WcKT>.
- [2] Los datos del autismo en España. <https://www.infosalus.com/asistencia/noticia-datos-autismo-espana-20191129142534.html>.
- [3] Autism Speaks. Autism Statistics and Facts. <https://www.autismspeaks.org/autism-statistics-asd>.
- [4] YoTambién. El Trastorno del Espectro Autista en cifras y datos. <https://www.yotambien.mx/actualidad/el-trastorno-del-espectro-autista-en-cifras-y-datos/>.
- [5] Raquel Saez. ¿Cómo gestionan los niños autistas sus emociones? <https://www.mundodeportivo.com/vidae/psicologia-bienestar/20211008/1001697475/como-gestionan-ninos-autistas-emociones-act-pau.html>.
- [6] Criptogen. Los tipos de robots educativos. <https://criptogen.com/los-tipos-de-robots-educativos/>.
- [7] IEBS (Escuela de Negocios de la Innovación y los Emprendedores). Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa. <https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/#:~:text=Por%20definici%C3%B3n%2C%20las%20metodolog%C3%ADas%20de%20los%20trabajos%20se%20realizan%20en%20entornos%20flexibles,%20circunstancias%20espec%C3%ADficas%20del%20entorno.>

## BIBLIOGRAFÍA

---

- [8] **ClickUp.** ClickUp. <https://app.clickup.com/>.
- [9] **GitHub.** GitHub. <https://github.com/>.
- [10] Digital Dream Labs. Cozmo. <https://www.digitaldreamlabs.com/pages/meet-cozmo>.
- [11] Digital Dream Lab. Vector. <https://www.digitaldreamlabs.com/pages/meet-vector>.
- [12] Bjorn W. What is model driven engineering? <https://explainagile.com/agile/model-driven-engineering/>.
- [13] Eclipse. <https://www.eclipse.org/>.
- [14] Eclipse Modeling Framework. <https://www.eclipse.org/modeling/emf/>.
- [15] OCLInEcore. <https://wiki.eclipse.org/OCL/OCLInEcore>.
- [16] Graphical Modeling Framework. <https://www.eclipse.org/gmf-runtime/>.
- [17] **EuGENia.** EuGENia. <https://www.eclipse.org/epsilon/doc/eugenia/>.
- [18] **Acceleo.** Acceleo. <https://www.eclipse.org/acceleo/>.
- [19] EMF.cloud. Emf.cloud. <https://www.eclipse.org/emfcloud/>.
- [20] EMF.cloud. Theia ecore tool. <https://github.com/eclipse-emfcloud.ecore-glsp>.
- [21] EMF.cloud. Coffee editor overview. <https://www.eclipse.org/emfcloud/#coffeeeditoroverview>.
- [22] **RoboComp.** RoboComp. <https://robolab.unex.es/index.php/robocomp/>.
- [23] **Ice.** Ice. <https://zeroc.com/>.
- [24] **Orca2.** Orca2. <http://orca-robotics.sourceforge.net/>.
- [25] **SmartSoft.** SmartSoft. [https://robmosys.eu/wiki/baseline:environment\\_tools:smartsoft:start](https://robmosys.eu/wiki/baseline:environment_tools:smartsoft:start).

## BIBLIOGRAFÍA

---

- [26] **Scratch.** Scratch. <https://scratch.mit.edu/about>.
- [27] **Python.** Python. <https://www.python.org/>.
- [28] Digital Dream Labs. Documentación de Cozmo SDK. <http://cozmosdk.anki.com/docs/>.
- [29] Kaloyan Tenchov. Documentación de PyCozmo. <https://pycozmo.readthedocs.io/en/stable/>.
- [30] **Digital Dream Labs.** Makers of AI Robotic Companions. <https://www.digitaldreamlabs.com/>.
- [31] **Nao.** Nao. <https://www.softbankrobotics.com/emea/en/nao>.
- [32] Verónica Mollejo. Idea del mes: Robots como elemento clave en la terapia con niños autistas. <https://www.redbull.com/es-es/basement-tecnologia-robots-ni%C3%B1os-autistas>, 2018.
- [33] **MIT Medi Lab.** MIT Medi Lab. <https://www.media.mit.edu/>.
- [34] Ognjen (Oggi) Rudovic. Measuring Engagement in Robot-Assisted Autism Therapy: A Cross-Cultural Study. <https://www.media.mit.edu/publications/measuring-engagement-in-robot-assisted-autism-therapy-a-cross-cultural-study/>, 2017.
- [35] **Robot Atent@.** Robot Atent@. [https://www.upm.es/UPM/SalaPrensa/NoticiasPortada/NoticiasAppPersonalPDI?fmt=detail&prefmt=articulo&id=a0910854af9b7710VgnVCM10000009c7648a\\_\\_\\_\\_\\_](https://www.upm.es/UPM/SalaPrensa/NoticiasPortada/NoticiasAppPersonalPDI?fmt=detail&prefmt=articulo&id=a0910854af9b7710VgnVCM10000009c7648a_____).
- [36] S. Trapani and M. Indri. "Task modeling for task-oriented robot programming", 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2017, pp. 1-8. doi:<https://doi.org/10.1109/ETFA.2017.8247650>.
- [37] D. Kerr, U. Nehmzow, and S. Billings. "Towards Automated Code Generation for Autonomous Mobile Robots", 2010. doi:<https://doi.org/10.2991/ag.2010.37>.

## BIBLIOGRAFÍA

---

- [38] Haiyang Hu, Jie Chen, Hanwen Liu, Zhongjin Li, and Liguo Huang. "Natural Language-Based Automatic Programming for Industrial Robots", Journal of Grid Computing, 2022, 20. doi:<https://doi.org/10.1007/s10723-022-09618-x>.
- [39] **Procolo de conexión Cozmo.** Procolo de conexión Cozmo. <https://pycozmo.readthedocs.io/en/stable/external/protocol.html>.
- [40] David Musat y Jennifer Pérez y Pedro Pablo Alarcón. Tutorial de introducción a EMF y GMF. <https://es.scribd.com/document/444942260/MDCIGSW-Tutorial-de-introduccio-n-a-EMF-y-GMF-D-Musat-J-Perez-P-Alarcon>.
- [41] Francisco Pérez y Juan de Lara. Hacia la definición de lenguajes específicos de dominio con sintaxis gráfica y textual. [https://repositorio.uam.es/bitstream/handle/10486/665849/definicion\\_perez\\_CWP\\_2006.pdf?sequence=1](https://repositorio.uam.es/bitstream/handle/10486/665849/definicion_perez_CWP_2006.pdf?sequence=1), 2006.
- [42] **Librería time.** Librería time. <https://docs.python.org/es/3.10/library/time.html?highlight=time#module-time>.
- [43] **Librería pyttsx3.** Librería pyttsx3. <https://pypi.org/project/pyttsx3/>.
- [44] **Librería PIL.** Librería PIL. <https://thecleverprogrammer.com/2021/08/21/python-imaging-library-pil-tutorial/>.
- [45] **Librería NumPy.** Librería NumPy. <https://aprendeconalf.es/docencia/python/manual/numpy/>.
- [46] **Librería os.** Librería os. <https://docs.python.org/3/library/os.html>.
- [47] **Librería tkinter.** Librería tkinter. <https://pythonbasics.org/tkinter/>.
- [48] **Digital Dream Labs.** Cozmo App. <https://play.google.com/store/apps/details?id=com.digitaldreamlabs.cozmo2&gl=US>.
- [49] **Gloria Díaz González.** Repositorio de GitHub. <https://github.com/GloriaDG22/GeneracionCodigoCozmo>.

## BIBLIOGRAFÍA

---

- [50] **Gloria Díaz González.** Herramienta de conexión con Cozmo. <https://github.com/GloriaDG22/GeneracionCodigoCozmo/tree/master/Herramientas>.

## **Anexos**

# Apéndice A

## Anexos

### A.1. Ficheros java del editor

En este anexo se incluyen los métodos modificados de los ficheros auto-generados que dan la funcionalidad al editor gráfico. Por un lado, se encuentran los relacionados con el meta-modelo del *Catalogue*, son los ficheros "ParameterDefinitionImpl.java" y "ComplexTaskImpl.java", respectivamente.

```
1 /**
2 * <!-- begin-user-doc -->
3 * <!-- end-user-doc -->
4 * @generatedNOT
5 */
6 @Override
7 public void eSet(int featureID, Object newValue) {
8     switch (featureID) {
9         case CataloguePackage.PARAMETER_DEFINITION__NAME:
10             setName((String)newValue);
11             // Si se intenta cambiar el nombre del ParamDef y esta vinculado (
12             //    ↗ boundTo)
```

## A.1. FICHEROS JAVA DEL EDITOR

---

```

12     // a un parameter, poner el nombre de dicho parametro
13     if(getBoundTo()!=null)
14         setName(boundTo.getName());
15     return;
16     case CataloguePackage.PARAMETER_DEFINITION__TYPE:
17         setType((ParameterType)newValue);
18         // Si se intenta cambiar el tipo del ParamDef y esta vinculado (boundTo
19             ↔ )
20         // a un parameter, poner el tipo de dicho parametro
21         if(getBoundTo()!=null)
22             setType(boundTo.getParamDefinition().getType());
23         return;
24     case CataloguePackage.PARAMETER_DEFINITION__BOUND_TO:
25         setBoundTo((Parameter)newValue);
26         // Si se intenta cambiar el Parametrer al que esta vinculado (boundTo
27         // el ParamDef, poner como nombre y tipo del ParamDef los del
28             ↔ Parameter
29         if(getBoundTo()!=null){
30             setName(boundTo.getName());
31             setType(boundTo.getParamDefinition().getType());
32         }
33         return;
34     }

```

```

1 /**
2 * <!-- begin-user-doc -->
3 * <!-- end-user-doc -->
4 * @generatedNOT

```

### A.1. FICHEROS JAVA DEL EDITOR

```
5  */
6 @Override
7 public void eSet(int featureID, Object newValue) {
8     switch (featureID) {
9         case CataloguePackage.COMPLEX_TASK__WORKFLOW:
10            setWorkflow((Workflow)newValue);
11
12            // Si se cambia el workflow asignado a un ComplexTask poner como
13            // → nombre del ComplexTask el del workflow
14            if(getWorkflow()!=null)
15                setName(workflow.getName());
16
17            return;
18    }
19
20    super.eSet(featureID, newValue);
21 }
```

Por otro lado, se encuentran los métodos del meta-modelo *Workflow*, que aparecen en los fichero "ParameterImpl.java" y "ActivityImpl.java", respectivamente.

```
1 /**
2 * <!-- begin-user-doc -->
3 * <!-- end-user-doc -->
4 * @generatedNOT
5 */
6 @Override
7 public void eSet(int featureID, Object newValue) {
8     ActivityImpl containerActivity;
9     System.out.println("eSet ParameterImpl " + getName());
10    switch (featureID) {
11        case WorkflowPackage.PARAMETER__NAME:
12    }
```



## A.1. FICHEROS JAVA DEL EDITOR

```
13     System.out.println("eSet ParameterImpl");
14     if(getName() != null){
15
16         setParamDefinition((ParameterDefinition)null);
17         containerActivity = (ActivityImpl)eContainer();
18         containerActivity.eSet(WorkflowPackage.
19             ACTIVITY__PARAMETERS, (String)newValue);
20     }else{
21
22         setName((String)newValue);
23         // Si existe una relacion del parametro con paramDefinition se
24         // queda el nombre de ParamDef
25         ifgetParamDefinition() != null){
26
27             System.out.println("set param def");
28             containerActivity = (ActivityImpl)eContainer();
29             setName(paramDefinition.getName() + " _ "
30                 containerActivity.getName());
31         }
32     }
33     return;
34 case WorkflowPackage.PARAMETER__VALUE:
35     System.out.println("eSet ParameterImpl => Setting
36             PARAMETER__VALUE: "+ getValue() + " => " + (String
37             newValue);
38     setValue((String)newValue);
39
40     return;
41 case WorkflowPackage.PARAMETER__PARAM_DEFINITION:
42
43     System.out.println("eSet Modificar definicion");
```

## A.1. FICHEROS JAVA DEL EDITOR

```
37     if(getParamDefinition()!=null){  
38  
39         setParamDefinition((ParameterDefinition)null);  
40         containerActivity = (ActivityImpl)eContainer();  
41         containerActivity.eSet(WorkflowPackage.  
42             ACTIVITY__PARAMETERS, (ParameterDefinition)  
43             newValue);  
44     }  
45 }  
46 return;  
47 }  
48 super.eSet(featureID, newValue);  
49 }  
50  
51 /**  
52 * <!-- begin-user-doc -->  
53 * <!-- end-user-doc -->  
54 * @generatedNOT  
55 */  
56 @Override  
57 public void eUnset(int featureID) {  
58     ActivityImpl containerActivity;  
59     switch (featureID) {  
60         case WorkflowPackage.PARAMETER__NAME:  
61             setName(NAME_EDEFAULT);  
62             return;  
63         case WorkflowPackage.PARAMETER__VALUE:  
64             setParamDefinition(null);  
65             containerActivity = (ActivityImpl)eContainer();  
66             containerActivity.eSet(WorkflowPackage.  
67                 ACTIVITY__PARAMETERS, (ParameterDefinition)  
68                 newValue);  
69     }  
70 }  
71  
72 //@@generatedNOT  
73 */  
74  
75 //@@generatedNOT  
76 */  
77  
78 //@@generatedNOT  
79 */  
80  
81 //@@generatedNOT  
82 */  
83  
84 //@@generatedNOT  
85 */  
86  
87 //@@generatedNOT  
88 */  
89  
90 //@@generatedNOT  
91 */  
92  
93 //@@generatedNOT  
94 */  
95  
96 //@@generatedNOT  
97 */  
98  
99 //@@generatedNOT  
100 */
```



## A.1. FICHEROS JAVA DEL EDITOR

```
64         setValue(VALUE_EDEFAULT);
65
66     return;
67
68     case WorkflowPackage.PARAMETER__PARAM_DEFINITION:
69
70         System.out.println("eUnSet ParameterImpl");
71
72         // Se entra aqui al borrar un parametro de la actividad => Volver a
73         // cargar los parametros de la definicion
74
75         // Borra el enlace entre el param actual y su paramDefinition
76
77         setParamDefinition((ParameterDefinition)null);
78
79         containerActivity = (ActivityImpl)eContainer();
80
81         containerActivity.eSet(WorkflowPackage.
82             ACTIVITY__PARAMETERS, null);
83
84         return;
85
86     }
87
88     super.eUnset(featureID);
89
90 }
```

```
1 /**
2 * <!-- begin-user-doc -->
3 * <!-- end-user-doc -->
4 * @generatedNOT
5 */
6 @Override
7 public void eSet(int featureID, Object newValue) {
8     System.out.println("eSet ActivityImpl " + getName());
9     switch (featureID) {
10         case WorkflowPackage.ACTIVITY__NAME:
11             setName((String)newValue);
12             System.out.println("Modificando NAME");
13         return;
14     }
15 }
```



## A.1. FICHEROS JAVA DEL EDITOR

```
14
15     case WorkflowPackage.ACTIVITY__PARAMETERS:
16         System.out.println("Modificando PARAMETERS");
17         System.out.println("Intento ilegal de modificar parametros");
18         copyParameter();
19         return;
20
21     case WorkflowPackage.ACTIVITY__TASK_DEFINITION:
22         System.out.println("Modificando TASK DEFINITION ");
23         if(getTaskDefinition()!=null){
24             // se ha cambiado la taskDefinition pero se ha vuelto a poner la
25             // → misma => no hacer nada!!
26             if (getTaskDefinition().getName().equals(((TaskDefinition)
27                 → newValue).getName())) {
28                 System.out.println("Intento de cambiar la taskdefinition por ella
29                 // → misma... No se hace nada");
30                 setTaskDefinition((TaskDefinition)newValue);
31             }
32             else {
33                 // Cargar la nueva definicion y copiar sus parametros
34                 System.out.println("cambio de taskdefinition... Copiar
35                 // → parametros");
36                 setTaskDefinition((TaskDefinition)newValue);
37             }
38         } // Cuando se agrega una TaskDefinition la primera vez
```



## A.1. FICHEROS JAVA DEL EDITOR

```
39     System.out.println("Asigna nueva TASK DEFINITION " + ((  
40         ↪ TaskDefinition)newValue).getName());  
41     setTaskDefinition((TaskDefinition)newValue);  
42  
43     if(((TaskDefinition)newValue).getName() != null){ //si no hay  
44         ↪ valor no hacer nada  
45         System.out.println("Asigna nueva PARAM DEFINITION ");  
46         // Cargar la nueva definicion y copiar sus parametros  
47         copyParameter();  
48     }  
49 }  
50 super.eSet(featureID, newValue);  
51 }  
52  
53 public void copyParameter(){  
54     ParameterImpl param;  
55     if (taskDefinition.getParamsDefinitions()!=null){  
56         getParameters().clear();  
57         for(int i=0; i<taskDefinition.getParamsDefinitions().size(); i++){  
58             // Para cada paramDefinition en la nueva TaskDefinition  
59             param = new ParameterImpl();  
60             // Ponemos su nombre como el del paramDefinition  
61             param.setName(taskDefinition.getParamsDefinitions().get(i).getName  
62             ↪ () + "_" + getName());  
63             // Enlazamos el parametro con su paramDefinition  
64             param.setParamDefinition(taskDefinition.getParamsDefinitions().get(i)  
65             ↪ );
```

## A.2. FICHEROS DEL EDITOR GRÁFICO

---

```

64         getParameters().add((Parameter) param);
65
66     }
67 }
68 }
```

## A.2. Ficheros del editor gráfico

En este anexo se incluye el fichero "Catalogue.emf", en el que se define el diseño del editor gráfico del meta-modelo *Catalogue*.

```

1 @Ecore(invocationDelegates=
2   "http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot", settingDelegates="
3     ↳ http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot",
4     ↳ validationDelegates="http://www.eclipse.org/emf/2002/Ecore/OCL
5     ↳ /Pivot")
6
7 @gmf
8 @namespace(uri="http://www.unex.es/catalogue", prefix="catalogue")
9 package catalogue;
10
11 @Ecore(constraints="TaskDefinitions_must_have_unique_names")@"
12   ↳ www.eclipse.org/emf/2002/Ecore/OCL/Pivot(
13     ↳ TaskDefinitions_must_have_unique_names="self.gathers->isUnique(
14       ↳ name)")
15 @gmf.diagram(rcp="true")
16 class Catalogue {
17   attr String[1] name;
18   val TaskDefinition[*] gathers;
19 }
```

## A.2. FICHEROS DEL EDITOR GRÁFICO

```
13
14 @Ecore(constraints="ParameterDefinitions_of_a_TaskDefinition_must
    ↪ _have_unique_names")@ "http://www.eclipse.org/emf/2002/Ecore/
    ↪ OCL/Pivot"(ParameterDefinitions_of_a_TaskDefinition_must_
    ↪ have_unique_names= "self.paramsDefinitions->isUnique(name)")
15 abstract class TaskDefinition {
16     attr String[1] name;
17
18     @gmf.affixed
19     val ParameterDefinition[*] paramsDefinitions;
20 }
21
22 @gmf.node(label="name", size="130,70", border.color="0,0,0")
23 class SimpleTask extends TaskDefinition {
24 }
25
26 @gmf.node(label="name", size="170,70", color="200,200,200", border.color="
    ↪ 0,0,0")
27 class ComplexTask extends TaskDefinition {
28 }
29
30 @gmf.node(label="name", label.icon="false", size="120,30", color="
    ↪ 144,187,180", border.color="0,0,0")
31 class ParameterDefinition {
32     attr String[1] name;
33     attr ParameterType[1] type;
34 }
35
36 enum ParameterType {
```

## A.2. FICHEROS DEL EDITOR GRÁFICO

---

```

37 NUMBER = 0;
38 BOOLEAN = 1;
39 STRING = 2;
40 }
```

A continuación, se muestra el fichero "Workflow.emf", en el que se define el diseño del editor gráfico del meta-modelo *Workflow*.

```

1 @"http://www.eclipse.org/OCL/Import"(ecore="http://www.eclipse.org/emf
   ↪ /2002/Ecore#/")
2 @Ecore(invocationDelegates="http://www.eclipse.org/emf/2002/Ecore/OCL/
   ↪ Pivot",settingDelegates="http://www.eclipse.org/emf/2002/Ecore/OCL
   ↪ /Pivot",validationDelegates="http://www.eclipse.org/emf/2002/Ecore/
   ↪ OCL/Pivot")
3 @namespace(uri="http://www.unex.es/workflow", prefix="workflow")
4 @gmf
5 package workflow;
6
7 @Ecore(constraints="Workflow_have_one_and_only_one_initial_task
   ↪ Workflow_have_one_or_more_final_task")
8 @"http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot|(
   ↪ Workflow_have_one_and_only_one_initial_task="self.tasks->
   ↪ selectByType (Initial)->size()=1",
   ↪ Workflow_have_one_or_more_final_task="self.tasks->selectByType
   ↪ (Final)->size()>=1")
9 @gmf.diagram
10 class Workflow {
11   attr String[1] name;
12   @gmf.compartment(collapsible="true")
13   val Task[3..*] tasks;
14   val Link[2..*] links;
```

## A.2. FICHEROS DEL EDITOR GRÁFICO

---

```

15 }
16
17 abstract class Task {
18     ref Link#target incoming;
19     ref Link#source outgoing;
20 }
21
22 @gmf.link(source="source", target="target", target.decoration="arrow", color
23   ↪ ="0,0,0")
24 class Link {
25     ref Task[1]#incoming target;
26     ref Task[1]#outgoing source;
27 }
28 @Ecore(constraints="Activities_have_one_and_only_one_input
29   ↪ _linkActivities_have_one_and_only_one_output_link")@"
30   ↪ http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"(Activities_have_
31   ↪ one_and_only_one_input_link="self.incoming->size() = 1",
32   ↪ Activities_have_one_and_only_one_output_link="self.outgoing->
33   ↪ size() = 1")
34 @gmf.node(label="name", size="130,70", border.color="0,0,0")
35
36 class Activity extends Task {
37     attr String[1] name;
38     @gmf.affixed
39     val Parameter[*] parameters;
40 }
41
42 @Ecore(constraints="One_and_only_one_output_link")
43 @http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"

```

## A.2. FICHEROS DEL EDITOR GRÁFICO

---

```

    ↳ One_and_only_one_output_link="self.incoming->size() = 0 and self.
    ↳ outgoing->size() = 1")
38 @gmf.node(label.placement="none", label.icon="false", size="70,70", figure="
    ↳ svg", svg.uri="platform:/plugin/RoboTaskFlow_v006/images/Initial.
    ↳ svg")
39 class Initial extends Task {
40 }
41
42 @Ecore(constraints="One_and_only_one_input_link")
43 @"http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"(
    ↳ One_and_only_one_input_link="self.outgoing->size() = 0 and self.
    ↳ incoming->size() = 1")
44 @gmf.node(label.placement="none", label.icon="false", size="70,70", figure="
    ↳ svg", svg.uri="platform:/plugin/RoboTaskFlow_v006/images/Final.svg
    ↳ ")
45 class Final extends Task {
46 }
47
48 @Ecore(constraints="Loops_have_one_and_only_one_input_link
    ↳ Loops_have_one_and_only_one_output_link
    ↳ Counter_Based_Loops_must_have_numIterations_greater_than_1")
49 @"http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"(
    ↳ Loops_have_one_and_only_one_input_link="self.incoming->size()
    ↳ = 1",Loops_have_one_and_only_one_output_link="self.outgoing->
    ↳ size() = 1",
    ↳ Counter_Based_Loops_must_have_numIterations_greater_than_1=
    ↳ "self.type=LoopType::COUNTER_BASED implies numIterations>1")
50 @gmf.node(label="name, type", label.pattern="{0}: {1}", label.icon="true",
    ↳ color="236,255,244", border.color="0,0,0")

```

## A.2. FICHEROS DEL EDITOR GRÁFICO

---

```

51 class Loop extends Task, Workflow {
52     attr LoopType[1] type;
53     attr int numIterations;
54 }
55
56 @Ecore(constraints="One_and_only_one_input_link")
57 @"http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"(
    ↳ One_and_only_one_input_link="self.outgoing->size() = 0 and self.
    ↳ incoming->size() = 1")
58 @gmf.node(label="question", label.icon="true", color="#227,198,244", border.
    ↳ color="0,0,0")
59 class Question extends Task {
60     attr String[1] question;
61     @gmf.compartment(collapsible="true", layout="list")
62     val Answer[2..*] answers;
63 }
64
65 @Ecore(constraints="Answers_must_belong_to_Question
    ↳ One_and_only_one_output_link")
66 @"http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot"(
    ↳ Answers_must_belong_to_Question="self.oclContainer().oclIsTypeOf
    ↳ (Question)",One_and_only_one_output_link="self.incoming->size()
    ↳ = 0 and self.outgoing->size() = 1")
67 @gmf.node(label="answer", label.pattern="- {0}", label.icon="false", color="
    ↳ 227,198,244", border.color="0,0,0")
68 class Answer extends Task {
69     attr String[1] answer;
70 }
71

```

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

---

```

72 @gmf.node(label="name, value", label.pattern="{0}:{1}", label.icon="false",
    ↪ size="130,30", color="144,187,180", border.color="0,0,0")
73 class Parameter {
74     attr String[1] name;
75     attr String[1] value;
76 }
77
78 enum LoopType {
79     WHILE = 0;
80     DO_WHILE = 1;
81     COUNTER_BASED = 2;
82 }
```

## A.3. Manual de implementación del editor gráfico

Este manual ha sido elaborado durante la creación de la herramienta PiLHaR y contiene los pasos seguidos para llevarla a cabo de principio a fin. Comienza con la creación del proyecto en Eclipse, continua por los meta-modelos, la generación automática del editor gráfico, los enlaces posteriores entre los meta-modelos, la generación de código y, finaliza, por la ejecución en el segundo Eclipse. Se listan a continuación:

- a. Crear nuevo proyecto
  1. Presionar Ctrl + N (equivalente a *File → New*).
  2. Seleccionar "*Empty EMF Project*" y añadir como nombre ***RoboTaskFlow***.
  3. Modificar el nombre de la carpeta ***model*** a ***metamodel***.
- b. Crear meta-modelo. Inicialmente, se siguen los pasos para crear el meta-modelo del Catálogo
  1. Seleccionar la carpeta ***metamodel*** y presionar Ctrl + N.

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

---

2. Seleccionar "*Ecore model*" y añadir como nombre **Catalogue.ecore**. Este fichero contendrá el meta-modelo del catálogo.
3. Seleccionar el paquete raíz incluido en el fichero .ecore y rellenar con las siguientes propiedades:
  - Name = **catalogue**
  - Ns Prefix = **catalogue**
  - Ns URI = **http://www.RoboTaskFlow.org/catalogue**
4. Guardar y cerrar el fichero .ecore
5. Para crear una representación gráfica asociada al meta-modelo hacer clic derecho en el fichero .ecore y seleccionar la opción "*Initialize Ecore Diagram...*".
  - I. Añadir **Catalogue.aird** como nombre y hacer clic en "Next"
  - II. Elegir *Entities in a Class Diagram* y hacer clic en "Next".
  - III. Seleccionar el paquete raíz del meta-modelo y hacer clic en "Finish".
6. Usando los elementos incluidos en la paleta, se crea el meta-modelo que se puede observar en la Figura A.1. Usando el tablón de propiedades se llenan los campos *EClasses*, *EAttributes*, *EReferences*, etc., incluidos en el metamodelo.
7. Guardar el fichero .aird y cerrar.
8. Abrir el fichero **Catalogue.ecore** usando la opción "*Sample Refelctive Ecore Model Editor*".
9. Validar el meta-modelo haciendo clic derecho en el paquete raíz y seleccionar *Validate*.

A continuación, hacemos los mismos pasos seguidos anteriormente para crear el meta-modelo del flujo de actividades.

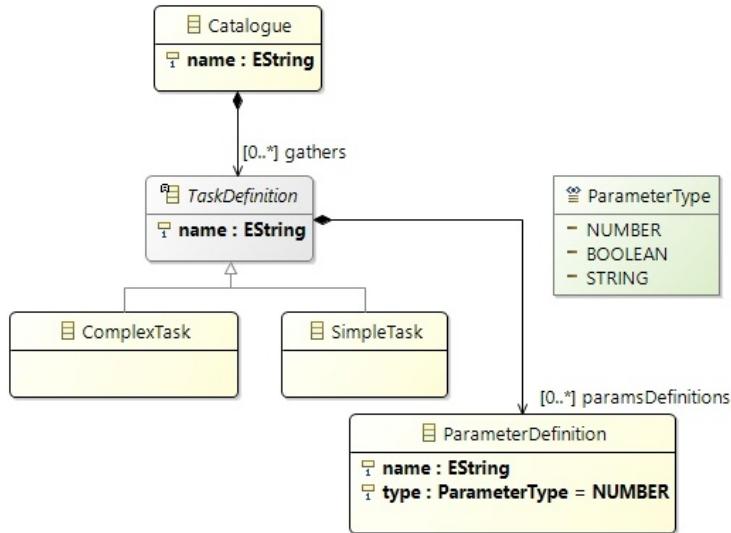


Figura A.1: Meta-metamodelo del catálogo

1. Seleccionar la carpeta ***metamodel*** y presionar Crtl + N.
2. Seleccionar "*Ecore model*" y añadir como nombre ***Workflow.ecore***. Este fichero contendrá el meta-metamodelo del flujo de actividades.
3. Seleccionar el paquete raíz incluido en el fichero .ecore y rellenar con las siguientes propiedades:
  - Name = **workflow**
  - Ns Prefix = **workflow**
  - Ns URI = **http://www.RoboTaskFlow.org/workflow**
4. Guardar y cerrar el fichero .ecore
5. Para crear una representación gráfica asociada al meta-metamodelo hacer clic derecho en el fichero .ecore y seleccionar la opción "*Initialize Ecore Diagram...*".
  - I. Añadir ***Workflow.aird*** como nombre y hacer clic en "*Next*"
  - II. Elegir *Entities in a Class Diagram* y hacer clic en "*Next*".

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

- III. Seleccionar el paquete raíz del meta-modelo y hacer clic en "Finish".
6. Usando los elementos incluidos en la paleta, se crea el meta-modelo que se puede observar en la Figura A.2. Usando el tablón de propiedades se llenan los campos *EClasses*, *EAttributes*, *Ereferences*, etc., incluidos en el metamodelo.

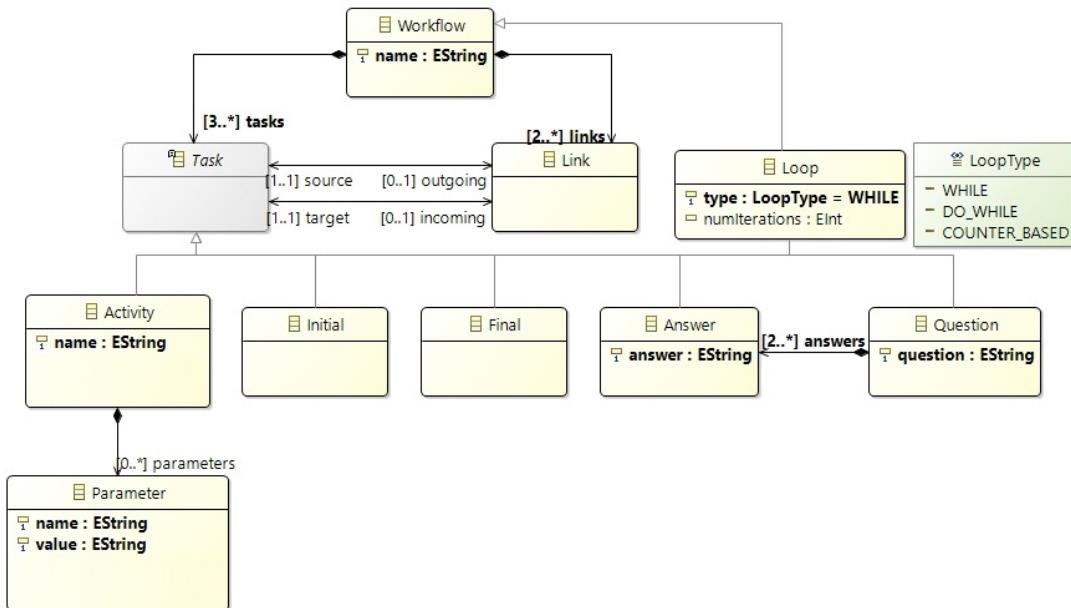


Figura A.2: Meta-modelo del flujo de actividades

7. Guardar el fichero .aird y cerrar.
8. Abrir el fichero **Workflow.ecore** usando la opción "Sample Refelctive Ecore Model Editor".
9. Validar el meta-modelo haciendo clic derecho en el paquete raíz y seleccionar *Validate*.
- c. Restricciones OCL. Para ello, es necesario abrir la representación textual de cada meta-modelo, que es equivalente al editor gráfico.
  1. Hacer clic derecho en el fichero **Catalogue.ecore** → *Open With* → *OCLInEcore Editor*.

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

---

2. Añadir las restricciones OCL correspondientes a este meta-modelo.
  3. Hacer clic derecho en el fichero **Workflow.ecore** → *Open With* → **OCLInEcore Editor**.
  4. Añadir las restricciones OCL correspondientes a este meta-modelo.
- d. Crear editor gráfico. Se explica como se desarrolla paso a paso cada editor gráfico para cada meta-modelo. En primer lugar, se realiza para el Catálogo.
1. Hacer clic derecho en el meta-modelo **Catalogue.ecore** y seleccionar la opción *"Generate Emfatic Source"*.
  2. Abrir el fichero generado **Catalogue.emf** y añadir las anotaciones de Eugenia (@gmf, @gmf.diagram, @gmf.node, etc.).
  3. Validar el meta-modelo haciendo clic derecho en el paquete raíz y seleccionar *Validate*.
  4. Hacer clic derecho en el fichero **Catalogue.emf** y seleccionar la opción *Eugenia* → *Generate GMF Editor*.
  5. Modificar el nombre de la carpeta con extensión .diagram.
    - I. Borrar el fichero **RoboTaskFlow.diagram** generado para crear otro **Catalogue.diagram** a continuación.
    - II. Abrir el fichero **Catalogue.gmfgen** y recorrer todos los paquetes modificando todos los parámetros con el nombre **"RoboTaskFlow"** por **"Catalogue"**. Por ejemplo: el parámetro con valor (*/RoboTaskFlow.diagram/src*) modificarlo por el valor (*/Catalogue.diagram/src*).
    - III. En ese mismo fichero, seleccionar el primer paquete y modificar el parámetro la extensión de los fichero por *"cat"* como se muestra en la ilustración A.3.
    - IV. Hacer clic derecho en el paquete raíz y seleccionar *"Generate diagram"*

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

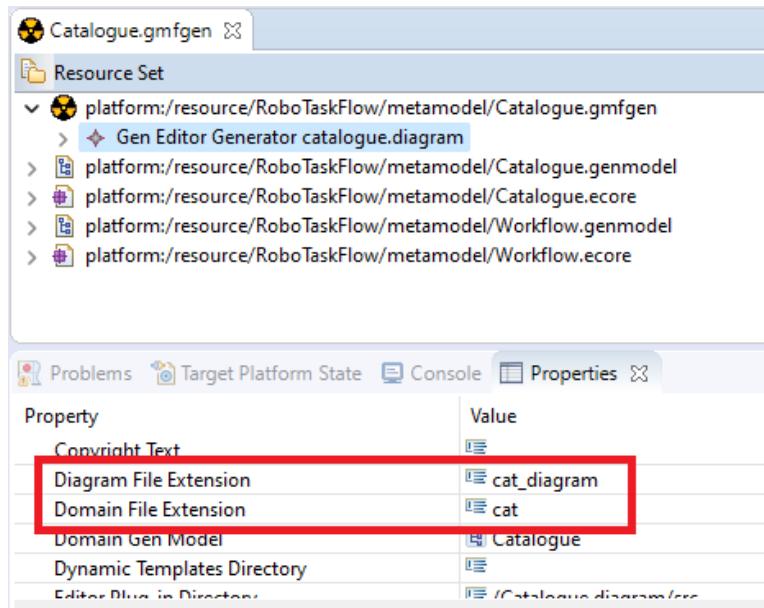


Figura A.3: Propiedades File Exetension modificadas

code”.

6. Modificar la extensión de los fichero en .genmodel.
  - I. Abrir el fichero **Catalogue.genmodel**.
  - II. Seleccionar el paquete raíz y modificar el parámetro *File Extensions* con el valor ”cat”.
  - III. Hacer clic derecho en ese paquete raíz y seleccionar ”Generate All”.

En segundo lugar, se realiza para el Flujo de actividades.

1. Hacer clic derecho en el meta-modelo **Workflow.ecore** y seleccionar la opción ”Generate Emfatic Source”.
2. Abrir el fichero generado **Workflow.emf** y añadir las anotaciones de EuGENia (@gmf, @gmf.diagram, @gmf.node, etc.).
3. Validar el meta-modelo haciendo clic derecho en el paquete raíz y seleccionar *Validate*.
4. Hacer clic derecho en el fichero **Workflow.emf** y seleccionar la opción *Eu-*

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

*genia* → *Generate GMF Editor*.

5. Modificar el nombre de la carpeta con extensión .diagram.
  - I. Borrar el fichero ***RoboTaskFlow.diagram*** generado para crear otro ***Workflow.diagram*** a continuación.
  - II. Abrir el fichero ***Workflow.gmfgen*** y recorrer todos los paquetes modificando todos los parámetros con el nombre **"RoboTaskFlow"** por **"Workflow"**. Por ejemplo: el parámetro con valor *(/RoboTaskFlow.diagram/src)* modificarlo por el valor *(/Workflow.diagram/src)*.
  - III. En ese mismo fichero, seleccionar el primer paquete y modificar la extensión de los fichero por **"wfl"** como se realizó anteriormente.
  - IV. Hacer clic derecho en el paquete raíz y seleccionar **"Generate diagram code"**.
6. Modificar la extensión de los fichero en .genmodel.
  - I. Abrir el fichero ***Workflow.genmodel***.
  - II. Seleccionar el paquete raíz y modificar el parámetro *File Extensions* con el valor **"wfl"**.
  - III. Hacer clic derecho en ese paquete raíz y seleccionar **"Generate All"**.
- e. Abrir editor gráfico.
  1. Seleccionar una o todas las carpetas del proyecto indistintamente.
  2. Hacer clic derecho y seleccionar *Run as... → Eclipse Application*

Esperamos a que se inicie el nuevo eclipse y se prueba que el editor tiene el diseño deseado, ya que aún falta por enlazar ambos meta-modelos. Para probarlo, hacemos lo siguiente:

1. Presionar Ctrl + N (equivalente a *File → New*).

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

---

2. Seleccionar *General* → *Project* y añadir como nombre ***EditorCozmo***.
3. Presionar **Ctrl + N** y seleccionar "*Catalogue Diagram*"
4. Modificar el nombre del fichero a ***Inicial.cat*** y seleccionar "*Next*".
5. Seleccionar "*Finish...*".
6. Presionar **Ctrl + N** y seleccionar "*Workflow Diagram*"
7. Modificar el nombre del fichero a ***Saludar.wfl*** y seleccionar "*Next*".
8. Seleccionar "*Finish...*".
9. Hacer modelos de ejemplo para dar el diseño deseado a los editores.

Las modificaciones en el fichero Emfatic se pueden realizar hasta este punto, ya que cuando comiencen las relaciones entre meta-modelos, no se podrá cambiar nada.

- f. Crear referencias entre meta-modelos.
  1. Cerrar el eclipse que contenía los editores gráficos.
  2. Abrir el fichero ***Catalogue.ecore***.
  3. Hacer clic derecho en cualquier parte de la pestaña recién abierta y seleccionar "*Load Resource*".
  4. Hacer clic sobre "*Browse Workspace*" y seleccionar *Workflow.ecore*.
  5. Añadir los parámetros correspondientes del meta-metamodelo *Catalogue* que se relacionan con el meta-metamodelo *Workflow*.
  6. Validar el meta-metamodelo haciendo clic derecho en el paquete raíz y seleccionar *Validate*.

Ahora, realizar los mismos pasos para el meta-metamodelo *Workflow*.

1. Abrir el fichero ***Workflow.ecore***.

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

---

2. Hacer clic derecho en cualquier parte de la pestaña recién abierta y seleccionar "*Load Resource*".
  3. Hacer clic sobre "*Browse Workspace*" y seleccionar *Catalogue.ecore*.
  4. Añadir los parámetros correspondientes del meta-modelo *Workflow* que se relacionan con el meta-modelo *Catalogue*.
  5. Validar el meta-modelo haciendo clic derecho en el paquete raíz y seleccionar *Validate*.
- g. Regenerar *catalogue.genmodel*.
1. Hacer clic derecho en el fichero ***Catalogue.genmodel*** y seleccionar "*Reload...*"
  2. Seleccionar como *Root packages*: ***catalogue*** y como *References generator models*: ***workflow***
  3. Abrir el fichero ***Catalogue.genmodel***.
  4. Seleccionar el paquete raíz y pulsar sobre "*Generate All*".
- Ahora, realizar los mismos pasos para el meta-modelo *Workflow*.
1. Hacer clic derecho en el fichero ***Workflow.genmodel*** y seleccionar "*Reload...*"
  2. Seleccionar como *Root packages*: ***workflow*** y como *References generator models*: ***catalogue***
  3. Abrir el fichero ***Workflow.genmodel***.
  4. Seleccionar el paquete raíz y pulsar sobre "*Generate All*".
- h. Añadir nuevas restricciones OCL. Este paso se vuelve a repetir, ya que ahora se han añadido nuevos parámetros sobre los que se aplican restricciones OCL.
1. Hacer clic derecho en el fichero ***Catalogue.ecore*** → *Open With* → *OCL-Editor*

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

---

*nEcore Editor.*

2. Añadir las restricciones OCL correspondientes a este meta-modelo.
3. Hacer clic derecho en el fichero **Workflow.ecore** → *Open With* → **OCLI-nEcore Editor**.
4. Añadir las restricciones OCL correspondientes a este meta-modelo.
- i. Modificar métodos auto-generados.
  1. Abrir la siguiente ruta *RoboTaskFlow* → *src* → *catalogue* → *impl*.
  2. Abrir los ficheros *"ParameterDefinitionImpl.java"* y *"ComplexTaskImpl.java"* y modificar los métodos *eSet* para crear el comportamiento deseado en los editores.
  3. Modificar la etiqueta **@generated** de los métodos *eSet* por **@generated-NOT**.
  4. Abrir la siguiente ruta *RoboTaskFlow* → *src* → *workflow* → *impl*.
  5. Abrir los ficheros *"ParameterImpl.java"* y *"ActivityImpl.java"* y modificar los métodos *eSet* para crear el comportamiento deseado en los editores.
  6. Modificar la etiqueta **@generated** de los métodos *eSet* por **@generated-NOT**.
- j. Modificar paleta del editor.
  1. Abrir fichero **Workflow.gmftool**.
  2. Abrir la ruta *Tool Registry* → *Palette* → *Tool Group Object*.
  3. Borrar *"Creation tool parameter"*.

A continuación, hay que comprobar que el fichero, que contiene las relaciones entre nodos, no haya sido modificado.

1. Abrir fichero **Workflow.gmfmap**.

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

2. Comprobar que el siguiente elemento seleccionado en la Figura A.4 tiene el parámetro *Tool* de la paleta a nulo.

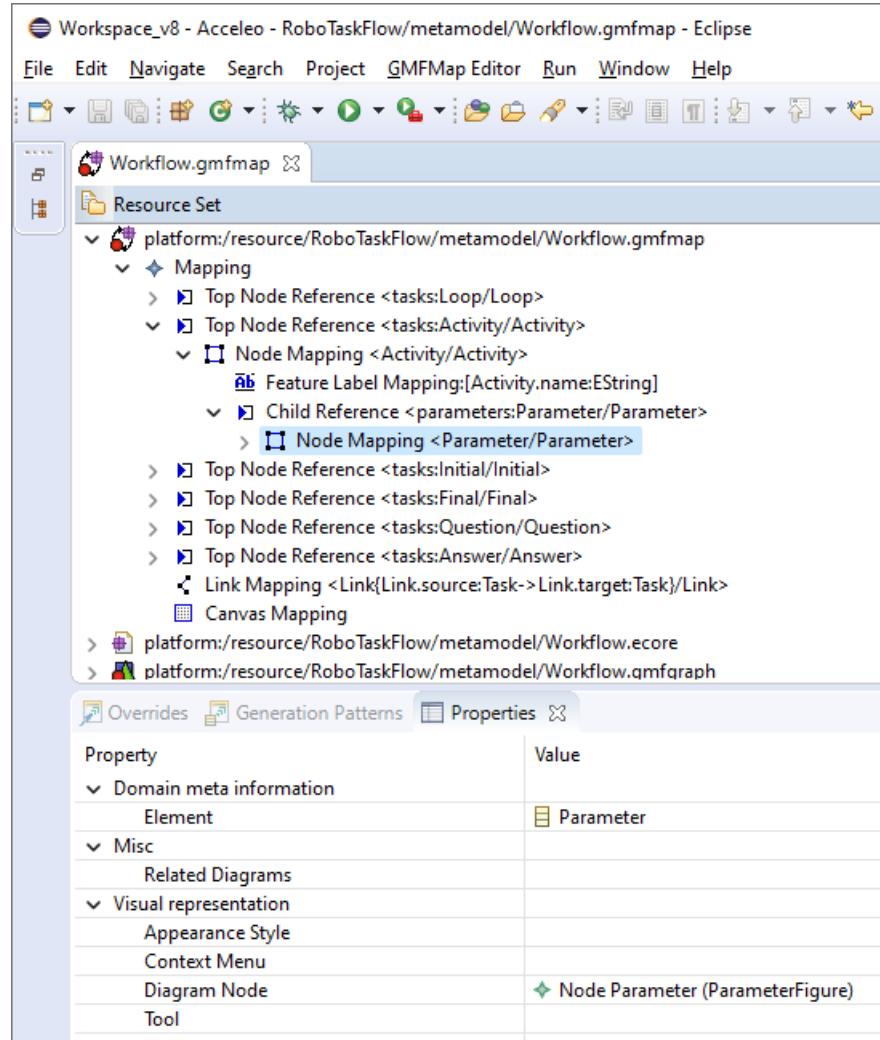


Figura A.4: Representación del modelo con el fichero gmfmap

3. Comprobar que el resto de *Node Mapping <>* tienen los tres parámetros relacionados con su misma clase como en el ejemplo de la siguiente Figura A.5.
4. Hacer clic derecho sobre el paquete **Workflow.gmfmap** y seleccionar la opción "*Create generator model*".
5. Hacer clic derecho sobre el paquete **Workflow.gmfgen** y seleccionar la opción "*Generate diagram code*".

### A.3. MANUAL DE IMPLEMENTACIÓN DEL EDITOR GRÁFICO

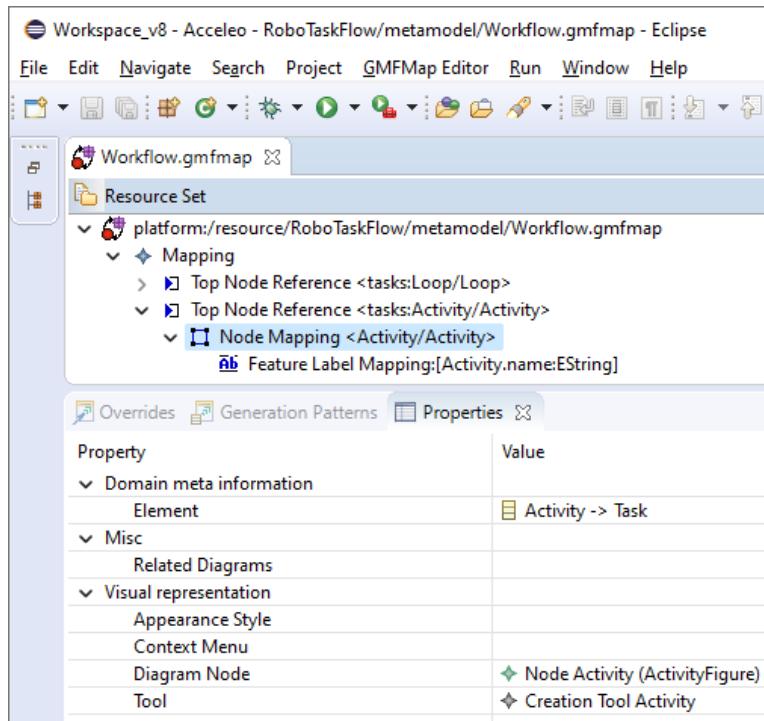


Figura A.5: Propiedades del nodo *Activity* del fichero gmfmap

k. Crear generador con Acceleo.

1. Presionar Ctrl + N y seleccionar "*Acceleo Project*".
2. Modificar el nombre a "...*workflow2cozmo*" y seleccionar "*Next*".
3. Añadir uri del meta-modelo flujo de actividades: + → Seleccionar "*Runtime Verison*" → Buscar "\*wor" → Seleccionar <http://www.RoboTaskFlow.org/workflow>.
4. Añadir uri del meta-modelo catalogo: + → Seleccionar "*Runtime Verison*" → Buscar "\*cat" → Seleccionar <http://www.RoboTaskFlow.org/catalogue>.
5. Añadir el parámetro *Template Name: generateCozmo*.
6. Seleccionar el tipo del parámetro que va a tener el método generar *Type: Workflow*.
7. Seleccionar "*Generate file*" y "*Main template*".

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

8. Editar el fichero **generate.mtl** para generar el fichero deseado.
1. Crear segundo Acceleo para el segundo eclipse.
  1. Seleccionar la capeta del proyecto de *Acceleo* → *New* → *Acceleo UI Launcher Project*
  2. Seleccionar "Next" → Seleccionar todos los archivos → "Next" → Añadir como nombre del generador: "...workflow2cozmoUI" y de la carpeta donde se almacenará (TareasComplejas) → "Finish".
- m. Ejecutar en el segundo eclipse.
  1. Seleccionar una o todas las carpetas del proyecto indistintamente.
  2. Hacer clic derecho y seleccionar *Run as...* → *Eclipse Application*
  3. Crear modelos de catalogo y flujo de actividades.
  4. Cuando un flujo de actividades ya esté creado y validado, hacer clic derecho en el modelo (no en el diagrama) y seleccionar *Acceleo Model to Text* → *Generate workflow2cozmoUI*

## A.4. Ficheros de generación de código con Acceleo

En este anexo se incluyen los ficheros relacionados con el generador de código. Inicialmente, se presenta el fichero "generate.mtl", que contiene método principal de generador.

```
1 [comment encoding = UTF-8 /]  
2 [module generate('http://www.RoboTaskFlow.org/workflow', 'http://www.  
    ↪ RoboTaskFlow.org/catalogue')]  
3  
4 [import org::eclipse::acceleo::module::workflow2cozmo::common::  
    ↪ generateProgram/]
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

5
6 [template public generateFiles(aWorkflow : Workflow)]
7 [comment @main/]
8 [if(not aWorkflow.ocIsTypeOf(Loop))][comment hace el fichero del workflow
   ↗ principal/]
9 [comment Fichero que contiene el codigo para el robot/]
10 [file (aWorkflow.name+'.py', false, 'UTF-8')]
11 [comment imports de todos las activities/]
12 [generateProgram(aWorkflow)/]
13 [/file]
14
15 [comment Fichero que lanza el programa al robot/]
16 [file ('EjecutarCozmo.bat', false, 'UTF-8')]
17 @echo off
18
19 echo Mandando [aWorkflow.name/] a Cozmo
20
21 python [aWorkflow.name/].py
22 [/file]
23 [/if]
24 [/template]
```

El fichero "generateProgram.mtl", genera el código del programa que ejecuta Cozmo.

```

1 [comment encoding = UTF-8 /]
2 [module generateProgram('http://www.RoboTaskFlow.org/workflow', 'http://'
   ↗ www.RoboTaskFlow.org/catalogue')]
3
4 [import org::eclipse::acceleo::module::workflow2cozmo::common::
   ↗ generateParameters/]
5 [import org::eclipse::acceleo::module::workflow2cozmo::common::
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

    ↵ generateParametersInterfaz/]

6 [import org::eclipse::acceleo::module::workflow2cozmo::common::generateTarea/]
7 [import org::eclipse::acceleo::module::workflow2cozmo::common::generateMain/]
8
9 [template public generateProgram(aWorkflow : Workflow)]
10 [comment Import general/]
11 import pycozmo
12 import time
13
14 [comment Imports de las tareas simples y las complejas que tenga el workflow/]
15 import TareasGenericas as mycozmo
16 [for (aTask:Task | aWorkflow.tasks)]
    [if(aTask.oclIsTypeOf(Activity))]
        [if(aTask.oclAsType(Activity).taskDefinition.oclIsTypeOf(
            ↵ ComplexTask))]

19 from [aTask.oclAsType(Activity).taskDefinition.oclAsType(ComplexTask).name
    ↵ /] import *
20
    [/if]
21     [elseif(aTask.oclIsTypeOf(Loop))]
22         [for (aTaskLoop:Task | aTask.oclAsType(Loop).tasks)]
23             [if(aTaskLoop.oclIsTypeOf(Activity))]
24                 [if(aTaskLoop.oclAsType(Activity).taskDefinition.
                    ↵ oclIsTypeOf(ComplexTask))]

25 from [aTaskLoop.oclAsType(Activity).taskDefinition.oclAsType(ComplexTask).
    ↵ name/] import *
26
    [/if][/if]
27     [/for]
28     [/if]
29 [/for]
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

```
30  
31 [comment Import para la interfaz/]  
32 import tkinter as tk  
33  
34 [comment Llamada del metodo que contiene todo el flujo de actividades/]  
35 def [aWorkflow.name.toLowerCase()]/(cli[aWorkflow.tasks.generateParameters()  
    ↪ /]):  
36 [comment Llamada a las tareas del flujo/]  
37 [aWorkflow.tasks.generateMetodos()/]  
38  
39 [comment Main del programa/]  
40 [generateMain(aWorkflow)]  
41 [/template]  
42  
43 [template public generateMetodos(aTask : Task)]  
44 [/template]  
45  
46 [template public generateMetodos(aInitial : Initial)]  
47 [comment Genera la siguiente tarea a la inicial. Es el comienzo del codigo.]/  
48 [generateTarea(aInitial.outgoing.target)]  
49 [/template]  
50  
51 [template public generateMetodos(aLoop : Loop)]  
52 [if(not (aLoop.type = LoopType::COUNTER_BASED))]  
53 [comment Metodo que llama a las tareas de dentro del bucle. Es llamado en la  
    ↪ interfaz/]  
54 def do[aLoop.name/]/([generateParametersLoop(aLoop)/]):  
55     [aLoop.tasks.generateMetodos()]  
56 [/if]
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

57 [/template]
58
59 [template public generateMetodos(aQuestion : Question)]
60 [comment Metodo que llama a las tareas de cada respuesta en las preguntas. Es
    ↪ llamado en la interfaz/]
61 [for(anAnswer:Answer | aQuestion.answers)]
62 def action[anAnswer.answer/][i/](ventanaQ[generateParametersInterfaz(
    ↪ anAnswer.outgoing.target)/]):
63 [generateTarea(anAnswer.outgoing.target)/]
64     ventanaQ.quit()
65
66 [/for]
67 [/template]

```

El fichero "generateParameters.mtl", crea las asignaciones a los parámetros de los métodos.

```

1 [comment encoding = UTF-8 /]
2 [module generateParameters( 'http://www.RoboTaskFlow.org/workflow', 'http
    ↪ ://www.RoboTaskFlow.org/catalogue')]
3
4 [template public generateParameters(aTask : Task)]
5 [/template]
6
7 [template public generateParameters(aActivity : Activity)]
8 [comment Asignacion de parametros/]
9 [for (aParam:Parameter | aActivity.parameters)before (', ') separator(', ')]
10 [if(aParam.paramDefinition.type = ParameterType::STRING)]
11 [aParam.name/] = "[aParam.value/]"
12 [elseif(aParam.paramDefinition.type = ParameterType::BOOLEAN)]
13 [if(aParam.value = 'Verdadero')]

```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

14 [aParam.name/] = True
15           [elseif(aParam.value = 'Falso')]
16 [aParam.name/] = False
17           [else]
18 [aParam.name/] = [aParam.value/]
19           [/if]
20           [else]
21 [aParam.name/] = [aParam.value/]
22           [/if]
23 [/for]
24
25 [/template]
26
27 [template public generateParameters(aLoop : Loop)]
28 [comment Recorre todas las tareas del bucle/]
29 [aLoop.tasks.generateParameters()/]
30 [/template]
```

El fichero "generateTarea.mtl", según el tipo de tarea llama al *template* correspondiente.

```

1 [comment encoding = UTF-8 /]
2 [module generateTarea('http://www.RoboTaskFlow.org/workflow', 'http://
   ↪ www.RoboTaskFlow.org/catalogue')]
3
4 [import org::eclipse::acceleo::module::workflow2cozmo::common::
   ↪ generateParametersInterfaz/]
5
6 [template public generateTarea(aTask : Task)]
7 [/template]
8
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

9 [template public generateTarea(anActivity : Activity)]
10   '-----[anActivity.name/] - [anActivity.taskDefinition.name.
11     ↪ toUpperCase()]/-----'
12 [comment Código de la llamada a la tarea/]
13 [if(anActivity.taskDefinition.oclIsTypeOf(ComplexTask))]
14   [anActivity.taskDefinition.name.toLowerCase()]/(cli[for (aParam:Parameter
15     ↪ | anActivity.parameters)before(' ',' ') separator(' ',' ')][aParam.
16     ↪ paramDefinition.name/]=[aParam.name/][/for]) [comment tarea
17     ↪ compleja/]
18
19 [else]
20   mycozmo.[anActivity.taskDefinition.name.toLowerCase()]/(cli[for (aParam:
21     ↪ Parameter | anActivity.parameters)before(' ',' ') separator(' ',' ' ) ][
22     ↪ aParam.name/][/for]) [comment tarea simple/]
23
24 [/if]
25 [generateTarea(anActivity.outgoing.target)/]
26 [/template]
27
28 [template public generateTarea(aQuestion : Question)]
29   '-----[aQuestion.question/] - QUESTION
30     ↪ -----'
31 [comment Crear interfaz/]
32   text_label = "[aQuestion.question/]"
33   mycozmo.hablar(cli, text_label)
34
35   ventanaQ=tk.Tk()
36   #set center screen window with following coordination
37   MyLeftPos = (ventanaQ.winfo_screenwidth() - 400) / 2

```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

31     myTopPos = (ventanaQ.winfo_screenheight() - 400) / 2
32     ventanaQ.geometry( "%dx%d+%d+%d" % (400, 400, MyLeftPos,
33                           ↪ myTopPos))
34
35
36     lbl = tk.Label(ventanaQ, text=text_label, font=("Arial Bold", 14))
37     lbl.pack(pady=15)
38
39 [comment Crear un boton por cada respuesta/]
40 [for(anAnswer:Answer | aQuestion.answers)]
41     answer[i/] = "[anAnswer.answer/]"
42     boton[i/]=tk.Button(ventanaQ, text=answer[i/], bg="#82c9dd", width =
43                           ↪ 9, font = ("Bold"),
44                           command=lambda: action[anAnswer.answer/][i/](ventanaQ[
45                           ↪ generateParametersInterfaz(anAnswer.outgoing.target)/]))
46
47     boton[i/].pack(pady=10)
48
49 [/for]
50
51 mycozmo.hablar(cli, [for(anAnswer:Answer | aQuestion.answers) separator(
52                           ↪ ' + " o " + ')answer[i/]/[for]])
53
54     ventanaQ.mainloop()
55
56 [/template]
57
58 [template public generateTarea(aLoop : Loop)]
59     '-----[aLoop.name/] - [aLoop.type/]-----
60     ↪   ''''
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

55 [comment WHILE o DO_WHILE: asigna variables para la interfaz/]
56
57 [comment WHILE/]
58 [if(aLoop.type = LoopType::WHILE)]
59     text_label="Hacemos la actividad [aLoop.name/]?""
60     mycozmo.hablar(cli, text_label)
61 [comment DO_WHILE: hace la primera iteracion y luego pregunta/]
62 [elseif(aLoop.oclAsType(Loop).type = LoopType::DO_WHILE)]
63     [for (aTask:Task | aLoop.tasks)]
64         [if(aTask.oclIsTypeOf(Initial))]
65 [generateTarea(aTask.outgoing.target)/]
66         [/if]
67     [/for]
68     text_label="Repetimos [aLoop.name/]?""
69     mycozmo.hablar(cli, text_label)
70 [/if]
71
72 [comment COUNTER_BASED o los otros tipos/]
73 [comment COUNTER_BASED: crea un for con el numero de iteraciones
   ↪ indicadas/]
74 [if(aLoop.type = LoopType::COUNTER_BASED)]
75     for i in range([aLoop.numIterations/]):
76         [for (aTask:Task | aLoop.tasks)]
77             [if(aTask.oclIsTypeOf(Initial))]
78 [generateTarea(aTask.outgoing.target)/]
79             [/if]
80         [/for]
81 [else] [comment WHILE o DO_WHILE: crea la interfaz/]
82     ventana=tk.Tk()

```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

83     #set center screen window with following coordination
84     MyLeftPos = (ventana.winfo_screenwidth() - 400) / 2
85     myTopPos = (ventana.winfo_screenheight() - 200) / 2
86     ventana.geometry( "%dx%d+%d+%d" % (400, 200, MyLeftPos,
87                           ↪ myTopPos))
88
89
90     lbl = tk.Label(ventana, text=text_label, font=("Arial Bold", 14))
91     lbl.pack(pady=15)
92
93     boton1=tk.Button(ventana, text="SI", bg="#6ce238", width = 9, font =
94                           ↪ ("Bold"),
95                           command=lambda: do[aLoop.name]/([generateParametersLoop(
96                           ↪ aLoop)/]))
95     boton1.pack(side=tk.LEFT, padx=45, pady=20)
96     boton2=tk.Button(ventana, text="NO", bg="#ff5649", width = 9, font =
97                           ↪ ("Bold"),
97                           command=ventana.quit)
98     boton2.pack(side=tk.RIGHT, padx=45, pady=20)
99     ventana.mainloop()
100
101 [/if]
102 [generateTarea(aLoop.outgoing.target)/]
103 [/template]
```

El fichero "generateParametersInterfaz.mtl", hace las llamadas a los parámetros para asignarselos al método correspondiente.

```

1 [comment encoding = UTF-8 /]
2 [module generateParametersInterfaz('http://www.RoboTaskFlow.org/workflow
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

---

```

    ↵ ', 'http://www.RoboTaskFlow.org/catalogue')]

3

4 [template public generateParametersInterfaz(aTask : Task)]
5 [comment Argumentos para los parametros de los metodos creados para una
   ↵ interfaz/]
6 [/template]

7

8 [template public generateParametersInterfaz(aLoop : Loop)]
9 [comment Recorre desde el inicio el workflow del loop/]
10 [for (aTaskLoop:Task | aLoop.tasks)]
11     [if(aTaskLoop.oclIsTypeOf(Initial))]
12 [generateParametersInterfaz(aTaskLoop.outgoing.target)/][/if]
13 [/for]
14 [/template]

15

16 [template public generateParametersInterfaz(anActivity : Activity)]
17 [for (aParam:Parameter | anActivity.parameters) before (', ') separator(', ')]
18 [aParam.name/][/for]
19 [generateParametersInterfaz(anActivity.outgoing.target)/]
20 [/template]

21

22 [template public generateParametersInterfaz(aQuestion : Question)]
23 [aQuestion.answers.outgoing.target.generateParametersInterfaz()/]
24 [/template]

25

26 [template public generateParametersLoop(aLoop : Loop)]
27 [comment Recorre desde el inicio el workflow del loop/]
28 [for (aTaskLoop:Task | aLoop.tasks)]
29     [if(aTaskLoop.oclIsTypeOf(Initial))]
```

#### A.4. FICHEROS DE GENERACIÓN DE CÓDIGO CON ACCELEO

```

30      [if (aTaskLoop.outgoing.target.oclIsTypeOf(Activity))]
31          [comment Inserta el conjunto de parametros sin la
            ↪ primera coma/]
32          [for (aParam:Parameter | aTaskLoop.outgoing.target.
            ↪ oclAsType(Activity).parameters) separator( ', '
            ↪ )]
33  [aParam.name/][/for]
34  [comment Llama recursivamente a las siguientes tareas/]
35  [generateParametersInterfaz(aTaskLoop.outgoing.target.outgoing.target)/][/if]
36      [/if]
37  [/for]
38 [/template]
```

El fichero "generateMain.mtl", hace la conexión con Cozmo y llama al método del flujo de actividades.

```

1 [comment encoding = UTF-8 /]
2 [module generateMain( 'http://www.RoboTaskFlow.org/workflow', 'http://
            ↪ www.RoboTaskFlow.org/catalogue')]

3
4 [template public generateMain(aWorkflow : Workflow)]
5 ' '----- MAIN -----' '
6 if __name__ == '__main__':
7     [comment Comenzar conexion/]
8     ' '-----Comenzar conexion-----' '
9     cli = pycozmo.Client()
10    cli.start()
11    cli.connect()
12    cli.wait_for_robot()
13
14    [comment Levantar cabeza/]
```



## A.5. MANUAL DE USUARIO

```
15     ' '-----Levantar cabeza----- '
16     angle = (pycozmo.robot.MAX_HEAD_ANGLE.radians - pycozmo.robot.
17             ↪ MIN_HEAD_ANGLE.radians) / 2.0
18     cli.set_head_angle(angle)
19
20     [comment Llamada a metodo/]
21     ' '-----Llamada a metodo inicial----- '
22     [aWorkflow.name.toLowerCase()]/(cli)
23
24     [comment Finalizar conexion/]
25     ' '-----Finalizar conexion----- '
26     cli.set_head_angle(pycozmo.robot.MIN_HEAD_ANGLE.radians)
27     time.sleep(1)
28
29     cli.disconnect()
30     cli.stop()
31 [/template]
```

## A.5. Manual de usuario

En este manual, se añade toda la información necesaria para que el usuario pueda seguir paso a paso las indicaciones y crear flujo de actividades a partir de tareas simples, tareas complejas, tareas complejas con parámetros o flujo de actividades a partir de tareas complejas.

### Creación de flujo de actividades a partir de tareas simples y generación de código

En este apartado, se detallan los pasos necesarios para crear un flujo de actividades a partir de tareas simples y ejecutar este en Cozmo.

- a. Creación de los ficheros. Inicialmente, se comienza con el diseño del flujo de actividades y para ello, se crea un nuevo diagrama.
  1. Hacer clic derecho en la carpeta "FlujoActividad", donde se desea insertar el diagrama y seguir la ruta *File → New → Other...*
  2. En el buscador, insertar "diag" y seleccionar *Examples → Workflow Diagram → Next.*
  3. Insertar un nombre, por ejemplo "Saludar" y siempre con la extensión ".wfl\_diagram".
  4. Seleccionar *Next.*
  5. Seleccionar la carpeta "modelos", que es donde se va a almacenar el fichero "Saludar.wfl" y pulsar *Finish.*
  6. Una vez abierto el diagrama, ir la ventana de propiedades y poner como nombre "Saludar".
- b. Inserción de objetos de la paleta.
  1. Crear un ejemplo de diagrama del flujo de actividades.
    - I. En la paleta, hacer clic sobre "Initial" y pulsar sobre el lugar donde se quiera colocar de la ventana de edición.
    - II. Hacer lo mismo con el objeto "Final" y "Activity" de la paleta.
    - III. En la tarea "Activity", añadir el nombre de la actividad que va a realizar Cozmo, por ejemplo "SubirBrazos".

## A.5. MANUAL DE USUARIO

---

- iv. Realizar este apartado de nuevo para otra "Activity" con nombre "Presentarse" (ver Figura A.6).

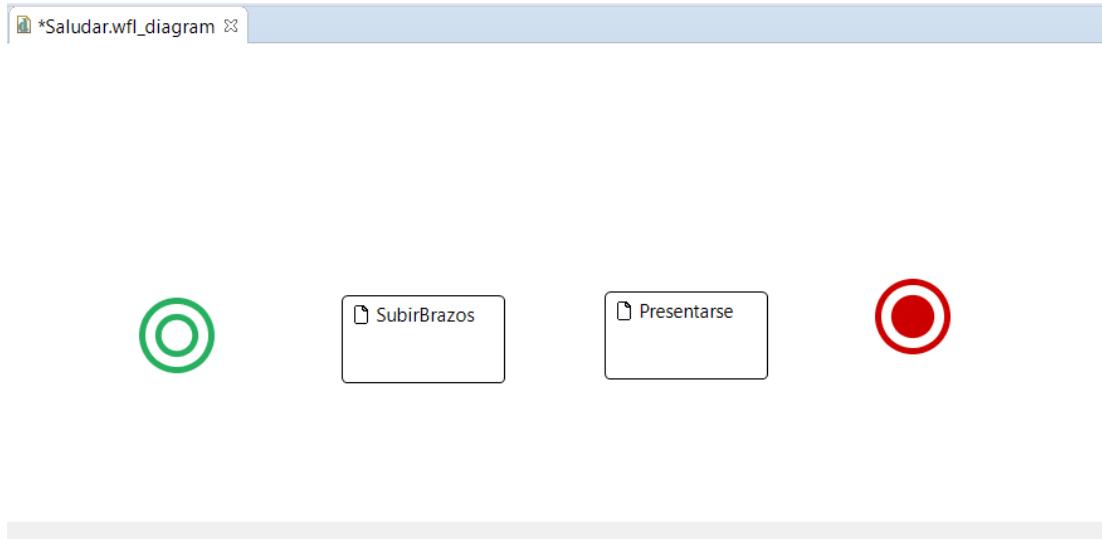


Figura A.6: Ejemplo de flujo de actividades sin acabar

2. Cargar el catalogo "Inicial" en este editor para poder crear relaciones entre ambos.
  - I. En cualquier parte de la ventana de edición, donde no haya ningún objeto, hacer clic derecho y seleccionar la opción "Load Resource".
  - II. Pulsar "Browse Workspace..." y seleccionar la ruta *Catalogue → Inicial.cat\_diagram* y pulsar *OK*.
  - III. Seleccionar *OK* de nuevo.
3. Asignar las propiedades a las tareas, en este caso, solo a las actividades.
  - I. En la ventana de las propiedades, rellenar "taskDefinition" con la definición de tarea que le pertenezca, en este caso, "Simple Task MoverBrazos". Y en la tarea "Presentarse", seleccionar "Simple Task Hacer" (ver Figura A.7).
  - II. Como se ha podido observar, han aparecido los parámetros que le pertenecen a cada definición de tarea. Ahora, le damos valores a estos

## A.5. MANUAL DE USUARIO

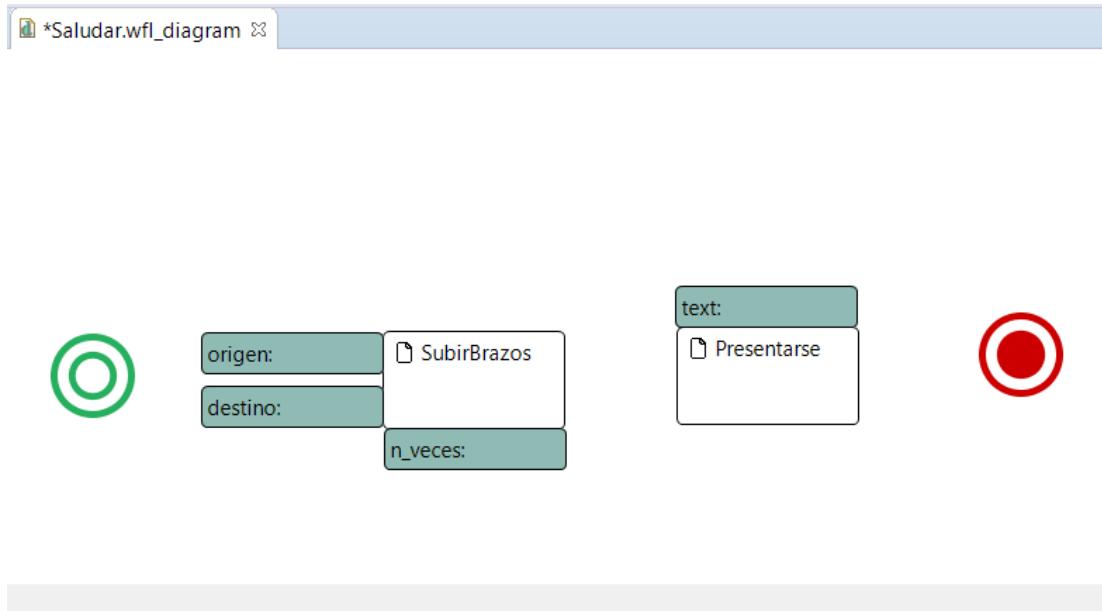


Figura A.7: Actividades con parámetros por definir

parámetros según las opciones indicadas en el manual A.6 (ver Figura A.8).

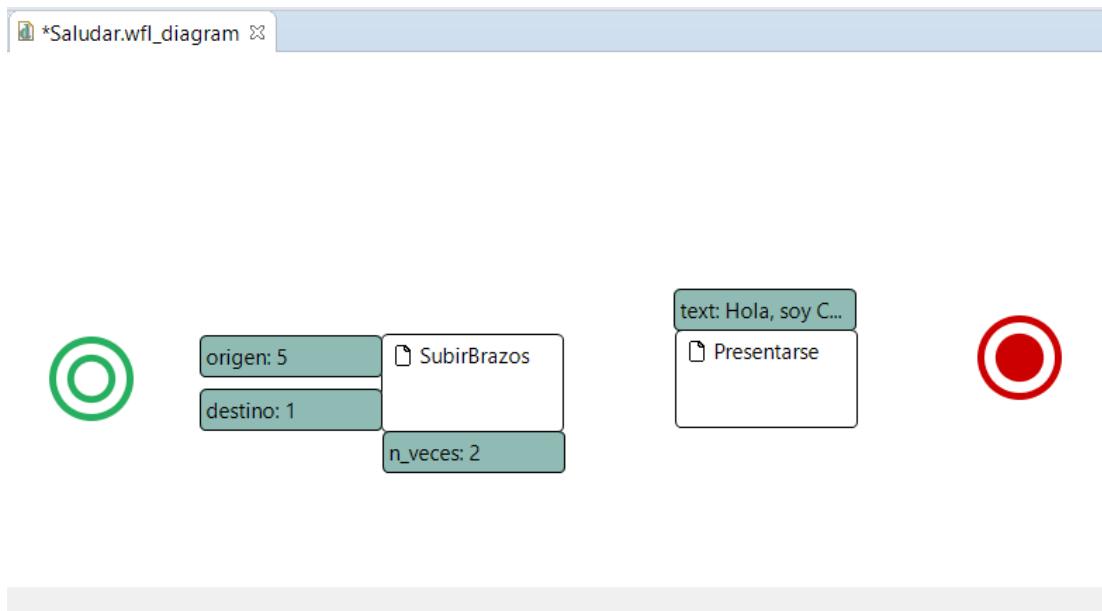


Figura A.8: Ejemplo de valores en las actividades

4. Unir las tareas en el orden en el que se quieren reproducir.

- I. Seleccionar el conector "link".

## A.5. MANUAL DE USUARIO

- II. Dejar pulsada la tarea que se desea que sea origen y soltar en la tarea destino.
- III. Hacer esto mismo con cada unión entre tareas. El resultado se puede observar en la Figura A.9.

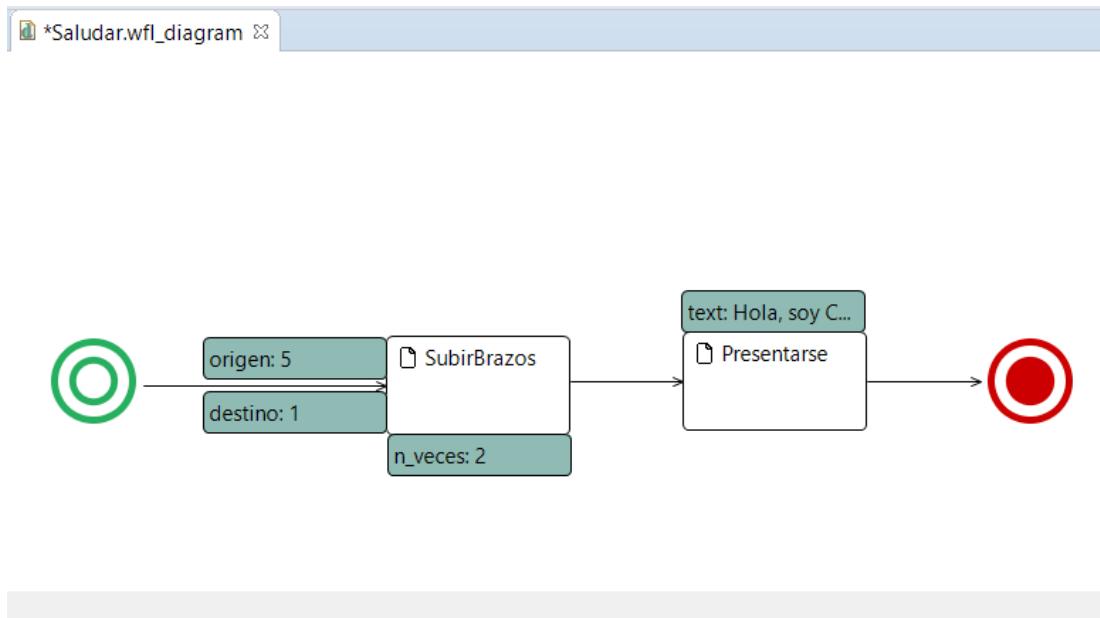


Figura A.9: Ejemplo flujo de actividad

- 5. Guardar a través de la barra de herramientas con la ruta *File → Save* o pulsando en el teclado *ctrl + S*.
- 6. Validar que el diagrama no tiene ningún error en el diseño. Mediante la barra de herramientas, seleccionar *Edit → Validate*. Nota: Si es correcto, no se mostrará nada. Si hay algún fallo, se mostrará una cruz en un círculo rojo y pasando el ratón sobre él indica el error.
- c. Generación de código y ejecución en Cozmo.
  - 1. Se va a generar el código que Cozmo va a realizar.
    - I. Seleccionar el fichero que contenga el flujo de actividades del que se desee generar el código dentro de la carpeta "Modelos", en este caso, el fichero es "Saludar.cat".

## A.5. MANUAL DE USUARIO

- II. Hacer clic derecho y seleccionar la opción "Acceleo Model to Text" y, a continuación, "Generate workflow2cozmoui"(ver Figura A.10).

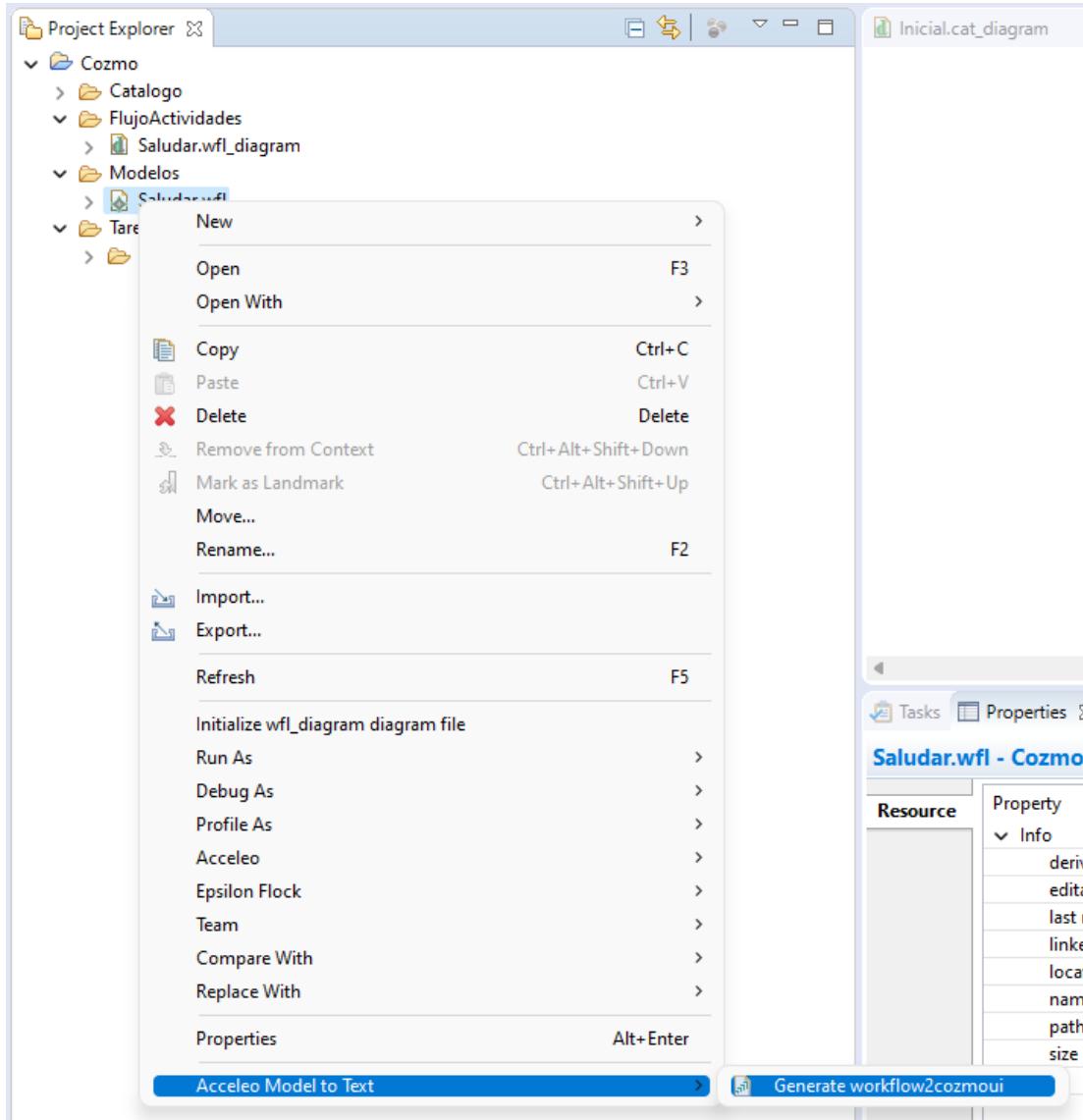


Figura A.10: Generación de código

2. Para ejecutar el programa, hacer doble clic en el fichero "EjecutarCozmo.bat"

## Creación de tareas complejas

En este apartado, se explica cómo una tarea compleja es añadida al catálogo.

## A.5. MANUAL DE USUARIO

- a. Editar el diseño del catálogo con una tarea compleja.
  1. En la paleta, hacer clic sobre "ComplexTask" y pulsar sobre el lugar, donde se quiera colocar, de la ventana de edición.
  2. Añadir el nombre la tarea compleja que se desee, por ejemplo "Saludar" (ver Figura A.11).

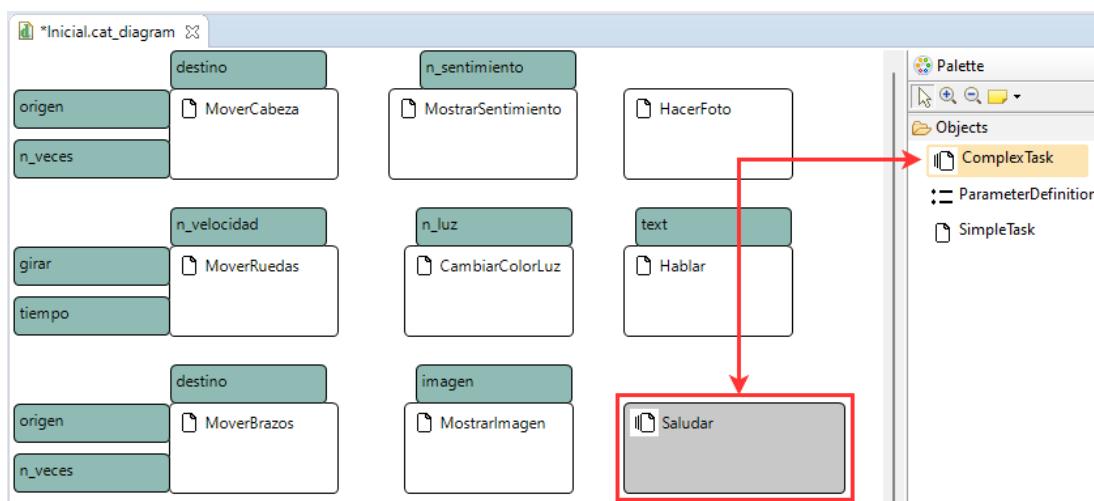


Figura A.11: Creación tarea simple

- b. Cargar el flujo de actividades "Saludar" en este editor para poder crear relaciones entre ambos.
  1. En cualquier parte de la ventana de edición, donde no haya ningún objeto, hacer clic derecho y seleccionar la opción "Load Resource".
  2. Pulsar "Browse Workspace..." y seleccionar la ruta *FlujoActividades* → *Saludar.wfl\_diagram* y pulsar *OK*.
  3. Seleccionar *OK* de nuevo.
- c. Asignar las propiedades
  1. Seleccionar la tarea compleja "Saludar" y en la ventana de las propiedades, llenar "workflow" con el flujo de actividad que le pertenezca, en este caso, "Workflow Saludar"(ver Figura A.12).

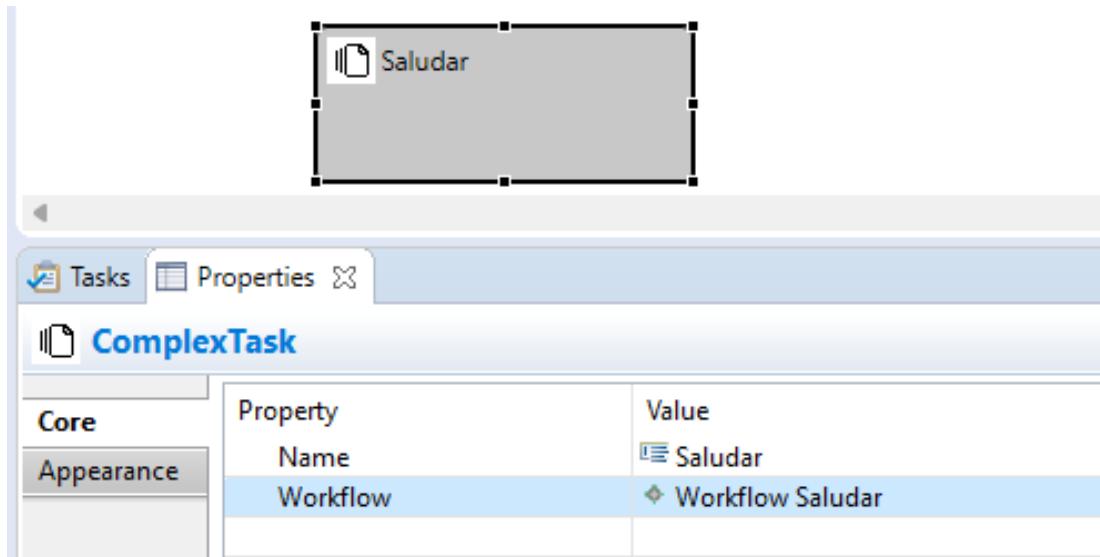


Figura A.12: Propiedades de la tarea compleja

### Tarea compleja con parámetro

Si se desea que el flujo de actividades relacionado con la tarea compleja cambie alguno de sus parámetros, se deben seguir los siguientes pasos.

- Realizar todos los pasos del apartado anterior.
- En la paleta se hace clic sobre "ParamDefinition" y se selecciona la tarea compleja creada.
- Añadir un nombre representativo para saberlo identificar y en la ventana de propiedades, asignar a "boundTo" el parámetro con el que se desea relacionar, en este caso "n\_veces" y "Parameter n\_veces", respectivamente. Se puede observar la relación en la Figura A.13.

### Creación de flujo de actividades a partir de tareas complejas

Un flujo de actividades con esta configuración se realiza como en la Sección A.5 a excepción del apartado 3. de **Inserción de objetos de la paleta**, que se cambia por el siguiente.

## A.6. MANUAL RÁPIDO DE USUARIO

---

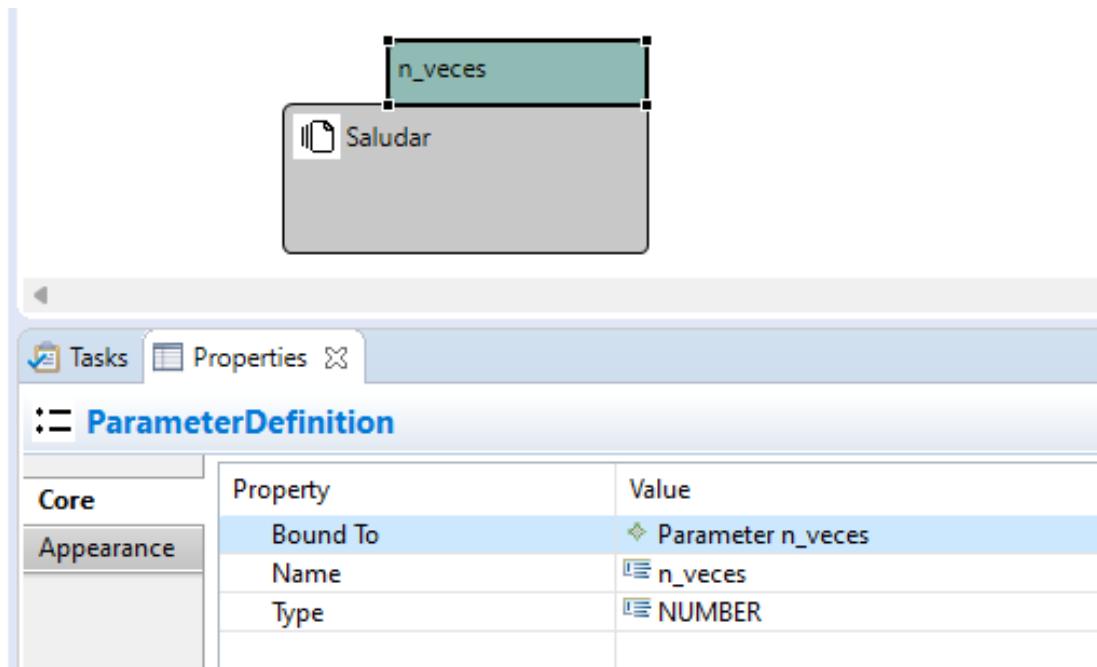


Figura A.13: Propiedades del parámetro de la tarea compleja

- I. En la ventana de las propiedades, llenar "taskDefinition" con la definición de tarea que le pertenezca, en este caso, "Complex Task Saludar" (ver Figura A.14). Y el resto de tareas como se ha hecho anteriormente (ver Figura A.15).
- II. Como se ha podido observar, ha aparecido el parámetro que se le ha definido en el catálogo en el apartado anterior. Ahora, se le asigna el valor según las opciones indicadas en el manual A.6 (ver Figura A.16).

El resultado del diagrama se puede observar en la Figura A.17. La generación de código y ejecución funcionan de la misma manera que en la Sección A.5.

## A.6. Manual rápido de usuario

En este anexo se listan los métodos disponibles en la librería implementada y a su vez, se listan sus parámetros con sus posibles valores (con la forma [valor]: [significado]).

- hablar(text)

## A.6. MANUAL RÁPIDO DE USUARIO

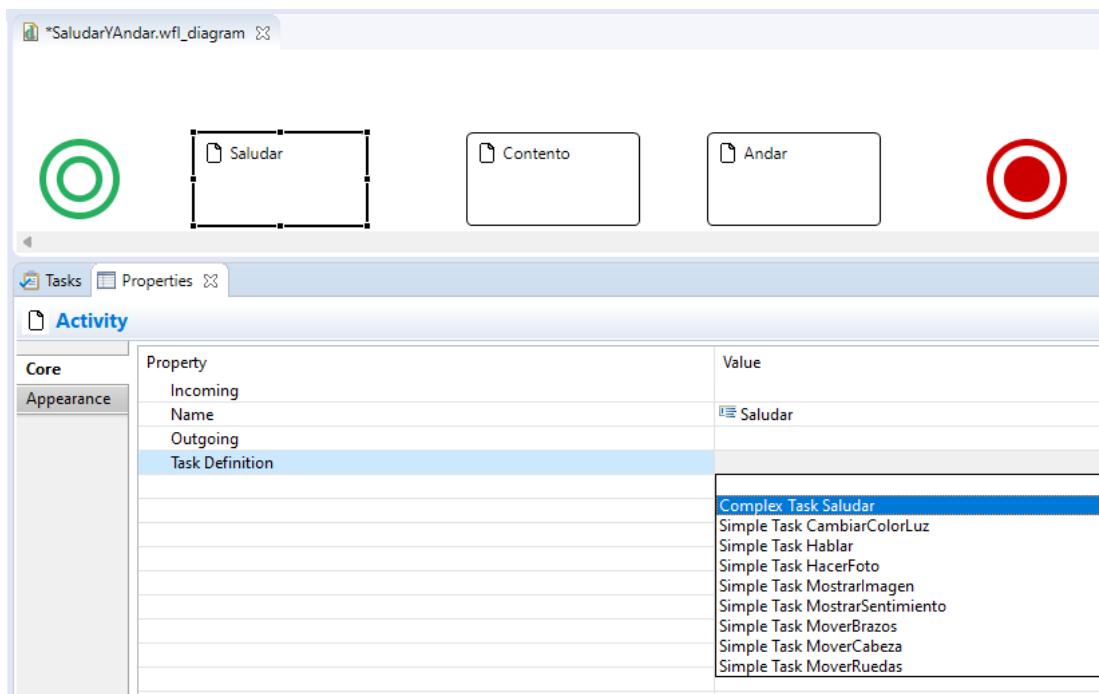


Figura A.14: Definición de "Activity" como tarea compleja

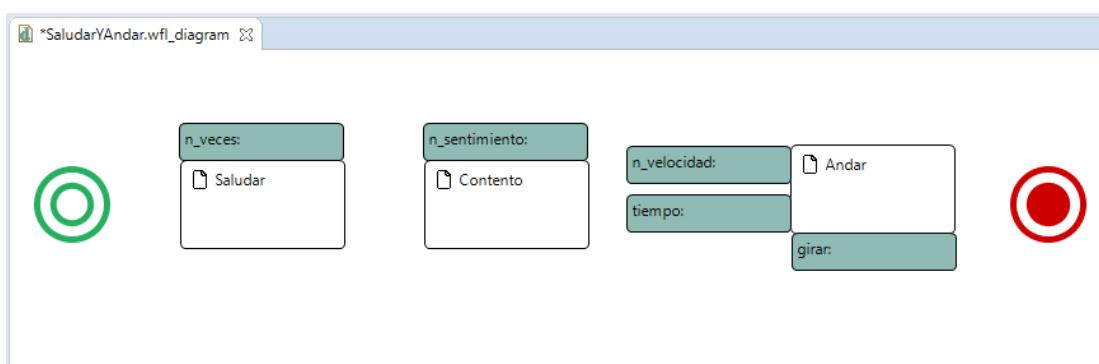


Figura A.15: Resultado de la asignación de definición de tareas

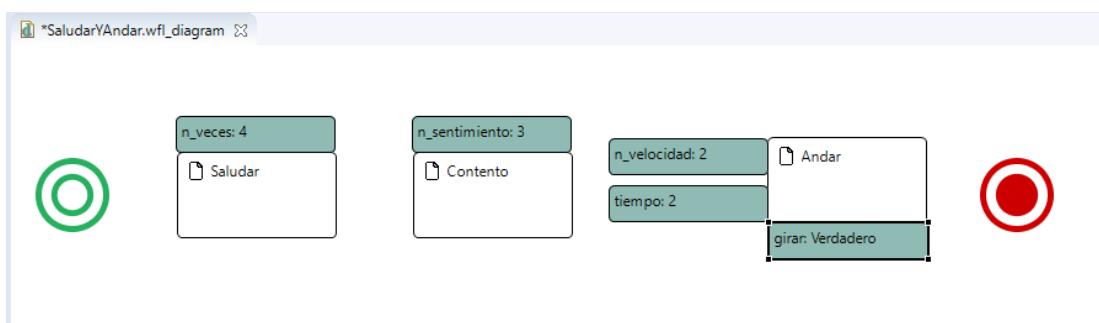


Figura A.16: Ejemplo de valores en las actividades

## A.6. MANUAL RÁPIDO DE USUARIO

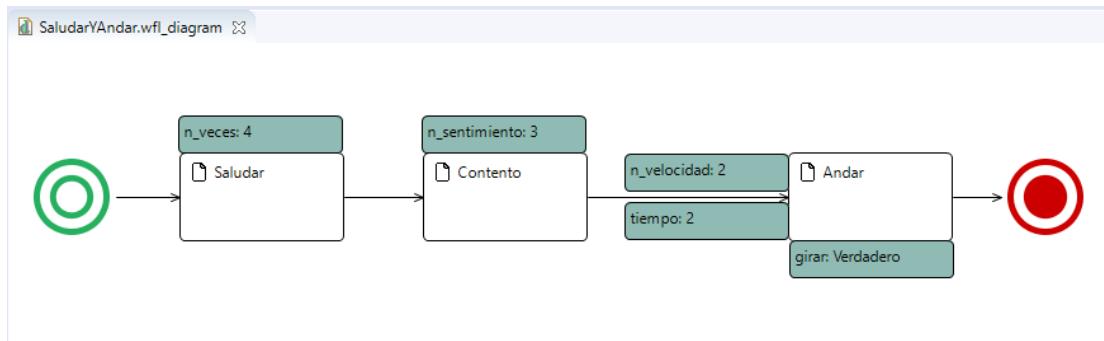


Figura A.17: Flujo de actividad con tarea compleja

- text
- moverBrazos(origen, destino, n\_veces)
  - origen y destino:
  - 1: Arriba
  - 2: Medio Alto
  - 3: Medio
  - 4: Medio Bajo
  - 5: Bajo
- n\_veces: desde 1 hasta 99.
- moverRuedas(n\_velocidad, tiempo, girar)
  - n\_velocidad:
    - 1: Rápido
    - 2: Medio
    - 3: Lento
    - 4: Hacia atrás
  - tiempo: se mide en segundo. Desde 1 hasta 99.

## A.6. MANUAL RÁPIDO DE USUARIO

---

- girar:
  - True o Verdadero
  - False o Falso
- moverCabeza (origen, destino, n\_veces)
  - origen y destino:
    - 1: Arriba
    - 2: Medio
    - 3: Bajo
  - n\_veces: desde 1 hasta 99.
- mostrarSentimiento(n\_sentimiento)
  - n\_sentimiento: la Figura A.18 muestra visualmente la cara de Cozmo con cada una de sus emociones.

0: Neutral	10: Culpable
1: Enfadado	11: Decepcionado
2: Triste	12: Vergonzoso
3: Contento	13: Horrorizado
4: Sorprendido	14: Escéptico
5: Disgustado	15: Molesto
6: Miedoso	16: Furioso
7: Suplicando	17: Sospechoso
8: Vulnerable	18: Rechazo
9: Desesperado	19: Aburrido

## A.6. MANUAL RÁPIDO DE USUARIO

20: Cansado

23: Asombrado

21: Con sueño

24: Emocionado

22: Confuso

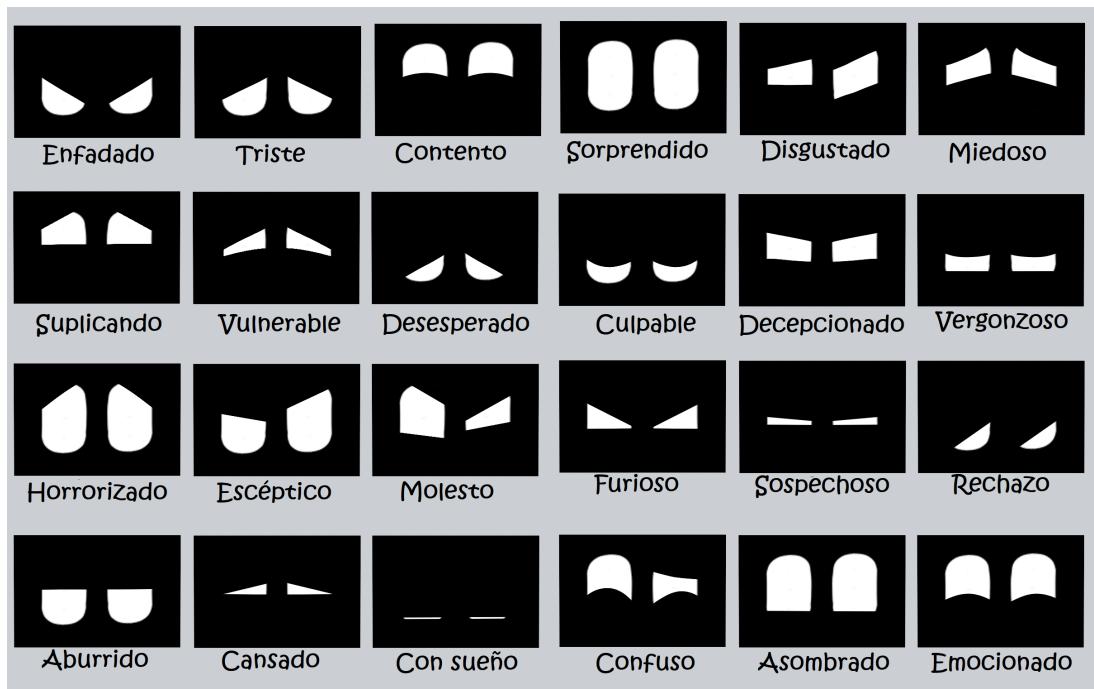


Figura A.18: Expresión de Cozmo con sus ojos

- hacerFoto: da como resultado una imagen con el nombre "camera.png".
- mostrarImagen (imagen)
  - imagen: hay que añadir el nombre de alguna de las imágenes que se encuentran en la carpeta "Recursos".
- cambiarColorLuz (n\_luz)

0: Red

1: Green

2: Blue

3: White

4: Off

## A.7. Manual del desarrollador

Este manual, detalla los pasos, que debe seguir el desarrollador, para crear un editor de Cozmo desde cero.

### Estructura del proyecto

Este apartado contiene el proceso de la creación de un nuevo proyecto y la estructura con las carpetas necesarias.

1. Inicialmente, se crea un proyecto para insertar todo el contenido perteneciente a Cozmo:
  - 1.1 En la barra de herramientas, se sigue la ruta *File* → *New* → *Other...*
  - 1.2 En el buscador, insertar "pro" y seleccionar *General* → *Project* → *Next*.
  - 1.3 Insertar un nombre, por ejemplo "EjemploCozmo".
2. A continuación, se crean las carpetas que formarán la estructura del proyecto:
  - 2.1 Se hace clic derecho sobre el proyecto creado y se sigue la ruta *File* → *New* → *Other...*
  - 2.2 En el buscador, insertar "fol" y seleccionar *General* → *Folder* → *Next*.
  - 2.3 Insertar un nombre, por ejemplo "Catalogo".
  - 2.4 Repetir este paso para crear cada carpeta. El resultado de la creación de carpetas se puede observar en la Figura A.19.
3. Para acabar con la estructura del proyecto se debe copiar la carpeta "TareasComplejas" en "EditorCozmo". El resultado de la estructura del proyecto se puede observar en la Figura A.20.

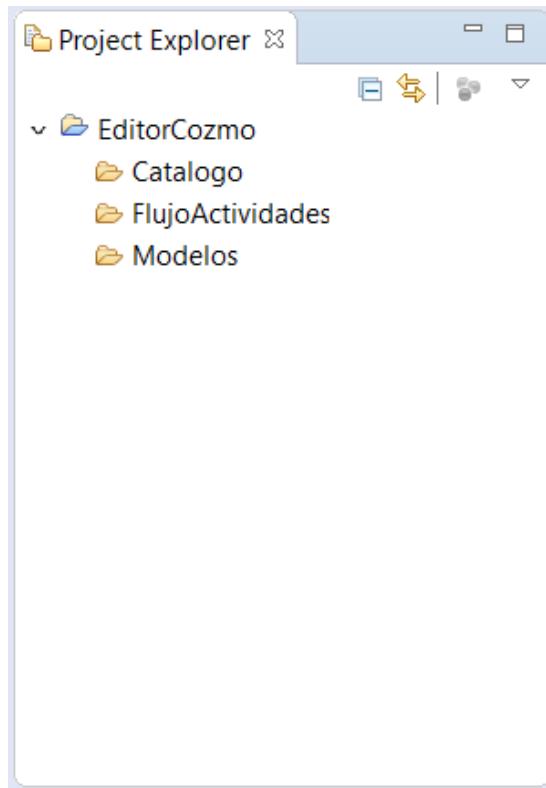


Figura A.19: Estructura de la creación de carpetas

## Creación del catálogo

En este apartado, se explica la creación de un nuevo catálogo y la inserción de tareas.

1. Ahora, vamos a empezar con el diseño del catálogo y para ello, creamos un nuevo diagrama.
  - 1.1 Hacer clic derecho en la carpeta "Catalogo", donde se desea insertar el diagrama y seguir la ruta *File → New → Other...*
  - 1.2 En el buscador, insertar "diag" y seleccionar *Examples → Catalogue Diagram → Next.*
  - 1.3 Insertar un nombre, por ejemplo "Inicial" y siempre con la extensión ".cat\_diagram".
  - 1.4 Seleccionar *Next* y a continuación, *Finish.*

## A.7. MANUAL DEL DESARROLLADOR

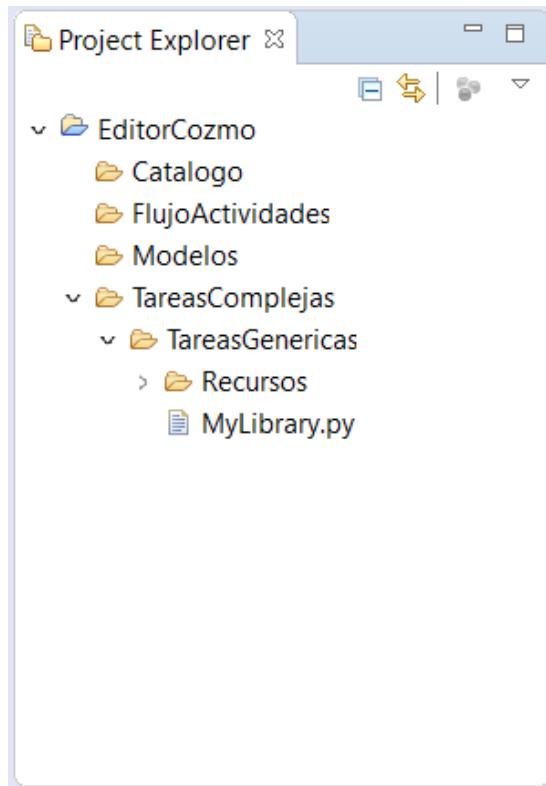


Figura A.20: Estructura final

- 1.5 Una vez abierto el diagrama, ir la ventana de propiedades y poner como nombre "Inicial".
2. Después, hay que crear la representación gráfica del catálogo
  - 2.1 En la paleta, hacer clic sobre "SimpleTask" y pulsar sobre el lugar, donde se quiera colocar, de la ventana de edición.
  - 2.2 Añadir el nombre del método de la librería que se quiera relacionar con la tarea simple, por ejemplo "Hablar" (ver Figura A.21). Nota: La primera letra puede ir tanto en minúsculas como en mayúsculas, el resto debe ser igual.
  - 2.3 Si el método no tiene parámetros, se habría acabado. Si tiene, en la paleta se hace clic sobre "ParamDefinition" y se selecciona la tarea simple creada.
  - 2.4 Añadir como nombre el mismo que tiene el parámetro de la librería, y en la

## A.7. MANUAL DEL DESARROLLADOR

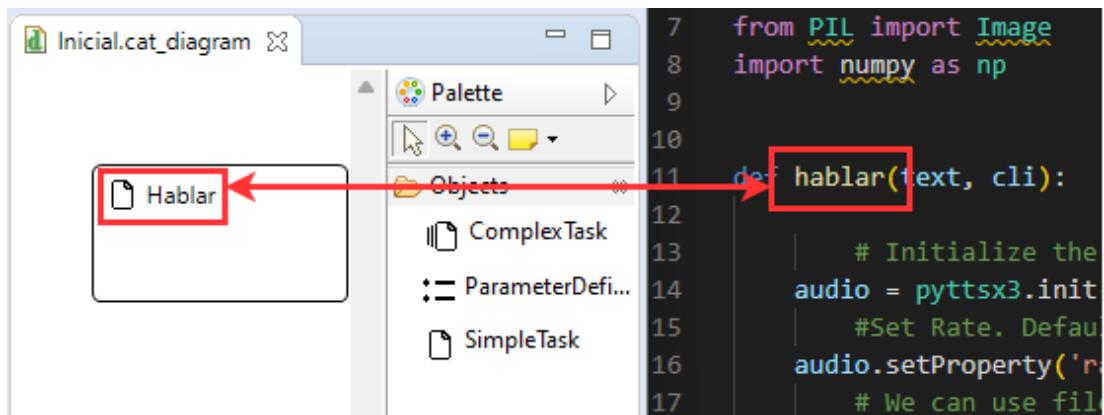


Figura A.21: Creación tarea simple

ventana de propiedades, seleccionar el tipo que le corresponde al parámetro, en este caso "text" y "STRING", respectivamente. Se puede observar la relación en la Figura A.22.

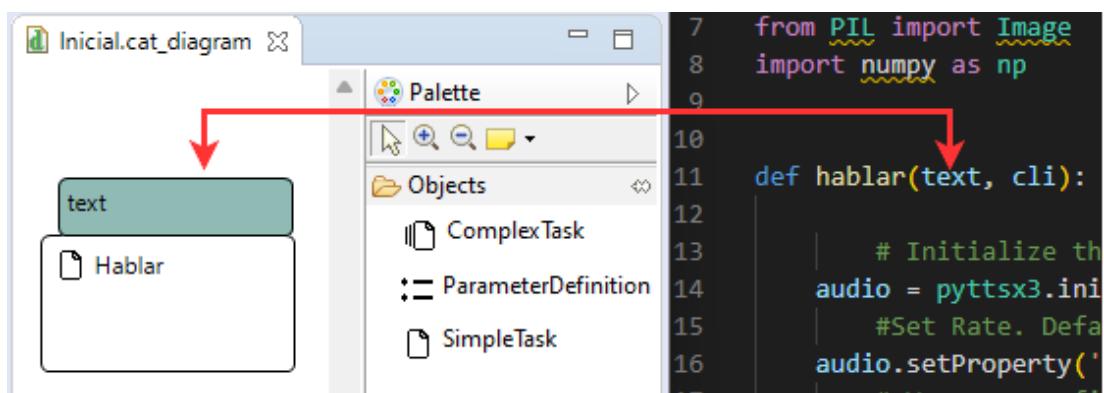


Figura A.22: Relación entre parámetros

3. Repetir este apartado para cada método de la librería. En la fecha en la que se ha terminado este documento, el catálogo al completo se puede observar en la Figura A.23.

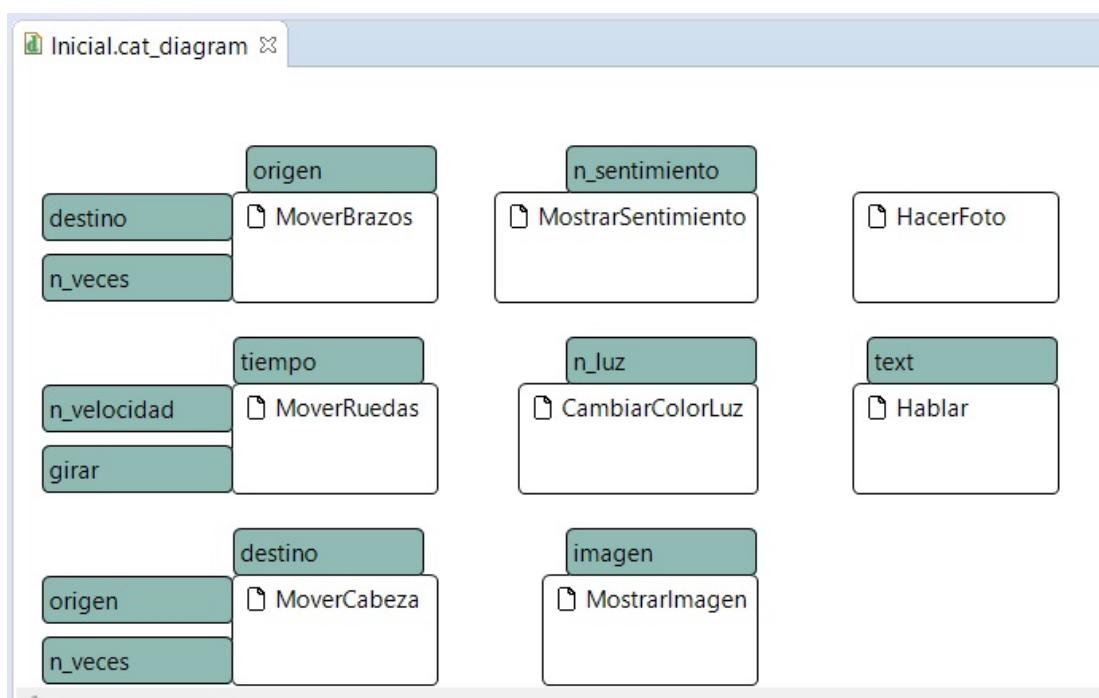


Figura A.23: Catálogo completo