



UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

Desarrollo Dirigido por Modelos en
Simulaciones de Inteligencia Ambiental

D. Pablo Campillo Sánchez

2017

Desarrollo Dirigido por Modelos en Simulaciones de Inteligencia Ambiental

Memoria que presenta para optar al título de Doctor en Informática

Pablo Campillo Sánchez

Dirigida por los doctores:

D. Juan Antonio Botía Blaya

D. Jorge J. Gómez Sanz

**Departamento de Ingeniería de la Información y las
Comunicaciones**

Facultad de Informática

Universidad de Murcia

Murcia, Enero de 2017

A mis padres

Agradecimientos

Quiero empezar los agradecimientos por mis padres, Juan y María, aquellos a los que le dedico la tesis. Esta supone el resultado y el reflejo de su continuo apoyo. Tengo mucha suerte tanto de la educación recibida por ellos en casa como de la universitaria, la cual os supuso un gran esfuerzo económico. Mil gracias por ser, en gran parte, responsables de la persona que soy hoy en día.

Otra persona clave en este trabajo es uno de mis directores, Juan Botía. Él me brindó la oportunidad de iniciarme en el mundo de la investigación, como técnico de apoyo, en varios proyectos. Desde entonces ha sido un placer trabajar con él. Gracias por transmitirme la pasión por este trabajo y tu profesionalidad, por tu apoyo, por compartir tus ideas y conocimiento conmigo, por ayudarme a ordenar mis ideas en mi cabeza, por diferenciar la fontanería de la investigación que tanto me tira. Gracias por todo, incluso por los tirones de orejas.

En aquellos inicios en proyectos de investigación también me siento afortunado de todos los que fueron mis compañeros en la sala 2.6 (Paco I, Paco II, Mayte, Alberto, Jesús, Vicky, el Tuenti, Andrés M., Andrés, Marwan, Carmen, Ramón, Tony, Ángel, Dani, Fran y Perico). Especial agradecimiento a Emilio con el que realicé mi primer JCR.

Un agradecimiento muy especial a Paula. Mi relación con ella y con la investigación se iniciaron prácticamente de forma simultánea. Mil gracias por tu amor y apoyo constante, por aguantarme las borderías cuando me invaden los estreses, por tu comprensión y sacrificio en muchos fines de semana y algunos veranos donde me aboqué al trabajo y resté tiempo a estar juntos. Gracias por estar siempre ahí, darme tu cariño y ayudarme en todo lo que puedes.

Muchas gracias a mi otro director, Jorge Gómez, el cual fue la gota que colmó el vaso para iniciar este trabajo a través de una beca predoctoral FPI. Gracias por brindarme la oportunidad de pertenecer, conocer y colaborar con el grupo GRASIA, son un grupo estupendo. Gracias por compartir tu conocimiento e ideas conmigo (¡cuántas herramientas nuevas!).

También quisiera agradecer a Virginia Dignum por acogerme y brindarme la oportunidad de trabajar en su departamento en TU Delft (Holanda) durante mi estancia de 3 meses. Nunca olvidaré la cariñosa acogida recibida por parte de todos los miembros del grupo, en especial la de mis compañeras de despacho, Karla y Alexia.

Por último, agradecer la financiación recibida por el Ministerio de Economía y Competitividad para la realización de este trabajo de tesis a través de los proyectos

“Social Ambient Assisted Living” (SociAAL, TIN2011-28335-C02-01/02) y “Desarrollo colaborativo de soluciones AAL” (ColosAAL, TIN2014-57028-R).

Resumen

La *Inteligencia Ambiental* es un paradigma que persigue crear entornos donde se ofrecen servicios a los usuarios. En dichos entornos los dispositivos están perfectamente integrados y los servicios se adaptan y son proactivos. En los últimos años, este paradigma se ha aplicado ampliamente como solución para la asistencia de personas mayores en sus hogares con el fin de mejorar su independencia y calidad de vida. Sin embargo, el desarrollo e investigación de este tipo de sistemas es costoso. Entre los costes más importantes se encuentran la contratación del personal, la compra de los dispositivos, el tiempo necesario para las pruebas o el alquiler de los espacios. Estos costes se deben a que muchas de las actividades de investigación es necesario realizarlas en entornos reales tales como los *Laboratorios Vivientes (Living Labs)*.

Una alternativa a estos laboratorios es la simulación, donde los objetos físicos se simulan: el espacio, los dispositivos y los usuarios. Sin embargo, el desarrollo de simulaciones 3D convincentes que reproduzcan las condiciones del laboratorio también son costosas. Se requiere de amplios conocimientos tanto de programación especializada como de creación de modelos 3D. En esta tesis se propone aplicar el desarrollo dirigido por modelos para facilitar la generación de las simulaciones. Esta propuesta requiere de un lenguaje, un generador de código y un marco de dominio. La idea consiste en que un generador permita crear simulaciones del marco de dominio de forma automática a partir de las especificaciones creadas con el lenguaje. El término automático significa que, tras la generación, el código fuente es completo y directamente ejecutable sin necesidad de modificarlo. De esta forma, se persigue que todo aquel capaz de modelar con el lenguaje pueda generar simulaciones 3D.

El lenguaje de modelado específico de dominio se ha llamado *SociAALML*. Inicialmente, el dominio se centra en la vida diaria de los enfermos de párkinson y sus familiares en sus hogares. Aunque este lenguaje también se puede aplicar a otros dominios. Por ejemplo, en el proyecto ColosAAL se aplica para tratar el caso de enfermedades como el Alzheimer. El lenguaje permite modelar todos los elementos de la simulación: elementos del espacio, comportamiento de los sujetos, enfermedad, interacciones, escenarios y despliegue de los dispositivos. El diseño y creación del lenguaje se ha basado en el estudio del dominio a partir de bibliografía y transcripciones realizadas a enfermos por miembros del proyecto. Además, la notación del lenguaje se ha procurado que sea cognitivamente eficiente siguiendo los principios de Moody, es decir, que sea fácil de entender y utilizar. Con esta característica de la notación, se busca ampliar el espectro

de usuarios del lenguaje no sólo a expertos sino también a trabajadores sociales, a médicos o incluso a los propios enfermos o familiares.

El marco de dominio diseñado corresponde al simulador *PHAT*. Este se ha diseñado y creado desde cero a partir del motor de juegos jME3 para que permita recrear los escenarios de la vida diaria de los enfermos de párkinson y sus familiares en sus hogares. Incluye funcionalidades como, por ejemplo, un motor físico para simular las caídas de los enfermos, iluminación, navegación de los sujetos virtuales por el escenario, sonidos, dispositivos virtuales, simulación de los síntomas de los enfermos, entre muchas otras. Todas estas funcionalidades las extiende el generador para implementar la semántica de las especificaciones creadas con SociAALML.

En esta tesis se describe el estudio realizado para generar estas herramientas y demostrar su eficacia aplicadas a dos casos de estudio. En el primero se modela la vida diaria de un enfermo de párkinson y la de su compañera a lo largo de una semana basándose en la transcripción de la entrevista. En el segundo caso, el modelado se centra en una actividad concreta, la de ver la televisión, y demuestra cómo el simulador se puede emplear para diseñar y validar una aplicación que trata de asistir al paciente.

Tanto el lenguaje como el simulador y el generador son totalmente abiertos y públicos. La intención es generar una comunidad de usuarios y desarrolladores que promuevan la simulación como herramienta para la investigación y el desarrollo de sistemas AAL. A la larga, esto debe ahorrar costes en el desarrollo de este tipo de tecnologías y, por consiguiente, hacerlas más asequibles y efectivas para los ciudadanos.

Este trabajo de tesis ha sido financiado con cargo a los proyectos del Ministerio de Economía y Competitividad “Social Ambient Assisted Living” (SociAAL, TIN2011-28335-C02-01/02) y “Desarrollo colaborativo de soluciones AAL” (ColosAAL, TIN2014-57028-R).

Índice

Agradecimientos	VII
Resumen	IX
1. Introducción	1
1.1. Contexto	1
1.1.1. Proyectos SociAAL y ColosAAL	2
1.2. Motivación	3
1.3. Objetivos	6
1.4. Metodología	7
1.5. Contribuciones	8
1.5.1. Publicaciones que avalan la tesis	10
1.6. Organización de la memoria de la tesis	12
2. Estado del arte	13
2.1. Los elementos principales de un sistema AmI	14
2.2. Requisitos generales del lenguaje y el simulador	17
2.2.1. Los principios para el diseño de notaciones visuales	19
2.3. Plataformas de simulación de entornos AmI	23
2.4. Plataformas de simulación genéricas	27
2.4.1. Los motores de juegos	27
2.4.2. Plataformas de mundos virtuales	31
2.4.3. Otras plataformas	32
2.4.4. La elección: jMonkeyEngine3	33
2.5. Las entrevistas a los enfermos de párkinson	34
2.6. Lenguajes de modelado	36
2.6.1. Lenguaje de modelado AAL: HCM-L	36
2.6.2. El lenguaje de modelado de UbikSim	38
2.6.3. Especificar la IA de los videojuegos: los árboles de comportamiento	40
2.6.4. Modelado formal de sistemas: Las Redes de Petri Coloreadas	44
2.6.5. Conclusiones	45
	XI

3. El lenguaje de modelado	47
3.1. Introducción	47
3.2. El meta-meta-modelo y la herramienta: INGENME	49
3.3. Meta-modelo de SociAALML	51
3.3.1. El espacio físico	53
3.3.2. Los sujetos a simular	61
3.3.3. El perfil social	64
3.3.4. El perfil de las actividades diarias	69
3.3.5. El perfil de las interacciones	79
3.3.6. El perfil de la enfermedad	85
3.3.7. El escenario	95
3.4. Conclusiones	102
4. El marco de dominio: PHAT	103
4.1. Requisitos	103
4.2. La arquitectura de PHAT	105
4.3. Conceptos básicos de jME3	107
4.4. El módulo del entorno	110
4.4.1. El entorno exterior de la vivienda	110
4.4.2. El interior de la vivienda	111
4.5. El módulo dedicado a los cuerpos de los sujetos virtuales	115
4.6. El módulo dedicado a los dispositivos	119
4.7. El módulo para acceder a los dispositivos virtuales	125
4.8. El módulo dedicado a la interfaz	129
4.8.1. Los agentes	130
4.8.2. Iniciar las simulaciones	134
4.8.3. Interfaz gráfica	136
4.8.4. Herramientas	137
4.9. El generador de código	137
4.10. Evaluación de los componentes software	141
4.11. Conclusiones	144
5. Casos de estudio	147
5.1. Caso de estudio 1: Modelado de la entrevista al paciente E9	150
5.1.1. El perfil social	151
5.1.2. El comportamiento de los sujetos	151
5.1.3. Modelado de la enfermedad	154
5.1.4. Validación del modelo	160
5.2. Caso de estudio 2: Ver la televisión	164
5.2.1. Descripción	164
5.2.2. Modelado de los escenarios	165

5.2.3. Diseño de la solución y evaluación	169
5.3. Valoración crítica de la plataforma a partir de los casos de estudio	174
5.3.1. Análisis de los modelos	174
5.3.2. Análisis del código generado	177
5.4. Uso de datos de sensores de la simulación	179
5.4.1. Efectividad de la simulación de un acelerómetro	180
5.4.2. Efectividad de la simulación de la cámara	182
5.4.3. Efectividad de la simulación del audio	185
5.5. Conclusiones	189
6. Conclusiones	191
6.1. Aportaciones	191
6.2. Líneas de investigación abiertas	193
6.3. Trabajos futuros	195
 I Apéndices	 197
A. Símbolos de los elementos del lenguaje SociAAML	199
A.1. Símbolos de los elementos del espacio físico	199
A.2. Símbolos de los elementos de los sujetos a simular	201
A.2.1. Los símbolos de los perfiles	201
A.2.2. Los símbolos del perfil social	202
A.2.3. Los símbolos del perfil de las actividades diarias	202
A.2.4. Los símbolos del perfil de las interacciones	205
A.2.5. Los símbolos del perfil de la enfermedad	206
A.2.6. Los símbolos del diagrama de simulación	208
B. API de los comandos de PHAT	209
B.1. El módulo del entorno	209
B.1.1. El entorno exterior de la vivienda	209
B.1.2. El interior de la vivienda	209
B.2. El módulo dedicado a los cuerpos de los sujetos virtuales	210
B.3. El módulo dedicado a los dispositivos	213
B.4. El módulo dedicado a los servidores	214
C. Esquemas descriptivos de la generación del código	217
C.1. Plantilla para generar las clases principales de la simulación	218
C.2. Plantilla para generar los agentes	221
C.3. Plantilla para gestionar los momentos del día	222
C.4. Plantilla para generar los momentos del día con sus actividades	223
C.5. Plantilla para la generación de las actividades	225

C.6. Plantilla para generar las secuencias de tareas	226
C.7. Plantilla para generar la clase encargada de implementar la enfermedad	228
C.8. Plantilla para generar las clases que implementan la evolución de los niveles de los síntomas	230
 Bibliografía	 231
 Lista de acrónimos	 244

Índice de figuras

1.1. Arquitectura del desarrollo dirigido por modelos.	9
2.1. Elementos principales de un Sistema de Inteligencia Ambiental.	14
2.2. Variables visuales [14]: definen un conjunto de técnicas gráficas elementales para la construcción de notaciones visuales.	20
2.3. Versión simplificada de la típica arquitectura de los elementos de un motor de juegos de Gregory [53].	28
2.4. Entorno de desarrollo de MASA-Life.	32
2.5. Captura de pantalla del SDK de jME3 donde se puede ver la maya de navegación generada.	34
2.6. Símbolos del lenguaje HCM-L [90].	37
2.7. Ejemplo de modelado de las actividades de la tarde con HCM-L [90].	37
2.8. Ejemplo de modelado de las actividades de la tarde con la notación de UbikSim.	40
2.9. Posibles símbolos para la notación de los árboles de comportamiento.	42
2.10. Ejemplo de modelado de las actividades de la tarde con BTs.	43
2.11. Ejemplo básico de un modelo CPN sobre un protocolo simple de red [65].	44
3.1. Los 4 niveles de instanciación de Kelly [71] necesarios para integrar el modelado del uso y la evolución de los sistemas.	48
3.2. Representación de las 5 entidades GOPRR de INGENME.	50
3.3. Representación de las entidades adicionales que ofrece INGENME para definir la sintaxis concreta y la documentación.	50
3.4. Plantilla para la representación de un objeto del lenguaje SociAALML.	53
3.5. Vistas del meta-modelo de SociAALML donde se incluyen los elementos espaciales.	55
3.6. Vistas de los diferentes elementos del modelo SociAALML de una vivienda de dos plantas.	58
3.7. Propuesta alternativa donde se integran todas las vistas de los niveles del edificio para cumplir con el principio de integración cognitiva de Moody.	61
3.8. Vista del meta-modelo que define los tipos de perfiles del sujeto.	64
3.9. Ejemplo de modelo de los perfiles de un enfermo de párkinson.	64
3.10. Vista del meta-modelo con los conceptos relacionados con el perfil social.	66

3.11. Modelos del perfil social del paciente y la pareja.	67
3.12. Vista del modelo como alternativa para modelar la red social global de todos los individuos.	69
3.13. Vista del meta-modelo para la agrupación de actividades a lo largo del día.	71
3.14. Vista del meta-modelo de las actividades del sujeto en un periodo del día.	72
3.15. Vista del meta-modelo con las entidades que definen las condiciones.	73
3.16. Vista del meta-modelo de las tareas que componen una actividad.	74
3.17. Ejemplo de modelo para definir los momentos del día.	77
3.18. Ejemplo de modelo de las actividades del sujeto para un periodo del día.	78
3.19. Secuencia completa de las tareas para que el sujeto se duche.	78
3.20. Modelos de árboles de comportamiento para modelar los momentos del día y las actividades.	81
3.21. Vista del meta-modelo de las interacciones.	83
3.22. Ejemplo de modelos para asistir al paciente con el mando a distancia.	84
3.23. Vista del meta-modelo de la enfermedad de Parkinson.	86
3.24. Vista del meta-modelo de la toma de medicamentos del enfermo de párkinson.	87
3.25. Vista del meta-modelo para definir la evolución de los niveles de intensidad de un síntoma.	88
3.26. Vista del meta-modelo para definir las modificaciones de las tareas según el nivel del síntoma.	89
3.27. Vista del meta-modelo para definir los tipos de relaciones según el tipo de filtro.	89
3.28. Ejemplo de modelo de la enfermedad de Parkinson.	91
3.29. Ejemplo de modelo donde se especifica la toma de medicamentos del enfermo de párkinson.	92
3.30. Ejemplo de modelo para describir la evolución del síntoma de pies congelados.	92
3.31. Limitaciones según los diferentes niveles de intensidad del síntoma de pies congelados.	93
3.32. Especificación para modelar las limitaciones en la tarea de andar para el síntoma de pies congelados empleando los árboles de comportamiento.	95
3.33. Vista del meta-modelo de un escenario de simulación.	98
3.34. Ejemplo de modelo para definir un escenario de simulación.	100
4.1. Los módulos que componen la arquitectura de PHAT.	106
4.2. Clases principales de jME3 para crear la simulación.	108
4.3. AppStates que forman los módulos de PHAT.	109
4.4. Terreno TwoHouses para simular el entorno exterior.	111

4.5. Captura de pantalla de la herramienta de diseño de interiores Sweet-Home3D. Los modelos 3D y las texturas empleadas están bajo licencia libre.	113
4.6. Captura de pantalla donde se muestra la malla de navegación generado con el SDK de jME3 para una vivienda.	114
4.7. Objetos de la vivienda con información, sonidos y/o emisores de partículas.	115
4.8. Metamodelo y modelo equivalente de las entidades de condición.	117
4.9. Huesos y animaciones que ofrece el modelo 3D del sujeto virtual.	118
4.10. Controlador KinematicRagdollControl para simular las caídas según las leyes de la física. Este genera automáticamente objetos rígidos entorno a los huesos para simular el efecto de la física sobre los mismos (colisiones y gravedad).	119
4.11. Clases controladores de PHAT para crear sensores con ejemplo del acelerómetro.	121
4.12. Simulación para probar el acelerómetro.	122
4.13. Simulación para probar la cámara.	123
4.14. Simulación para probar la micrófonos.	124
4.15. Ejemplo de dispositivos asociados al sujeto virtual. Uno se encuentra fijado al pecho y el otro a la muñeca izquierda.	125
4.16. Diferentes arquitecturas para acceder al hardware virtual de la simulación.	127
4.17. Esquema del mecanismo implementado en PHAT para permitir que las aplicaciones AAL accedan a los sensores y actuadores de su dispositivo virtual.	129
4.18. Diagrama UML de las clases principales que implementan los comportamientos.	131
4.19. Diagrama UML de las clases PHAT que implementan la enfermedad, los síntomas y las limitaciones.	133
4.20. Log de PHAT que muestra las tareas y las actividades de los sujetos virtuales.	137
4.21. Capturas de pantalla donde se visualizan los identificadores de los objetos de la vivienda “House3room2bath”.	138
4.22. Ejemplo sencillo de generación de código a partir de una plantilla.	139
4.23. Comprobación de los niveles de calidad del código realizado por SonarQube.	143
4.24. Resultados del análisis de la mantenibilidad del código realizado por SonarQube.	144
5.1. Perfil social del paciente E9.	151
5.2. Los ADLs del matrimonio y el voluntario.	152
5.3. Actividades que realiza la compañera del paciente E9 por las mañanas después de asearse y vestirse.	153
5.4. Actividades que realiza el paciente E9 tras desayunar y asearse.	154

5.5. Diagramas de secuencia de tareas genéricas organizados por habitación.	155
5.6. Diagramas de secuencia de las tareas que corresponden a realizar una comida sentado en la silla con id <i>Chair1</i> .	155
5.7. Filtros del paciente E9 relacionados con la actividad de vestirse.	156
5.8. Filtros del paciente E9 para modelar velocidad de desplazamiento y las caídas.	157
5.9. Evolución de los niveles de los síntomas del paciente E9.	158
5.10. Extractos de los diagramas con las entidades para recordar al paciente que debe tomar la medicación y la respuesta del paciente realizando la toma del mismo.	159
5.11. Diagrama del perfil de enfermedad de Parkinson del sujeto E9.	159
5.12. Diagrama de simulación donde participan todos los sujetos y la simulación se inicia el lunes 4 de enero a las 8 de la mañana.	161
5.13. Error al modelar la tarea <i>DoWoodworking1h</i> al incluir la tarea de ir al aseo.	163
5.14. Secuencia de tareas que definen la actividad de ver la televisión.	165
5.15. Perfil de la enfermedad de Parkinson con los síntomas de temblores y pies congelados y sus correspondientes limitaciones cuando el nivel del síntoma es medio.	166
5.16. Diagrama de filtros que alteran el comportamiento del paciente cuando el nivel del síntoma de pies congelados es medio.	166
5.17. Diagrama de filtros que alteran el comportamiento del paciente cuando el nivel del síntoma de temblores es medio.	167
5.18. Diagrama inicial de simulación del escenario con el paciente con niveles medios en sus síntomas de pies congelados y temblores.	167
5.19. Captura de pantalla correspondiente al log del escenario con valor de semilla 2.	168
5.20. Esquema del diseño de la solución AAL para dotar al paciente de más autonomía.	170
5.21. Diagrama de simulación para evaluar el prototipo que asiste al paciente para cambiar de canal mediante comandos de voz.	172
5.22. Diagrama con la secuencia de tareas para simular el fallo al pulsar el botón 5 (pulsando el 8), su frustración, y su respuesta realizando la misma tarea pero con la voz.	172
5.23. Captura de pantalla justo después de que el paciente diga “five” (cinco).	173
5.24. Resultado del análisis de SonarQube aplicado al código generado a partir del modelo creado para la entrevista al paciente E9.	178
5.25. Resultado del análisis de SonarQube aplicado al código generado a partir del modelo creado para el caso de uso de ver la televisión.	179
5.26. Sistema de coordenadas de los acelerómetros en los dispositivos Android.	181

5.27. Diagrama de simulación para recrear la situación del sujeto virtual bebiendo con el dispositivo sujeto a la muñeca derecha y así poder guardar las aceleraciones recibidas en un fichero.	181
5.28. Comparativa eje a eje de los datos obtenidos de los acelerómetros virtual y real mientras se realiza el gesto de beber.	183
5.29. Fotogramas por segundo obtenidos por el dispositivo Android real y el simulado.	184
5.30. Comparativa del vídeo capturado desde la simulación y su equivalente en el mundo real.	185
5.31. Distancias máximas a las que es posible leer diferentes tamaños de códigos QR (5, 2.5 y 1 cm) en UbikMobile, PHAT y el mundo real (Nexus S).	187
5.32. Diagramas principales para probar la interacción por voz del sujeto virtual con la aplicación de demo de CMU Sphinx.	188
5.33. Sujeto virtual interaccionando con la aplicación a través de comandos de voz. La aplicación emplea la librería CMU Sphinx para su reconocimiento.	188
C.1. Parte de la plantilla de las simulaciones que corresponde a la inicialización de la fecha, la hora, la semilla y el modelo 3D de la vivienda. . . .	218
C.2. Parte de la plantilla de las simulaciones que corresponde a la inicialización de los sujetos virtuales.	219
C.3. Parte de la plantilla de las simulaciones que corresponde a la inicialización de los dispositivos.	220
C.4. Plantilla para generar los agentes e instanciar a su ADL, su gestor de eventos y la enfermedad.	221
C.5. Plantilla para generar los periodos del día que conforman las actividades diarias de la persona.	222
C.6. Parte de la plantilla para definir la clase que definirá las actividades de un periodo de tiempo.	223
C.7. Parte de la plantilla que crea las actividades y sus relaciones para un periodo del día.	224
C.8. Plantilla para la generación de las actividades.	225
C.9. Parte de la plantilla que genera la clase y el constructor de una secuencia de tareas.	226
C.10. Parte de la plantilla que genera las subtareas de una secuencia de tareas.	227
C.11. Parte de la plantilla para crear la clase que gestiona la enfermedad añadiendo los síntomas y estableciendo sus niveles iniciales.	228
C.12. Parte de la plantilla que crea las cadenas de filtros y las añade a los síntomas según el nivel.	229
C.13. Plantilla para generar el código relativo a la evolución de los síntomas.	230

Índice de Tablas

1.1. Tabla de publicaciones que avalan la tesis.	11
2.1. Tabla simuladores AmI.	25
2.2. Tabla lenguajes comportamientos.	45
3.1. Evaluación espacio físico según los principios de Moody	60
3.2. Evaluación perfil social según los principios de Moody	68
3.3. Evaluación cualitativa perfil ADL según los principios de Moody	80
3.4. Evaluación perfil de enfermedad de Parkinson según los principios de Moody	96
3.5. Evaluación notación elementos para simulaciones según los principios de Moody	101
4.1. Tabla resumen SLOCs componentes	142
5.1. Tabla hechos caso 1	162
5.2. Tabla resumen de situaciones en función de la semilla	170
5.3. Tabla estadísticas diagramas caso 1	175
5.4. Tabla estadísticas diagramas caso 2	176
A.1. Notación elementos de un edificio	199
A.2. Notación elementos de una planta	200
A.3. Notación elementos de una habitación	200
A.4. Notación perfiles asociados al humano	201
A.5. Notación elementos del perfil social	202
A.6. Notación elementos del periodos del día	202
A.7. Notación elementos de las actividades del día	203
A.8. Notación elementos de las tareas. Parte 1	204
A.9. Notación elementos de las tareas. Parte 2	205
A.10. Notación elementos de las interacciones	205
A.11. Notación elementos de los síntomas del Parkinson	206
A.12. Notación elementos de la toma de medicamentos	206
A.13. Notación elementos de la evolución de los síntomas	207

A.14. Notación elementos del periodos del día	207
A.15. Notación elementos de iniciación de la simulación	208

Índice de Códigos

4.1. Interfaz <code>PHATInitializer</code> para facilitar la creación de simulaciones en PHAT	134
4.2. Ejemplo de clase principal para crear un escenario en PHAT	135

Capítulo 1

Introducción

Piensa, cree, sueña y atrévete.

Walt Disney

RESUMEN: Este capítulo comienza poniendo en contexto la tesis y presentando sus motivaciones y objetivos para después describir los pasos que se han seguido para alcanzar dichos objetivos. Finaliza introduciendo las contribuciones del trabajo realizado y cómo se ha estructurado este documento.

1.1. Contexto

El paradigma **AmI** (*Ambient Intelligence*, **Inteligencia Ambiental**) busca crear *entornos ubicuos* que ofrezcan a los usuarios *servicios sensibles al contexto* [119, 2, 100].

El término “ubícuo” lo introdujo Weiser [142] en 1991 para referirse al concepto de la perfecta integración de los dispositivos en la vida cotidiana de sus usuarios. Es decir, la idea consiste en que los dispositivos dejan de ser un conjunto de aparatos independientes y pasivos, para pasar a estar interconectados, ser proactivos y formar parte natural del entorno y de los usuarios.

Los servicios o aplicaciones sensibles al contexto [3] (Context-Aware) son aquellos elementos software que son capaces de ofrecer sus funcionalidades dependiendo del contexto. Day y Abowd [3] definen el contexto como:

“El contexto es cualquier información que se pueda usar para caracterizar la situación de una entidad. Un sujeto, lugar u objeto que se considere relevante para la interacción entre el usuario y la aplicación, incluyendo al propio usuario y a la aplicación, puede ser una entidad”.

Por tanto, un sistema AmI se adapta y ofrece de forma proactiva los servicios adecuados a los usuarios. Los servicios son adecuados si resultan de interés para el usuario. Para ello, el servicio debe conocer en cada momento los deseos, necesidades e intereses del usuario (contexto). El objetivo de estos sistemas es asistir, facilitar y hacer más eficiente el desempeño de las actividades diarias de sus usuarios [10].

La creación de servicios sensibles al contexto puede resultar una tarea muy compleja. La efectividad de un servicio depende de la capacidad del sistema para identificar en cada momento la actividad que está realizando el usuario e, inclusive, predecir aquella que realizará o le gustaría realizar. Conseguir esta información es un gran reto que se vuelve mayor si se persigue asistir al usuario en cualquier lugar y ante cualquier situación. Por ello, actualmente, debido a la complejidad del problema, los estudios sobre este paradigma se limitan a espacios concretos dependiendo del dominio de aplicación. Algunos ejemplos de estos espacios son hospitales [125, 41], museos [111], vehículos [16] y hogares [45]. En concreto, el campo de investigación en AAL (*Ambient Assisted Living*, Vida cotidiana asistida por el entorno) [73] tiene como objetivo aplicar el paradigma de los sistemas AmI para mejorar la autonomía y la calidad de vida de las personas mayores en sus hogares. En la última década, los gobiernos europeos están promoviendo esta línea de investigación debido al envejecimiento de la población, tal y como se refleja en los Retos de la Sociedad del programa nacional y del programa Horizonte 2020¹. Un reflejo de dicha inversión son los proyectos “Social Ambient Assisted Living” (SociAAL) y “Desarrollo colaborativo de soluciones AAL” (ColosAAL) financiados por el Ministerio de Ciencia e Innovación y donde se desarrolla este trabajo.

1.1.1. Proyectos SociAAL y ColosAAL

El proyecto SociAAL² se inició en 2012 y finalizó en 2015. El consorcio lo formaron la Universidad de Murcia y la Universidad Complutense de Madrid. Además, colaboró la asociación Parkinson Madrid. El equipo de personas que participaron en el proyecto fue interdisciplinar: sociólogos, ingenieros de conocimiento, de Inteligencia Artificial y de software. Entre ellos, tanto los directores del trabajo como el candidato han formado parte del proyecto. El candidato se incorporó en 2013 gracias a una beca predoctoral FPI.

El objetivo del proyecto fue promover el desarrollo de sistemas AAL para un colectivo concreto, el de los enfermos de párkinson y sus familiares. La enfermedad de Parkinson es neurodegenerativa y crónica. Esta se produce por la pérdida progresiva de una sustancia en el cerebro llamada dopamina [19]. Los síntomas de la enfermedad están relacionados con el deterioro de las habilidades motoras, ya que la carencia de esta sustancia hace que los músculos tiemblen y se vuelvan rígidos [64]. Debido a este progresivo deterioro, los enfermos son cada vez menos autónomos y, por consiguiente, su calidad de vida y la de los que les rodean se ve mermada.

¹Artículo Horizon 2020: <http://ec.europa.eu/programmes/horizon2020/en/h2020-section/health-demographic-change-and-wellbeing>

²Web de SociAAL:<http://grasia.fdi.ucm.es/sociaal>

Las herramientas, métodos y técnicas que se emplean para la construcción de servicios AmI aplicables a sistemas AAL necesitan aún alcanzar un estado de madurez que no tienen. El proyecto ofreció un marco metodológico que diera apoyo a las actividades de modelado, de simulación y de implementación para la creación de sistemas AAL.

Para alcanzar dicho objetivo, el proyecto se propuso obtener dos resultados. Por un lado, entender el tipo de aplicaciones AAL que demandan los pacientes con enfermedad de Parkinson. Por el otro, ofrecer una infraestructura que permita a la industria crear soluciones más eficientes y más asequibles. Para ello, la asociación Parkinson Madrid facilitaron la realización de un estudio de campo sociológico consistente en unas entrevistas a enfermos de párkinson y a sus familiares. El objetivo de las entrevistas fue recabar información sobre el estado de los pacientes, sus necesidades y el tipo de asistencia tecnológica que podría serles de utilidad. Por ejemplo, la mayoría de los sujetos manifestaban un gran interés por algún mecanismo que mejorase su movilidad y, además, fuese capaz de detectar las caídas del enfermo y de notificar dichos incidentes a algún familiar. Para más detalles sobre las entrevistas consultar el apartado 2.5. Esta tesis ha contribuido al segundo resultado del proyecto (el de la infraestructura) como se verá a lo largo de todo el documento.

El trabajo continuó en el proyecto ColosAAL³ centrándose, esta vez, en el efecto de enfermedades neurodegenerativas en el comportamiento. Usando como punto de partida las herramientas desarrolladas en SociAAL, se desarrolló nueva funcionalidad para expresar actividades donde los procesos cognitivos tuvieran un papel dominante, como es el caso del dialogo entre dos personas. Los avances hechos en ColosAAL en términos de mejoras en la notación y en las herramientas de soporte forman parte de esta tesis aunque dichas mejoras no se mencionan explícitamente. Otras mejoras específicas del dominio de ColosAAL, tales como el prototipado de dispositivos y la gestión de diálogos, se ha dejado aparte por requerir mayor experimentación.

En definitiva, la tesis se enmarca dentro de dos proyectos que abarcan dos contextos principalmente, uno tecnológico y otro social. El contexto tecnológico corresponde a la complejidad de los sistemas AmI. El contexto social tiene que ver con el deterioro de la calidad de vida de los enfermos y su familiares.

1.2. Motivación

Cunningham et al. mantienen que la tecnología AmI puede mejorar la autonomía y la calidad de vida de las personas mayores, y en concreto, la de las que sufren la enfermedad de Parkinson [33]. Para ello, realizaron un estudio sobre las tecnologías disponibles en aquel periodo (hasta 2009) y que estaban dirigidas a la asistencia de este tipo de enfermos. Sin embargo, la conclusión principal fue que la mayoría de las soluciones existentes eran muy invasivas. Como solución, propusieron el empleo de tecnologías AmI (o AAL), ya que precisamente persiguen una filosofía no invasiva. No obstante, a pesar de disponer cada día de hardware y software más avanzado, la

³Sitio web de ColoSAAL: <http://grasia.fdi.ucm.es/colosaal/>

creación de un sistema AmI totalmente equipado puede resultar muy complejo y costoso [101, 138, 39].

Una parte importante del problema reside en la necesidad de *entender y adaptarse a los requerimientos particulares de los usuarios finales*, las personas adultas y los cuidadores, como demuestran estudios recientes [5]. Una inadecuada realización de esta tarea, lleva a no generar soluciones AAL eficaces y bien aceptadas por los usuarios finales [61].

Un prototipo es una implementación parcial de un sistema. Se crea expresamente para aprender más sobre un problema o sobre una solución al problema. El uso de prototipos puede ser una forma de ayudar al cliente (o futuro usuario) a que valide los requisitos y a que le surjan otros nuevos [36]. Sin embargo, *la validación y evaluación de este tipo de prototipos AAL en el mundo real* es costosa y compleja [103]. El comportamiento de estos sistemas se basa en el contexto. Este contexto se genera a partir de la interacción dinámica de los elementos principales de los sistemas AmI: el espacio físico, los usuarios, los dispositivos y el software [21, 57]. Por tanto, se hace imprescindible disponer de estos recursos para la validación y evaluación de los prototipos. Dependiendo del tipo y del número de recursos necesarios, el coste que supone adquirirlos puede no ser asumible para un grupo de investigación o una compañía. Este problema económico se agudiza aún más si resulta necesario *evaluar la escalabilidad del sistema* [126, 98]. Es decir, analizar cómo se comporta el sistema al aumentar el número de usuarios, de dispositivos y al ampliar el espacio físico donde el sistema se despliega. En cualquier caso, aunque se disponga de todo lo necesario, llevar a cabo las pruebas del sistema AAL en el mundo real de forma *repetitiva y controlada* no es sencillo y consume mucho tiempo [109]. Esta tarea supone, por un lado, coordinar todos los elementos para que se produzcan las condiciones específicas de interés y, por el otro, comprobar la respuesta del sistema ante el contexto generado. Por último, hay que añadir el *coste de respetar y mantener la privacidad e integridad del usuario*. Determinadas situaciones son difíciles de reproducir, por ejemplo, un incendio en la vivienda o que el paciente se caiga por las escaleras.

En definitiva, todos estos problemas suponen que el desarrollo e investigación de este tipo de sistemas sea costoso y, por consiguiente, el producto final no sea asequible para la mayor parte de los usuarios [76].

Durante el transcurso de este trabajo, no se encontró un estándar concreto para el desarrollo de sistemas AmI. Por el contrario, sí que se hallaron algunos trabajos [58, 54, 72] donde se proponen metodologías o guías de desarrollo. En todas las propuestas se enfatiza la importancia de implicar más al usuario en las fases del desarrollo del sistema. No obstante, también todas ellas mantienen la idea de basar las evaluaciones de los diseños del sistema en un entorno real. El enfoque más conocido para desarrollar sistemas AAL es el *Living Lab* [72] (laboratorio viviente). El enfoque abarca desde la concepción del sistema hasta su comercialización. Al igual que en las propuestas anteriores, este se caracteriza por involucrar al usuario final desde la concepción del sistema hasta su prueba. Este enfoque suele requerir de una vivienda prototipo especialmente diseñada para este fin. Además, la vivienda debe disponer de los dispositivos necesarios

donde desplegar el software y de todos los elementos para que los usuarios puedan recrear sus actividades diarias como si estuviesen en su vivienda. Por último, idealmente, el entorno físico del Living Lab debe estar equipado con dispositivos disfrazados que permitan el seguimiento y registro de los datos del correspondiente experimento (e.g. micrófonos, cámaras). Aunque los Living Labs suponen un gran avance, siguen sin resolver los problemas relacionados con la falta de presupuesto, la repetición y el control de las pruebas, los costes relativos al tiempo y la dificultad a la hora de evaluar el sistema ante ciertos escenarios peligrosos.

Una alternativa al método de Living Lab es la simulación. El uso de esta tecnología es común en otros campos tales como la robótica (donde es popular el simulador GAZEBO ⁴ para el diseño de robots y prueba de algoritmos sin necesidad de tener el robot físico) o como en la creación de sistemas ópticos (donde se puede utilizar el simulador Zemax⁵ para evaluar diseños sin necesidad de disponer de los componentes ópticos). No obstante, esta idea de aplicar la simulación a la Inteligencia Ambiental no es nueva. De hecho, son varios los autores los que promueven la simulación 3D como herramienta para el desarrollo de sistemas AmI. Bainbridge [9] hace un repaso sobre el potencial de los mundos virtuales como herramienta para la investigación científica en campos como la sociología, la teoría del comportamiento y la computación centrada en el humano. Ya que estos mundos virtuales facilitan, por un lado, que los sujetos se relacionen a través de su avatar sin necesidad de estar físicamente en el mismo espacio y, por otro lado, que los investigadores puedan observar y analizar más fácilmente lo que ocurre a lo largo del experimento. Aarts y Ruyter [1] apoyan la idea de que los entornos de programación para usuarios finales de sistemas AmI incorporen un mecanismo que permita validar lo programado a través de visualización 3D, animaciones y simulación. De esta forma, los propios usuarios pueden validar si lo que especificaron era o no lo que deseaban. Tang et al. [133] sostienen la importancia de la simulación como herramienta de prototipado rápido que permita dar soporte al diseño de sistemas AmI. Sernani et al. [127] también barajan la posibilidad de utilizar entornos virtuales 3D durante el desarrollo de sistemas AmI y para realizar algunas pruebas previas al despliegue en el entorno real.

Por tanto, la simulación como herramienta de desarrollo e investigación de sistemas AmI podría eliminar los problemas que no solucionan los Living Labs. Como consecuencia del uso de simulaciones, muchos elementos del sistema pasan ahora a ser simulados: el espacio físico, los usuarios y los dispositivos. Por ello, se reducen los costes asociados a estos elementos. La evaluación de la escalabilidad tampoco supone un problema, pues implica aumentar el número de elementos simulados. También se solucionan los problemas de control y repetitividad en las pruebas. Tampoco es necesario preocuparse por la privacidad e integridad de los usuarios, ya que estos son sujetos virtuales. Además, la simulación permite compartir con la comunidad científica los experimentos realizados. No obstante, no se pretende que la simulación reemplace a las pruebas en el mundo real, sino que sea una herramienta complementaria. Una herramienta con la cual se pueda

⁴Sitio web de GAZEBO: <http://gazebosim.org/>

⁵Sitio web de Zemax: <http://www.zemax.com/>

facilitar la comprensión inicial del comportamiento del sistema y, además, sea el primer lugar donde se conciben y ensayan las primeras versiones del software sin necesidad de disponer del resto de elementos del sistema (espacio físico, usuarios y dispositivos).

A pesar de las numerosas ventajas que podría aportar la simulación al desarrollo de los sistemas AmI, el desarrollo de simulaciones 3D realistas que incluyan el entorno (e.g. una vivienda), los dispositivos (e.g. teléfonos inteligentes o electrodomésticos), y los sujetos (e.g. los usuarios, pacientes o cuidadores) también es una tarea compleja y costosa que requiere de personal especializado. Desde hace aproximadamente 20 años, la comunidad ha ido proponiendo nuevos simuladores AmI cada vez más avanzados, según la evolución de las tecnologías 3D. Sin embargo, parece que no se ha llegado a imponer ninguna propuesta y la gran mayoría no están accesibles al resto de la comunidad. Por otro lado, aunque son simuladores específicos, la generación de dichas simulaciones son costosas y la mayoría requieren conocimientos de programación. Este trabajo propone como alternativa, un lenguaje gráfico, de alto nivel y específico de dominio. Este lenguaje debería permitir a usuarios no expertos en simulación crear especificaciones que correspondan a situaciones problemáticas de la vida diaria de los enfermos de párkinson y sus familiares. Estas especificaciones se podrían considerar una forma de caso de uso. Además, el lenguaje facilitaría la generación automática de código a partir de sus especificaciones. La generación automática significa que no es necesario modificar el código generado. Este código generado recogería la semántica de la especificación en forma de simulación 3D.

De esta propuesta se destacan varias ventajas. El lenguaje sirve como mecanismo formal para recoger requisitos del usuario y mecanismo de comunicación entre los distintos actores. Además, acorde con las ideas de [1], las especificaciones se pueden validar mediante la visualización 3D de sus simulaciones equivalentes. En definitiva, la intención es que el lenguaje y su generación automática faciliten y hagan más accesible el desarrollo de este tipo de simulaciones.

1.3. Objetivos

Con el fin de contribuir en la idea de que el modelado realista de escenarios AmI puede ser una herramienta con futuro en el proceso de desarrollo de este tipo de productos, y en concreto para el caso de la asistencia de enfermos de párkinson, se han establecido los siguientes objetivos específicos:

1. Identificar y capturar pero sobre todo expresar modelando información sobre la vida diaria de un paciente de párkinson de tal forma que pueda ser posteriormente analizada y procesada.
2. Proponer soluciones para interpretar esta información y crear simulaciones realistas que reproduzcan la vida de los pacientes. Estas simulaciones deben permitir el despliegue del sistema AAL a evaluar y la interacción entre los sujetos virtuales y el sistema.

1.4. Metodología

Para alcanzar los objetivos anteriores, se ha llevado a cabo el siguiente plan de trabajo:

- *Estudio del estado del arte de plataformas de simulación.* Se ha realizado una búsqueda, y posterior análisis, de los simuladores de Inteligencia Ambiental existentes hasta la fecha. El objetivo del estudio es encontrar una plataforma de simulación a partir de la cual basar la generación de código. No obstante, al no disponer de ningún simulador que satisficiera las necesidades, se hizo también necesario un estudio más amplio para decidir un marco de trabajo a partir de la cual desarrollar nuestra propia plataforma de simulación.
- *Estudio del dominio de la enfermedad de Parkinson.* Con el fin de familiarizarse con el dominio de la enfermedad de Parkinson, se ha buscado y estudiado literatura relacionada con la materia. También se ha llevado a cabo un análisis detallado de las transcripciones de las entrevistas realizadas a los enfermos de párkinson por compañeros del proyecto SociAAL.
- *Definición de la plataforma de simulación.* Esta plataforma permite recrear de forma realista escenarios de la vida cotidiana de los enfermos de párkinson y sus familiares. Para ello, resulta necesario estudiar la forma de modelar la vivienda, los dispositivos y los sujetos virtuales. Dedicando mayor esfuerzo al modelado de los sujetos, para así conseguir que estos sean capaces de comportarse según la física (especialmente los temblores y las caídas), realizar tareas domésticas de la vida diaria y reproducir los síntomas de la enfermedad.
- *Estudio del estado del arte de los lenguajes de modelado.* Se estudiaron y analizaron las propuestas de lenguajes de modelado con el fin de comprobar si eran apropiados. Para ello, los mismos deberían ser capaces de especificar los comportamientos y situaciones que reflejasen el día a día de los enfermos de párkinson y sus familiares. En este proceso se prestó especial atención a las situaciones problemáticas donde el enfermo necesita asistencia y/o puede sufrir un accidente.
- *Definición del lenguaje de modelado y el generador.* Este lenguaje permite modelar el entorno, los dispositivos y los sujetos. No obstante, este se centra en el comportamiento de los sujetos. Estos se definen en base a perfiles: social, actividades diarias, interacciones y enfermedad de Parkinson.
- *Validar la propuesta a través de casos de estudio.* Para validar la propuesta de este trabajo se han llevado a cabo dos casos de estudio. La finalidad es mostrar la filosofía de uso de la propuesta y las ventajas que aporta.

Matizar que, una vez que el simulador proporcionó las funcionalidades básicas, el proceso de definición del lenguaje, el generador y la plataforma de simulación ha

sido realmente un proceso iterativo. En este proceso iterativo, primero se identificaba un nuevo concepto. Luego se incluía en el lenguaje a través de su meta-modelo y su representación. Posteriormente, se comprobaba cómo reflejar dicho concepto en el simulador y se implementaba si era necesario. Por último, se realizaban los cambios oportunos en el generador para que considere los nuevos elementos del lenguaje fruto del nuevo concepto.

1.5. Contribuciones

La propuesta de esta tesis ha sido utilizar el MDD (*Model-Driven Development*, Desarrollo Dirigido por Modelos) para la generación de simulaciones 3D convincentes que reproduzcan situaciones de la vida cotidiana de los enfermos de párkinson en sus hogares. En el MDD se utilizan los modelos como elemento principal en el proceso de desarrollo del producto, en nuestro caso la simulación, en lugar del código. Dichos modelos se transforman en código haciendo uso de un generador. Dependiendo de los modelos, la generación podrá ser totalmente automática o no. En el caso de que los modelos sean específicos del dominio, más abstractos serán los conceptos y más fácil será la generación automática del código [71]. Por el contrario, en el caso de UML [124], los modelos son de propósito general, pensados para modelar cualquier sistema. Por ello, frecuentemente, la generación de código en UML se limita a generar las plantillas de las clases. Por estas ventajas, se ha optado por modelos específicos del dominio, cuyas siglas en inglés son DSM (*Domain-Specific Modeling*, Modelado Específico del Dominio). Los elementos principales del DSM son el lenguaje, el generador y el marco de dominio. Los cuales, forman parte de las contribuciones de esta tesis. Un diagrama de la arquitectura con estos elementos se muestra en la Figura 1.1. Los elementos se describen a continuación:

- El *Lenguaje específico del dominio* proporciona un mecanismo de abstracción que permite definir la especificación (el modelo) en términos del dominio en lugar de en términos de programación. En nuestro caso, SociAALML (*Social Ambient Assisted Living Modeling Language*, Lenguaje de modelado para la sociedad de la vida asistida de las personas) es el lenguaje específico del dominio propuesto para definir los escenarios domésticos de los enfermos de párkinson. El lenguaje se define a partir de un meta-modelo y tiene asociado una notación y unas herramientas. Destacar el trabajo realizado en la notación (creación de todos los símbolos) para que esta sea cognitivamente eficiente y resulte más fácil su aprendizaje y comprensión. El lenguaje es el resultado del conocimiento adquirido sobre el estudio de la enfermedad a través de la literatura y el estudio de las transcripciones de las entrevistas a los enfermos y sus familiares. En el Capítulo 3 se encuentran todos los detalles.
- El *Generador* extrae la información de los modelos y la transforma en código que extiende o utiliza del *Marco del dominio*. El generador se ha implementado como

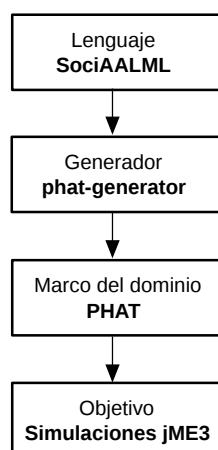


Figura 1.1: Arquitectura del desarrollo dirigido por modelos.

un módulo de PHAT (*Physical Human Activity Tester*, Prueba de la actividad física humana) y se encarga de generar simulaciones a partir de los modelos creados con SociAALML.

- El *Marco del dominio* proporciona interfaces entre el código generado y la plataforma subyacente. Este elemento se utiliza para facilitar la generación del código. PHAT es nuestra propuesta que hace las veces de marco de dominio, es decir, un conjunto de módulos que ofrecen interfaces para la creación de elementos en la plataforma subyacente. La plataforma subyacente es el motor de juegos jME3 (*jMonkeyEngine3*, Motor de juegos en Java). Y los elementos a crear son los virtuales del sistema AmI, tales como la vivienda, los sujetos con sus comportamientos, la enfermedad y los dispositivos. Sin este marco de dominio, resultaría mucho más complicado la generación del código directamente a jME3.

Esta forma de desarrollo permite subir el nivel de abstracción permitiendo especificar la solución (la simulación de un escenario) en términos del dominio. Es decir, en el lenguaje se encuentran conceptos como humano, actividad, tarea, enfermedad, síntoma y limitación. Esto permite que individuos interesados en el dominio (un trabajador social, un médico, un familiar o el propio enfermo), y que no tienen conocimientos de programación, puedan especificar y generar simulaciones. Además, limitar el dominio de aplicación del lenguaje, el generador y el marco permite que la generación de código sea totalmente automática. En DSM, el código generado es funcional, legible y eficiente si se desarrolla correctamente el generador [71].

Otras aportaciones de este trabajo hacen referencia a los casos de estudio abordados. Estos suponen una contribución de conocimiento sobre la materia. Por un lado, dicho conocimiento se encuentra en los modelos creados con el lenguaje para recrear dichos casos. Y, por otro lado, a partir de la experiencia obtenida a la hora de desarrollar dichos casos de estudio, suponen una propuesta sobre la forma de usar este tipo de herramientas.

Finalmente, destacar que todas estas herramientas son gratuitas, de código abierto y están disponibles a través de la web. En la misma, también se encuentran vídeos demostrativos así como un tutorial para iniciarse en el uso de estas herramientas. Así, se busca promover y facilitar el estudio y el desarrollo de sistemas de AmI con ayuda de la simulación. De hecho, unos estudiantes del grado en informática han empleado el simulador para evaluar su sistema de detección de caídas presentado en su trabajo final de grado. También, un estudiante de doctorado, Marlon Cádenas, ha iniciado su tesis en el desarrollo de algoritmos de reconocimiento de actividades basándose en el simulador PHAT.

1.5.1. Publicaciones que avalan la tesis

Son varias las publicaciones científicas en las que el candidato ha contribuido. Además, todas están relacionadas con la simulación como medio para dar soporte al desarrollo de sistemas de Inteligencia Ambiental.

El autor ha colaborado en tres trabajos donde se intenta demostrar la utilidad de la simulación para la validación y prueba de sistemas AmI. El primero consiste en la prueba de un sistema de localización en interiores basado en Bluetooth [83]. El segundo trata sobre la validación del gestor de contexto FAIRE [42, 37] desarrollado por Fernández-de-Alba et al. El último trabajo [46, 50], utiliza la simulación para ayudar a definir las reglas de un sistema normativo. El sistema normativo monitoriza la simulación y cuantifica si esta corresponde a un escenario donde se fomenta la autonomía del individuo o bien se fomenta su dependencia con respecto a un sistema AmI.

En [23, 24] se identifican problemas que pueden surgir a la hora de probar dispositivos móviles como teléfonos inteligentes en el laboratorio y se propone la simulación como solución. En la revista [29] se desarrolla más en profundidad la idea del uso de la simulación 3D para probar aplicaciones en teléfonos inteligentes. Se estudia la realización de dichas pruebas tanto de forma manual (siendo el usuario el que controla su avatar) como de forma automática (definiendo el comportamiento de los sujetos virtuales y su interacción con el dispositivo). Para ello, se amplió el simulador UbikSim [29] para soportar dispositivos virtuales conectados a dispositivos Android reales o emuladores. Esta nueva herramienta se llamó UbikMobile.

UbikSim supone una aportación significativa en la simulación AmI. No obstante, también carece de ciertas funcionalidades necesarias para aportar realismo en el dominio que nos ocupa. Algunas de estas funcionalidades de las que carece son la física, las animaciones de los sujetos virtuales, la iluminación y renderización de audio. Por ejemplo, UbikSim no permite obtener las aceleraciones producidas por la caída de un sujeto virtual. En [28] se identifican estas carencias y se propone el uso de motores de juegos como componente software a partir del cuál crear el simulador PHAT.

El prototipo de PHAT se ha presentado en varios congresos en la categoría de demostración, aplicándose a diferentes casos de estudio. Estos congresos son: PAAMS 2014 [25] y 2016 [51], AAMAS 2015 [27] y DRT4ALL [47]. Este resultó segundo en el caso del PAAMS 2014, finalista en el caso del DRT4ALL y primero en el PAAMS

2016. Además, la herramienta se ha presentado a la comunidad a través de talleres en dos ocasiones. El primero se realizó a expertos en simulación social en la conferencia internacional de simulación social en Barcelona, ESSA 2014⁶. El segundo tuvo lugar en Brasil.

El lenguaje SociAALML se presentó en el track de Desarrollo de Software Dirigido por Modelos de las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015) [48].

Por último, se estudió la viabilidad de emplear SociAALML y PHAT en enfermos de Alzheimer [26], concluyendo que el lenguaje permite modelar ciertos aspectos pero otros no. Esto es debido, fundamentalmente, a que el párkinson afecta al aparato locomotor y el Alzheimer a la cognición.

Tipo	Título	Acrónimo	Rank	Ref.
Capítulo (1)	"Multi-Agent Based Social Simulation Applied to Validation of Location Services"	-	-	[83]
Revistas (4)	"Testing Context-Aware Services based on Smartphones by Agent Based Social Simulation"	JAISE'13	1,063 (Q2)	[29]
	"Opportunistic Control Mechanisms for Ambience Intelligence World"	ESA'14	2,24 (Q1)	[37]
	"Development of Sensor Based Applications for the Android Platform: an Approach Based on Realistic Simulation"	ADCAIJ'13	-	[24]
	"Domain independent regulative norms for evaluating performance of assistive solutions"	PMC'16	1,719 (Q2)	[50]
Congresos (7)	"Simulation based software development for smart phones"	ISAmI'12	-	[23]
	"Opportunistic Sensor Interpretation in a Virtual Smart Environment"	IDEAL'12	C	[42]
	"PHAT: Physical Human Activity Tester"	HAIS'13	C	[28]
	"SociAALML: Lenguaje de Modelado para Escenarios de Inteligencia Ambiental"	JISBD'15	-	[48]
	"Achieving Parkinson's disease patient autonomy through regulative norms"	PAAMS'15	-	[46]
	"Modelling and Simulation of Alzheimer's Disease Scenarios"	ANT'16	-	[26]
	"A Multiple Data Stream Management Framework for Ambient Assisted Living Emulation"	KES IIMSS'16	C	[51]
Demos (4)	"Agent based simulation for creating ambient assisted living solutions"	PAAMS'14	-	[25]
	"A framework for developing multi-agent systems in ambient intelligence scenarios"	AAMAS'15	A*	[27]
	"SociAAL: Un entorno 3D para la simulación de soluciones AAL"	DTR4AAL'15	-	[47]
	"Building Prototypes Through 3D Simulations"	PAAMS'16	-	[49]

Tabla 1.1: Tabla resumen de las publicaciones que avalan la tesis.

La recopilación de todas las publicaciones en las que ha contribuido el candidato se encuentran agrupadas por tipo su tipo en la Tabla 1.1. Además, por cada trabajo se muestra el título, el acrónimo donde se publicó el trabajo (revista o conferencia), el ranking que ocupa dicha revista o conferencia y la referencia a la misma. Los criterios de valoración establecidos para las revistas difieren de aquellos empleados para los congresos. En el caso de las revistas, estas se rigen por un factor de impacto (JCR)⁷ que a su vez las agrupa en cuartiles de más a menos relevancia: Q1, Q2, Q3 y Q4. En el caso de los congresos, los criterios de valoración están menos estandarizados. Se han tomado los del portal CORE⁸, el cual se rige por varios criterios de valoración y también agrupa los congresos por importancia. Pero a diferencia de los cuartiles de los JCR, estas agrupaciones no son proporcionales. Las categorías de congresos de mayor a menor relevancia son: A*(4%), A(14%), B(26%), C(51%) y otros (5%).

⁶ESSA 2015: <http://www.bsc.es/essa/home-page>

⁷Herramienta de Análisis del JCR®: <https://www.recursoscientificos.fecyt.es/servicios/indices-de-impacto>

⁸Portal CORE de conferencias: <http://portal.core.edu.au/conf-ranks/>

1.6. Organización de la memoria de la tesis

Este trabajo de investigación se encuentra organizado en los siguientes capítulos:

- **Capítulo 2 - Estado del arte.** Comienza presentando los elementos principales que forman parte de los sistemas AmI. Después realiza un análisis de las propuestas existentes en cuanto a simuladores de entornos virtuales donde desplegar dichos sistemas. El capítulo finaliza realizando un estudio acerca de los lenguajes enfocados a modelar los elementos del sistema, en especial el comportamiento de los sujetos.
- **Capítulo 3 - El lenguaje de modelado.** Define el lenguaje específico de dominio SociAALML, el cual está diseñado para el modelado de escenarios domésticos de enfermos de párkinson y sus familiares. Para ello, introduce la herramienta INGENME y su meta-meta-modelo, ya que a partir de los cuales se crea el meta-modelo que define el lenguaje. El lenguaje se presenta por aspectos. Cada aspecto consta de una introducción, una propuesta de meta-modelo, la notación y ejemplos de modelos y, por último, una evaluación de la propuesta.
- **Capítulo 4 - El marco de dominio: PHAT.** Describe la arquitectura del simulador PHAT y las funcionalidades que incorpora para poder simular escenarios realistas de la vida diaria de los enfermos de párkinson y sus familiares en sus hogares. Una vez presentado el lenguaje y el simulador, se procede a explicar el generador de código. Es decir, el proceso por el cual, a partir de las especificaciones creadas con SociAALML se generan las simulaciones basadas en PHAT.
- **Capítulo 5 - Casos de estudio.** Intenta demostrar a través de dos casos de uso que las herramientas propuestas en este trabajo permiten: (1) modelar los escenarios de interés de la vida diaria de los enfermos de párkinson, (2) validar dichos modelos mediante su simulación 3D y (3) evaluar y validar las soluciones AAL en dichas simulaciones.
- **Capítulo 6 - Conclusiones.** Presenta las conclusiones obtenidas en esta tesis y las líneas futuras de trabajo a las que ha dado lugar.

Capítulo 2

Estado del arte

No puedes sobrevivir si no conoces el pasado.

Oriana Fallaci

RESUMEN: Este capítulo comienza presentando los elementos principales de un sistema AmI e identificando aquellos que serán simulados. Posteriormente, se establecen los requisitos para la plataforma de simulación y para el lenguaje de modelado. Basándose en dichos requisitos, se realiza un estudio de las plataformas de simulación y de los lenguajes de modelado que se consideran mejores candidatos.

En el capítulo anterior se ha podido comprobar que los elementos de un sistema AmI pueden ser tanto software (aplicaciones y servicios), como hardware (dispositivos con sus sensores y actuadores) y, también, sujetos (los usuarios). Una simulación por ordenador consiste en recrear el comportamiento de un modelo [95]. En nuestro caso nos interesa reproducir el comportamiento de los elementos para generar el contexto que alimentará las aplicaciones del sistema a través de los sensores virtuales. Los elementos software ya definen por sí mismos su comportamiento y, por tanto, no es necesario recrearlo. Por el contrario, aquellos elementos físicos, como el entorno, los dispositivos (sensores y actuadores) y los usuarios, necesitan tanto un modelo de comportamiento como una representación visual.

La propuesta para generar las simulaciones es dirigida por modelos. Así que se necesita principalmente un lenguaje para definir los elementos y una plataforma de simulación que sirva de base para generar las simulaciones a partir del lenguaje. Para ello, primero (en la sección 2.1) se analizará más en detalle las características de los elementos del sistema que se quiere modelar. Una vez se tiene una idea más clara de los elementos del sistema, en la sección 2.2 se establecen los requisitos principales para modelar y simular dichos elementos. Dado que la simulación de entornos inteligentes no es nueva, en la sección 2.3 se hace un repaso a las propuestas ya existentes con el ánimo de poder utilizar alguna de ellas. Se verá que son muy pocas las propuestas que

están disponibles en la red para que la comunidad pueda contribuir. Y las pocas que se ofrecen no cumplen con los requisitos. Por ello, en la sección 2.4 se hace un estudio de motores de juegos y plataformas más genéricas que sirvan de punto de partida sobre las que construir el nuevo simulador. En la sección 2.5 se aporta información de las entrevistas que realizaron los sociólogos: el objetivo, el tipo de entrevista, la metodología y los puntos que abordaban. Por último, en la sección 2.6 se presentan los lenguajes de modelado considerados para crear las especificaciones de los escenarios de la vida diaria de los enfermos de párkinson y sus familiares.

2.1. Los elementos principales de un sistema AmI

Tal y como se avanzó en la introducción, no existe un estándar sobre el desarrollo de sistemas de Inteligencia Ambiental. Este estándar sería de utilidad para ayudarnos a identificar los elementos más importantes de un sistema de este tipo. No obstante, existen varios trabajos que abordan esta tarea de identificación [10, 143, 61]. Todos ellos coinciden en que un sistema AmI debe constar al menos de: un entorno inteligente (o ubícuo), un middleware, los usuarios, los servicios y las aplicaciones.

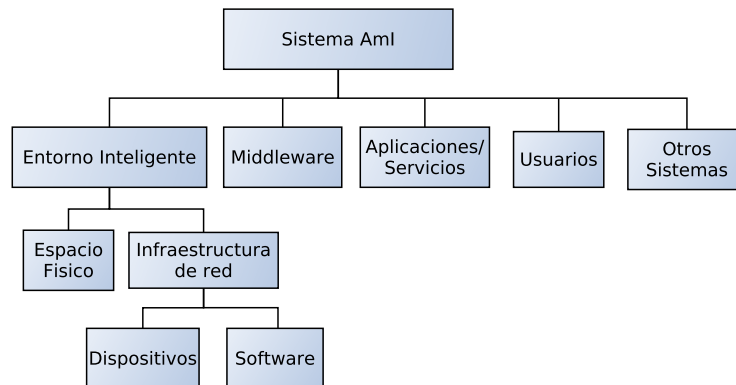


Figura 2.1: Elementos principales de un Sistema de Inteligencia Ambiental.

La Figura 2.1 muestra un diagrama que corresponde a una posible organización de los elementos de un sistema AmI. El uso de este tipo de representación es común en Ingeniería de Sistemas. Cada caja es un elemento del sistema que, a su vez, puede contener más elementos. Un elemento representado en el diagrama no indica que este tenga que crearse necesariamente. Por ejemplo, los usuarios no se crean, así como tampoco puede serlo algunos elementos hardware y software. En la parte superior del diagrama se encuentra el sistema AmI. Este se compone de uno o varios entornos inteligentes, un middleware, aplicaciones y servicios, usuarios, y otros sistemas que sean necesarios (e.g. un servicio que se encargase de controlar la medicación de un enfermo podría necesitar tener acceso al sistema sanitario).

El *entorno inteligente* se puede considerar en sí mismo un sistema. En este trabajo se entenderá el entorno inteligente en el mismo sentido que Augusto [100], quien lo

define como la infraestructura física que da soporte al sistema. Este se compone de dos elementos principales, por un lado, del espacio físico y, por el otro, de la infraestructura de red.

El *espacio físico* es uno de los elementos fundamentales del sistema pues corresponde al lugar donde se desplegará el sistema y donde los usuarios realizan sus actividades. Se considera que el espacio físico comprende tanto los elementos estructurales, así como los muebles y otros objetos relevantes para el sistema. Como se avanzó en la introducción, los estudios sobre sistemas AmI se suelen centrar en un dominio y en un espacio físico concreto. Lo mismo ocurre en este trabajo, ya que este se va a centrar en las actividades diarias de los enfermos de párkinson en sus hogares. Y, por tanto, el espacio físico se restringe a la vivienda del enfermo. Este elemento del sistema muy raramente se construirá. Sin embargo, sí es habitual que los enfermos adapten su vivienda para facilitar su autonomía. Por ejemplo, cambiar la bañera por el pie de ducha, o instalar asas donde sujetarse. Conocer la distribución de la vivienda, y algunos de los objetos que esta contiene, podría ser información importante a tener en cuenta para el diseño de un sistema AAL.

La *infraestructura de red* se compone de dispositivos interconectados (incluyendo su software o firmware), los cuales se despliegan en el espacio físico para obtener información del entorno y actuar sobre el mismo. Puede que el espacio físico ya disponga de algún tipo de infraestructura. Por ejemplo, una red Wifi y algunos teléfonos inteligentes. No obstante, seguramente la solución AmI requerirá otros nuevos, como por ejemplo, sensores de presión o de puerta. Normalmente, en un sistema AmI los dispositivos se suelen clasificar en sensores y actuadores. Los *sensores* son aquellos dispositivos capaces de medir magnitudes físicas del entorno. Estos datos son procesados por los servicios y las aplicaciones del sistema para inferir información o contexto. Por ejemplo, puede haber un sensor de presión instalado en una silla. Si detecta presión, las aplicaciones pueden asumir que hay alguien sentado. Si es por la mañana y se sabe que la silla corresponde a una de las que se encuentran en la cocina, además, se podría asumir que ese alguien está desayunando. Esta asunción podría reforzarse si se sabe que previamente se abrió la puerta del frigorífico, gracias a un sensor de puerta instalado en el mismo. Al fin y al cabo, los sensores son fundamentales para que el sistema pueda inferir qué actividad o tarea está realizando el usuario en cada momento para poder tomar las acciones oportunas. Inferir qué actividad está realizando el usuario no es sencillo y, de hecho, la línea de investigación **Activity Recognition** [11] se dedica a ello. Los *actuadores* son aquellos dispositivos que son capaces de modificar el entorno. Por ejemplo, un aparato de aire acondicionado puede modificar la temperatura de una habitación y una pantalla puede ofrecer información visual a un usuario. Para que los actuadores y los sensores resulten útiles, estos deben estar accesibles a los servicios y las aplicaciones. La línea de investigación **Sensor Networking** [62] se dedica precisamente al estudio de grandes infraestructuras de sensores. Estas infraestructuras crean una topología dinámica a través de unos protocolos de red propios, de forma que facilitan el acceso a los sensores y su mantenimiento. A este tipo de dispositivos simples que contienen o bien un sensor o un actuador y algún mecanismo para que el siste-

ma se comunique con él, se le suele llamar mota. Estos son relativamente económicos y se emplean para “sensorizar” una vivienda. Por otro lado, los llamados dispositivos inteligentes (e.g. relojes, pulseras y teléfonos) también cobran relevancia en el diseño de este tipo de sistemas. Estos dispositivos incorporan un conjunto de sensores (GPS, luminosidad, acelerómetro, giroscopio, etc.), actuadores (altavoz, pantalla, vibración, etc.) y gran capacidad de interconexión a través de Bluetooth, WiFi, NFC, 3G y 4G. Además, estos dispositivos suelen ser personales y cada vez son más comunes entre la población, sobre todo los teléfonos inteligentes. Actualmente, en España, el 71 % de la población adulta posee un teléfono inteligente [113]. Esto hace que resulte interesante aprovecharlos como infraestructura de red en un sistema doméstico AAL para ahorrar costes. No obstante, no son adecuados para emplearlos, por ejemplo, como sensor de presión debajo de una silla pues resultaría bastante más costoso que emplear una mota. Sin embargo, en términos de simulación esta diferencia de coste no se produce. Por lo que sí se podría emplear para la evaluación del sistema AAL. En esta tesis el foco principal está en este tipo de dispositivos y concretamente en aquellos que llevan instalado el popular sistema operativo Android. Las razones de esta elección se verán más adelante.

El *middleware* de un sistema AmI es un elemento software que puede ayudar a esconder la complejidad y heterogeneidad de la infraestructura de red, facilitar la gestión de los recursos (descubrimiento de servicios y gestión de contexto) y ofrecer una API (*Application Programming Interface*, Interfaz de programación de aplicaciones) para el desarrollo de aplicaciones y servicios [117]. La infraestructura de red puede ser muy compleja y estar compuesta por gran variedad de dispositivos. Permitir que los servicios y aplicaciones accedan directamente a los dispositivos puede no ser una buena decisión de diseño. Tanto el formato de los datos que manejan estos dispositivos, como la forma de obtenerlos pueden variar de un dispositivo a otro. Esto provoca que si, por ejemplo, se desea cambiar el fabricante de unos sensores, seguramente haya que cambiar todas las aplicaciones y servicios que dependan de dichos sensores. En lugar de usar este diseño, es más recomendable utilizar un middleware que haga de intermediario entre las aplicaciones y la infraestructura de red. El middleware trabaja con contexto en lugar de con datos. Por ello, es necesario definir un modelo del contexto que dependerá del dominio de la aplicación. El firmware de los dispositivos se debe modificar para hacerlos compatibles con el middleware si estos no lo son. Si resulta complicado o imposible modificar el firmware de los dispositivos, entonces se puede crear uno o varios adaptadores. Estos adaptadores estarían conectados a los dispositivos y se encargarían de hacer la traducción de datos a contexto. En este caso, si se cambia un tipo de dispositivo, entonces solo hay que modificar el firmware o crear un nuevo adaptador. En el ámbito del AAL, dos middleware bastante relevantes son OpenAAL¹ y UniversAAL². En cualquier caso, una tarea fundamental en el desarrollo de un sistema AmI es decidir el middleware que mejor se adapte a las necesidades del sistema a desarrollar. Aunque

¹Sitio web de OpenAAL: <http://openaal.org>

²Sitio web de UniversAAL: <http://www.universaal.org>

resulta básico y conveniente este elemento para el desarrollo de sistemas AmI, por falta de tiempo, las aplicaciones prototipo se han desarrollado directamente en Android.

Los *servicios y aplicaciones* del sistema son los elementos que implementan las funcionalidades del sistema. Estos utilizan el middleware para coordinarse enviando y recibiendo contexto. Los servicios suelen crearse para ofrecer funcionalidades a otras aplicaciones o servicios. Mientras que las aplicaciones ofrecen las funcionalidades a los usuarios. Por ejemplo, se puede pensar en una aplicación de mando a distancia para un teléfono inteligente de un enfermo de párkinson donde hay una interfaz con botones para cada acción. Dependiendo del nivel de temblores, la interfaz de la aplicación podría variar cambiando el tamaño de los botones según corresponda. Por el contrario, un servicio se puede encargar de inferir el nivel de temblores en el que se encuentra el enfermo.

Los *usuarios del sistema* son los elementos más importantes y todo se diseña pensando en ellos. Recuérdese que el objetivo de un sistema de Inteligencia Ambiental es el de ofrecer los servicios adecuados en cada momento a los usuarios de forma no invasiva y proactiva. Esto supone conocer muy bien los comportamientos y preferencias de los usuarios. Por ello, como se avanzó en la introducción, las metodologías para el desarrollo de este tipo de sistemas se suelen inspirar en la que propuso Norman [105]. Esta metodología se centra en los usuarios, de forma que estos intervienen continua y activamente en el diseño del sistema.

2.2. Requisitos generales del lenguaje y el simulador

Una vez identificados y algo más caracterizados los elementos de un sistema AmI, en esta sección se presentan los requisitos generales que se consideran deseables tanto para el lenguaje de modelado como para la plataforma de simulación. Estos requisitos van orientados a que ambas herramientas permitan a gente no experta en simulación, generar simulaciones 3D que reflejen el día a día de los enfermos de párkinson y sus familiares. Teniendo identificados estos requisitos, se podrá evaluar si son adecuadas las soluciones actuales para el caso que nos ocupa.

Tanto el lenguaje como la plataforma de simulación deberían incorporar conceptos del dominio, es decir, los elementos del sistema AmI en un entorno doméstico. Normalmente, cada elemento del sistema a simular necesita un identificador, un rol, una representación y un comportamiento. El *rol* indica el tipo de elemento, por ejemplo, una silla, una mesa o un humano. El *identificador* es fundamental para poder diferenciar unos elementos de otros, sobre todo si estos tienen el mismo rol. Una buena *representación de los objetos* permite diferenciarlos y dar realismo durante la simulación 3D. Todas las plataformas de simulación utilizan los modelos 3D como medio de representación de los objetos. Estos modelos 3D se suelen crear con herramientas específicas de modelado como Blender³. Sin embargo, la creación de modelos con este tipo de herramientas es una tarea especializada y costosa. Por ello, una alternativa a su

³Sitio web de Blender: <https://www.blender.org/>

creación es reutilizar modelos, bien de forma gratuita o comprándolos. Crear este tipo de representaciones desde el lenguaje, por ejemplo una vivienda, resultaría muy complicado. El lenguaje debería más bien permitir especificar, por ejemplo, qué modelo 3D de vivienda se desea emplear en la simulación. El *comportamiento de los elementos* es el más determinante para dotar de realismo a la simulación. Este trabajo se centra en este aspecto de los elementos. En las plataformas de simulación, la especificación de los comportamientos de los objetos puede variar. Algunas plataformas ofrecen lenguajes propios o scripts para especificar dichos comportamientos.

A continuación, se describe brevemente, para cada tipo de elemento del sistema a simular, qué se entiende por su representación y su comportamiento. Además, se dan pistas acerca del soporte que podría ofrecer una plataforma de simulación para implementar estos aspectos y de cómo lo modelaría el lenguaje:

- *El espacio físico.* Su representación corresponde a la vivienda, los muebles y los objetos. Su comportamiento viene dado principalmente por la física, el sonido y la iluminación. En las plataformas de simulación estos comportamientos se especifican principalmente modificando propiedades, por ejemplo, el peso de un objeto o el factor de rozamiento. Por tanto, sería suficiente con que el lenguaje recogiese dichas propiedades.
- *La infraestructura de red.* Este trabajo se centrará fundamentalmente en dispositivos inteligentes cuya representación puede ser suficiente empleando un cubo plano con una pantalla (si la ofreciese). El comportamiento es la parte que más interesa. Este corresponde al de los sensores, los actuadores y la propagación de la señal de las comunicaciones (e.g. Wifi o Bluetooth). Estos elementos permiten a las aplicaciones reales interaccionar con el mundo virtual. Dependiendo de cómo se implementen estos elementos, las simulaciones resultarán más o menos realistas. Por ejemplo, sería conveniente introducir modelos de ruido a la hora de generar los valores de los sensores virtuales. Simular el comportamiento de los sensores y actuadores es la parte más específica de la simulación y, por ello, las plataformas de propósito general no incorporan estos elementos. Se considera que, al igual que con el espacio físico, la plataforma debería implementar el comportamiento de los sensores y actuadores. El lenguaje debería ser responsable de configurar y desplegar dichos sensores. Por ejemplo, para un sensor de presencia, el lenguaje permitiría especificar el alcance del sensor, frecuencia de muestreo, posición y orientación.
- *Los sujetos.* Son los elementos más complejos. Su representación 3D corresponde al cuerpo de un humano el cuál lleva asociado un esqueleto y unas animaciones para representar sus acciones (e.g. andar o sentarse). Sería interesante poder variar en gran medida su representación en base a la edad, la altura o el peso y, además, poder seleccionar la ropa. Esto facilitaría diferenciar unos sujetos virtuales de otros durante la simulación. El comportamiento de los sujetos virtuales viene dado por la física y por una secuencia de acciones. Las acciones se pueden generar de forma autónoma o pueden venir dadas por un sujeto real a través del

teclado y el ratón. En el primer caso, se dice que el sujeto virtual es un bot o un agente y, en el segundo, que es un avatar. En este trabajo se persigue generar el comportamiento de los agentes de forma que realicen las acciones que correspondan a un escenario dado. Es importante que la plataforma ofrezca un conjunto de acciones adecuadas para reflejar el comportamiento de los enfermos de párkinson y sus familiares en sus hogares. Igualmente, el lenguaje debe permitir especificar las actividades de los agentes a lo largo del día, las limitaciones debidas a la enfermedad y sus respuestas ante determinados eventos.

El middleware, las aplicaciones y los servicios son elementos del sistema AmI que no se simulan. No obstante, la plataforma de simulación debe permitir que estos se puedan conectar a la simulación para que así puedan interaccionar con el mundo virtual. Resulta conveniente que dicha interacción sea transparente desde el punto de vista del software. Para ello, se podrían aplicar diferentes opciones: a nivel del emulador modificándolo, a nivel de contexto a través del middleware o con soluciones a medida específicas. Las diferencias entre uno y otro se verán en la sección dedicada a la plataforma (Capítulo 4). Por ahora, simplemente decir que, si los dispositivos lo permiten, los autores consideran que el uso de su emulador es la opción más acertada.

De forma específica, en cuanto al lenguaje, se persigue principalmente que facilite a cualquier persona relacionada con el sistema AAL entender y especificar modelos. Para ello, como se avanzó en la introducción, se propone un lenguaje específico del dominio, es decir, que en su vocabulario se traten términos cotidianos de la vida diaria de los enfermos de párkinson en su hogares. Y, también, que su notación sea visual y siga los principios de Moody [96]. Estos principios establecen unas pautas para crear notaciones cognitivamente eficientes, es decir, fáciles de aprender y entender. Estos principios se ven en el apartado siguiente.

2.2.1. Los principios para el diseño de notaciones visuales

Las *notaciones visuales* se usan en todas las etapas del proceso de desarrollo del software, desde la ingeniería de requisitos hasta el mantenimiento. Estas juegan un rol particularmente crítico en la comunicación con los usuarios finales y los clientes. Las notaciones visuales transmiten información de manera más efectiva que el texto, sobre todo a usuarios con formación no técnica [6]. Además, los diagramas pueden transmitir información de forma más concisa [38] y precisa que el lenguaje ordinario [14, 75]. También, la información representada visualmente es más propensa a ser recordada por el efecto de superioridad de imagen [52].

Moody [96] define un conjunto de principios para el diseño de notaciones visuales. El objetivo de estos principios es diseñar notaciones visuales cognitivamente eficientes. La *eficacia cognitiva* se define como la velocidad, facilidad y precisión con la que una representación puede ser procesada por una mente humana [75]. Resulta curioso que este aspecto tan importante en las notaciones, no se haya tenido en cuenta a la hora de diseñar UML [97].

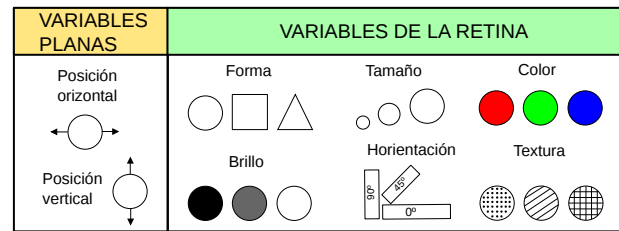


Figura 2.2: Variables visuales [14]: definen un conjunto de técnicas gráficas elementales para la construcción de notaciones visuales.

Los principios para el diseño de notaciones visuales efectivas cognitivamente son [96]:

1. **Principio de claridad semiótica (Semiotic Clarity):** Debe haber una correspondencia 1:1 entre las construcciones semánticas y los símbolos gráficos. Si este principio no se cumple, pueden surgir cuatro anomalías. La *redundancia simbólica* ocurre cuando es posible utilizar más de un símbolo gráfico para representar el mismo concepto semántico. La *sobrecarga simbólica* es el efecto inverso, se utiliza un mismo símbolo para representar dos o más conceptos. El *exceso simbólico* aparece cuando se crean símbolos que no corresponden a ningún concepto semántico. Y, por último, el *déficit simbólico* se produce cuando existen conceptos que no tienen asignada una representación simbólica.
2. **Principio de discriminabilidad perceptual (Perceptual Discriminability):** Símbolos diferentes deben ser fácilmente distinguibles. La discriminabilidad está principalmente determinada por la distancia visual entre símbolos. Esta se mide por el número de variables visuales en las que se diferencian, ver Figura 2.2, y el tamaño de esas diferencias. No obstante, la variable visual más importante es la forma y se debe usar como mecanismo principal para distinguir entre construcciones semánticas diferentes. Además de la forma, se recomienda emplear codificación redundante, es decir, usar más de una variable visual como puede ser el color. De esta forma, una entidad se reconoce tanto por el color como por la forma. En los lenguajes es común incluir texto para diferenciar las relaciones, sin embargo es una forma cognitivamente ineficiente de representación. El texto tiene una distancia visual igual a cero. No obstante, este es una forma efectiva de distinguir entre instancias de símbolos pero no para tipos de símbolos.
3. **Principio de transparencia semántica (Semantic Transparency):** Seleccionar representaciones visuales cuya apariencia sugieran su significado. Las representaciones semánticamente transparentes reducen la carga cognitiva porque tienen incorporados mnemotécnicos: su significado puede ser percibido directamente o fácilmente aprendido. Los iconos son más apropiados que los símbolos abstractos para las notaciones visuales donde los usuarios se están iniciando. Esto se debe a que los iconos generan diagramas más comprensibles y atractivos.

Los iconos se encuentran de manera generalizada en interfaces gráficas de usuario y en cartografía pero raramente se usa en ingeniería del software [96].

4. **Principio de gestión de la complejidad: Incluir mecanismos explícitos para lidiar con la complejidad.** La gestión de la complejidad hace referencia a la habilidad de una notación visual para representar información sin sobrecargar la mente humana. En este contexto, la complejidad equivale al número de elementos que tiene un diagrama. Este principio no lo cumplen algunos de las notaciones visuales de la ingeniería del software (e.g. Modelo Entidad-Relación o i^*). Para representar efectivamente situaciones complejas, las notaciones visuales proporcionan mecanismos como la modularización y las estructuras jerárquicas. La modularización divide el sistema en partes más pequeñas. La jerarquía permite representar los sistemas en diferentes niveles de abstracción.
5. **Principio de integración cognitiva: Incluir mecanismos explícitos para soportar la integración de información procedente de diferentes diagramas.** Representar sistemas utilizando múltiples diagramas provoca una demanda cognitiva adicional para el lector. Este esfuerzo adicional se debe a que el lector tiene que integrar la información de diferentes diagramas mentalmente y recordar dónde se encuentran. Para solventar esto, las notaciones deben incorporar mecanismos explícitos para soportar integración conceptual y perceptual. El primer tipo de integración consiste en agregar información de diferentes diagramas en una representación mental coherente del sistema. Los diagramas que recogen esta agregación se llaman *diagramas de contextualización*. El segundo tipo de integración se consigue facilitando la navegación y la transición entre diagramas. Un ejemplo de este tipo de integración son los *mapas de navegación*.
6. **Principio de expresividad visual: Aplicar correctamente las capacidades de las variables visuales.** La *expresividad visual* se define como el número de variables visuales que emplea una notación. Si esta no incluye ninguna variable visual, se dice que no tiene expresividad visual. Por el contrario, si las incluye todas, se dice que la notación está saturada visualmente.
7. **Principio de codificación dual: Incluir texto para complementar los gráficos.** De acuerdo con la teoría de la codificación dual, utilizar texto y gráficos conjuntamente para transmitir información es más efectivo que emplear cada uno por separado. Además, de acuerdo con el principio de contigüidad espacial, incluir anotaciones (explicaciones en forma de texto) en los propios diagramas es mucho más efectivo que tenerlas de forma separada.
8. **Principio de economía gráfica: El número de símbolos gráficos diferentes debe ser cognitivamente manejable.** La complejidad gráfica se define como el número de símbolos gráficos en una notación, es decir, el tamaño de su vocabulario visual [104]. Cada vez que se introduce un símbolo nuevo en la notación se reduce su eficacia cognitiva. La capacidad humana para discriminar entre

alternativas perceptiblemente distintas es de entorno a 6 categorías [93]. Esto se aplica cuando la representación consta solo de una variable visual, por ejemplo, la forma. Sin embargo, incluir el número de variables visuales (e.g. colores) y, sobre todo, el empleo de iconos amplía esta limitación en gran medida.

9. **Principio de ajuste cognitivo: Disponer de diferentes dialectos visuales según la situación.** El *ajuste cognitivo* es el resultado de la interacción entre la representación, la tarea y la persona. La mayoría de las notaciones solo ofrecen una representación visual para todo. Sin embargo, en el contexto de la ingeniería del software, existen, al menos, dos situaciones donde es conveniente crear múltiples dialectos visuales. La primera es relativa a las diferencias en cuanto a las habilidades de modelado entre el experto y el no experto. Esta diferencia sugiere que al menos haya un dialecto visual para expertos (pro) y otro para no expertos (lite). La otra situación surge cuando resulta necesario hacer uso de la notación tanto para bocetos a mano como para diseños finales en ordenador. Una notación que contenga colores, iconos y formas 3D resulta tediosa su reproducción a mano. Además, su correcta representación dependería en gran medida de las habilidades de sus usuarios a la hora de dibujar. Por ello, este principio sugiere un dialecto para cada tarea, uno para bocetos a mano y otro para ordenador.

Para resumir, tanto el simulador como el lenguaje deben incorporar los elementos del sistema AmI, o facilitar su creación. El lenguaje debe permitir especificar o configurar la representación y el comportamiento de los elementos virtualizados del sistema. La plataforma debe implementar la semántica de lo especificado con el lenguaje. Además, de forma específica, el simulador debe:

1. Facilitar una interfaz para conectar las aplicaciones y los servicios del sistema AAL a la simulación.
2. Ser una solución implementada disponible gratuitamente en la web para poder reutilizarla y de código abierto.

Y el lenguaje debe:

1. Incorporar el concepto de evento o interrupción. Estos son necesarios para modelar situaciones donde un sujeto interrumpe su actividad para atender a un evento, por ejemplo, que el paciente se ha caído.
2. Facilitar la reutilización de los comportamientos. Para ello, se considera que las acciones ofrecidas por el lenguaje deben poder parametrizarse y que debe existir elementos para hacer referencia a comportamientos previamente definidos.
3. Facilitar a cualquier persona relacionada con el sistema AAL entender y especificar modelos con el lenguaje. Es decir, que el lenguaje siga los principios de Moody.

4. Disponer de un editor para crear las especificaciones y un motor que reproduzca la semántica de las especificaciones creadas con el editor. Si no, al menos debe contener la información suficiente para que a partir de las especificaciones se genere la simulación automáticamente.

2.3. Plataformas de simulación de entornos AmI

Esta sección presenta el análisis realizado a las propuestas de simulación obtenidas tras la realización del estado del arte. Todas estas propuestas se centran en crear entornos virtuales para evaluar sistemas AmI o entornos inteligentes. El análisis se basa en comprobar en qué medida y cómo cumplen con los requisitos vistos en la sección anterior. Es decir, qué soporte ofrecen las propuestas para dotar de representación y de comportamiento a los elementos virtualizados del sistema AmI.

Concretamente, por cada propuesta se ha recogido: el nombre, el año de publicación de la misma, información sobre qué aspectos del modelado del espacio físico, los dispositivos y los sujetos virtuales y cómo los soportan y, por último, los mecanismos que incorporan para conectar el sistema AmI a la simulación.

En cuanto al espacio físico, se comprueba si la representación del mismo es en 2D o en 3D. Además, interesa saber si la propuesta ofrece alguna herramienta para crear dicha representación, por ejemplo, un editor de viviendas. Referente al comportamiento, se comprueba si la plataforma ofrece algún soporte para simular propiedades del entorno, por ejemplo, la física o la temperatura. Además, se indica con qué herramienta, o de qué forma, se facilita la especificación del comportamiento del espacio físico.

En lo relativo a los dispositivos, también se ha prestado atención a qué tipo de representaciones (2D o 3D) y de comportamientos ofrecen las plataformas. En el caso de la especificación de los comportamientos, esta puede venir dada por código o por algún modelo. Al igual que con el espacio físico, se ha comprobado si la plataforma ofrece alguna herramienta para crear tanto la representación como el comportamiento de los dispositivos. Para el caso de los dispositivos, se ha considerado si la plataforma ofrece algún mecanismo para simular la red. Esta característica podría formar parte del comportamiento de los dispositivos. Entre otras cosas, el simulador de red podría indicar la potencia con la que llega la señal WiFi a los dispositivos. Este nivel de potencia dependería de la posición de la fuente, de la del receptor y de los obstáculos que se hallasen entre ambos.

Los sujetos a simular es el último elemento del sistema AmI a tener en cuenta. La representación de un sujeto virtual puede ser en 2D a través de una imagen o en 3D gracias a un modelo. A su vez, un modelo 3D de un sujeto virtual puede ser estático o dinámico. Se considera que un modelo 3D es estático si no incorpora animaciones. Normalmente, durante la simulación, el modelo simplemente se desliza por el entorno. Un modelo 3D es dinámico si, por el contrario, dispone de animaciones. En cuanto al comportamiento de los sujetos, se identifican dos aproximaciones bien diferenciadas. Cuando el sujeto es un avatar o es un bot. En el primer caso, el sujeto virtual es

controlado por un sujeto real y, por tanto, su comportamiento depende de las acciones del real. En el segundo caso, el comportamiento del sujeto virtual viene dado por un agente, que de forma autónoma decide las acciones a realizar en cada momento. Esta última opción es más compleja que la primera. El comportamiento depende de la capacidad de percepción y lo sofisticado que sea la implementación del agente. La especificación del comportamiento del agente se puede realizar mediante codificación o a través de un lenguaje específico.

Por último, a lo largo de las propuestas se identifican tres tipos de mecanismos para conectar las aplicaciones y servicios del sistema AmI a la simulación. A través del middleware, a través del emulador y/o utilizando otras propuestas específicas. En este trabajo se considera que el uso del emulador es el mecanismo más atractivo para conectar las aplicaciones al sistema AmI. Este garantiza la transparencia y no requiere modificar el código de la aplicación cuando se desea pasar a pruebas reales. Así se minimizan los errores que puedan surgir al hacer los cambios en el código. El inconveniente es que requiere modificar el código del emulador o poder acceder al mismo mediante algún mecanismo.

En la Tabla 2.1 se encuentra resumida la información que se ha extraído de los trabajos científicos que tratan sobre las plataformas de simulación de entornos inteligentes. En la primera columna se muestra el nombre de la plataforma de simulación y en la siguiente el año de su publicación. En la tercera columna se encuentra el tipo de (1) representación (R) y de (2) comportamiento (C) del espacio físico que ofrece la plataforma, indicando, para cada caso, si se propone o facilita una herramienta para su creación. La cuarta columna corresponde a los dispositivos y muestra, al igual que el espacio físico, (1) el tipo de representación y de (2) comportamientos con sus herramientas y, además, indica si ofrece un (3) simulador de red. La quinta columna corresponde a los mecanismos empleados para conectar al software del sistema AmI a la simulación. Se diferencian tres opciones: (1) a través del middleware, (2) utilizando el emulador de cada tipo de dispositivo y (3) con otro mecanismo propuesto por los autores. Por último, la columna sexta recoge el soporte que ofrecen las plataformas para simular a los sujetos. Por un lado, se muestra la (1) representación (R) diferenciando entre 2D/imágenes, 3D estático (sin animaciones) y 3D con animaciones. Por otro lado, se muestra si ofrece (2.1) avatares, es decir, que los sujetos virtuales puedan ser controlados por sujetos reales, y también si dispone de (2.2.) Bots indicando cómo se especifica el comportamiento y si se dispone de alguna herramienta.

Si se analiza la información recogida en la Tabla 2.1, se pueden extraer varias conclusiones:

- Atendiendo al año de publicación de las propuestas, se puede comprobar cómo estas se han ido produciendo desde el año 2002 hasta el 2014. Esto nos indica que es un problema abierto el cual todavía no se ha solucionado.
- La mayoría de los simuladores están enfocados a recrear mundos virtuales en 3D (20 de 26). Las representaciones en 3D aportan mayor realismo.

Plataforma de simulación	Año	Espacio físico 1. R/Herramienta 2. C/Herramienta	Dispositivos 1. R/Herramienta 2. C/Herramienta 3. Simulador de red	Interconexión 1. Middlewa- re 2. Emulador 3. Otro	Sujetos 1. R/Herramienta 2.1. Avatar/ Herramienta 2.2. Bots/ Herramienta
GLS [126]	2002	1. ?/? 2. ?/?	1. ?/? 2. Código/?	1. QoS- DReAM	1. ?/? 2.2. Código/?
QuakeSim [21]	2002	1. 3D/GtkRadiant 2. Física/Quake	1. 3D/? 2. Código/?		1. 3D animado 2.1. 1ª persona
UbiWise [12]	2003	1. 3D/GtkRadiant 2. Física/Quake	1. 3D/MilkShape 2. XML and Java/ WISE	1. Context Toolkit	1. 3D animado 2.1. 1ª persona
TATUS [109]	2004	1. 3D/Hammer 2. Física/Half-Life	1. 3D/? 2. Código Wireless	3. Proxy	1. 3D animado 2.1. 1ª persona 2.2 Bots/ GUI
3DSim [101]	2005	1. 3D/? 2. ?/?	1. 3D/? 2. Código	1. UPnP	1. 3D animado 2.1. 3ª persona
GST [120]	2006	1. 2D/? 2. Código/Layer API	1. 2D/? 2. Código/Reference Layer	2. RCEU	1. 2D/? 2.2. Código/Pipeline de filtros
UbiREAL [103]	2006	1. 3D/? 2. Magnitudes fisi- cas/Sim. Externo	1. 3D/? 2. Código/? 3. Capa TCP/IP		1. 3D/? 2.2. Pos. y act./ GUI
UbiBuilding [106]	2007	1. 2D/? 2. ?/?	1. 2D/? 2. Código/?	1. UPnP	1. 2D/? 2.1. Static/ ?
SitCom [35]	2008	1. 3D/? 2. ?/?	1. 3D/USUed 2. Código	1. SitMod	1. 3D/? 2.2. Pos., act./ USEd
V-PlaceSims [79]	2008	1. 3D/PlaceMaker 2. ?/?	1. 3D/? 2. Script/PlaceLab		1. 3D animado /Place- Lab 2.1. / V-PlaceSims
ISS [138]	2009	1. Imágen/? 2. Código/?	1. Imágen/? 2. Código/?	1. Context Server	1. Imágen/? 2.1. 3ª persona/ GUI
SFSL [114]	2009	1. 3D/Second Life 2. ?/Second Life	1. 3D/Second Life 2. LSL Script/ Se- cond Life	3. Twiwn- World- Mediator	1. 3D animado/Second Life 2.1. 1ª persona/ clien- te Second Life
SIMACT [17]	2010	1. 3D/SketchUp 2. ?/?	1. 3D/? 2. XML Script/ SI- MACT	3. Base de da- tos	?
SHSim [77]	2010	1. Imágen/? 2. Código/?	1. Imágen/? 2. Código/?	1. OSGi	1. Imágen/? 2.1. Pos. y act./ GUI
APEX [131]	2010	1. 3D/OpenSim 2. ?/OpenSim	1. 3D/OpenSim 2. CPN/ CPN Tools	3. Módulo de comunicación	1. 3D animado /Open- Sim 2.1. 3ª persona/ clien- te OpenSim
SHCFramework [141]	2011	1. 3D/OpenSim 2. ?/OpenSim	1. 3D/OpenSim 2. LSL script	3. Base de da- tos	1. 3D animado /Open- Sim 2.1. 3ª persona/ clien- te OpenSim
InSitu [108]	2011	1. 3D/Hammer 2. Física/TATUS	1. 3D/? 2. Código/?	3. Update Router	1. 3D animado/Ham- mer 2.1. 1ª persona 2.2. Pos. y Dur./?
DIEToolChain [39]	2011	1. 3D/ SweetHo- me3D 2. Física/SH3D	1. 3D/? 2. Código/?	1. ROS[115] 2. Android	
Persim3D [57]	2012	1. 3D/SH3D 2. Física/?	1. 3D/? 2. Código/?	3. Base de da- tos	1. 3D animado 2.2. Bot/ Basado esce- narios previos.
PerDE [134]	2012	1. 3D/OpenSim 2. Física/OpenSim	1. 3D/OpenSim 2. PerDL/Generador de código	1. BASE[13]	1. 3D animado/Open- Sim 2.1. 3ª persona/ clien- te OpenSim
DiaSim [20]	2013	1. 2D/DiaSim Edi- tor 2. Stimulis/DiaS- pec	1. 2D/DiaSim 2. Código/ DiaSpec	1. DiaSuite[15] 2. Generado con DiaSpec	1. 2D 2.1. Código/ DiaSpec
DAI Virtual Lab [43]	2013	1. 3D/? 2. Física/?	1. 3D/? 2. DAI Macros/ GUI	1. DAI[22]	2.1. 1ª persona/ GUI
UbikSim [128]	2013	1. 3D/UbikEditor	1. 3D/UbikEditor 2. Código	1. OCP[102]	1. 3D estático/ UbikE- ditor 2.2 HAB/-
UbikMobile [29]	2013	1. 3D/UbikEditor	1. 3D/UbikEditor 2. Código	1. OCP[102] 2. Librería Android	1. 3D estático/ UbikE- ditor 2.2 HAB/-
SimCon [87]	2014	1. 3D/Hammer 2. Física/TATUS	1. 3D/? 2. Modelo/SimCon- fig	3. Contums	1. 3D animado/? 2.1. 3ª persona 2.2. Pos. y Dur.
VR-Smart Ho- me [55]	2014	1. 3D, IFC/? 2. ?/?	1. 3D/? 2. SmartObjects/?		2.1. 1ª persona/ GUI
Leyenda: R→Representación; C→Comportamiento; Pos.→Posición; Acc.→Acciones; Dur.→Duración; GUI→GUI (<i>Graphical User Interface</i> , Interfaz gráfica de usuario)					

Tabla 2.1: Tabla de simuladores de entornos AmI.

- Respecto a la representación del espacio físico, algunas plataformas incorporan herramientas para ayudar a crear los modelos 3D. Pero solo DIEToolChain, UbikSim y UbikMobile ofrecen una específica para viviendas. Todas ellas basada en SweetHome3D⁴, una herramienta de código abierto enfocada al diseño de interiores.
- Algo menos de la mitad de los simuladores (11/26), incorporan motores físicos. Esta parte nos resulta interesante para poder simular las caídas de los enfermos de párkinson.
- En cuanto a los dispositivos, algunos incluyen herramientas para añadir dispositivos al escenario, pero no son realmente para crearlos. Es decir, no se puede editar la forma y a la misma vez dotarlos de sensores y actuadores. En cuanto a los mecanismos para especificar el comportamiento de los dispositivos, la mayoría se realizan a través de scripts o código. Las excepciones son APEX (utiliza modelos de Redes Coloreadas de Petri), PerDE, DiaSim, DAI Virtual Lab y SimConfig (estos utilizan lenguajes propios). El módulo de simulación de red solo lo incorporan los simuladores TATUS y UbiREAL.
- La forma de interconectar el software con el entorno virtual es variado y cuatro de ellos no lo especifican. El método más utilizado es a nivel del middleware, 13 de los simuladores soportan esta opción. Esta es una opción bastante sencilla y que puede ser útil en la mayoría de los casos. El segundo método más empleado son soluciones a medida. Resultan convenientes cuando se quiere trabajar a nivel más bajo, por ejemplo, con datos en vez de con contexto. Por último, el menos usado es el que los autores consideran el más interesante, aquel que emplea el Emulador. Esta baja popularidad es normal ya que muchos de los simuladores están enfocados a la fase de diseño donde se crean prototipos y no se decide los dispositivos concretos donde se desplegarán.
- Relativo a la representación de los sujetos virtuales, bastantes simuladores soportan modelos 3D animados. Destacar aquellos basados en OpenSim, ya que las aplicaciones cliente permiten configurar la apariencia del avatar fácilmente seleccionando los tipos de recursos disponibles (e.g., el pelo, ropa, tono de la piel, entre otros).
- En cuanto al comportamiento de los sujetos, resultan interesantes para la tesis aquellas plataformas que facilitan el uso de Bots o Agentes. Existen algunas propuestas que sí lo soportan y ofrecen algún tipo de herramienta para especificar dichos comportamientos. No obstante, excepto UbikSim y UbikMobile, el resto ofrecen unos comportamientos muy básicos los cuales se basan simplemente en especificar una secuencia de acciones que el agente ejecuta a lo largo del tiempo.

Ahora bien, atendiendo a las propiedades extraídas de los simuladores y los requisitos establecidos, se va a seleccionar aquella propuesta que se considera la más adecuada

⁴Sitio web de SweetHome3D: <http://www.sweethome3d.com/>

para los propósitos de este trabajo de tesis. Si se pone el foco en aquellas propiedades que son más difíciles de implementar (los sujetos animados 3D y la física) y que se consideran fundamentales para aportar realismo a las simulaciones, la lista de plataformas se reduce a menos del tercio: QuakeSim, UbiWise, TATUS, InSitu, Persim3D, PerDE y SimCon. Si además se consideran aquellas que soportan el comportamiento de los sujetos a través de Bots, las opciones se reducen a TATUS, InSitu, Persim3D y SimCon. Pero se debe tener en cuenta que SimCon, InSitu son versiones de TATUS creadas por el mismo grupo de investigación y cuyas plataformas no se encuentran disponibles en la red. Por tanto, la comunidad científica y los desarrolladores no pueden aprovecharse ni contribuir en las mismas como es nuestra intención. Persim3D es un simulador que tampoco está disponible en la red y que sus comportamientos son generados a partir de valores de sensores de escenarios reales. Esta forma de especificar los comportamientos no es la que se busca. No obstante, Persim3D destaca por el trabajo realizado a la hora de ofrecer y modelar sensores virtuales para percibir el entorno virtual.

Los simuladores que se han encontrado accesibles a través de la web son: UbiREAL⁵, SIMACT⁶, DAI Virtual Lab⁷ y UbikSim⁸. Pero ninguno de estos dispone de un motor físico para simular caídas de los sujetos virtuales. Por tanto, de las 26 propuestas de simulación estudiadas, se considera que ninguna es adecuada para ser seleccionada como plataforma de simulación. Así que en el siguiente apartado se estudian motores de juegos y otras plataformas más generales en las que basar nuestra propuesta de simulador.

2.4. Plataformas de simulación genéricas

Tras comprobar que no es posible aprovechar ninguna de las propuestas de simuladores AmI estudiadas, en esta sección se exploran las alternativas que se han considerado más relevantes para facilitar el desarrollo de nuestro propio simulador. Estas alternativas son los motores de juegos, las plataformas de mundos virtuales y otras alternativas más específicas.

2.4.1. Los motores de juegos

Un motor de juegos es una colección de módulos que ni especifican directamente el comportamiento del juego (lógica del juego) ni el entorno del juego (nivel de datos). Estos elementos se organizan en capas como es común en sistemas de gran escala. Debido a esta complejidad, hay un gran interés en promover su reutilización y en encontrar la mejor manera de organizar correctamente sus elementos. Esto ha llevado a crear motores de juegos específicos para un juego, pero con la flexibilidad suficiente como para poder reutilizarlo en el desarrollo de otros juegos de la misma familia [80]. Los

⁵Sitio web de UbiREAL: <http://ubireal.org/>

⁶Código fuente de SIMACT: <https://sourceforge.net/projects/simact/>

⁷Sitio web de DAI Virtual Lab: <http://web.ua.es/en/dai/dai-virtual-lab.html>

⁸Sitio web de UbikSim: <https://github.com/emilioserra/UbikSim/wiki>

motores de juegos normalmente vienen acompañados de un conjunto de herramientas que facilitan su creación, conocido como SDK (*Software Development Kit*, Kit de Desarrollo de Software), y de un componente de ejecución donde corren los componentes del juego.

Game-Specific Subsystem			
Front End	Gameplay Foundations		
Visual Effects			
Scene Graph	Skeletal Animation	Online Multiplayer	Audio
Low-Level Renderer		Collision & Physics	HID
Resource Manager			
Core System			

Figura 2.3: Versión simplificada de la típica arquitectura de los elementos de un motor de juegos de Gregory [53].

En la Figura 2.3 se puede ver la arquitectura de los componentes que intervienen en el funcionamiento de un juego en tiempo de ejecución. Analizando las funcionalidades que ofrece cada componente, se puede comprobar cómo muchos de ellos son útiles para crear simulaciones convincentes:

- *Resource Manager*. El gestor de recursos proporciona una interfaz unificada para acceder a cualquier tipo de activo del juego. Los activos son los recursos multimedia y modelos 3D que se usan para componer la escena 3D de la simulación, por ejemplo, la vivienda, los sujetos virtuales y los dispositivos.
- *Rendering Engine*. El motor de renderizado es uno de los componentes más complejos y caros de cualquier motor de juegos. Este motor incluye un renderizador de bajo nivel, un grafo de la escena para organizar los elementos visuales, un generador de efectos visuales (e.g. sistemas generadores de partículas o de iluminación) y una interfaz para facilitar la creación de la GUI del juego. Estos componentes de renderizado permiten generar visualizaciones 3D de la simulación más realistas. De esta forma, cuanto más realista sea la visualización 3D, más fácil le podría resultar a un usuario validar el escenario. Además, este componente define el concepto de cámara para, a partir del cual, generar la escena 3D. Al permitir definir varias cámaras, esto facilita enormemente simular el sensor de cámara.
- *Collision and Physics*. Normalmente, la física y la detección de colisiones están estrechamente relacionados. Estos facilitan la creación de simulaciones más dinámicas y realistas donde los elementos tangibles (e.g. objetos, sujetos y muros) se comportan de acuerdo a las leyes de la física. Esto nos facilita, por ejemplo, simular las caídas de los sujetos en distintas condiciones e implementar sensores virtuales tales como el acelerómetro y el giroscopio.

- *Skeletal Animation*. La animación por esqueleto permite que los sujetos reproduzcan movimientos previamente guardados. Estos movimientos se componen de rotaciones y traslaciones de los huesos. Estas animaciones se consideran muy interesantes para poder representar las acciones de los sujetos. De forma que, mediante la visualización 3D de la simulación, el observador pueda reconocer a simple vista qué está haciendo el sujeto en cada momento y cuál es su estado. Por ejemplo, si el paciente de párkinson tiene temblores en una mano, el sujeto virtual debe reproducir esos temblores.
- *Human Interface Devices (HID)*. Los dispositivos de interfaz humana permiten al motor de juegos procesar las entradas que provengan del jugador. Las entradas en nuestro caso por parte del observador de la simulación son pasivas y se limitan a navegar por el escenario, parar la simulación o tomar una captura de pantalla.
- *Audio*. El audio es un componente muy valorado en los juegos ya que ayuda a que el jugador se sumerja en el mundo virtual del juego. En nuestro caso, el audio es interesante en varios sentidos. A nivel de dispositivo debe poder crearse un sensor micrófono que perciba los sonidos en función de su posición y teniendo en cuenta las atenuaciones. En relación con los sujetos, permitir que estos puedan hablar resulta interesante. Así, estos podrían, por ejemplo, interactuar con las aplicaciones a través interfaces de voz. En relación con el entorno, para dotarlo de realismo y así ayudar al observador a entender el escenario, es también importante el ruido ambiente. Por ejemplo, el ruido de los pasos al andar o el ruido de los platos y del agua al fregar.
- *Online Multiplayer*. El módulo de multijugador online permite que varios jugadores puedan participar al mismo tiempo en un mismo mundo virtual pero cada uno desde su propio ordenador. Esta funcionalidad es interesante cuando se mezclan sujetos virtuales que son controlados por usuarios reales (avatares) con aquellos controlados por programas (bots o agentes). En este trabajo se considera solo la opción del control por agentes, así que este módulo, en principio, no resulta de interés.
- *Gameplay Foundation System*. Este módulo es la capa que une el código relativo a la jugabilidad (es decir, las acciones que se pueden realizar en el juego) con los elementos del motor de juegos anteriores. Este módulo suele variar dependiendo de la familia de juegos a la que esté enfocado el motor. Los motores de juegos de propósito general que no están enfocados a ningún juego no tienen esta capa. En nuestro caso, esta capa debería contener elementos para facilitar la creación y gestión de los activos del simulador (tanto su lógica como su representación). Los tipos de activos deberían ser los elementos del sistema AmI vistos en el apartado 2.1: la vivienda, los dispositivos y los sujetos (enfermos y familiares).
- *Game-Specific Subsystem*. Este componente contiene el código específico del juego. En el caso de la propuesta planteada en este trabajo, este módulo se generaría

automáticamente a partir de las especificaciones realizadas con el lenguaje de modelado (SociAAML).

Por tanto, los motores de juegos parecen ser un buen punto de partida para desarrollar entornos virtuales dinámicos y realistas ya que ofrecen soporte para la física, la iluminación, el audio, la animación de sujetos, entre otros [28]. Trenholme y Smith [135] también apoyan esta idea. Ellos afirman que los motores de juegos modernos pueden ser unas herramientas beneficiosas en la investigación de entornos inteligentes debido a que son componentes software probados rigurosamente para dar un buen rendimiento, confiables y fáciles de usar.

Uno de los primeros juegos que se reutilizaron para tareas de investigación fue el Quake III Arena⁹. Este era un videojuego de tipo FPS (First-Person Shooter) multijugador que además permitía una gran cantidad de modificaciones. Estas posibilidades las aprovecharon los simuladores QuakeSim [21] y UbiWise [12] permitiendo a varios usuarios, cada uno desde su ordenador, conectarse en un mismo escenario virtual para evaluar el sistema de Inteligencia Ambiental. UbiWise además reutilizó la funcionalidad de las armas. Estas fueron sustituidas por dispositivos como por ejemplo el prototipo de una cámara de fotos con conexión WiFi. En este caso, el usuario en vez de disparar balas hacía fotos. Casi al mismo tiempo, otro motor de juego llamado Half-Life¹⁰ fue también empleado para estos propósitos. Primero por el simulador TATUS [109] y recientemente por SimCon[87]. Como ya se ha dicho, realmente, SimCon es una extensión de TATUS.

Ambos motores, Quake III Arena y Half-Life, ofrecían mundos virtuales 3D con sistemas de iluminación y audio. También, los dos eran gratuitos y, al menos, el código fuente de sus SDKs era código abierto (no los componentes de ejecución). Sin embargo, ambos están desfasados y carecen de motores físicos y otras nuevas características que dan realismo a la simulación.

Existen propuestas de simuladores de entornos virtuales que utilizan motores de juegos modernos. Los motores en los que se basan las propuestas analizadas son Source Engine¹¹, Unity¹², y jMonkeyEngine¹³. Todos ellos son motores de juegos de propósito general y tienen características más avanzadas: física, efectos visuales y de partículas, animaciones de sujetos, entre otros:

- *Source Engine* es desarrollado por Valve y es fruto de las lecciones aprendidas en Half-Life. Se utilizó para desarrollar el videojuego Half-Life 2 entre otros. El simulador InSitu [108] emplea este motor.
- *Unity* es un motor de juegos comercial muy popular. En el momento de la elección de motores, la versión gratuita venía con funcionalidades limitadas. Sin embargo,

⁹Sitio web de Id Software: <http://www.idsoftware.com/>

¹⁰Sitio web de la comunidad de desarrolladores Valve: <https://developer.valvesoftware.com/>

¹¹Sitio web de Source Engine, Valve: <https://developer.valvesoftware.com/>

¹²Sitio web de Unity 3D: <http://unity3d.com/>

¹³Sitio web de jMonkeyEngine: <http://jmonkeyengine.org/>

en estos momentos es gratuita la versión profesional. En cualquier caso, tanto el SDK como el entorno de ejecución son de código cerrado y no se pueden modificar. Los investigadores creadores de los simuladores UbiSim [76] y Persim3D [57] optaron por este motor.

- *jMonkeyEngine (jME)* es de código libre (tanto el entorno de ejecución como el SDK) y está codificado en Java. Tiene una comunidad de desarrolladores muy activa que están continuamente mejorándolo. Su licencia es BSD y, por tanto, se puede emplear para cualquier propósito. Los autores del simulador SIMACT[17] eligieron este motor como punto de partida.

2.4.2. Plataformas de mundos virtuales

El desarrollo de un simulador a partir de un motor de juegos es una tarea dura. Por ello, otros autores han desarrollado sus simuladores basándose en plataformas de simulación de mundos virtuales. Estas plataformas alojan los mundos en servidores y los usuarios pueden explorarlos a través de sus avatares. Además, los usuarios pueden modificar su propio entorno creando o comprando objetos en tiempo de ejecución. Estos pueden, por ejemplo, editar el terreno o crear una vivienda. El problema es que la creación de los objetos es a partir de elementos muy básicos tales como cubos y esferas. No obstante, permite importar objetos 3D ya creados.

Como se puede ver, este tipo de plataformas parecen muy adecuadas para crear simuladores donde los usuarios, a través de sus avatares, evalúan entornos de Inteligencia Ambiental. Para ello, debería incluirse objetos de tipo dispositivo a los cuales se les pueda añadir sensores y actuadores virtuales. Estos dispositivos deben poder conectarse al exterior (a emuladores o a un middleware) para que las aplicaciones del sistema AmI puedan interaccionar con el entorno y los avatares.

*Second Life*¹⁴ es el precursor de este tipo de plataformas, desarrollado en 2004 y activo desde entonces. Second Life proporciona una API (Application Programming Interface) desde su propio lenguaje llamado “Linden Scripting Language”. A cualquier objeto virtual se le puede asociar uno o más scripts para controlar el comportamiento del objeto. Toda la creación del entorno, objetos y su comportamiento se puede realizar de forma remota desde clientes. Gracias a esta accesibilidad, en 2009, Prendinger et al. propusieron un simulador basado en Second Life [114] que se llamará SFSL. Diseñaron un componente Twin-World-Mediator cuya principal misión era conectar el sistema de Inteligencia Ambiental con el mundo virtual. Los autores identificaron limitaciones en el uso de esta plataforma. Por un lado, había un retardo de 0.2 segundos en el movimiento de los objetos y, por el otro, una limitación en la memoria para cada script de 16 KB. Según ellos, esta restricción de memoria hace difícil implementar y probar de forma efectiva algoritmos más complejos. Por ello, como trabajo futuro plantearon migrar su simulador a la versión de código abierto, *OpenSimulator*¹⁵, que por aquel

¹⁴Sitio web de Second Life: <http://secondlife.com/>

¹⁵Sitio web de Open Simulator: <http://opensimulator.org/>

entonces estaba naciendo. En este caso, la única limitación de memoria la establece la máquina donde esté corriendo el servidor de *OpenSimulator*. El lenguaje con el cual se programa la plataforma es C#. Los simuladores APEX [131] y SHCFramework [141] están basados en OpenSim.

2.4.3. Otras plataformas

Se han encontrado propuestas para facilitar la implementación de los comportamientos de los agentes en entornos virtuales. Estas plataformas, se encargan de detalles de bajo nivel como, por ejemplo, los conflictos entre las animaciones. El objetivo es que los investigadores puedan centrarse en el modelado del comportamiento de los sujetos y no en este tipo de detalles de bajo nivel.

Una de las primeras plataformas en dar este tipo de soporte se llamaba ACE (Agent Common Environment) [66]. Esta propuesta implementaba los requisitos básicos y necesarios para que un agente se desenvolviese en un mundo virtual. Para ello definía interfaces para que los agentes fuesen capaces de percibir y actuar de forma compartida, coherente y sincronizada.

ADAPT [130] es otra propuesta reciente basada en el motor de juegos Unity. Esta facilita la navegación de los sujetos virtuales por el entorno, ofrece un mecanismo para combinar animaciones asignándole pesos y la posibilidad de especificar el comportamiento de los sujetos virtuales de forma gráfica. La especificación de los comportamientos se realiza a partir de árboles de comportamiento con parámetros [129]. Este tipo de especificación se verá más adelante.

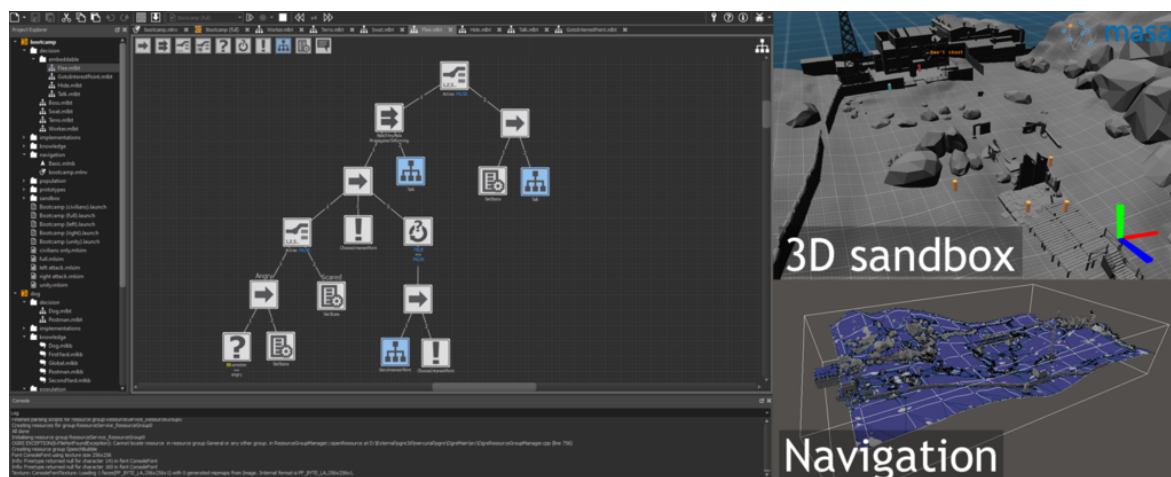


Figura 2.4: Entorno de desarrollo de MASA-Life.

También existen propuestas comerciales como MASA Life ¹⁶. Esta se define como una herramienta para crear Inteligencia Artificial, la cual se centra en el comportamiento de entidades autónomas del mundo virtual facilitando su creación y depuración. Al

¹⁶Sitio web de MASA Life: <http://www.masalife.net/>

igual que la propuesta anterior, emplea los árboles de comportamiento como lenguaje de especificación. Las especificaciones se pueden crear, modificar y depurar en tiempo de ejecución gracias a su SDK, ver Figura 2.4. MASA Life ofrece un renderizado básico del entorno virtual el cual es suficiente para depurar los comportamientos. Una vez depurados los comportamientos, la plataforma facilita mecanismos para conectar sus agentes a plataformas comerciales como Unity.

Como se puede comprobar, estas plataformas facilitan el desarrollo de agentes autónomos en el mundo virtual. No obstante, estas tienen varios inconvenientes. ACE está solo disponible para Windows y se abandonó en 2008. ADAPT está basada en Unity y, por tanto, los módulos relativos a la simulación son de código cerrado. MASA Life es comercial, tenía un coste de entorno a 500 euros al año y no es de código abierto.

2.4.4. La elección: jMonkeyEngine3

Tal y como refleja el estudio realizado, existen bastantes alternativas a la hora de elegir una plataforma a partir de la cual basar un simulador de entornos virtuales aplicado a la Inteligencia Ambiental. No obstante, el abanico de opciones se reduce drásticamente si se establecen algunos requerimientos. Principalmente, estos requerimientos son que la propuesta esté implementada y accesible de forma gratuita y, además, sea de código libre. La finalidad de estos requisitos es que el simulador a desarrollar pueda estar disponible a toda la comunidad investigadora y esta pueda tanto beneficiarse de su uso como contribuir en su mejora.

Considerando los requisitos anteriores, la lista de opciones se reduce a OpenSim y jME3. Podría pensarse que OpenSim es la mejor opción ya que está diseñado para crear mundos virtuales donde pueden interaccionar los avatares. No obstante, al estar precisamente OpenSim centrado en avatares, este no ofrece funcionalidades que faciliten el uso de agentes para controlar los sujetos. Por ejemplo, OpenSim no soporta “pathfinding”, es decir, dados dos puntos del entorno obtener un camino coherente que los una. Además, el soporte que ofrece para el sonido es muy básico. El nivel de realismo que se puede alcanzar en esta plataforma parece inferior. Por ejemplo, la posibilidad de insertar física es relativamente reciente. Otro inconveniente hace referencia a la necesidad de tener que lidiar con la arquitectura cliente-servidor. Finalmente, basar el simulador en esta plataforma requeriría un esfuerzo adicional ya que habría que familiarizarse tanto con el lenguaje C# como con el lenguaje script de OpenSim.

En cambio, jME3 es un motor de juegos de propósito general que está desarrollado 100 % en Java, lenguaje de programación con el cual los autores están muy familiarizados. Tiene un buen sistema de sonido basado en la librería OpenAL¹⁷ y una buena física gracias al motor Bullet¹⁸. Además, ofrece un generador de maya de navegación para que los agentes puedan utilizar el “pathfinding” para desplazarse por el escenario. jME3 tiene una comunidad muy activa, buena documentación con muchos tutoriales, y un SDK integrado en el IDE Netbeans con plugins para ampliar su funcionalidad. Por

¹⁷Sitio web de OpenAL: <https://www.openal.org/>

¹⁸Sitio web de Bullet: <http://bulletphysics.org/>

ejemplo, en la Figura 2.5 se puede ver el SDK del jME3 donde se ha generado la malla de navegación tras cargar el modelo 3D de una vivienda. En el Capítulo 4, dedicado al diseño de nuestro simulador PHAT, se puede encontrar más información sobre jME3 y sobre cómo se ha creado el simulador a partir de este motor de juegos.

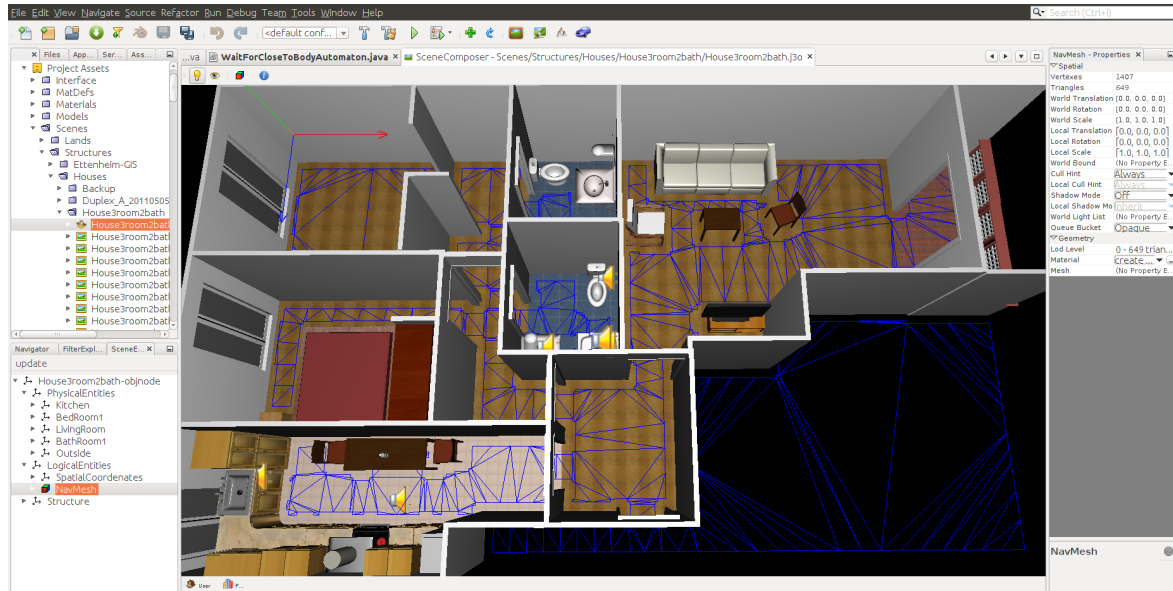


Figura 2.5: Captura de pantalla del SDK de jME3 donde se puede ver la maya de navegación generada.

2.5. Las entrevistas a los enfermos de párkinson

Tal y como avanzó en la contextualización del trabajo, una de las primeras tareas del proyecto SociAAL fue la de recabar información de primera mano acerca de los enfermos de párkinson y sus familiares.

La selección de los sujetos se realizó en base al estadio al que pertenecía el enfermo de párkinson. Según Hoehn y Yahr estos son cinco [60], los cuales se caracterizan por el nivel de discapacidad que manifiesta el paciente:

- *Estadio 1 (E1)*: Los síntomas se manifiestan sólo en un lado del cuerpo, normalmente, produciendo muy poca o ninguna discapacidad funcional.
- *Estadio 2 (E2)*: Los síntomas se manifiestan en ambos lados del cuerpo, pero se conserva el equilibrio.
- *Estadio 3 (E3)*: Incapacidad moderada. El paciente manifiesta problemas de equilibrio aunque es capaz de estar de pie y andar sin ayuda. Algunas actividades las realiza con dificultad y dependiendo del empleo que tenga podrá seguir desempeñándolo.

- *Estadio 4 (E4):* Incapacidad grave. La enfermedad se ha desarrollado por completo. Todavía puede andar y permanecer de pie sin ayuda pero con más dificultad. La incapacidad para realizar las actividades cotidianas aumenta.
- *Estadio 5 (E5):* Confinado en la cama o en silla de ruedas. El paciente es totalmente dependiente.

Las entrevistas las realizaron los sociólogos del proyecto a enfermos de párkinson cuya evolución de la enfermedad se encontraba entre mitad y avanzada (estadio E3 y E4). De acuerdo con expertos consultados, los enfermos en estos estadios son los que más se podrían beneficiar de algún tipo de asistencia tecnológica para mejorar su autonomía. Esto se debe a que los pacientes sólo son dependientes en determinadas situaciones puntuales. Lo cual resulta más factible que estos puedan dejar de ser dependientes, o menos dependientes, con algún tipo de ayuda tecnológica. Por ejemplo, si el paciente se queda parado de forma que no puede iniciar la marcha, hacer sonar música puede ayudarles a proseguir, tal y como manifiesta uno de los pacientes de las entrevistas.

Las entrevistas fueron diseñadas y realizadas por los sociólogos Millán Arroyo y Lucila Finkel, con ayuda de la asociación Parkinson Madrid. El objetivo de las entrevistas era conocer el estado de los pacientes y sus familiares, sus limitaciones y su posición frente a las soluciones tecnológicas. La metodología que emplearon consistió en reunirse con el enfermo y su pareja. Primero se realizaban las preguntas al enfermo y, posteriormente, a la pareja. El tipo de entrevista era estructurada pero abierta, es decir, la entrevista estaba programada con una serie de preguntas encaminadas a conversar con el entrevistado en la misma secuencia y con sus mismas palabras. Además, se permitía que el otro sujeto hiciese matizaciones o comentarios. Los puntos que se abordaron durante la entrevista fueron:

- *Información personal general:* la edad, estudios, ocupación y cuestiones relativas al entorno familiar (e.g. con quién vive y si tienen hijos).
- *Inicios de la enfermedad:* la edad de inicio y los primeros síntomas.
- *Otros problemas de salud:* e.g., tensión o artrosis.
- *Descripción de las actividades diarias:* piden que describan lo que los sujetos hacen a lo largo de un día, haciendo especial hincapié en aquellas tareas que les resultan más complicadas.
- *Síntomas y medicación:* qué síntomas sufren, cuál le afecta más en su día a día, cómo gestionan las tomas de los medicamentos y si les supone algún problema.
- *Asistencia:* reciben ayuda por parte de familiares o personal contratado.
- *Las relaciones sociales:* mantienen el contacto con sus amigos, reciben o realizan visitas.

- *Relación de los sujetos con la tecnología:* cómo se manejan con dispositivos tales como el teléfono, el mando a distancia o el ordenador.
- *Sugerencias de soluciones tecnológicas:* ideas por parte del sujeto acerca de cómo la tecnología podría hacerle la vida más fácil. Aquí también se manifestaba la aptitud del sujeto hacia las tecnologías.

Los sociólogos realizaron un total de 17 entrevistas, las cuales fueron grabadas y, más tarde, transcritas con el consentimiento de los sujetos e informándoles del uso que se haría de dicha información donde se respetaría siempre el anonimato. Estas transcripciones son las que se han empleado en este trabajo como fuente de conocimiento para, por un lado, complementar el entendimiento del dominio y, por el otro, para el diseño del lenguaje. El objetivo es que se pueda formalizar dicho conocimiento a través del lenguaje SociAALML y, posteriormente, se vea reflejado en una simulación 3D de forma automática. Esto se comprobará en el primer caso de estudio, apartado 5.1. En este apartado se muestra cómo se extrae y formaliza toda la información relativa a la entrevista del paciente E9. No obstante, se demostrará que el nivel de detalle dado en las entrevistas no es suficiente para recrear las actividades diarias de los sujetos en una simulación, ya que tampoco era este el objetivo de las entrevistas.

En el siguiente apartado se presentan algunos lenguajes que se podrían considerar para el modelado, pero teniendo en cuenta que dicho lenguaje debe, además, permitir a un generador crear la simulación de forma automática.

2.6. Lenguajes de modelado

Una de las partes más importantes de la tesis está relacionada con el diseño de un lenguaje de modelado específico de dominio. Con este lenguaje se persigue poder especificar formalmente situaciones de la vida diaria de los enfermos de párkinson y sus familiares y, además, que a partir de dichas especificaciones se puedan generar automáticamente simulaciones 3D.

En esta sección se presentan y analizan lenguajes de modelado que se han considerado relevantes. Para ello, el trabajo se centra en la idoneidad de los mismos para modelar el comportamiento de los enfermos de párkinson. Estos lenguajes van desde el dominio de los motores de juegos hasta el dominio de la AAL, pasando por lenguajes de agentes y de especificación de sistemas.

2.6.1. Lenguaje de modelado AAL: HCM-L

Michael y Mayr [86, 90] proponen un lenguaje HCM-L (Human Cognitive Modeling - Language) para recoger el conocimiento sobre las actividades diarias de las personas mayores y describir el posible conjunto de acciones que siguen para realizar las actividades. La finalidad del lenguaje es que dichas especificaciones sirvan como base de conocimiento a un sistema AAL para asistir a sus usuarios cuando quieren realizar una

tarea y no recuerdan bien los pasos a seguir, por ejemplo, reproducir un DVD en la televisión. El lenguaje tiene una notación visual que sigue los principios de Moody y un editor [4] basado en la plataforma de meta-modelado ADOxx¹⁹. De esta forma, y al igual que en esta tesis, los autores persiguen que un usuario no experto sea capaz de entender los diagramas e incluso crearlos.

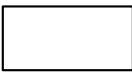
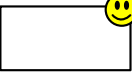
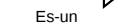
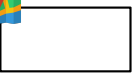
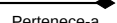
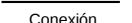
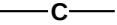
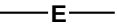
Elementos básicos	Cosa	Conexión	Operación
 Cosa	 Persona	 Es-un	 Operación con pre y post condición
 Operación	 Lugar	 Pertenece-a	
 Conexión	 Atributo	 Propiedad	
 Flujo		 Llamada	 Operación con suboperaciones
		 Participación	
		 Ejecución	

Figura 2.6: Símbolos del lenguaje HCM-L [90].

El lenguaje define el concepto de unidades de comportamiento que, a su vez, se componen de operaciones. Cada operación puede tener pre-condiciones y post-condiciones indicando cómo se relacionan con otras operaciones. Además, HCM-L incluye conceptos para modelar entidades y sus relaciones. Este define el concepto Cosa (Thing) para modelar objetos físicos o abstractos del mundo, los cuales se pueden relacionar con elementos tipo Conexión (Connection). Los autores del lenguaje definen dos tipos especiales de Cosa: Persona (Person) y Localización (Location Thing). En la Figura 2.6 se puede comprobar los símbolos utilizados para cada elemento del lenguaje.

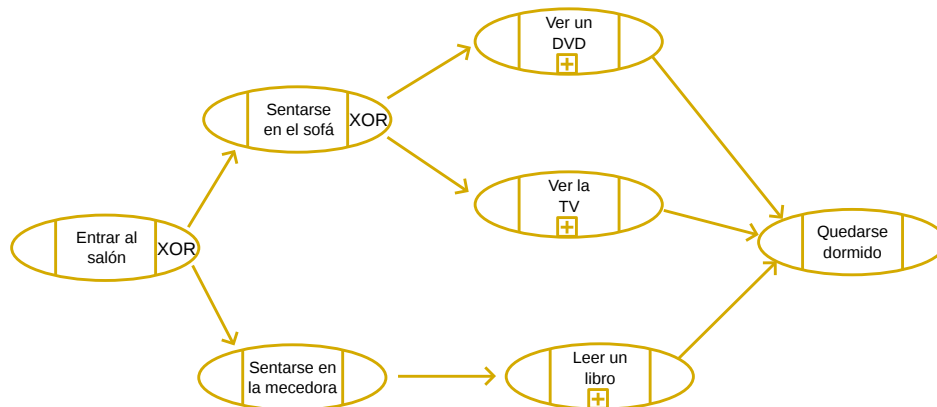


Figura 2.7: Ejemplo de modelado de las actividades de la tarde con HCM-L [90].

Si se trata de especificar alguna actividad con el lenguaje, se puede comprobar que es bastante expresivo y relativamente sencillo, no necesitando emplear muchos elementos.

¹⁹Sitio web de ADOxx: <https://www.adoxx.org>

Por ejemplo, en la Figura 2.7, se modela la actividad de pasar la tarde. Esta consiste en llegar al salón y sentarse en la mecedora para leer un libro, o sentarse en el sofá para o bien ver la televisión o bien ver un DVD. Tras realizar cualquiera de las 3 opciones, la última operación consiste en quedarse dormido. Fíjese cómo las post-condiciones XOR indican que tras realizar una operación, hay varias alternativas excluyentes. Sin embargo, no se indica si alguna tiene más probabilidad o si depende de alguna condición. Además, el lenguaje está enfocado a modelar solo el comportamiento de un sujeto y no incluye ni eventos, ni actividades cooperativas, ni interacciones. Tampoco incluye parámetros en las operaciones, por lo que hace que dichas operaciones sean menos reutilizables. Por otro lado, aunque en los trabajos se muestra el uso de un editor, este no está disponible, ni tampoco existe un motor que implemente la semántica de las especificaciones. Esto último tiene sentido porque la finalidad del lenguaje no es recrear las especificaciones, si no comprobar que se cumplen.

2.6.2. El lenguaje de modelado de UbikSim

El simulador UbikSim [128] se equipó con dos módulos de comportamiento a partir de los trabajos realizados por Campuzano et al. [30]. Los módulos se llaman HAB (*Hierarchical Automaton of Behaviours*, Comportamientos de autómatas jerárquicos) y HAI (*Hierarchical Automaton of Interactions*, Interacciones de autómatas jerárquicos). Ambos módulos se basan en una entidad principal llamada Automaton. Cada Automaton tiene una prioridad y puede tener una lista de Automaton hijos ordenados por prioridad y, a parte, un Automaton por defecto. El Automaton padre va delegando en los hijos. Cuando un hijo termina, pasa al siguiente de mayor prioridad. Si no hay más hijos, se pasa el control al Automaton por defecto si es que existe uno. Una vez se han terminado todos los Automaton hijos (y el por defecto) de un Automaton, dicho Automaton se considera terminado. En cualquier momento, se puede incluir un Automaton hijo. Si la prioridad del nuevo hijo es mayor que la del que se está ejecutando, este se interrumpe y se le da paso al nuevo hijo. Este es un mecanismo muy conveniente para incluir dinámicamente eventos en los comportamientos de los modelos. Fíjese que los Automaton se organizan formando una jerarquía donde la idea es que en cada nivel inferior se define el comportamiento con grano más fino. Existen un tipo especial de Automaton que no tiene hijos llamado *SimpleState*. Este suele corresponder a acciones simples del comportamiento.

Tras estos trabajos, en la extensión de UbikSim para soportar dispositivos móviles (UbikMobile), se propuso una notación gráfica para modelar el comportamiento de agentes basados en estos módulos [29]. Aquí solo se verá la notación dedicada al módulo HAB, para más información sobre este y el HAI consultar el artículo.

La notación se basa en un grafo dirigido que contiene los agentes involucrados en la simulación y sus comportamientos. Los tipos de nodos que puede contener el grafo son:

- *Simulaciones*. Es el nodo raíz del grafo y solo puede existir uno por grafo.

- *Agentes*. Nodos conectados a la simulación que representan a agentes con comportamientos.
- *Roles*. Nodos conectados a agentes que definen comportamientos. Un agente puede tener varios roles dependiendo de la situación. Por tanto, dependiendo de la situación se comportarán de una manera o de otra.
- *Comportamientos*. Nodos que se pueden asociar entre sí para definir comportamientos cada vez más complejos. Debe haber un nodo raíz que se asocie con un nodo de tipo rol. Cada nodo de estos corresponde a un Automaton.

Cada enlace entre dos comportamientos representa una transferencia de control del nodo origen al destino. Cuando un nodo posee más de una posible transición, estas deben ser evaluadas para decidir cuál tomar. Para ello, los enlaces se pueden etiquetar con la tupla $x.y.z$, donde x es un número natural indicando su prioridad para ser evaluado frente al resto; y la condición para que la transición se cumpla; y z corresponde al número de veces máximo que se puede transferir el control al nodo destino.

Los símbolos empleados para la notación son muy simples y no siguen los principios de Moody. Todos los elementos (Simulaciones, Agentes, Roles y Comportamientos) se representan con la forma de círculo. En la Figura 2.8, muestra un ejemplo de la notación donde se modela la misma situación recogida con HCM-L. En este caso, se crea un Automaton padre llamado “Actividades de la tarde” a partir del cual se define el resto del comportamiento. Primero se realiza el comportamiento de entrar al salón al tener mayor prioridad. Posteriormente, se comprueba la condición de leer, si esta es cierta, se realiza el comportamiento “Leer en la mecedora”. Este se compone de la acción sentarse en la mecedora y, luego, leer un libro. Si es falsa la condición de leer, entonces la de no leer será cierta (“no leer?”), y se pasará al comportamiento de entretenerse con elementos multimedia. Este comportamiento se inicia sentándose en el sofá y, posteriormente, según la condición “TV”, se realizará el comportamiento de ver la TV o la de ver un DVD. Cuando se terminan de realizar todos los comportamientos que dependen de las actividades de la tarde, se realiza aquel definido por defecto, es decir, “Quedarse dormido”.

Fíjese que en este caso ha sido necesario definir dos comportamientos adicionales: “Leer en la mecedora” y “Entretener con multimedia”, y condiciones específicas ya que no existe el operador XOR. No obstante, en esta definición, la tarea de leer, por ejemplo, se puede reutilizar en otros diagramas. Al igual que en el lenguaje HCM-L, la reutilización de los comportamientos es limitada al no disponer de parámetros. La notación es una propuesta que no se llegó a implementar, por lo que no existe editor ni generador de la notación a código para los módulos HAB y HAI.

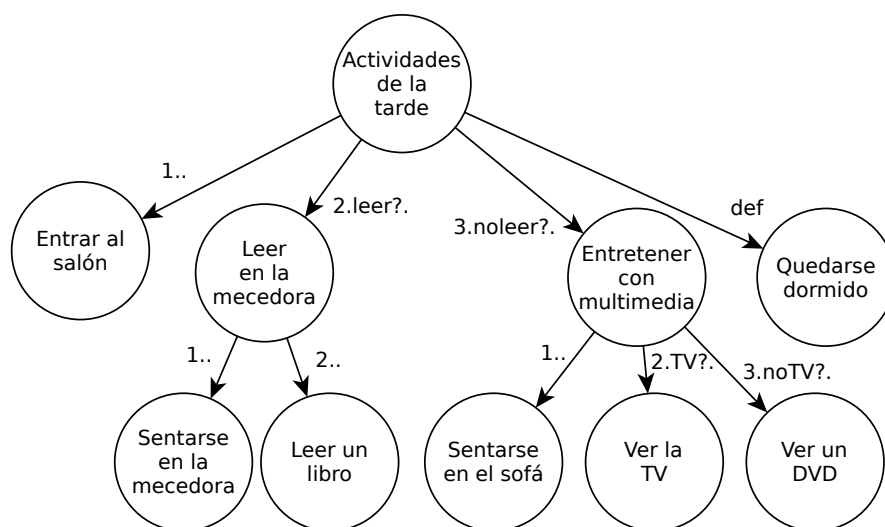


Figura 2.8: Ejemplo de modelado de las actividades de la tarde con la notación de UbikSim.

2.6.3. Especificar la IA de los videojuegos: los árboles de comportamiento

La IA de un sujeto virtual en un videojuego es aquella parte del código que toma las decisiones sobre qué acciones debe tomar el sujeto basándose en su estado interno y en el del entorno. A lo largo de la historia se han empleado varias técnicas.

La incorporación de necesidades a los sujetos virtuales es una de las técnicas más comunes en videojuegos para definir el comportamiento de los mismos [31]. Estas necesidades cambian en función del tiempo y con ellas las acciones del sujeto. La elección de la siguiente acción que debe tomar el sujeto virtual se basa en la mejor forma de satisfacer el conjunto de necesidades. Estas necesidades están bien definidas por los estudios de Abraham Maslow sobre la teoría de la motivación humana [85] donde identifica un conjunto de necesidades y las ordena de forma piramidal, de forma que en la base se encuentran las más básicas e importantes y en la cima las más complejas y menos necesarias. Sin embargo, esta técnica basada en necesidades se considera adecuada para modelar comportamientos a largo plazo, donde no se desea especificar que el sujeto realice una serie concreta de acciones. Con esta técnica, resulta complicado saber a priori la configuración de las necesidades iniciales del sujeto para que se comporte exactamente como deseamos.

Los árboles de comportamiento (BTs) [94, 32] es una técnica para especificar la IA de los sujetos virtuales que se volvió muy popular. Halo 2 [63] fue uno de los primeros videojuegos en utilizar y describir ampliamente cómo aplicó esta técnica. Según Millington [94], los BTs son una síntesis de una serie de técnicas que durante tiempo han girado en torno a la IA: HFSM, Scheduling, Planning, y Action Execution. Como se vio en la sección 2.4 dedicada a las plataformas de simulación genéricas,

las plataformas ADAPT y MASA Life también utilizan esta técnica. Además, una de las ventajas de los BTs, es que han demostrado ser una herramienta que permite a diseñadores, programadores y expertos en IA crear comportamientos complejos [94]. Así que en este apartado se verán los BTs en más detalle.

Los BTs recuerdan a las máquinas de estados jerárquicas (HFSM). La mayor diferencia entre ambos radica en el tipo de elemento principal de construcción. En vez de emplear estados, los BTs se construyen a partir de *tareas*. Una tarea tiene asignada una cantidad de tiempo de ejecución y devuelve un valor. Este valor puede ser verdadero, si la tarea se completó con éxito, o falso, en caso contrario. Las tareas se pueden combinar generando árboles de comportamiento. Estos árboles se evalúan de arriba abajo y de izquierda a derecha. Las tareas corresponden a los nodos del árbol. Estos nodos pueden ser de tipo hoja si no contienen más nodos, o de tipo rama, en caso contrario. Los BTs definen dos tipos de nodos hoja:

- *Condiciones*. Comprueban el valor o el estado de alguna propiedad y devuelven verdadero o falso según corresponda. Por ejemplo, comprobar si se observa al enfermo.
- *Acciones*. Modifican el estado de alguna propiedad. Si se consigue completar la acción, este nodo devuelve verdadero y, si no, devuelve falso. Por ejemplo, coger un objeto o desplazarse hasta una habitación.

Los nodos de tipo rama se llaman *tareas compuestas*. Los BTs pueden definir varios de este tipo. Los más básicos son:

- *Secuencias*. Devuelven un valor verdadero si y solo si todos sus hijos lo hicieron. En cuanto uno de ellos devuelve falso, la secuencia también lo hará e ignorará al resto de hijos. Las secuencias pueden modelar tareas en las que es importante realizar todas las subtareas. Por ejemplo, en la tarea de vestirse, en cierta medida es importante el orden, y que el sujeto se ponga todas las prendas.
- *Selectores*. Devuelven verdadero en cuanto uno de sus hijos lo hace, ignorando al resto. Estos son útiles cuando se quiere modelar una tarea que se puede completar de varias formas pero se tiene una preferencia. Por ejemplo, el subárbol para modelar la tarea de sentarse en el salón. Seguramente habrá varios sitios disponibles, y el sujeto tendrá preferencias, pero puede ocurrir que algunos sitios estén ocupados. Con este selector se podría poner en orden las tareas de sentarse en cada sitio ordenadas de mayor a menor preferencia. El sujeto se sentará en el primer sitio disponible que más le guste.

Además, los BTs ofrecen un tipo concreto de tarea compuesta llama *decorador*, la cual se caracteriza porque solo tiene un hijo. Su función es la de modificar el comportamiento del hijo. Se pueden definir varios tipos de decoradores. El más sencillo es el de *negación*. Si la tarea hija devuelve verdadero, el decorador devolverá falso, y a la

inversa. Otro ejemplo de decorador es el de *iteración*. La semántica de este decorador podría consistir en repetir la ejecución del hijo hasta que este consigue completar la tarea (devuelva verdadero) o se alcance un determinado número de intentos. Si se consigue la tarea, el iterador devuelve verdadero y si, por el contrario, se alcanza el número máximo de intentos, este devuelve falso.

Por último, el lenguaje define un nodo especial de tipo *referencia* (o en inglés lookup), que permite incluir en dicha posición una instancia del árbol al que hace referencia. De esta forma se gestiona la complejidad, ya que permite reducir el número de elementos por diagrama.

Fíjese que, de todos los elementos que definen el lenguaje, los dependientes del dominio son aquellos que corresponden a los nodos hoja, es decir, las condiciones y las acciones. Por ejemplo, un BT diseñado para modelar el comportamiento de un enfermo de párkinson en su casa tendrá diferentes condiciones y acciones de aquel diseñado para modelar un marine en combate.

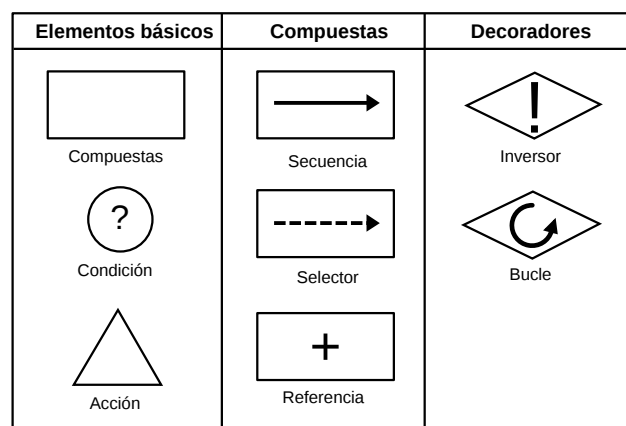


Figura 2.9: Posibles símbolos para la notación de los árboles de comportamiento.

No se ha encontrado una notación establecida o ampliamente utilizada para los BTs. Por ello, en la Figura 2.9, se muestra un ejemplo de los símbolos que se podrían emplear para los elementos que se han definido.

Continuando con el ejemplo de la actividad de pasar la tarde, la Figura 2.10 muestra el BT que corresponde a dicha actividad. A este diagrama se le han añadido unos números encerrados en unos círculos a la izquierda de los elementos secuencia y selector para facilitar la explicación. El primer selector de secuencia (1) empieza delegando en la tarea de entrar al salón. Una vez realizada, se aplica el selector (2) que a su vez depende de dos secuencias, una que equivaldría a la tarea de leer sentado en la mecedora (3) y la otra con la de entretenerse con contenidos multimedia (4). La tarea de leer depende de la condición “leer”, si esta es cierta, realiza la tarea de sentarse en la mecedora y, posteriormente, la de leer un libro. La secuencia (3) devolverá verdadero al selector (2) y, por tanto, este también acabará devolviendo verdadero a la secuencia raíz (1), la cual realizará la tarea de quedarse dormido. En caso contrario, si la condición de leer es falsa, la secuencia (3) se interrumpe y devuelve falso al selector (2). En este

caso, el selector pasará a evaluar la secuencia correspondiente a entretenerse con el multimedia (4). En este subárbol, lo primero que se realiza es la tarea de sentarse en el sofá y después el selector (5). Fíjese que se vuelve a combinar un selector (5) con uno de secuencia (6). En conjunto, son equivalentes a la sentencia de control de flujo **if... else** de los lenguajes de programación. Es decir, si es cierta la condición de ver la televisión se realiza dicha tarea, si no se realiza la de ver un DVD.

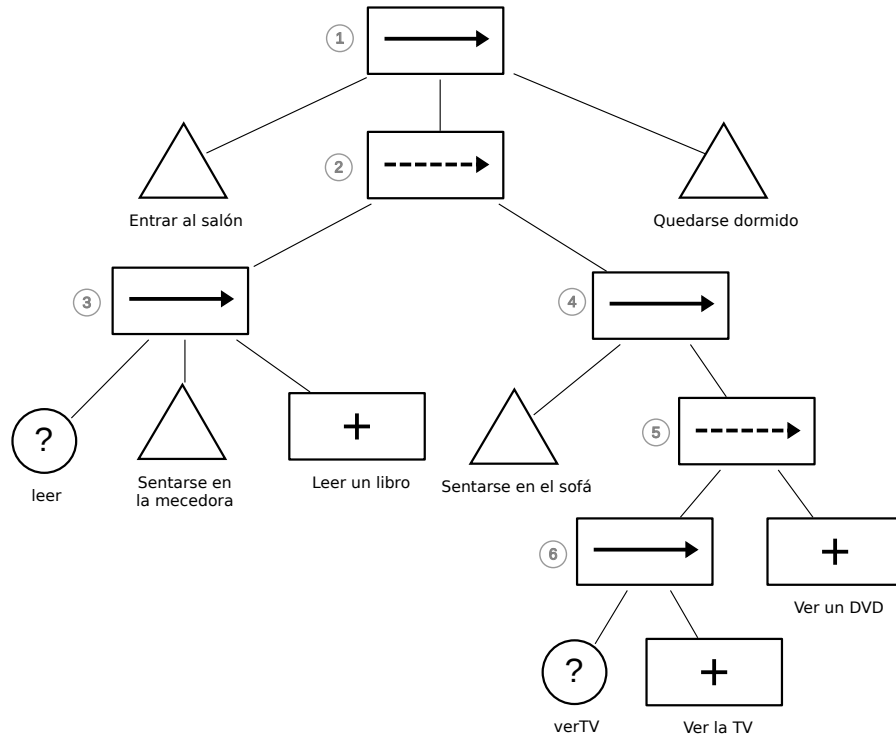


Figura 2.10: Ejemplo de modelado de las actividades de la tarde con BTs.

Los BTs no tienen notación pero al constar de pocos elementos, se ha propuesto una que sigue los principios de Moody. Si se emplease este lenguaje, sería recomendable ampliar la simbología añadiendo símbolos nuevos tanto para los tipos de tareas como para las condiciones. Al igual que en el caso de la notación para HAB, ha sido necesario crear dos condiciones, una para leer y otra para ver la televisión. Sin embargo, con los BTs ha sido necesario emplear más elementos todavía (sobre todo secuencias y selectores) para especificar lo mismo. En total, BTs ha necesitado el doble de elementos que HCM-L. Esto implica que los diagramas de los BTs contendrán por lo general más elementos y será más difícil cumplir el principio de gestión de la complejidad, aunque se disponga de nodos referencia. La reutilización de los comportamientos, al igual que en el lenguaje HCM-L, es limitada al no disponer de parámetros. Este lenguaje tampoco incluye mecanismos para reaccionar ante eventos. No obstante, existen propuestas como las de Shoulson et al. [129] y Kapadia et al. [68] donde se trata de solventar estas limitaciones. Los primeros proponen lo que ellos llaman Parameterizing Behavior Trees (Árboles de Comportamiento con parámetros), donde las tareas pueden tener

para el control acceso de usuarios a una biblioteca. No obstante, el lenguaje está dirigido a expertos en diseño de sistemas y con unos conocimientos mínimos de programación. Este no cumple con el principio de transparencia semántica, es decir, la representación de la notación no evoca el significado de los conceptos y resulta difícil entender los diagramas. A modo de ejemplo que ilustre esta dificultad, se puede ver el diagrama de la Figura 2.11. Este diagrama modela un protocolo simple de comunicación. Sin embargo, resulta difícil entender exactamente su comportamiento. Por ello, las Redes Coloreadas de Petri no se consideran apropiadas como lenguaje de modelado para futuros usuarios no expertos como médicos y trabajadores sociales.

2.6.5. Conclusiones

Tras presentar los lenguajes, se puede comprobar que ninguno cumple con todas las características que se buscan, ver la Tabla 2.2 que resume dicho análisis. De entrada, al no ser lenguajes específicos del dominio, todos tienen en común que necesitan ser ampliados para incorporar las acciones/tareas/limitaciones específicas del dominio que nos ocupa.

Lenguaje	Moody	Eventos	Reutilización	Editor	Motor	Online
HCM-L [86, 90]	Sí	No	No	Sí	No	No
HAB [30, 29]	No	Sí	No	No	HAB	Sí
BTs [94, 32]	No	SEs [68]	PBT [129]	JBT	JBT	Sí
CPN [55]	No	Sí	Sí	Sí	Sí	Sí

Tabla 2.2: Tabla comparativa de lenguajes para especificar comportamientos.

Las Redes Coloreadas de Petri se descartan directamente por resultar muy complejas de manejar por personal no especializado.

HCM-L podría ser una opción interesante por su forma de especificar los comportamientos. No obstante, tiene muchos otros inconvenientes. Este no aborda el hecho de que una tarea o actividad pueda ser interrumpida por un evento y tampoco modela actividades o tareas conjuntas. Además, el editor no está disponible y, debido a que no está enfocado a recrear comportamientos (si no a reconocerlos), no dispone de un motor.

La propuesta de notación para el módulo de comportamientos HAB se queda a medio. Esta necesita una notación que siga los principios de Moody, que sus entidades de comportamiento incluyan parámetros y que ofrezca un editor. Por el contrario, su motor sí se encuentra disponible con el módulo HAB disponible en UbikSim.

Los BTs (si se incluyen las mejoras) parece una buena solución. Existe un editor y un motor (JBT) que se pueden emplear como punto de partida. No obstante, las construcciones que hay que realizar con los árboles de comportamiento, se aproximan mucho a las de la programación, por lo que no son una notación adecuada. Este hecho se refleja con más detalle en el Capítulo 3 dedicado al lenguaje. En dicho capítulo se comparan las especificaciones realizadas con SociAALML frente a aquellas realizadas con BTs.

Capítulo 3

El lenguaje de modelado

Solo hay mundo donde hay lenguaje.

Martin Heidegger

RESUMEN: Este capítulo define el lenguaje específico de dominio SociAALML. Primero, expone unos requisitos generales y presenta las herramientas utilizadas para su creación. Posteriormente, por cada aspecto del lenguaje: se establecen unos requisitos, se define la parte del lenguaje a través de su meta-modelo, se ilustra su notación a través de pequeños ejemplos y, finalmente, se realiza una evaluación. El capítulo finaliza con unas conclusiones globales sobre el lenguaje SociALLML.

3.1. Introducción

Disponer de simulaciones que recreen la vida cotidiana de los enfermos de párkinson en sus viviendas a modo de *Living Lab virtual* puede ayudar en el proceso de desarrollo de soluciones AAL, incentivando su investigación y reduciendo costes de desarrollo. Sin embargo, con los medios que existen actualmente es muy complejo y costoso realizar dichas simulaciones. En esta tesis se propone el desarrollo dirigido por modelos para generar las simulaciones. Si se recuerda la arquitectura del MDD de la Figura 1.1, el primer elemento corresponde al del lenguaje. Un *lenguaje específico del dominio* proporciona el nivel de abstracción para que sus usuarios realicen especificaciones empleando conceptos del dominio del problema [137]. Este alto nivel de abstracción, junto con el uso de conceptos del dominio, facilita que individuos sin conocimientos técnicos pero conocedores del dominio (e.g. médicos o trabajadores sociales) sean capaces de crear especificaciones. Estas especificaciones deben contener la suficiente información para generar de forma automática la simulación y, además, otro tipo de información relevante del dominio del problema que no se simule pero que sea de utilidad para los expertos.

Dicho tipo de información se extrae de las entrevistas. SociAALML es el nombre del lenguaje propuesto para estos fines y el cual se presenta en este capítulo.

Para crear un lenguaje de modelado hay que definir su sintaxis y su semántica. La sintaxis se descompone en abstracta y concreta. La *sintaxis abstracta* se encarga de definir la estructura y las reglas gramaticales del lenguaje, mientras que la *sintaxis concreta* define los símbolos de la notación del lenguaje y su representación. La sintaxis se suele especificar formalmente utilizando un meta-modelo. La semántica especifica el significado de cada concepto del dominio que se ha modelado.

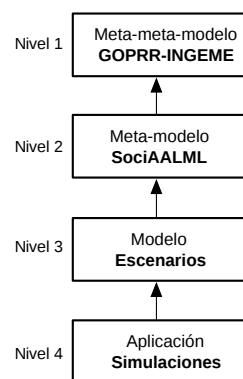


Figura 3.1: Los 4 niveles de instancia de Kelly [71] necesarios para integrar el modelado del uso y la evolución de los sistemas.

Sin embargo, para definir el meta-modelo del lenguaje hace falta a su vez otro lenguaje, tal como expone Kelly [71] con sus 4 niveles de instancia, ver Figura 3.1. Cada nivel superior emplea los conceptos del nivel inferior para definirse. El primer nivel lo constituye el meta-meta-modelo que se utiliza para definir lenguajes. En este nivel se encuentra INGENME, el cual está basado en el meta-meta-lenguaje GOPRR. INGENME ofrece herramientas para definir lenguajes y generar automáticamente sus editores. Esto facilita la creación de los modelos de los lenguajes. En el apartado 3.2 se verán más detalles sobre GOPRR e INGENME. En el segundo nivel se encuentra el meta-modelo de SociAALML que se define a partir los conceptos y herramientas de INGENME. Los modelos generados con SociAALML (los cuales contienen los conceptos específicos del dominio tales como sujetos, tareas o síntomas) pertenecen al tercer nivel. El apartado 3.3 cubre estos dos niveles. Por cada aspecto del lenguaje, primero se explican los detalles sobre el meta-modelo de SociAALML (donde se trata tanto la sintaxis abstracta y concreta como su semántica) y, después, se muestra la expresividad del lenguaje con modelos de ejemplo. En el último nivel se encuentra la aplicación o, en nuestro caso, el código del simulador. Tanto los detalles del simulador como la generación de instancias del mismo a partir de los escenarios se verán en el Capítulo 4.

3.2. El meta-meta-modelo y la herramienta: INGENME

SociAALML se construye a partir del meta-editor INGENME¹. Este ofrece un lenguaje de meta-modelado basado en GOPRR [70] y es similar al Ecore del Eclipse Modeling Framework, que es más habitual. Los autores se han decantado por INGENME por su facilidad para crear de forma rápida editores autocontenidos y reutilizables como artefactos Maven.

INGENME ofrece los 5 elementos básicos de GOPRR e incorpora algunos adicionales para cuestiones de representación y documentación. Los elementos de GOPRR son aquellos necesarios para definir el lenguaje, estos son:

- *Grafos (Graphs)*, permiten el concepto de agregación ya que contienen un conjunto de objetos con sus relaciones (y sus roles). En INGENME se denomina *MetaDiagram* y tiene la relación «*Contains*» para indicar los elementos que pertenecen al diagrama (*MetaObjects* y *MetaRelationships*).
- *Objetos (Objects)*, consisten en elementos de diseño independientes e identificables. Ejemplos de objetos son una Entidad en un Diagrama Entidad Relación o un Proceso en un Diagrama de Flujo de Datos. En INGENME se llaman *MetaObject*.
- *Propiedades (Properties)*, son los atributos de los objetos y solo se puede acceder a ellos a través de su objeto o relación. Los atributos suelen aparecer como etiquetas de texto en los diagramas y contienen datos individuales, por ejemplo, un nombre, un campo de texto o un número. El equivalente en INGENME son los *PropertyField*. Al resto de elementos se les puede asociar una propiedad a través de la relación «*Has*». Para definir la propiedad, *PropertyField* tiene un atributo *WrappedType* que permite que un atributo pueda ser un tipo básico (cadena de texto, entero o decimal), un objeto o incluso una referencia a un diagrama. Las referencias a diagramas permiten crear relaciones del tipo agregación, composición o jerarquía.
- *Relaciones (Relationships)*, son asociaciones entre objetos y también pueden tener propiedades. INGENME facilita la creación de relaciones básicas 1:1 entre dos objetos con la relación «*BinaryRel*». También tiene un elemento llamado *MetaRelationship* para definir relaciones más complejas necesitando para ello el último elemento de GOPRR, el rol.
- *Roles (Roles)*, definen la forma en la que los objetos participan en una relación. Los roles también pueden tener propiedades. En INGENME, este elemento se llama *MetaRole* y, como indica GOPRR, sirve para modelar cómo participa un objeto en una relación. Un *MetaRole* se relaciona con un *MetaObject* con la relación «*PlayedBy*».

¹Sitio web de INGENME: <http://ingenme.sf.net>

La representación de INGENME para estos elementos se encuentra en la Figura 3.2.

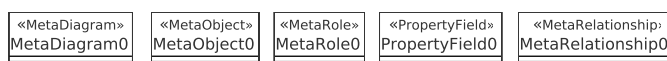


Figura 3.2: Representación de las 5 entidades GOPRR de INGENME.

Los elementos extra incluidos en INGENME son también cinco. Dos de ellos son *BasicRepresentation* y *VisualRepresentation*, los cuales se emplean para especificar cómo representar los objetos y las relaciones, es decir, definir la sintaxis concreta. Tanto *MetaObjects* como *MetaRelationships* se pueden representar por estos elementos utilizando la relación «*VisualizedAs*». *BasicRepresentation* simplemente requiere indicar dos imágenes, una para representar al elemento en la botonera del editor y otra para representar al elemento en el panel de edición del diagrama. *VisualRepresentation* permite hacer representaciones mucho más ricas empleando *SwiX^{ml}*². *SwiX^{ml}* se diseñó para definir componentes gráficos del Swing de Java en formato XML. *VisualRepresentation* dispone de un campo donde se puede indicar su representación utilizando *SwiX^{ml}*. Gracias a este lenguaje se pueden hacer representaciones más expresivas al facilitar, por ejemplo, incluir varios iconos o mostrar los valores de los atributos de la entidad. Además, con INGENME se pueden crear enlaces si una propiedad de un campo hace referencia a un diagrama, mejorando así la navegación. El elemento *PreferredOrder* se incluye para poder indicar tanto el orden de los atributos en una entidad como si se desea que alguno no aparezca. Toda entidad que pueda tener propiedades (*PropertyField*) se puede asociar a un *PreferredOrder* a través de la relación «*PropsOrderedAs*». Como muchos otros lenguajes, INGENME también ofrece un elemento para hacer anotaciones las cuales se pueden o no asociar a una entidad. Por último, a todas las entidades principales de INGENME (excepto a *PropertyField*) se les puede asociar un elemento *Documentation* a través de la relación «*DocumentedBy*». Este elemento tiene dos campos, uno para la descripción y otro para indicar recomendaciones de uso. La representación empleada por INGENME para estos elementos extra se puede ver en la Figura 3.3.

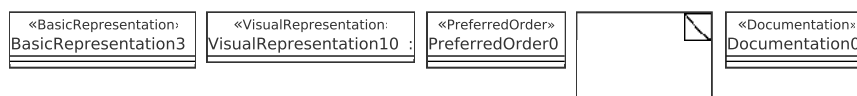


Figura 3.3: Representación de las entidades adicionales que ofrece INGENME para definir la sintaxis concreta y la documentación.

Destacar que INGENME solo permite definir cómo representar las entidades en el diagrama y no cómo representar u organizar elementos del editor, característica que sí la implementa GEF³ de Eclipse. No obstante, GEF no es sencillo de utilizar y tampoco está del todo conseguido, por ello, se ha optado por INGENME aunque presente esta limitación.

²Sitio web de *SwiX^{ml}*: <http://www.swixml.org/>

³Sitio web de GMF de Eclipse <http://www.eclipse.org/gef/>

3.3. Meta-modelo de SociAALML

La finalidad principal del lenguaje es permitir generar especificaciones que modelen el día a día de aquellos elementos no software relacionados con un sistema AAL y que a partir de dichas especificaciones sea posible generar simulaciones 3D que reflejen la semántica de ese día a día. Como se vio en la sección 2.1, los elementos más recurrentes en un sistema de este tipo son: el espacio físico, los dispositivos y su infraestructura, el middleware, las aplicaciones y servicios y, finalmente, los usuarios. Por tanto, SociAALML se debe centrar en modelar el espacio físico, la parte hardware de los dispositivos (sensores y actuadores) y, sobre todo, el elemento más complejo: el comportamiento de los sujetos. Aunque no se modele el comportamiento de las aplicaciones, se hace necesario hacer referencia a artefactos software que irán instalados en los emuladores.

El modelado del espacio físico (es decir, la vivienda con sus habitaciones, sus muebles y objetos) es necesario para hacer referencia a sus elementos. Por ejemplo, para especificar que un sujeto vaya a una habitación y se siente en una silla concreta. También es importante para indicar los despliegues de los dispositivos virtuales, por ejemplo, que un dispositivo se asocie a una silla o que un sensor de presencia se instale en la esquina de una habitación. Estos conceptos se ven en el apartado 3.3.1.

El modelado de los dispositivos resulta importante desde dos perspectivas. Una que contempla la interacción del usuario con los mismos, esto es, qué interfaces y funcionalidades ofrecen a los usuarios. Por ejemplo, la televisión se puede encender pulsando el botón o con un comando de voz. La otra perspectiva tiene que ver con aspectos más técnicos del diseño de la solución AAL. Estos aspectos se refieren a las características y a las configuraciones de los sensores y de los actuadores, los cuales influirán en el funcionamiento del software conectado al simulador. Por ejemplo, la precisión del sensor o la zona que es capaz de cubrir. No obstante, esta parte del lenguaje no se ha llevado a cabo en esta tesis ya que se sale de su ámbito. Esta está centrada en modelar el comportamiento humano. Así que este aspecto del lenguaje se deja como trabajo futuro. Esto no significa que no haya sensores y actuadores virtuales, sino que desde el lenguaje SociAALML, no se podrán configurar los mismos. Básicamente, se han modelado los dispositivos Android como un objeto que el sujeto puede coger y dejar, fijárselo a una parte del cuerpo y pulsar en posiciones de la pantalla dadas por coordenadas. Por ello, no hay una sección para los dispositivos en este capítulo, pero sí la hay para el caso del simulador (Capítulo 4). Solo en el apartado 3.3.7, dedicado a la definición de los escenarios, se puede comprobar cómo se han creado elementos en el lenguaje indispensables para permitir desplegar los dispositivos en el entorno virtual e instalar y arrancar el software a evaluar.

El lenguaje debe permitir especificar el comportamiento diario de los individuos que habitan en la vivienda y que serán los futuros usuarios del sistema AAL. Para ello, este debe recoger, además de sus actividades diarias, su comportamiento ante determinados eventos (un sujeto se cae o llaman al timbre de la puerta) y cómo varía dicho comportamiento en función de sus síntomas, en el caso de que el sujeto sufra de

la enfermedad de Parkinson (por ejemplo, no puede cambiar de canal de televisión por el síntoma de temblor en las manos). La parte del lenguaje que se dedica a la definición del comportamiento de los sujetos se trata desde el apartado 3.3.2 al 3.3.6.

Además de los elementos propios del sistema, el lenguaje debe incorporar una serie de conceptos para definir los escenarios a partir de dichos elementos previamente modelados. Por ejemplo, entidades para indicar posición inicial de los habitantes en el escenario o los síntomas iniciales del enfermo, así como la configuración de los dispositivos para su despliegue. Esta parte del lenguaje se aborda en el apartado 3.3.7.

Pero el lenguaje no debe limitarse solo a recoger dichos conceptos. Tal y como se avanzó en el apartado 2.2, dedicado a los requisitos, se considera fundamental que el lenguaje de modelado cumpla las siguientes características:

- Los usuarios del lenguaje, tales como los trabajador sociales, los usuarios, los desarrolladores e investigadores, deben ser capaces de crear especificaciones con el lenguaje y comprenderlas. Para ello el lenguaje debe (1) tener un nivel de abstracción alto que utilice conceptos cotidianos y del dominio de la enfermedad de Parkinson, y (2) tener una notación visual fácil de aprender y entender, es decir, ser cognitivamente eficiente. Para ello, el lenguaje debería seguir los principios para el diseño de notaciones visuales definidos por Moody [96].
- Las especificaciones deben poder reutilizarse y combinarse. Por ejemplo, tomar limitaciones de un enfermo de párkinson modelado previamente y hacerle unas pequeñas modificaciones para modelar a otro enfermo.

La notación influye en gran medida a la hora de conseguir que individuos no expertos sean capaces tanto de realizar especificaciones como de entenderlas. Para ello, se ha prestado especial atención a la hora de diseñar los iconos para la representación de las instancias de los elementos del lenguaje. En todo momento, se ha buscado que estos sean auto-explicativos. Además, la notación permite mostrar valores relevantes de las propiedades del objeto para su mejor comprensión. Se ha creado un símbolo gráfico por cada objeto del lenguaje para cumplir el *principio de claridad simbólica* de Moody y así conseguir eficacia cognitiva en la notación. Las representaciones de los objetos siguen un patrón más o menos uniforme, ver Figura 3.4. De arriba a bajo, primero se muestra el identificador del objeto (*id*). A continuación una línea horizontal tras la cual se muestran los valores de los atributos del objeto. Si se muestra un atributo que hace referencia a un diagrama, este consiste en el nombre del diagrama en letras azules y subrayadas. Por último, se encuentra el icono que sirve para identificar fácilmente el tipo de entidad. En aquellos objetos donde no se muestra ningún atributo, la línea horizontal se suprime. El lenguaje define una gran cantidad de objetos, más de 100. Por ello, en las siguientes secciones se ha decidido no listar la representación de cada objeto y emplearlas directamente en los ejemplos. Si en algún momento, no se entiende bien la representación de un elemento, se puede recurrir al Apéndice A. Este apéndice contiene las representaciones de todos los objetos del lenguaje con una descripción asociada.

En los siguientes apartados se discutirá qué persigue cada aspecto del lenguaje más en profundidad. Por cada uno de los aspectos, se presentará una propuesta para formar

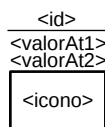


Figura 3.4: Plantilla para la representación de un objeto del lenguaje SociAALML.

parte del meta-modelo que definirá el lenguaje. Cada propuesta tratará de perseguir los objetivos y características aquí expuestas. Tras cada propuesta de meta-modelo, se mostrará un ejemplo de modelo y, por último, una evaluación.

3.3.1. El espacio físico

3.3.1.1. Introducción

Según las entrevistas, las actividades que realizan la mayor parte del tiempo los pacientes se desarrollan en su vivienda. Aunque también hacen referencia a cómo de fácil o difícil les resulta desenvolverse cuando tienen que, por ejemplo, ir a hacer la compra o coger el transporte público. En todo modelado específico de dominio, cuanto más concreto sea y menos elementos abarque, más fácil será el desarrollo del lenguaje, el generador y el marco del dominio [71]. Por ello, el entorno se restringe a la vivienda de los enfermos ya que es donde pasan la mayor parte del tiempo y en donde se desplegará las soluciones AAL.

Aun así, una vivienda es compleja, tiene muchos elementos: plantas, muros, puertas, sistema eléctrico, fontanería, habitaciones, sanitarios, muebles, entre otros. Pretender que a partir de un lenguaje del tipo que nos ocupa se genere el código que represente tanto el aspecto 3D de los elementos como su comportamiento, es muy complejo. Esta complejidad se agudiza aún más si se persigue que sea asequible para un usuario no experto. Por tanto, la parte del meta-modelo dedicado a los conceptos del espacio se va a centrar en:

- Proporcionar una vista general de la distribución de la vivienda.
- Identificar las habitaciones para poder hacer referencias a ellas y así poder especificar tareas tales como que un sujeto vaya al salón.
- Identificar y asignar objetos a cada habitación para, igual que antes, poder hacer referencia a los mismos. Por ejemplo, en un salón debe poder identificarse objetos como el sofá, la mesa, las sillas o la televisión para así poder especificar actividades cotidianas, por ejemplo, ver la televisión. Es importante también la identificación de objetos a la hora del diseño del entorno AAL de forma que se pueda asociar dispositivos a elementos físicos del entorno. Por ejemplo, asignar un sensor de presión a una silla.

Los arquitectos son los profesionales encargados de diseñar las edificaciones, entre ellas las viviendas. Estos utilizan desde hace ya muchos años aplicaciones del tipo

CAD (*Computer-Aided Design*, Diseño asistido por ordenador) para sus diseños. Los diseños, simplificando, consisten en dibujar los planos, es decir, las vistas de la edificación. Una vez creadas las vistas, estas aplicaciones permiten “levantar” los planos para facilitar el modelado 3D del edificio. Sin embargo, esto solo aporta información de representación (formas, texturas, posiciones). BIM (*Building Information Modeling*, Modelado de la información de edificios) se podría considerar un lenguaje específico para el dominio de la construcción de edificios. Según Eastman [40], los modelos generados deben contener información geométrica precisa y los datos necesarios para dar soporte a su construcción, a su fabricación y a las actividades de ejecución necesarias para construir un edificio. Además, este lenguaje con sus herramientas persigue objetivos muy parecidos a los de esta tesis. Según Azhar [8, 7] estos son: incremento de la colaboración dentro del proyecto, reducción de costes, mejor gestión del tiempo y una mejor relación con los clientes.

Si se consideran los simuladores AmI vistos en el apartado 2.3 sobre el estado del arte, solo hay dos simuladores que utilizan este estándar como modelos espaciales. Uno es VR-SmartHome [55, 56], el cual extiende el estándar IFC (*Industry Foundation Classes*, Clases de Fundación de la Industria) incluyendo Smart Objects y tareas del usuario. Por un lado, los Smart Objects contienen la información necesaria para que los usuarios puedan interactuar con ellos y acceder a sus funcionalidades. Por otro lado, definen actividades y tareas con sus *pasos*. Cada *paso* requiere de una o más funcionalidades necesarias para su realización. Por tanto, a través de las funcionalidades se relacionan tareas con Smart Objects. El otro simulador es SimCon [88, 87], el cual también extiende el lenguaje para integrarlo con un lenguaje de modelado de sensores (SensorML⁴) y así enriquecer la descripción de los mismos para su simulación.

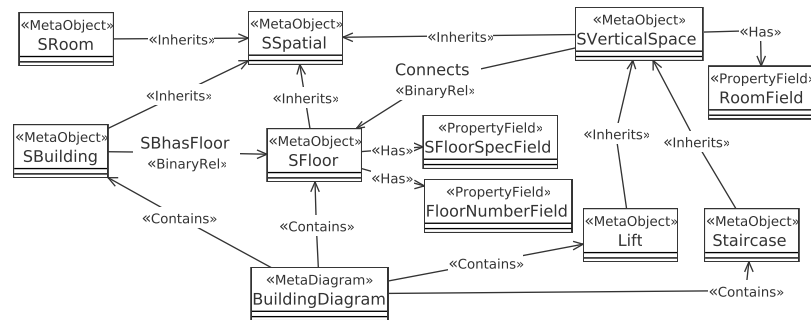
Sin embargo, estudios como [110] destacan la complejidad del modelo IFC, su costosa curva de aprendizaje y la escasez de herramientas. Debido a estas desventajas, la falta de tiempo y puesto que el modelado del espacio no es una prioridad en esta tesis se ha desechado el uso del estándar IFC como modelo espacial. En su lugar, se ha creado una especificación propia, sencilla y suficiente para nuestros propósitos que se presenta a continuación a través de su meta-modelo.

3.3.1.2. Propuesta de Meta-modelo

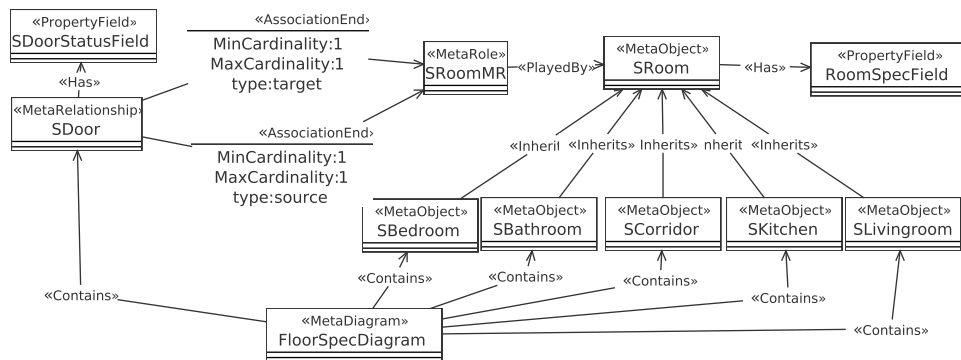
La parte del meta-modelo de SociAALML encargado de definir los conceptos espaciales se divide en tres vistas. Cada vista corresponde a un nivel de detalle: edificio, planta y habitación. Los meta-modelos de cada vista se encuentran en la Figura 3.5.

Un edificio (*SBuilding*) consta de una o varias plantas (*SFloor*). Cada planta tiene un número (*FloorNumberField*) y una especificación, es decir, referencia a un diagrama *FloorSpecDiagram* que define dicha planta. Además, las plantas pueden estar conectadas por espacios verticales *SVerticalSpace* tales como escaleras (*Staircase*) y/o ascensores (*Lift*). Un espacio vertical, además tiene referencias a las habitaciones que conecta (con el campo *RoomField*). Las habitaciones deben pertenecer a plantas dife-

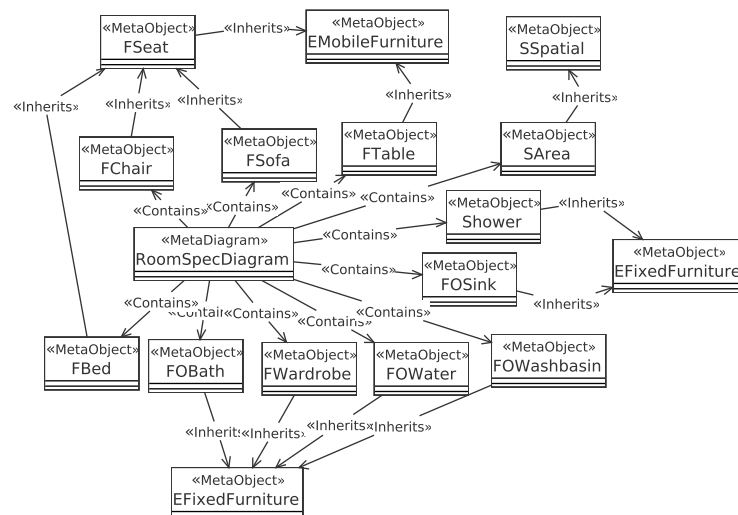
⁴Sitio web de SensorML: <http://www.opengeospatial.org/standards/sensorml>



(a) Vista del meta-modelo con los conceptos principales del edificio.



(b) Vista del meta-modelo de los conceptos de una planta del edificio.



(c) Vista del meta-modelo de las entidades de la habitación.

Figura 3.5: Vistas del meta-modelo de SociAALML donde se incluyen los elementos espaciales.

rentes. Fíjese que todos los elementos son espacios y, por ello, heredan del meta-objeto *SSpatial*. Este meta-objeto representa el concepto más genérico de espacio, entendiéndose como una extensión delimitada que puede contener objetos y donde los sujetos pueden realizar sus actividades. Los conceptos principales del edificio se encuentran en la vista del meta-modelo de la Figura 3.5a.

En una planta, la entidad principal es la habitación (*SRoom*). Las habitaciones suelen estar conectadas por puertas. La puerta se modela con la meta-relación *SDoor*. Cada puerta solo conecta un par de habitaciones que siguen el meta-rol *SRoomMR*. Las puertas tienen una propiedad *SDoorStatusField* para indicar el estado de la puerta (abierta, cerrada o cerrada con llave). Cada habitación, al igual que ocurre con las plantas, se define utilizando otro conjunto de conceptos a los cuales se hace referencia a través de la propiedad *RoomSpecField*. Además, resulta importante diferenciar entre tipos de habitación: dormitorio (*SBedroom*), baño (*SBathroom*), pasillo (*SCorridor*), cocina (*SKitchen*) y salón (*SLivingroom*). A la hora de especificar actividades, estas se suelen realizar en un tipo de habitación concreta, por ejemplo, cocinar o ducharse. Los elementos del meta-modelo que definen los conceptos de la planta se muestran en la Figura 3.5b.

Por último, las habitaciones se caracterizan por tener muebles y objetos. En esta vista se recogen los objetos que según la experiencia se consideran relevantes para la especificación de las actividades diarias de los individuos y para el despliegue del sistema AAL. Se diferencia entre muebles fijos (*EFixedFurniture*) como una bañera (*FOBath*) o un armario (*FWardrobe*), y entre aquellos que se pueden mover (*EMobileFurniture*), por ejemplo, una mesa (*FTable*). Dicha diferenciación resulta útil a la hora de generar el código para definir el comportamiento del objeto según las leyes de la física. Dentro de los móviles también se identifican aquellos que sirven para sentarse (*FSeat*), e.g. una silla (*FChair*) o un sofá (*FSofa*). A continuación se describe la finalidad de los objetos que se consideran más interesantes:

- *FChair* representa a una silla. Este es un objeto muy empleado en el día a día mientras se realizan actividades, como por ejemplo comer o estar con el ordenador. Por ello, resulta básico para especificar los comportamientos. Además, en los sistemas AAL no es raro instalar sensores de presión en algunas sillas para facilitar la detección de la actividad del sujeto.
- *Shower* representa a una ducha. Este es un elemento básico para especificar la actividad de ducharse. Además, las personas mayores lo temen debido a que se pueden producir caídas por resbalones. Podría resultar interesante considerar para el diseño del sistema AAL instalar sensores de flujo de agua para detectar más fácilmente la actividad del sujeto.
- *FOSink* representa a los lavabos donde se realizan las actividades de aseo tales como afeitarse, lavarse las manos y los dientes. Debido a la falta de precisión en las manos de los enfermos de párkinson, las actividades en dicho sanitario pueden resultar complicadas.

- *FWardrobe* representa un armario donde se guarda la ropa. Otra de las actividades que le cuesta a los enfermos de párkinson es vestirse, siendo especialmente problemáticas las tareas de abrocharse los botones (por la falta de precisión en los dedos) y las de ponerse los calcetines (debido a la rigidez que sufren). Un sensor de puerta podría indicar cuándo el paciente tiene intención de vestirse para que el sistema AAL lo sepa y pueda asistirlo de alguna manera.

Por último, en una habitación también se pueden distinguir elementos que no son objetos sino espacios y que se han llamado áreas (*SArea*). Este concepto es interesante para poder especificar en qué zonas de la habitación el sujeto realiza determinadas actividades. Por ejemplo, se puede especificar una zona del lado derecho de la cama donde el sujeto suele vestirse. Los conceptos de interés relacionados con una habitación los define la vista del meta-modelo de la Figura 3.5c.

3.3.1.3. Notación y modelos

Las Tablas A.1, A.2 y A.3 del Apéndice A contienen un ejemplo de la notación para cada tipo de meta-entidad definidas aquí, y van acompañadas de una descripción.

Para ejemplificar la expresividad y las posibilidades de esta parte del meta-modelo, supóngase que se quiere modelar una vivienda de dos plantas donde uno de los habitantes tiene una discapacidad que le impide subir y bajar por las escaleras. Por esta razón, la vivienda dispone de un ascensor. Tanto el ascensor como las escaleras conectan los pasillos de ambas plantas. Esta descripción más general de la vivienda correspondería con el modelo de la Figura 3.6a. En ella puede verse la vivienda con identificador *SBuilding0* y que a través de la relación *SBhasFloor* conecta dos plantas, *SFloor0* y *SFloor1*. Se sabe qué planta es cuál, gracias a que cada una muestra su nivel, “*SFloor0 :0*” y “*SFloor1 :1*”. Por último, hay una instancia que representa las escaleras (*Staircase1*) y otra el ascensor (*Lift1*). Ambas conectan las dos plantas a través de la relación *Connects* y ambas a través de las mismas habitaciones (*SCorridor0* y *SCorridor1*). Estos pasillos están definidos en las vistas de cada una de las plantas, a la cual se puede acceder a través de los enlaces que poseen cada instancia de planta, *SFloor0* tiene *Floor0* y *SFloor1* tiene *Floor1*.

La vivienda está distribuida de forma que la cocina y el salón se encuentran en la planta baja y las habitaciones en la planta alta. En la Figura 3.6b se encuentra la parte del modelo que define la planta baja. En esta puede verse el pasillo *SCorridor0* que da acceso a un aseo (*SBathroom0*), a un salón (*SLivingroom0*) y a una cocina (*SKitchen0*) a través de puertas representadas por la relación *SDoor*. Al igual que las plantas, cada habitación puede tener una especificación de los elementos que contiene y a la que se puede acceder fácilmente a través de su enlace. El pasillo *SCorridor* forma también parte de la entrada de la vivienda y, por ello, como muestra la Figura 3.6c, contiene una mesa (*FTable3*) que representa una consola y un área (*SArea0*) junto a la puerta principal que correspondería a la zona de la entrada. El baño de la planta baja (*SBathroom0*) hace las veces de aseo pues, tal como indica la Figura 3.6d, solo contiene un váter y un lavabo. La cocina dispone de un fregadero, una mesa, dos sillas y un área

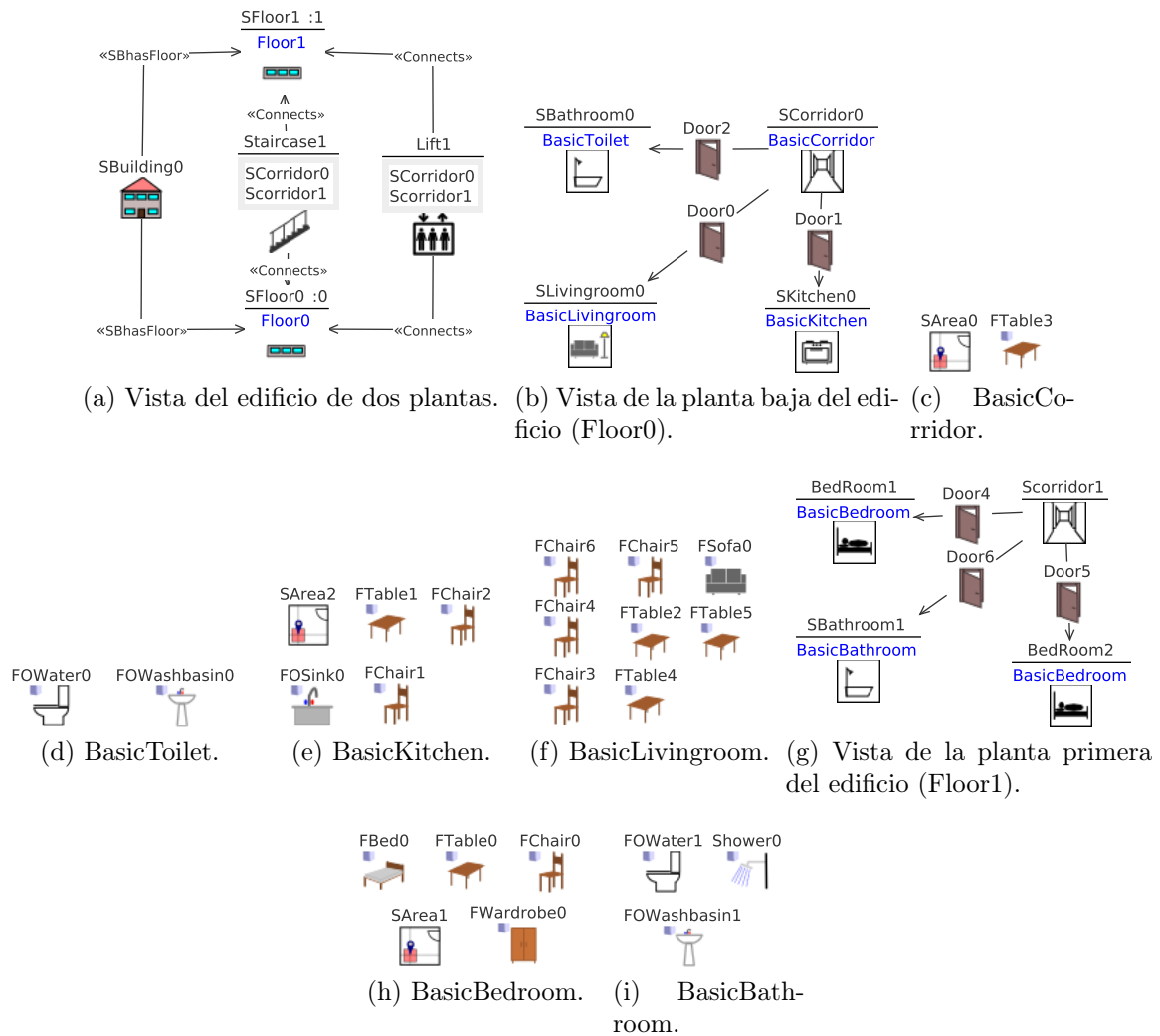


Figura 3.6: Vistas de los diferentes elementos del modelo SociAALML de una vivienda de dos plantas.

(*SArea2*) junto a los fogones para especificar actividades de cocinar, ver Figura 3.6e. El salón es también comedor, por ello, tiene una mesa de comer (*FTable2*) con cuatro sillas (*FChair3-6*) y, por otro lado, un sofá (*FSofa0*), una mesa de café (*FTable5*) y una mesa para la televisión (*FTable4*), ver Figura 3.6f.

En la Figura 3.6g se encuentra la parte del modelo que define la planta alta. En esta puede verse el pasillo *SCorridor1* que da acceso a un baño (*SBathroom1*), y a dos habitaciones (*Bedroom0* y *Bedroom1*). En este caso, el pasillo no contiene ningún elemento y, por tanto, no tiene asignado una vista de habitación. En la vivienda, las dos habitaciones contienen el mismo número y tipo de elementos, por ello, tanto *Bedroom1* como *Bedroom2* comparten la misma especificación de habitación de la Figura 3.6h. Esta contiene una cama (*FBed0*), una mesa (*FTable0*) con su silla (*FChair0*), un armario (*FWardrobe0*) y un área para poder vestirse (*SArea1*). Por último, el baño de la planta alta, además de disponer de un váter y un lavabo como el de la planta baja, dispone de una ducha (*Shower0*). Por ello, no se puede reutilizar la especificación del aseo de la planta baja y es necesario crear otra (*BasicBathroom*, Figura 3.6i).

3.3.1.4. Evaluación

Como ya avanzamos, la parte del lenguaje dedicada al espacio físico no es crítica para la especificación de los escenarios ya que su responsabilidad se restringe a identificar los espacios de la vivienda y sus elementos, y a organizarlos de forma lógica como se ha visto. Aun así, destacar que la necesidad de definir 3 vistas para especificar los espacios y los elementos de una vivienda finalmente resultan demasiadas. Para definir un edificio sí podría ser adecuado, pues suelen disponer de muchas plantas y habitaciones.

En cuanto a la evaluación de la notación, esta se resume en la Tabla 3.1, donde se recoge cualitativamente en qué medida se cumple o no cada principio de Moody. Se demuestra que la notación sigue casi todos los principios de Moody para conseguir una eficacia cognitiva. Uno en los que falla tiene que ver con la integración cognitiva. Esto se debe a que INGENME no permite la integración de vistas para tener una visión más general de los elementos. Por ello, puede resultar un poco complicado mantener una visión global de toda la vivienda. Este principio podría resolverse si INGENME permitiese mostrar subvistas en una misma vista parecido a como muestra la Figura 3.7. Esta integra todas las vistas la Figura 3.6. La propuesta también falla en el ajuste cognitivo, ya que la notación no soporta dialectos, es decir, diferentes notaciones dependiendo del usuario o del soporte. Por ejemplo, un dialecto para los familiares y otro para los trabajadores sociales, y un dialecto más simplificado para la notación cuando se quiera escribir a mano y no a ordenador.

Como trabajo futuro se considera interesante utilizar el formando estándar para edificios IFC ya que este se toma como referente de modelo para el ciclo de vida en la construcción de edificios. Los autores consideran un gran avance que el desarrollo e implantación de un sistema de Inteligencia Ambiental formara parte, desde el principio, del ciclo de vida de los edificios.

Principios de Moody	Comentario
Claridad simbólica	Se cumple totalmente al existir una representación por cada meta-objeto.
Discriminabilidad perceptual	Se cumple bastante bien al emplear iconos que ofrecen distancia visual. Además, se utiliza el cuadrado como redundancia visual de forma para las habitaciones y un cubo de color lila para los objetos. Se podría pensar en que incluir, además, algún color de fondo, pero se considera que eso provocaría saturación visual.
Transparencia semántica	Se cumple totalmente al utilizar iconos que evocan el significado semántico de lo que representan.
Gestión de la complejidad	Se gestiona la complejidad al emplear la jerarquía para organizar los elementos de más generales a más concretos: elementos de edificio, de planta y de habitación. De esta forma, el número de elementos por diagrama no es elevado.
Integración cognitiva	INGENME no permite elementos de integración de varios diagramas en uno para tener vistas generales. Sin embargo, sí facilita elementos de navegación a través de los enlaces para ir a nuevos diagramas, tal y como se ejemplificó con las entidades de tipo planta y habitación.
Expresividad visual	Tiene expresividad visual sin llegar a saturar al emplear solo 2 elementos visuales pero ser los más expresivos (los iconos y las formas).
Codificación dual	Se cumple al incluir texto junto con los elementos gráficos (identificadores y algunas valores de los atributos) y también al permitir incluir notas en los diagramas.
Economía gráfica	Hay 21 elementos gráficos, tantos como objetos definidos para que se cumpla el principio de claridad simbólica.
Ajuste cognitivo	No lo cumple al no proporcionar dialectos.

Tabla 3.1: Evaluación cualitativa de la notación para los elementos del espacio físico según los principios de Moody.

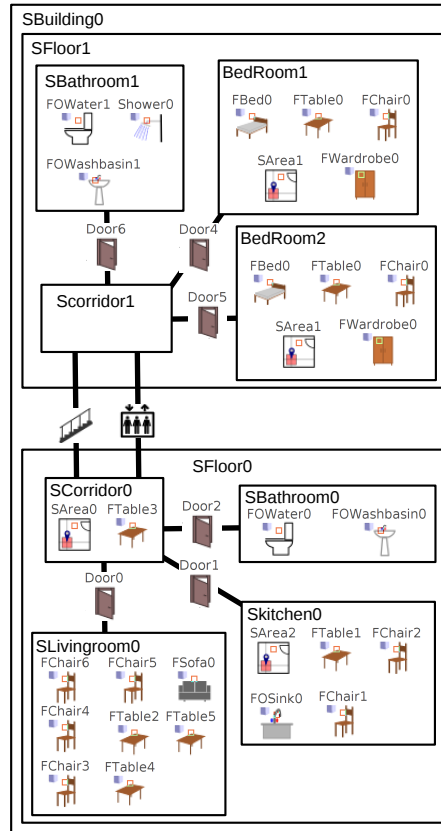


Figura 3.7: Propuesta alternativa donde se integran todas las vistas de los niveles del edificio para cumplir con el principio de integración cognitiva de Moody.

3.3.2. Los sujetos a simular

3.3.2.1. Introducción

El sujeto se podría decir que es la entidad más importante y compleja en torno a la cuál giran muchos otros conceptos también importantes. El lenguaje se centra en los aspectos del comportamiento diario de los habitantes de la vivienda y no en su representación 3D. Dependiendo de cómo se definan y combinen los comportamientos de los sujetos, se producirán unas situaciones u otras. Por un lado, interesa poder especificar situaciones concretas y deterministas. Con esto se persigue poder orquestar y predecir exactamente lo que hará cada sujeto virtual para que se simule la situación problemática que se desea solucionar con el sistema AAL. El modelado y la simulación de estas situaciones problemáticas nos permite utilizar el lenguaje para la toma y validación de requisitos. Por otro lado, también se persigue la posibilidad de definir comportamientos de los sujetos menos deterministas y que se desarrollen a lo largo de días enteros. La finalidad, en este caso, es poner a prueba el sistema AAL ante situaciones no consideradas en un principio.

A partir de la experiencia de la lectura de las entrevistas, los enfermos y familiares describen su día a día a nivel de actividad y sin entrar en mucho detalle en la secuencia de tareas que realizan para completar dichas actividades. No obstante, sí que enfatizan qué tareas no pueden realizar los enfermos cuando algún síntoma alcanza determinados niveles. Por ejemplo, “cuando tengo muchos temblores no soy capaz de coger las pastillas y estas se me acaban cayendo”. También destacan quién responde ante situaciones eventuales como que llamen al timbre o suene el teléfono, y otras referentes a quién asiste al enfermo dependiendo de quién se encuentre en la vivienda y de qué tipo de asistencia necesite. Por ejemplo, en el contexto de una pareja muy mayor, el que hace de cuidador podría ayudar al enfermo a darle las medicinas, pero seguramente no podría ayudarlo a levantarse si se cae. Por último, al inicio de las entrevistas siempre se recaba información personal, social y cultural que los expertos consideran interesante. Aunque esta información no se ve reflejada en la simulación, el lenguaje la incluye para facilitar su gestión. Por tanto, la parte del lenguaje dedicada a modelar a los sujetos debe permitir:

- Modelar el comportamiento de los sujetos centrándose en las actividades diarias, pero permitiendo diferentes niveles de detalle de forma que se gestione la complejidad.
- Especificar comportamientos repetibles de forma determinista y no determinista.
- Recoger la información personal, profesional, social y cultural.
- Modelar eventualidades y especificar cómo reacciona cada individuo ante las mismas.
- Describir la enfermedad de Parkinson y cómo se ve modificado el comportamiento normal del enfermo. Estas alteraciones en el comportamiento pueden provocar diferentes situaciones. Algunas de ellas consistirán en la incapacidad del sujeto para realizar ciertas tareas (o sí será capaz de realizarlas pero de forma más lenta) o en que sufrirá algún accidente, por ejemplo, que se caiga.

En el apartado 2.6 del estado del arte, se analizaron varios tipos de lenguajes atendiendo a varios requerimientos: principios de Moody, modelado de eventos, reutilización de modelos, disponibilidad de editor y motor, y disponibilidad en la web. Sin embargo, ninguno de ellos satisfizo completamente dichos requerimientos. En el siguiente apartado se presenta la propuesta de SociAALML para definir los comportamientos de los sujetos de forma que se cumplan estos requerimientos generales y los específicos que se acaban de identificar.

3.3.2.2. Propuesta de Meta-modelo

Para conseguir los objetivos planteados, la propuesta de esta tesis para modelar el comportamiento de los sujetos se basa en perfiles. Existe un perfil base que describe

las actividades diarias y el resto de perfiles alteran dicho comportamiento base. Como puede verse en la Figura 3.8, un individuo se representa por un objeto *Human*. Cada perfil, *HumanProfile*, se asocia a un humano a través de la relación «*ProfileOf*». Se han identificado 4 perfiles:

- *ADLProfile* es el perfil base que define las actividades de la vida diaria de un sujeto, es decir, especifica la lista de actividades y tareas que realiza un individuo a lo largo del día, suponiendo que no hay interrupciones ni problemas. Para manejar la complejidad y poder abstraerse, el comportamiento se organiza en 3 niveles: (1) el de los momentos del día, (2) el de las actividades y (3) el de las tareas. En la sección 3.3.4, se verán estos niveles en más detalle.
- *SocialProfile* corresponde a la información social y personal del sujeto. Toda entrevista a familiar y a enfermo empieza recopilando este tipo de información. Como se verá más detalladamente en la sección 3.3.3, este perfil solo recoge la información y no influye en el comportamiento del sujeto.
- *InteractionProfile* define comportamientos eventuales, es decir, qué hace el sujeto cuando ocurre algo relevante como que llamen al teléfono o un paciente solicita ayuda. Un evento tiene prioridad sobre las tareas de las actividades definidas en el *ADLProfile*. Cuando se especifica que se atienda un determinado evento llevando a cabo una actividad determinada, el sujeto suspende su tarea actual lo antes posible para realizar dicha actividad y, tras finalizar, reanuda la actividad anterior. Este perfil se discutirá más en detalle en la sección 3.3.5.
- *DiseaseProfile* corresponde al perfil de la enfermedad. Tiene un campo de inicio de enfermedad (*BeginningDateField*), pues suele ser relevante cuánto tiempo lleva el paciente sufriendo la enfermedad. Se distingue dos tipos de perfiles de enfermedad, el del Parkinson (*ParkinsonsProfile*) propio del dominio y uno genérico (*GenericDisease*), ya que es muy común que los pacientes, al ser de avanzada edad, padezcan otras enfermedades tales como diabetes, tensión o reuma. Sin embargo, como se verá en la sección 3.3.6, el lenguaje se centra en modelar el Parkinson que es el dominio que nos ocupa.

Todos los perfiles tienen una propiedad que hace referencia a un diagrama en el cual están incluidos los principales elementos para definir el perfil. Por ejemplo, el perfil social (*SocialProfile*) lo define el diagrama al que referencia la propiedad *SocialSpecDiagField*. Un humano solo puede tener un perfil del tipo *ADLProfile*, *SocialProfile* y *ParkinsonSpecDiag*. Resulta evidente que un mismo sujeto no puede estar físicamente en dos sitios a la vez por lo que solo debe asociársele una especificación de actividades diarias. Lo mismo ocurre con el perfil social y el del Parkinson. En cambio, sí pueden emplearse varios perfiles de enfermedad genérica si cada uno corresponde a diferentes enfermedades. También, sería posible tener varios perfiles de interacción, por ejemplo, se puede crear un perfil para interacciones relacionadas con eventos de asistencia al paciente y otro para eventos de dispositivos.

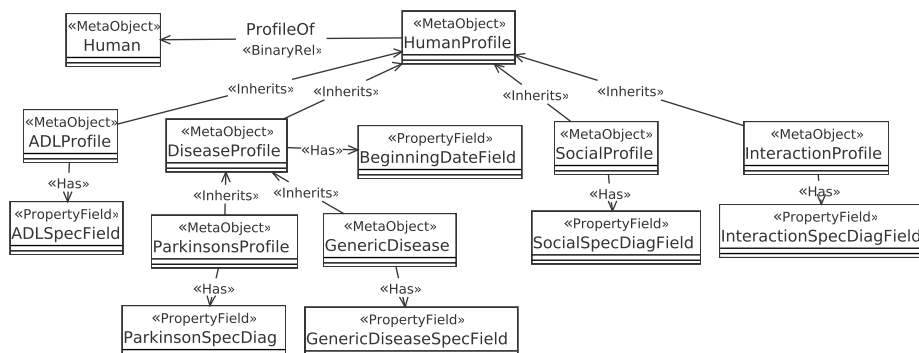


Figura 3.8: Vista del meta-modelo que define los tipos de perfiles del sujeto.

3.3.2.3. Notación y modelos

Si se desea obtener detalles sobre cómo se representan los nuevos elementos definidos, consultar la Tabla A.4 en el Apéndice A. Con estos elementos se pueden crear diagramas que dan una idea muy general de cómo va a ser el comportamiento del sujeto. Como ejemplo, véase la Figura 3.9. Este modelo nos dice que el sujeto es un enfermo de párkinson que también padece otra enfermedad, reuma. Además, las actividades definidas para el paciente corresponden solo a la noche. Por último, se puede ver un perfil social y de interacciones, pero no nos aportan pistas sobre qué aspectos modelan. Fíjese que el resto sí que lo hacen gracias a que los nombres de los diagramas que definen cada perfil son representativos. Por tanto, la expresividad recae, también en parte, en las buenas prácticas del usuario del lenguaje que crea los modelos.

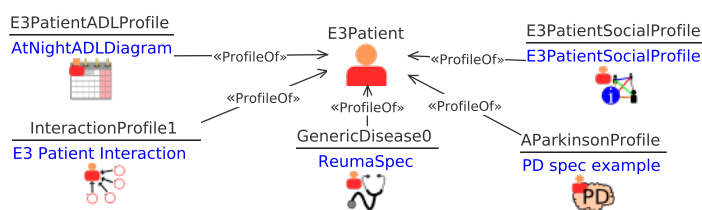


Figura 3.9: Ejemplo de modelo de los perfiles de un enfermo de párkinson.

3.3.3. El perfil social

3.3.3.1. Introducción

El perfil social se centra en recopilar la información socio-cultural, profesional y personal del sujeto que se encuentra, principalmente, en el resumen al inicio de las entrevistas a los pacientes y familiares. El siguiente texto es el resumen correspondiente al paciente E3:

“Varón de 71 años viviendo con su mujer de 68. Lleva 20 años con párkinson. E3 avanzado con un grado de dependencia media. De clase social media. Ex-sargento de la Guardia Civil. Estudios medios.”

El objetivo de este perfil es recoger, organizar y visualizar los datos de carácter socio-cultural, profesional y personal, siendo dichos datos:

- La edad, el sexo y el domicilio. El domicilio es importante para saber cuáles son los miembros de la unidad familiar.
- Clase social a la que pertenece.
- Nivel de estudios.
- La profesión y si sigue en activo o no.

Además, nos resulta interesante reflejar las relaciones que existen entre los individuos. Por ejemplo, quién es la pareja de quién, pues suele ser uno de ellos el cuidador del otro. Los hijos también juegan un rol importante ayudando cuando pueden. En ocasiones, a parte de los hijos y de la pareja del enfermo, estos pueden tener contratado a un asistente que acuda a la vivienda para cuidar del enfermo y/o hacer tareas del hogar.

3.3.3.2. Propuesta de Meta-modelo

La parte del meta-modelo dedicado al perfil social se ha inspirado en [69] donde se define una ontología con información personal para un sistema de gestión de información personal, ver Figura 3.10. La entidad principal es *PersonalInfo* de la que cuelgan el resto de entidades. Esta tiene las propiedades para la edad del paciente (*AgeField*), para el sexo (*SexField*), la clase social (*SocialClassField*) y el nivel de estudios (*StudiesLevelField*). También se indica aquí la vivienda del sujeto a través de la relación «*LiveIn*». La entidad *Profession* está pensada para recoger la información profesional del sujeto. Sus propiedades *ProffesionNameField* y *ActiveWorkerField* registran su profesión y si está en activo, respectivamente. Por último, se indica la red social del sujeto, es decir, qué tipo de relación tiene el sujeto al que pertenece el perfil con el resto de individuos. La red social la representa *SocialNetwork* y a través de su rol *SocialSourceMR* puede relacionarse con humanos con el rol *SocialTargetMR*. Las relaciones más relevantes que aparecen en el entorno del enfermo de párkinson son:

- *isSpouseOf* indica quién es la pareja del sujeto. Por lo general, la pareja es el individuo que más cuida al enfermo si la salud se lo permite. Por tanto, normalmente la pareja también se relacionará con el paciente con la relación de cuidador tal y como se ve más adelante.
- *isParentOf* indica de quién es padre o, lo que es lo mismo, quiénes son sus hijos. Las entrevistas demuestran que los hijos, sobre todo si viven cerca y tienen una clase social baja, se implican bastante en la vida de los pacientes realizando tareas de cuidados.

- *isCaregiverOf* está pensada para el individuo que es un asistente o cuidador, y relaciona al sujeto que cuida. Este puede ser la pareja del paciente, un hijo, una hija o una persona contratada. Esa persona puede ser que no solo cuide al paciente, sino también a la pareja de la misma, ya que en muchos casos esta se encuentra también con cierta discapacidad y/o edad avanzada.

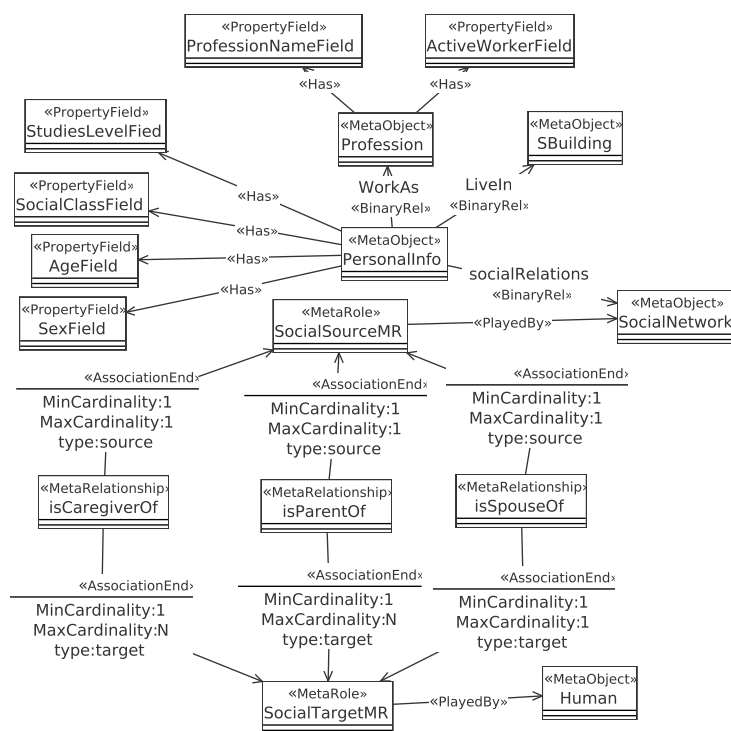


Figura 3.10: Vista del meta-modelo con los conceptos relacionados con el perfil social.

3.3.3.3. Notación y modelos

Para seguir en la línea de conseguir eficiencia cognitiva en el lenguaje y, en este caso, mostrar la información socio-cultural, personal y profesional fácilmente, se han diseñado las representaciones para los conceptos nuevos que muestra la Tabla A.5 del Apéndice A. Destacar que esta vez se han utilizado representaciones para las tres relaciones de la red social, lo que permite leer y entender las notaciones con más rapidez.

La Figura 3.11 muestra dos modelos de perfil social resultado de la lectura de la entrevista a la familia del paciente de párkinson E9. Se puede ver en la Figura 3.11a cómo el modelo recoge toda la información del resumen y alguna más relativa a las relaciones sociales, e.g. que tiene un hijo y una hija. El modelo de la Figura 3.11b recaba la información de la esposa del paciente resultado de leer la entrevista. La mujer tiene 68 años, clase social y estudios medios como el marido, vive en la misma casa que el marido (E9House) y todavía trabajaba como ama de casa. También se refleja que comparten los mismo hijos y que su pareja es E9Patient, a la cual se encarga de cuidar.

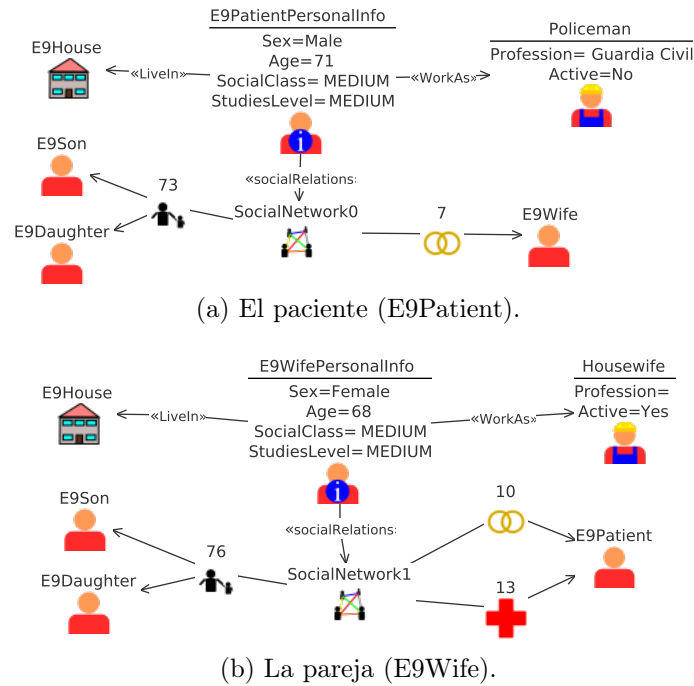


Figura 3.11: Modelos del perfil social del paciente y la pareja.

3.3.3.4. Evaluación

Esta parte del lenguaje no es muy crítica pues de momento solo se emplea para recoger, organizar y visualizar la información socio-cultural, profesional y personal de los sujetos involucrados en el sistema AAL. Como se ha visto en los modelos resultantes, estos recaban todos los datos y se organizan en una sola vista donde se puede ver fácilmente toda la información referente a un sujeto. De nuevo, en la Tabla 3.2 se hace una valoración cualitativa de cómo esta parte de la notación se ajusta a los principios de Moody.

Un error de diseño identificado, hace referencia a que toda la información está distribuida y replicada. Por ejemplo, como muestran los modelos de la Figura 3.11, en la parte de la red social hay muchas redundancias donde se indica dos veces quiénes son los hijos de cada uno, quién es la pareja del otro (propiedad recíproca) y quién cuida a quién. Esto puede llevar a inconsistencias en la información y a no facilitar el mantenimiento de los modelos. También resulta engorroso hacerse una idea mental de todas las relaciones, pues hay que ir mirando todos los perfiles sociales de los individuos. Se considera más interesante hacer un solo diagrama donde se muestren las relaciones sociales que existen entre todos los individuos involucrados y, por otro lado, la información personal y profesional. Un ejemplo de lo que podría ser esta vista de la red social global es la que muestra la Figura 3.12. Se puede comprobar cómo este diagrama, que recuerda a un árbol genealógico, recoge toda la información relativa a las relaciones de todos los individuos de forma concisa y clara. Fíjese que se ha incluido un campo nuevo en la relación *«isCaregiverOf»* con un número entero. Este se ha añadido para

Principios de Moody	Comentario
Claridad simbólica	Se cumple totalmente al existir una representación por cada meta-objeto.
Discriminabilidad perceptual	Se cumple bastante bien al emplear iconos que ofrecen distancia visual. En este caso no se utiliza redundancia gráfica como la forma para identificar tipos de entidades. Pero, por el contrario, también se emplean iconos para las relaciones, lo que permite identificar rápidamente qué relación existe entre el sujeto al que pertenece el perfil y el resto.
Transparencia semántica	Se cumple totalmente al emplear iconos que evocan el significado semántico de lo que representan.
Gestión de la complejidad	En este caso, no es necesario gestionar la complejidad pues son pocos los elementos de los modelos creados.
Integración cognitiva	No es necesaria la integración de diagramas pues no cuelgan diagramas de este.
Expresividad visual	Tiene expresividad visual gracias a los iconos y al texto que los acompañan pero se considera que se mejoraría utilizando formas para diferenciar los tipos de elementos ya que son parecidos entre sí (la mayoría se basan en el icono que representa al humano).
Codificación dual	Se cumple al incluir texto junto con los elementos gráficos (identificadores y algunas valores de los atributos) y también al permitir incluir notas en los diagramas.
Economía gráfica	Se cumple pues hay solo 5 tipos de objetos y 6 tipos de relaciones de las cuales 3 tienen representación.
Ajuste cognitivo	No lo cumple al no proporcionar dialectos.

Tabla 3.2: Evaluación cualitativa de la notación para los elementos del perfil social según los principios de Moody.

ilustrar cómo podría el perfil social modificar el comportamiento de los individuos. El número indica la prioridad por defecto de asistencia al sujeto. La hija es la que tiene mayor prioridad (1) y la compañera es la siguiente (2). Estas prioridades se traducirían en que siempre que se encuentre la hija en vivienda de visita o para ayudar, será ella la que asista al paciente por defecto. Si no se encuentra la hija, es la compañera la que ayudaría al paciente. El hijo solo ayudaría cuando no tenga más remedio, es decir, cuando fuese el único disponible.

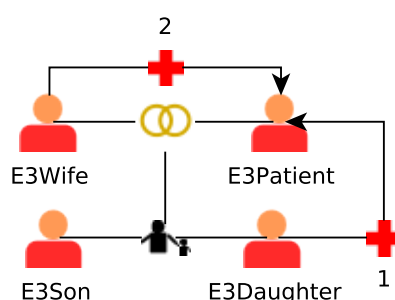


Figura 3.12: Vista del modelo como alternativa para modelar la red social global de todos los individuos.

3.3.4. El perfil de las actividades diarias

3.3.4.1. Introducción

Para modelar las situaciones de interés y definir los escenarios resulta fundamental poder especificar las tareas o las acciones que han de realizar cada individuo. Pero definir el comportamiento de todo un día como una secuencia de acciones resultaría inmanejable. Además, los familiares y pacientes describen su día a día a nivel de actividad, por ejemplo, primero desayunar, luego asearse, después vestirse, etc. Y es en este nivel donde describen cómo varían sus actividades en función del día de la semana, de cómo se encuentren o del tiempo que haga. Por ejemplo, si llueve, en vez de salir a caminar, se queda en casa y hace ejercicio con la bicicleta estática o simplemente ve la televisión. Sin embargo, en las entrevistas, detallan poco la secuencia de tareas que realizan para llevar a cabo cada actividad. Especificar con detalle cada actividad es necesario para que el resultado de la simulación sea realista. El número de actividades que realizan los sujetos en un día también puede ser elevado, así que se considera necesario un método para agrupar las actividades. Los objetivos del perfil ADL son permitir:

- modelar el comportamiento de los sujetos centrándose en las actividades diarias pero disponiendo también de diferentes niveles de detalle de forma que se gestione la complejidad,

- especificar variaciones en las actividades dependiendo del día o estado del sujeto,
- modelar comportamientos de forma determinista y no determinista (aunque sean no deterministas deben ser repetibles),
- reutilizar los comportamientos.

Como se vio en la sección 3.3.2, la gran mayoría de los simuladores enfocados a la Inteligencia Ambiental no ofrecen un lenguaje para modelar los comportamientos. SitCom[35] solo permite modelar el comportamiento a nivel de acciones a lo largo del tiempo. Por el contrario, InSitu[107] solo se queda en el nivel de actividad, las cuales se definen con un nombre, una localización y una duración. Esta especificación solo permite simulaciones poco realistas, pues consisten en un sujeto virtual yendo de una habitación a otra y permaneciendo de pie asumiendo que está realizando la actividad asignada. La notación propuesta por UbikMobile permite realizar especificaciones cada vez con más detalle gracias a su jerarquía, pero al igual que ocurre con los árboles de comportamiento y el lenguaje HCM-L [90] no se diferencia entre momentos del día, actividades y tareas. Estas diferenciaciones se consideran importantes para facilitar la definición y organización de los comportamientos.

3.3.4.2. Propuesta de Meta-modelo

La Figura 3.13 muestra los elementos empleados para dividir el día en franjas. Normalmente, las personas distinguen entre mañana, medio día, tarde y noche. Cada franja está limitada por un hecho concreto, por ejemplo, el inicio de la mañana puede corresponder con la hora a la que el individuo se levanta o el medio día a partir de la hora de comer. Esto son convenciones que dependen de países y culturas, por ello, SociAALML no establece unas franjas fijas si no que se pueden definir tantas como se quieran empleando los elementos que se verán a continuación. La semántica de estos elementos es:

- *TimeInterval* es el meta-objeto que representa un periodo de tiempo del día. Tiene el campo *ActivitySpecField* para especificar las actividades que se encuentran en dicho periodo de tiempo. Las actividades en un periodo de tiempo empiezan si es la hora de inicio de dicho periodo (para ello se utiliza la siguiente entidad) o, si no hay una hora fija, cuando han acabado todas las actividades del periodo anterior. Por el contrario, puede ocurrir que se terminen todas las actividades pero no es la hora de que empiece el siguiente periodo del día. Por ello, la propiedad *DefaultActivityField* indica la actividad por defecto. Las actividades por defecto deberían ser aquellas que se pueden interrumpir en cualquier momento como descansar en el sofá o ver la televisión.
- *ClockTime* es un meta-objeto que se puede utilizar para indicar en qué momento (hora, minutos y segundos) empieza un periodo de tiempo. Las dos entidades se relacionan con la relación *TIStartTime*. Cuando es la hora, se termina la actividad

que el sujeto estuviese realizando lo antes posible para pasar a realizar la primera actividad del momento del día que toca.

- *NextTI* es la meta-relación que se emplea para indicar cuál es el siguiente periodo de tiempo del día. Los periodos de tiempo a lo largo de un día son secuenciales, ocurren uno detrás de otro, y son siempre los mismos. Cuando acaba el último periodo del día se pasa otra vez al primero de forma implícita. El primer periodo del día es aquel que no lo referencia ningún otro.

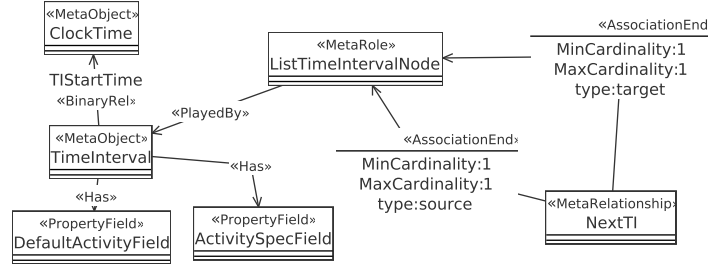


Figura 3.13: Vista del meta-modelo para la agrupación de actividades a lo largo del día.

El siguiente nivel de detalle al de los periodos de tiempo del día son las actividades. Además, este es el nivel que los entrevistados emplean cuando se les pregunta que describan un ejemplo representativo de lo que hacen a lo largo de un día y qué les supone. De acuerdo con la teoría de la actividad [78], una actividad consiste en un conjunto de acciones para conseguir normalmente un objetivo o cubrir una necesidad. Ejemplos de actividades son desayunar, ducharse, dormir, ver la televisión, entre otras. En este caso, las actividades en un periodo de tiempo del día no tienen por qué ser siempre las mismas como ya se comentó en la introducción. Por ejemplo, un individuo puede que no se duche todos los días (y más si resulta una tarea costosa), puede que unos días haga un tipo de ejercicio (e.g. ir a caminar) y otros días otro (e.g. montar en bicicleta estática), o solo vaya 2 días a la semana a hacer la compra.

Atendiendo a las necesidades anteriores, los elementos del meta-modelo que definen el diagrama de actividades (*ActivityDiagram*) se muestran en la Figura 3.14. La semántica de los elementos de esta vista del meta-modelo es:

- *BActivity* representa a una actividad y tiene una propiedad (*SeqTaskDiagramField*) que hace referencia a un diagrama de tareas que la describe. Una actividad termina cuando lo han hecho sus tareas o es hora de que empiece el siguiente periodo del día.
- *NextActivity* indica la siguiente actividad (*BActivity*) a realizarse o el elemento de control de flujo (*IFFlowControl*) a ser evaluado. Como el extremo de la relación puede llevar a dos objetos diferentes, se define el rol *TActivityNode* que es jugado por ambos. Un actividad solo puede relacionarse con una actividad o con un

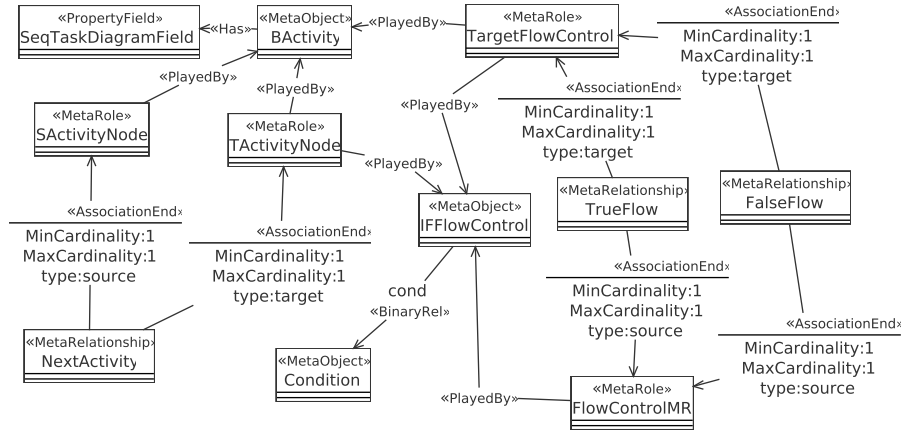


Figura 3.14: Vista del meta-modelo de las actividades del sujeto en un periodo del día.

control de flujo pero no con ambas a la vez. Tanto si se asocia a otra actividad como a un control de flujo, hasta que no termine la actividad no se evalúa el siguiente elemento.

- *IFFlowControl* se emplea para poder especificar diferentes alternativas de actividad en un periodo de tiempo. Este meta-objeto se puede relacionar con un meta-objeto condición (*Condition*) que al evaluarse puede devolver dos resultados, verdadero o falso. Este objeto tiene otras dos relaciones cuyo significado está relacionado con la condición. La meta-relación *TrueFlow* indica la siguiente actividad a realizar si la condición se evalúa como cierta. Por el contrario, la meta-relación *FalseFlow* indica la siguiente actividad a realizar si la condición se evalúa como falsa.
- *Condition* representa una condición o un hecho que puede ser cierto o no. En la Figura 3.15 se puede ver la vista del meta-modelo relativo a las entidades de condición para flujos de actividades. El más inmediato es *CDayOfTheWeek*, empleado para comprobar si es un día concreto de la semana (indicado por el atributo *DayOfTheWeekField*). Otro tipo de condición importante es *CProb* que se basa en la probabilidad uniformemente distribuida entre el 0 y el 1 y se indica con el atributo *ProbVarField*, siendo 0 nada probable y 1 totalmente probable. Esta condición permite modelar comportamientos no deterministas si el atributo toma un valor distinto de 0 y 1. El estado del sujeto es importante también para elegir qué actividades realizar. Por ello, la meta-entidad *CSymptom* comprueba si el sujeto sufre un determinado nivel de un síntoma indicados por los campos *PFIIntensityLevelField* y *PDSymptomTypeField*. También puede variar la elección de las actividades dependiendo de si se encuentra solo en la vivienda o no. Para ello se definen dos condiciones opuestas: *CInside* y *COutside*, ambas con el atributo *HumanTarget* y evalúan si el individuo indicado está en la vivienda o no, respectivamente. Por otro lado, hay situaciones donde se quiere que se realice una o varias actividades hasta que se cumpla una condición, por ejemplo que ocu-

rra un determinado evento (*CEvent*), pase una determinada cantidad de tiempo (*CTimer*) o se alcance una hora (*CTime*).

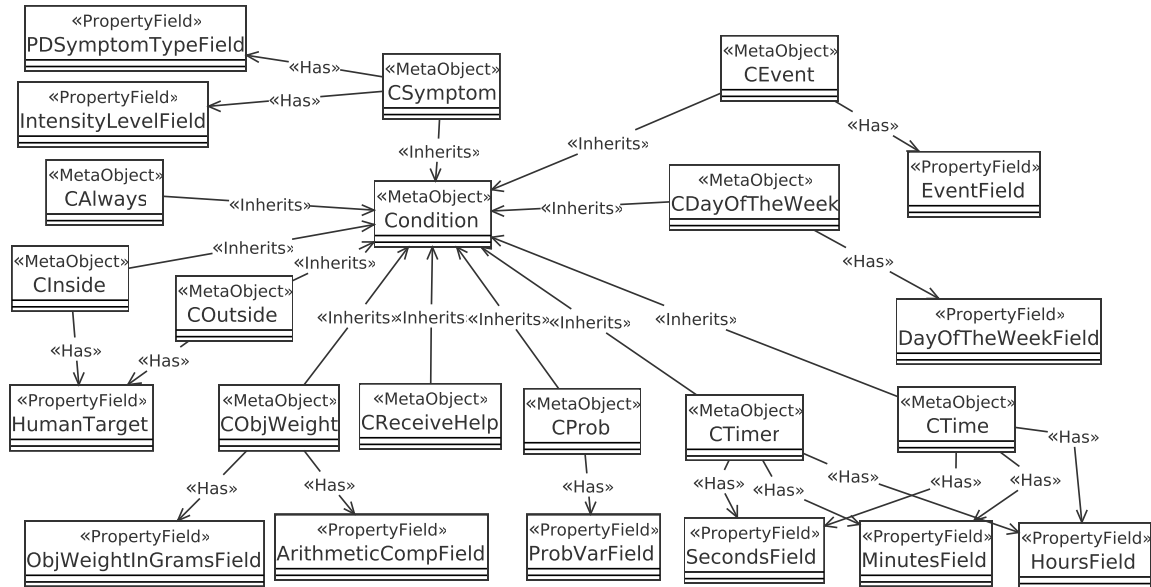
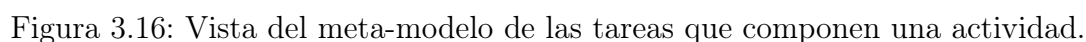


Figura 3.15: Vista del meta-modelo con las entidades que definen las condiciones.

Las tareas corresponden a las acciones de los sujetos y se combinan para realizar las actividades. Dado que el objetivo es reproducir una actividad concreta realizando una serie de pasos, se asume que las tareas no pueden fracasar. Además, también se asume que el número de tareas que se realizan son fijas para una actividad y que no pueden variar, como sí ocurre en los diagramas de actividades empleando los controles de flujo.

Partiendo de las entrevistas y de la experiencia obtenida del uso del lenguaje, las tareas de SociAALML se definen en el meta-modelo de la Figura 3.16. Se han definido estas tareas en base a dos motivos principales. El primer motivo atiende a la lógica, tareas básicas para llevar a cabo las actividades como por ejemplo desplazarse o sentarse. El segundo motivo es resultado de identificar aquellas tareas que le resultan complicadas a los pacientes de párkinson, por ejemplo, ponerse los calcetines o coger cosas pesadas y/o pequeñas.

Todas las tareas tienen un campo llamado *CanBeInterruptedField* que indica si la actividad puede interrumpirse o no. Esta decisión es personal y depende de la situación. Por ejemplo, una tarea de vestirse se puede interrumpir o no dependiendo del contexto. Si los dos miembros de la pareja están en la habitación vistiéndose y el paciente necesita ayuda, no es necesario que el que juega el rol de cuidador termine de vestirse para ayudarlo. Además, una tarea (*BTask*) puede ser simple (*BSimpleTask*) o compuesta (*BCompTask*). Las tareas compuestas son útiles por dos razones: (1) porque reducen la complejidad de los diagramas al poder limitar su número de tareas y (2) porque permiten especificar la forma en la que se llevan a cabo las tareas. Se definen dos tipos



- *BGoToTask* representa la tarea de desplazarse andando a un lugar concreto especificado por el campo *SpaceToGoField* que hace referencia a un meta-objeto *SSpatial*. De forma opcional, se puede indicar la velocidad (en m/s) a la que se desplazará. Esto permite modelar, por ejemplo, la urgencia.

- *GoToBodyLoc* representa la tarea de desplazarse andando a la posición donde se encuentra el individuo especificado en el campo *HumanTarget* que hace referencia a un meta-objeto *Human*. Al igual que la tarea anterior, de forma opcional, se puede indicar la velocidad (en m/s) a la que se desplazará.
- *SayTask* representa la tarea de comunicar el mensaje especificado en el campo *MessageField* a través del habla.
- *FallTask* representa la tarea de caerse al suelo. Esta es una tarea que desgraciadamente los enfermos de párkinson suelen realizar (no voluntariamente por supuesto). Un diagrama de actividad no debería contener una tarea de este tipo pues no es algo que se desee o se realice por costumbre. Se verá su uso más adelante en la sección dedicada a la enfermedad.
- *DropObj* representa la acción de que a un individuo se le caiga de las manos un objeto. Los enfermos de párkinson pierden la precisión en los dedos y suele pasarles que los objetos se les caigan al suelo, sobre todo con los pequeños (e.g. las pastillas o los cubiertos para comer). Al igual que la tarea anterior, esta es una tarea no deseable y que no debería estar en un diagrama de tareas.
- *TapXYTask* corresponde a la acción de pulsar con el dedo una pantalla en una posición concreta. La posición se especifica con coordenadas (x,y) con los campos *XPosOnScreen* e *YPosOnScreen*, respectivamente. El dispositivo sobre el que se realizará la acción se especifica con el campo *TargetSmartphone*. Esta también puede resultar una tarea complicada si el enfermo manifiesta temblores.
- *WaitForBodyClose* es una tarea de sincronización la cual consiste en esperar hasta que el individuo indicado en el campo *HumanTarget* esté próximo o, si no se indica, cuando cualquiera lo esté.

Todas estas tareas tienen una duración variable dependiendo de lo que tarde en realizarse la acción, por ejemplo, desplazarse a un sitio o decir una frase. Se distingue un tipo de tareas simples (*BSimpleDurationTask*) que tienen una duración fija que se especifica con el campo *BtaskDuration*. Estas tareas son:

- *BGetUpFromBed* y *BWakeUpTask* representan meterse y levantarse de la cama, respectivamente. Aunque en un principio, para un individuo sano estas acciones son inmediatas, a determinados enfermos de párkinson les cuesta mucho y pueden llegar a necesitar ayuda.
- *SitDown* y *StandUp* corresponden a sentarse y levantarse de un asiento, respectivamente. La primera tiene el campo *SeatField* opcional, este indica el sitio donde sentarse. Si se especifica se sentará en dicha silla, si no, lo hará en la más próxima. Esta tarea puede ser problemática, al igual que la de meterse en la cama, pero de forma menos acusada.

- *PutOnTask* y *TakeOffTask* representan las tareas ponerse y quitarse el objeto indicado en el campo *WearableObjField*. Este objeto puede ser una prenda de ropa pero también podría ser un reloj. Este tipo de tareas pueden ser complicadas dependiendo del objeto, como ya se ha comentado, ponerse los calcetines y abrocharse la camisa son tareas que acaban siendo complicadas para los enfermos.
- *Drink* y *Eat* simbolizan beber y comer, respectivamente. Cada uno tiene un campo para indicar lo que comen (*DrinkItemField*) o beben (*EatableItemField*). Este tipo de tareas tampoco les resultan sencillas a los enfermos de párkinson. Dependiendo de su estado, necesitarán más o menos tiempo para realizar la tarea o que sufran algún inconveniente (e.g. que se les caiga la comida o los cubiertos).
- *BPickUpTask* y *BLeaveTask* corresponden a la tareas de coger y dejar cosas, respectivamente. El objeto que se coge o deja se indica en el campo *PhysicalMobObjField*. Además, la tarea *BLeaveTask* tiene otro atributo llamado *DestinyField* para indicar dónde dejar el objeto. Dependiendo del tamaño y/o el peso del objeto, estas tareas también pueden ser complicadas, sobre todo la primera.
- *BUseTask* es una tarea bastante genérica que puede representar muchas cosas dependiendo del objeto que se indique en el campo *BUseObjectField*. Por ejemplo, si el objeto es una ducha, lo que representa es ducharse, si es un lavabo representará lavarse las manos o si es un fregadero corresponderá a la tarea de fregar los platos. No se ha encontrado necesario detallar más este tipo de tareas pues no les supone un problema. Por ejemplo, abrir y cerrar el grifo del agua al usar el fregadero. En otros dominios como el Alzheimer, donde a los enfermos se les olvida realizar tareas, seguramente estas sí serían necesarias, pues podría darse la situación no deseada en la que el enfermo se dejase el grifo abierto.
- *WaitTask* es la tarea más genérica. Se emplea para representar cualquier tarea que implique tiempo y no se haya contemplado de momento en el lenguaje. Por ejemplo, puede representar que está barriendo el suelo durante 5 minutos.

3.3.4.3. Notación y modelos

La notación para definir los periodos del día es sencilla pues solo dispone de dos meta-objetos, ver Tabla A.10 del Apéndice A.

La Figura 3.17 muestra un ejemplo de cómo definir los periodos del día con SociAAML. A las 8:00 empieza el día para el sujeto y realiza las actividades de prepararse (*GetReadyTime*). Una vez que está listo, este realiza las actividades de la mañana (*MorningActivities*) definidas en *Do morning activities*. Si ya ha terminado de hacer las actividades de la mañana y le ha sobrado tiempo para empezar con las de comer, se dedicará a descansar (*Rest for a while*). A las 13:00 es su hora de comer y realiza las actividades para ello (*LunchActivities*). Una vez ha hecho la comida, ha comido y recogido, empiezan las actividades de la tarde (*AfternoonActivities*), las cuales se prolongan hasta que son las 20:00, hora de cenar (*DinerActivities*). Otra vez, se define

una tarea por defecto para especificar qué hacer si se acaban las actividades de la tarde y todavía no es la hora para el siguiente momento del día. En este caso, la actividad por defecto consiste en ver la televisión. Después de la cena, empieza el periodo más tranquilo del día (*Evening activities*) que se extiende hasta las 23:00, momento en el que empieza el periodo de las actividades nocturnas (*At night*). Estas consisten en prepararse para ir a la cama y dormir. Fíjese que dormir es la actividad por defecto. Este periodo acabará cuando sean las 8 de la mañana y empiece de nuevo el periodo *GetReadyTime*. Existe una relación implícita entre el primer y el último elemento del día. Esto nos permite no tener que emplear un elemento extra para indicar cuál es el primer momento del día.

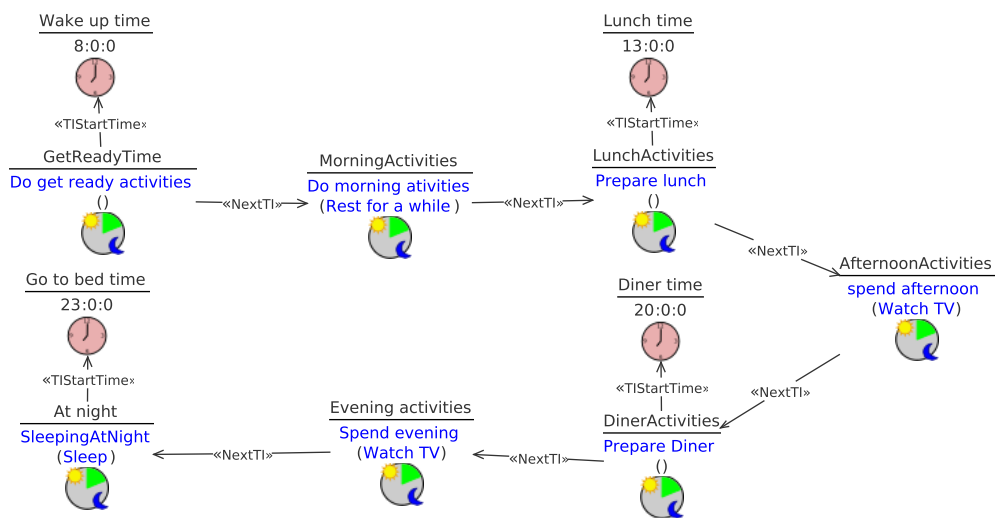


Figura 3.17: Ejemplo de modelo para definir los momentos del día.

La notación de los elementos que definen las actividades de los periodos del día se encuentran en la Tabla A.10 del Apéndice A. En esta ocasión, se utilizan dos variables visuales para diferenciar tipos de entidades. Un rombo con bordes azules representa a las actividades y un pentágono dorado con un símbolo de interrogación representa a las condiciones.

En la Figura 3.18 se muestra un ejemplo de actividades para un periodo de tiempo de la mañana. La primera actividad es desayunar (*Have Breakfast*), y por el nombre del diagrama de tareas, la actividad la debe desempeñar en la silla 1. Una vez ha desayunado, si es jueves, se ducha (*Have a shower*) y después se viste (*Get dressed*), si no, se viste directamente. Cuando ya está vestido, con la misma probabilidad (*Half of the time*), puede que, o bien vea la televisión, o bien lea el periódico.

La notación de los elementos que definen las tareas se encuentran en la Tabla A.8 y A.9 del Apéndice A. Nótese cómo las tareas se identifican con un cuadrado verde. También es común a todas aquellas tareas del tipo duración que se muestre la duración de la forma “For x seconds”, donde x corresponde con el número de segundos que estará realizando dicha tarea.

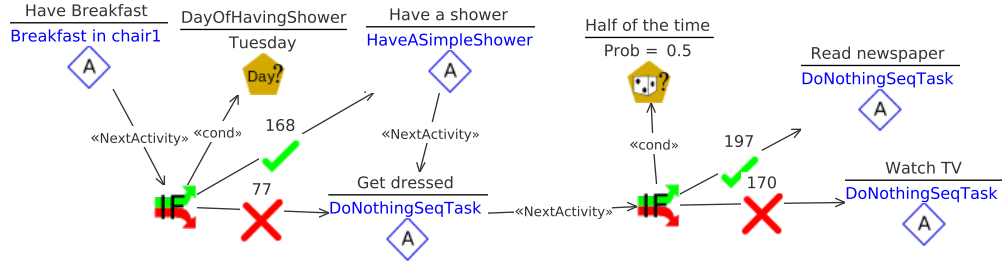
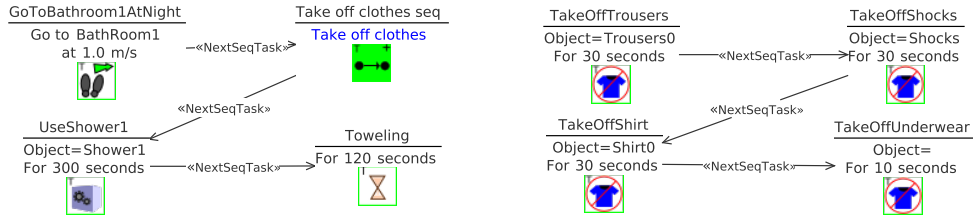


Figura 3.18: Ejemplo de modelo de las actividades del sujeto para un periodo del día.



(a) Primer nivel de secuencia de tareas (b) Subsecuencia de tareas para desvestirse para la ducha.

Figura 3.19: Secuencia completa de las tareas para que el sujeto se duche.

Un ejemplo de especificación para modelar la actividad de ducharse se muestra en la Figura 3.19. Esta se compone de dos diagramas, el principal de la Figura 3.19a y una secuencia de tareas para desvestirse, en la Figura 3.19b. La primera tarea para ducharse consiste en que el sujeto vaya al baño. Después, realice la secuencia de tareas para quitarse la ropa, y una vez desnudo, se meta en la ducha durante 5 minutos. Finalmente, se seca el cuerpo durante 2 minutos. Fíjese cómo la tarea de quitarse la ropa es compuesta y referencia a otro diagrama. De esta forma, se consiguen diagramas más simples y reutilización de comportamientos. Por ejemplo, la subsecuencia de tareas para desvestirse se podría emplear en otra actividad que consista en ponerse el pijama.

3.3.4.4. Evaluación

Como se ha podido comprobar los modelos resultantes permiten modelar los comportamientos en diferentes niveles de detalle: momentos del día, actividades y tareas. Además, las actividades pueden variar atendiendo a diferentes condiciones, tales como el día de la semana o el estado del sujeto. Variando los valores de las condiciones de tipo probabilidad, los comportamientos se pueden volver deterministas o no deterministas. Por último, esta jerarquía de comportamientos (sobre todo con la definición de tareas compuestas) permite la reutilización de comportamientos como ya se vio en el ejemplo de quitarse la ropa. Además, tanto las tareas y como las condiciones aceptan parámetros, los cuales también favorecen la reutilización. No obstante, se ha identificado un error de diseño en el lenguaje que limita la reutilización debido a dos razones principales. Por un lado, las actividades y tareas compuestas, al igual que en los árbo-

les de comportamiento, no permiten parámetros y, por tanto, limitan su reutilización. Por ejemplo, la tarea de desayunar necesita especificar la silla donde se sentará para desayunar. Un miembro de la pareja podría desayunar en la silla con identificador *Chair0* y el otro en la *Chair1*. Por lo que no siempre se puede reutilizar la secuencia de tareas para sujetos diferentes. Lo mismo ocurre con tareas que hacen referencia a lugares. Por ejemplo, para ducharse lo primero que se hace es ir al baño *BathRoom1*. Si la nueva vivienda tiene identificado el baño con otro nombre no se puede reutilizar dicho comportamiento.

La evaluación cualitativa de la notación para esta parte de los comportamientos de los sujetos se encuentra en la Tabla 3.3.

Para ilustrar la diferencia entre esta propuesta y los BTs para modelar los comportamientos, se han creado dos BTs. El primer diagrama, Figura 3.20a, representa los momentos del día de la Figura 3.17. El segundo diagrama (Figura 3.20b), recoge las actividades de la mañana definidas en el diagrama de la Figura 3.18. Recordar que la Tabla 2.9 contiene una leyenda con la representación de los elementos de los árboles de comportamiento. A primera vista, se puede comprobar cómo la notación posee una eficiencia cognitiva menor. Además, el número de elementos necesarios en los BTs para definir lo mismo es mayor. En el caso de los momentos del día, son necesarios 17 frente a 11 en SociAALML. Para el diagrama de las actividades, esta diferencia es menos acusada, 12 frente a 9. Sin embargo, la lectura de SociAALML resulta más intuitiva gracias a la notación y a las relaciones dependientes del dominio. Resulta, por ejemplo, algo complicado modelar el hecho de que los jueves es el día de ducharse utilizando el operador de selección. La idea es: tengo éxito si no es jueves (con el operador de inversión) ya que no hay que ducharse, si no, es jueves y hay que ducharse. Se debe llevar cuidado con los operadores secuenciales de los árboles de comportamiento, pues un fracaso no permitirá que se sigan ejecutando el resto de opciones.

3.3.5. El perfil de las interacciones

3.3.5.1. Introducción

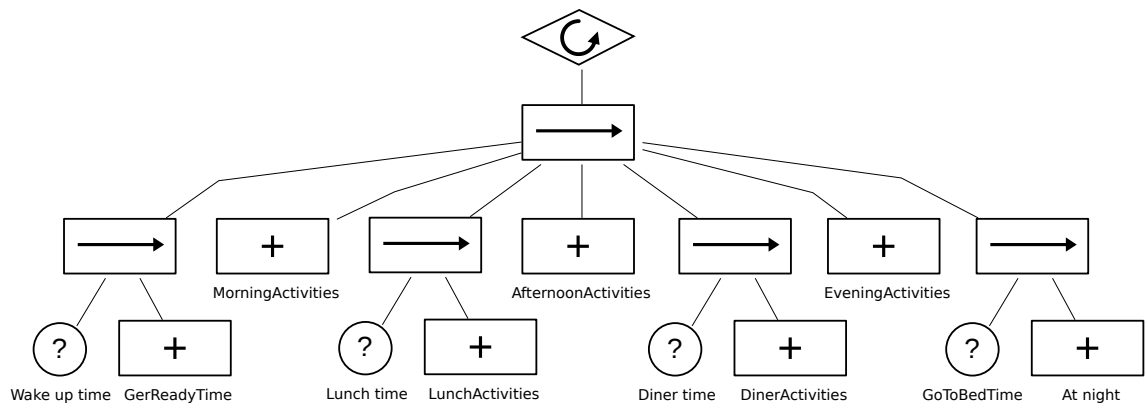
Hasta ahora, se han dado mecanismos para especificar las actividades y tareas que realiza un sujeto a lo largo del día. Pero, a lo largo de todo un día, pueden ocurrir situaciones o eventualidades (e.g. que llamen al teléfono o que el paciente necesite ayuda) que interrumpan las actividades de un sujeto.

Dependiendo del tipo de entidades que toman parte en la interacción, estas pueden ser:

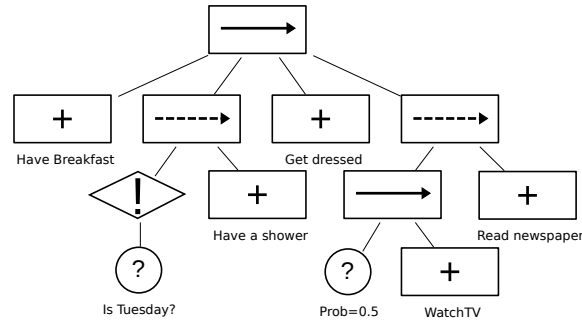
- *Sujeto-entorno*, la cuales se realizan, principalmente, a través de las tareas descritas en la sección anterior.
- *Sujeto-dispositivo* son aquellas secuencias de acciones que realiza un sujeto con un dispositivo. En el perfil de las actividades diarias se definieron también acciones para interactuar con un dispositivo: coger el dispositivo con una mano, dejar

Principios de Moody	Comentario
Claridad simbólica	Se cumple totalmente al existir una representación por cada meta-objeto.
Discriminabilidad perceptual	Se cumple bastante bien al emplear iconos que ofrecen distancia visual. También se utilizan formas y colores para distinguir entre tipos de entidades. El círculo relleno de gris para los momentos del día, el rombo con contorno azul para las actividades, el pentágono dorado para las condiciones y el cuadrado verde para las tareas.
Transparencia semántica	Se cumple totalmente al incluir iconos que evocan el significado semántico de lo que representan.
Gestión de la complejidad	Se gestiona la complejidad al emplear la jerarquía de elementos más generales a más concretos, es decir, momentos del día, actividades, tareas compuestas y simples. De esta forma, el número de elementos por diagrama no es elevado.
Integración cognitiva	Como ya se ha comentado, INGENME no permite elementos de integración de varios diagramas en uno para tener vistas generales. Sin embargo, sí facilita elementos de navegación a través de enlaces para ir a nuevos diagramas tal como muestra los elementos momento del día, actividad y las tareas compuestas.
Expresividad visual	Tiene expresividad visual sin llegar a saturar al emplear 3 elementos visuales (los iconos, la forma y el color en algunos casos).
Codificación dual	Se cumple al incluir texto junto con los elementos gráficos (identificadores y algunas valores de los atributos) y también al permitir incluir notas en los diagramas.
Economía gráfica	Hay 33 elementos gráficos ya que hay muchos elementos para las tareas y las condiciones y se desea cumplir el principio de claridad simbólica.
Ajuste cognitivo	No lo cumple al no proporcionar dialectos.

Tabla 3.3: Evaluación cualitativa de la notación para los elementos del perfil ADL según los principios de Moody.



(a) Modelo de los momentos del día equivalente al diagrama de momentos del día de la Figura 3.17.



(b) Modelo de las actividades de la mañana que equivale al de la Figura 3.18.

Figura 3.20: Modelos de árboles de comportamiento para modelar los momentos del día y las actividades.

el dispositivo en algún mueble y, la más importante, pulsar sobre una posición de la pantalla. Esta última permite que el sujeto virtual pueda interaccionar con las aplicaciones. El dispositivo interacciona a través de sus sensores y actuadores. Sin embargo, los actuadores no están modelados y, por tanto, no es posible que el sujeto virtual reaccione a los dispositivos, por ejemplo cuando suene el timbre o el teléfono. Se considera necesario incluir esta funcionalidad en trabajos futuros a corto plazo.

- *Sujeto-sujeto* son las más complicadas. Dejando a parte medios de comunicación tecnológicos, los humanos se pueden comunicar con el habla y con gestos. Esta comunicación permite, por ejemplo, realizar tareas de forma sincronizada, por ejemplo, que un cuidador asista a un enfermo de párkinson. La parte del lenguaje dedicada a las actividades de la vida diaria incluye tareas propias de la comunicación (e.g. la de decir un mensaje), sin embargo, estas no aportan ninguna semántica. En este apartado se trata este tipo de interacciones.

Cuando se genera el código de las simulaciones, son los agentes los que implementan el comportamiento de los sujetos. Los mecanismos que ofrecen los sistemas multi-agentes para comunicarse (por ejemplo, el estándar FIPA) pueden parecer un buen mecanismo. En UbikMobile se utiliza una propuesta basada en este estándar [30]. Pero en los mundos virtuales, los agentes deberían hacer uso de su cuerpo y del entorno para comunicarse como si fueran sujetos reales. Este uso de gestos hace que los protocolos fijos del estándar FIPA no sean suficientes [139]. A este tipo de agentes que se desenvuelven en entornos virtuales con cuerpo se les llama IVA (*Intelligent Virtual Agent*, Agente inteligente virtual) y es un campo de investigación en sí mismo conseguir que estos agentes se comuniquen con otros empleando tanto lenguaje verbal como no verbal. Lo ideal sería conseguir crear un lenguaje que consiguiese comunicaciones tan realistas como los que persiguen los agentes IVA, pero esto supondría otra tesis. En vez de esto, se ha optado por un enfoque muy sencillo pero suficiente para permitir alterar el comportamiento de los sujetos virtuales y que interaccionen entre ellos de forma básica. Este enfoque se basa en eventos inspirados en los Smart Events [132] de Stoker. El evento contiene toda la información de la interacción y toma el control de los participantes. Cuando se acaba el evento, cada agente vuelve a tomar el control de sí mismo. Esta idea también es aplicada a los árboles de comportamiento en [68], donde las acciones del árbol asociado al evento tienen una etiqueta con el participante que tiene que realizar la acción. De esta forma, en un solo árbol se coordinan las acciones de los participantes. En SociAALML se emplea una alternativa todavía más sencilla. Los eventos se pueden asociar a las tareas y cuando estas se realizan, los eventos se generan pudiendo ser percibidos por los agentes. Cada agente tiene definido cómo reacciona ante cada evento. Esta reacción puede consistir en no hacer nada o realizar una actividad. A continuación se ve más en detalle esta propuesta.

3.3.5.2. Propuesta de Meta-modelo

Si se vuelve a la vista del meta-modelo dedicado a definir las tareas de los comportamientos, Figura 3.16, se puede comprobar que a las tareas se les pueden asociar eventos con la relación «*ProducesEvent*». Esto significa que cuando se realice la tarea, el o los eventos asociados se generarán. Un evento tiene un campo que indica la prioridad o importancia del evento con tres niveles (LOW, MEDIUM, HIGH). Los eventos pueden ser percibidos o no por otros agentes. De momento, se asume que todos los eventos se propagan por el aire y tienen un alcance de 10 metros. De esta forma, se tiene un mecanismo sencillo y potente de identificar tareas y notificarlas. Por ejemplo, se puede asociar un evento a una tarea de hablar con el mensaje “¿Me puedes ayudar con los botones de la camisa?” o a una que corresponda a una caída (ya que esta producirá sonido).

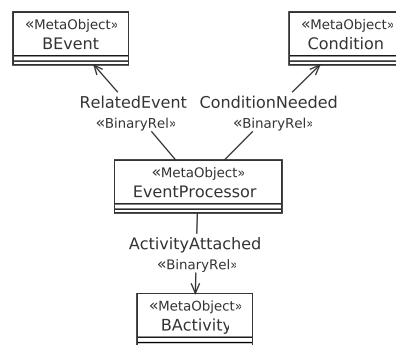


Figura 3.21: Vista del meta-modelo de las interacciones.

Para especificar cómo reacciona cada individuo a los eventos se crea un meta-objeto nuevo (*EventProcessor*) y tres meta-relaciones (*RelatedEvent*, *ConditionNeeded* y *ActivityAttached*) como muestra la vista del meta-modelo de la Figura 3.21. El meta-objeto se conecta con las nuevas meta-relaciones con otros meta-objetos ya conocidos. Para indicar cómo se va a reaccionar ante un evento, lo primero es indicar el evento con la relación «*RelatedEvent*». Ante un mismo evento no se tiene por qué reaccionar siempre de la misma manera. Por ejemplo, si el enfermo tiene contratado un cuidador y necesita asistencia, no lo hará el familiar sino el cuidador. Por tanto, con la relación «*ConditionNeeded*», se indica en qué condiciones se va a reaccionar ante el evento. Por último, cuando la condición se cumple, con la relación «*ActivityAttached*» se indica qué actividad realizará el sujeto. Cuando se dan las condiciones para realizar una tarea asociada a un evento, el sujeto interrumpe la tarea de la actividad que esté realizando en ese momento para realizar la especificada. Recordar que las tareas pueden marcarse como que no se pueden interrumpir. Si durante el evento se está realizando una tarea que no se puede interrumpir, la actividad se interrumpirá tan pronto como dicha tarea termine. Una vez terminada la actividad asociada al evento, el sujeto reanuda la actividad que estaba realizando. Si se está realizando la actividad asociada a un evento y se produce otro evento, el cual es percibido y se decide atenderlo, pueden ocurrir dos cosas dependiendo de las prioridades. (1) Si la prioridad del nuevo evento es superior

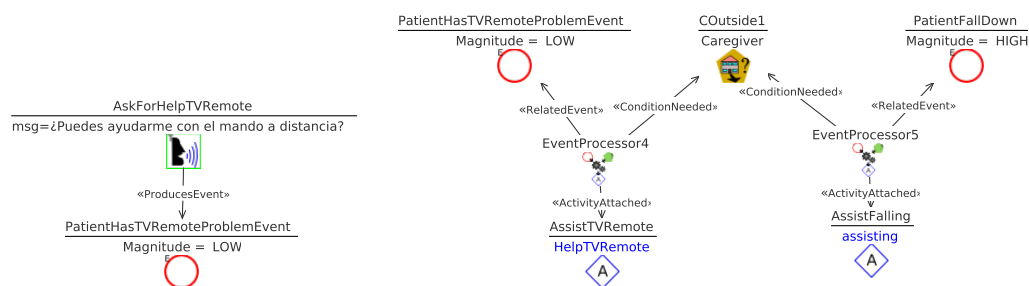
a la del evento actual, la actividad se suspende (al igual que ocurre con las normales) y se realizará la nueva actividad más prioritaria. Cuando ésta termine, se retomará la actividad del evento interrumpido. (2) Si la prioridad del nuevo evento es igual o inferior a la del evento actual, el evento se atenderá (se realizará la actividad) después de finalizar el actual.

Para decidir si se procesan los eventos, además de las condiciones definidas previamente, se define una nueva cuyo meta-objeto es *CReceiveHelp*. Esta condición evalúa si el sujeto está recibiendo ayuda. Si alguien ya está ayudando al paciente, puede no ser necesario ir a ayudarlo.

3.3.5.3. Notación y modelos

La notación para definir los eventos y cómo reaccionar ante ellos se muestran en la Tabla A.10 del Apéndice A.

Para ilustrar el uso de este mecanismo y modelar la interacción de los sujetos, supóngase que un paciente quiere pedir ayuda para ver la televisión ya que debido al temblor de las manos no consigue pulsar los botones del mando de manera adecuada. En este caso, el paciente pide ayuda con la tarea *Say* y el mensaje “¿Puedes ayudarme con el mando a distancia?”, como muestra la Figura 3.22a. Por otro lado, en la Figura 3.22b se puede ver un ejemplo de lo que podría ser el diagrama de interacción de un familiar. Especifica cómo reaccionar ante dos eventos: el paciente tiene problemas con el mando a distancia (*PatientHasTVRemoteProblemEvent*) y el paciente se ha caído (*PatientFallDown*). Nótese cómo el evento relativo a la caída tiene prioridad alta y el de ayudar con el mando a distancia tiene prioridad baja. Se puede ver que en ambos casos la condición para intervenir depende de que el asistente no se encuentre en casa (*COutside1*). Pero la actividad a realizar variará dependiendo del evento. En caso de necesitar ayuda con el mando a distancia, el familiar realiza las tareas correspondientes definidas en el diagrama *HelpTVRemote* y, en caso de una caída, responderá llevando a cabo las tareas del diagrama *assisting*.



(a) Ejemplo de modelo donde una tarea de hablar produce un evento. (b) Ejemplo de modelo del diagrama de interacciones de un familiar.

Figura 3.22: Ejemplo de modelos para asistir al paciente con el mando a distancia.

3.3.5.4. Evaluación

El mecanismo presentado para modelar las interacciones de los sujetos es sencillo y suficiente para especificar interacciones simples como las mencionadas. Un paciente solicita ayuda y el familiar o asistente le ayuda o realiza la tarea que el paciente no puede realizar, por ejemplo, encender la televisión. No obstante, la propuesta tiene una pequeña desventaja. Para entender qué ocurre ante un determinado evento, es necesario comprobar varios diagramas, al menos todos los de interacción de los sujetos modelados. Pero este inconveniente se alivia gracias al editor. Este incorpora una herramienta para que muestre todos los diagramas que contienen una entidad concreta, en este caso el evento de interés.

Esta propuesta basada en eventos es suficiente para realizar gran cantidad de escenarios. Sin embargo, resulta insuficiente para especificar actividades conjuntas donde hay que coordinar las acciones de cada sujeto. Como trabajo futuro se considera interesante incluir el concepto de Smart Event [132] al lenguaje, para poder coordinar actividades conjuntas de forma más simple y centralizada.

3.3.6. El perfil de la enfermedad

3.3.6.1. Introducción

El párkinson es una enfermedad crónica, neurodegenerativa y progresiva que afecta principalmente al aparato locomotor provocando cada vez más dependencia en el paciente a la hora de realizar actividades físicas. Esta dependencia lleva a una disminución considerable de la calidad de vida del enfermo y de sus familiares [82].

Modelar la enfermedad es fundamental ya que el comportamiento del paciente se ve limitado y, en consecuencia, el de los familiares y cuidadores se ve alterado. Estas situaciones donde se disminuye la calidad de vida de los individuos son las que se quieren recrear en la simulación para, posteriormente, ser solucionadas con el sistema AAL. Tras el estudio de las entrevistas, los libros de la asociación de Parkinson Madrid [84, 123, 91] y algunos trabajos sobre la enfermedad como [82] (donde se hace un estudio de cómo afecta el Parkinson a los enfermos en la realización de actividades cotidianas), se han identificado los conceptos más importantes sobre la enfermedad que debería recoger el lenguaje:

- Información sobre el estadio de la enfermedad (del E1 al E5).
- Información sobre los síntomas que padece el paciente. Esta debe poder describir cómo evoluciona el síntoma en términos de niveles de intensidad. No se puede hacer una relación directa entre estadios, síntomas y su evolución pues cada paciente de párkinson experimenta los síntomas y se le manifiestan de forma diferente [91, p.13].
- Las limitaciones que provoca cada síntoma dependiendo de su nivel de intensidad y de cómo reacciona el paciente ante dichas limitaciones.

- Información sobre los medicamentos, cuándo han de tomarlos y en qué cantidad para aliviar los síntomas. Para el tratamiento de la enfermedad de Parkinson, es importante que los enfermos tomen la cantidad adecuada de medicamentos a la ahora indicada para así poder aliviar los síntomas el máximo tiempo posible [59].

3.3.6.2. Propuesta de Meta-modelo

Para cumplir con los objetivos concretos para la enfermedad, se crean bastantes elementos nuevos en el meta-modelo. Para que los modelos sean manejables, se dividen los conceptos en cuatro vistas: la descripción general de la enfermedad, la de las tomas de los medicamentos, la evolución de los síntomas y la de las limitaciones de los síntomas.

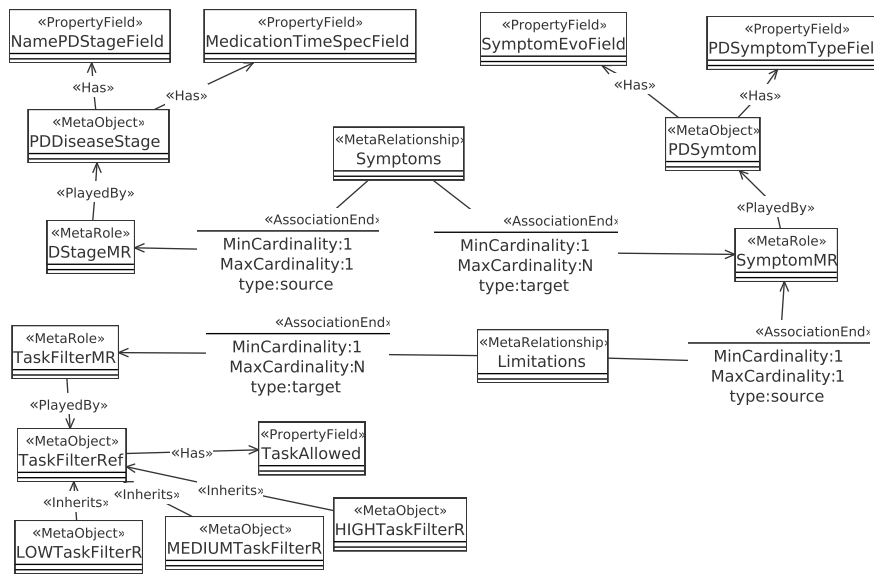


Figura 3.23: Vista del meta-modelo de la enfermedad de Parkinson.

Los elementos para describir los conceptos más importantes de la enfermedad se encuentran en diagrama *ParkinsonSpecDiagram*. Estos elementos se definen en el meta-modelo de la Figura 3.23. La entidad principal es *PDDiseaseStage*, la cual tiene un campo para definir el estadio en el que se encuentra la enfermedad (*NamePDStageField*) y otro (*MedicationTimeSpecField*) que hace referencia a un diagrama donde se recoge la información relativa a la medicación. Además, cada etapa de la enfermedad de Parkinson lleva asociada uno o varios síntomas (*PDSymptom*) a través de la meta-relación «*Symptoms*». Un síntoma tiene dos campos *PDSymptomTypeField* y *SymptomEvoField*. El primero contiene el nombre del tipo de síntoma, se puede especificar uno nuevo escribiéndolo o seleccionar uno de los más comunes que están ya definidos. El segundo campo contiene una descripción de cómo evoluciona el nivel de intensidad del síntoma. De las entrevistas se ha extraído que la intensidad de un síntoma se puede clasificar de forma suficiente en 4 estados: no se manifiesta (NONE), se manifiesta levemente (LOW), se manifiesta moderadamente (MEDIUM) y se manifiesta intensamente

(HIGH). Por otro lado, cada síntoma tiene asociadas unas limitaciones (*TaskFilterRef*) a través de la relación «*Limitations*». Dependiendo del nivel de intensidad del síntoma, se aplican las limitaciones definidas para dicho nivel (*LOWTaskFilterR*, *MEDIUMTaskFilterR*, *HIGHTaskFilterR*). Fíjese que no hay un meta-objeto para cuando no se manifiesta el síntoma (este sería *NONETaskFilterR*), pues al no haber síntoma no hay limitaciones. A través del campo *TaskAllowed* se referencia al diagrama (*FilterDiagram*), el cual define las limitaciones como se verá más adelante.

Las tomas de los medicamentos del paciente se especifican en el diagrama *MedicationTimeDiagram* cuyos elementos se encuentran en el meta-modelo de la Figura 3.24. El modelo resultante debe recoger a qué horas, qué medicinas y en cuánta cantidad debe tomar el paciente de párkinson. Para ello, se define la entidad *MedicationTime* que representa una hora del día en la que hay que tomar medicamentos. La hora se especifica a través del campo *ClockTimeField*. Utilizando la relación *MedicinesIntakes* se asocian horas con tomas (*MedIntake*). Cada toma indica la cantidad o dosis a tomar con el campo *DosesNumberField* y el o los medicamentos (*Medication*) con la relación *med*. Se ha identificado un tipo específico de toma que en vez de ser oral es intravenoso (*MedInjection*) aunque no se suele emplear.

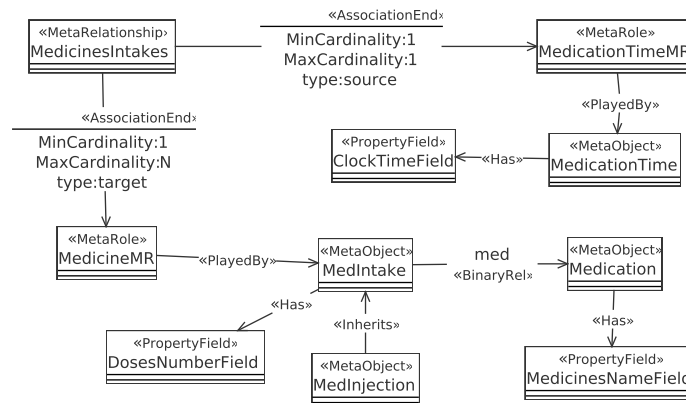


Figura 3.24: Vista del meta-modelo de la toma de medicamentos del enfermo de párkinson.

La evolución y sintomatología de cada enfermo de párkinson depende de cada sujeto, por ello, se da gran flexibilidad y no se restringe los síntomas por etapa. Dependiendo del paciente que se esté modelando, se puede definir el conjunto de síntomas necesarios para dicho paciente. Durante las entrevistas, los enfermos de párkinson relatan cómo evolucionan los síntomas en términos de tiempo. Normalmente, indican cómo se encuentran tanto tiempo después de la última toma de medicamentos, pero también puede depender de otros factores, e.g. que tengan un mal día. Por tanto, para cada síntoma se modela cómo evoluciona, es decir, cómo varían su niveles de intensidad a través de un autómata de estados finitos definido por el meta-modelo de la Figura 3.25. El estado del síntoma lo representa el meta-objeto *SymptomLevelState* y dependiendo de su nivel de intensidad, se definen cuatro meta-objetos que especializan al anterior: *NONESymptomLevelState*, *LOWSymptomLevelState*, *MEDIUMSymptomLevelState* y *HIGHSymptomLevelState*,

cada uno representa el nivel de intensidad nulo, bajo, medio y alto respectivamente. Las transiciones se definen con la meta-relación *SymptomTransition* que tiene como origen (*PreCondSymptomMR*) uno o varios estados (*SymptomLevelState*) y cero o más condiciones (*Condition*). En el extremo de la relación (*PostCondSymptomMR*) solo se puede encontrar un estado. Téngase en cuenta que se necesitará estados adicionales para un mismo nivel de intensidad. Normalmente, serán necesarios dos, uno para indicar que el nivel del síntoma está en disminución porque, por ejemplo, se ha tomado la medicina, así que la siguiente transición será un estado con un nivel inferior. El otro estado será necesario para indicar que el nivel del síntoma está en aumento y la siguiente transición será hacia un estado con un nivel superior de síntoma.

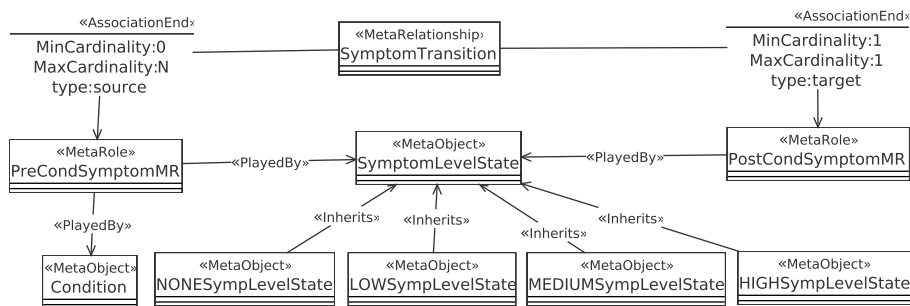


Figura 3.25: Vista del meta-modelo para definir la evolución de los niveles de intensidad de un síntoma.

Las limitaciones provocadas por los síntomas se modelan utilizando los elementos definidos en la vista del meta-modelo de la Figura 3.26. Hay que tener en cuenta que las limitaciones se definen a nivel del comportamiento de las tareas. Ejemplos de tareas comunes que se ven afectadas son andar o ponerse los calcetines debido a la rigidez, comer o abrocharse la camisa debido a los temblores, entre otras. Normalmente, los pacientes necesitan asistencia para determinadas tareas de una actividad y no durante toda la actividad. Por ello, esta parte del meta-modelo persigue describir cómo se ven afectadas dichas tareas. La idea general consiste en definir filtros o elementos que seleccionan, alteran o sustituyen las tareas. El meta-objeto *FTaskFilter* corresponde al concepto general de filtro. Éste puede estar relacionado con un meta-objeto condición (*Condition*) para indicar bajo qué condiciones se aplica el filtro. Dependiendo de si la condición se cumple o no, el filtro se puede unir con otros filtros con dos tipos de relaciones (ver Figura 3.27): *«NextFilter»* y *«FAlternative»*. Si la condición es cierta, el filtro se aplica a la tarea de entrada y la tarea resultante se manda al siguiente filtro indicado por la relación *«NextFilter»*. Por el contrario, si la condición es falsa, el filtro no se aplica a la tarea de entrada y esa misma tarea se manda al siguiente filtro indicado por la relación *«FAlternative»*. Puede verse que se definen varios tipos de filtros, algunos juegan el rol *SourceFilter* y otros el de *TargetFilter*, dependiendo de si pueden mandar una tarea a otro filtro o recibirla, respectivamente.

Los tipos de filtros definidos son:

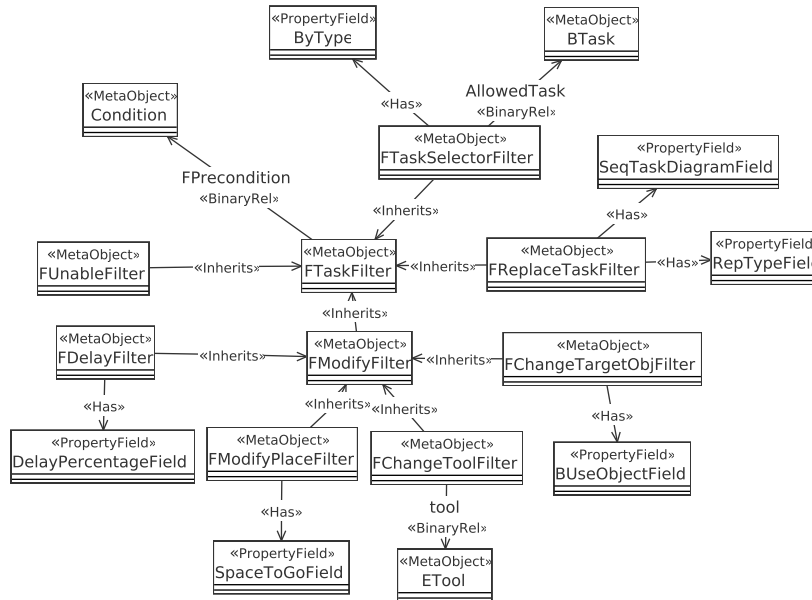


Figura 3.26: Vista del meta-modelo para definir las modificaciones de las tareas según el nivel del síntoma.

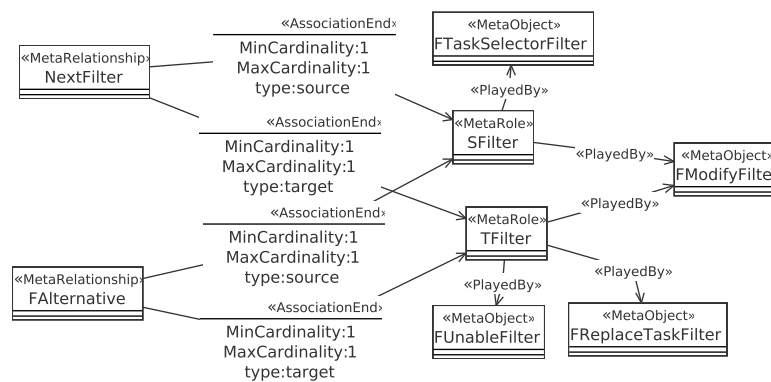


Figura 3.27: Vista del meta-modelo para definir los tipos de relaciones según el tipo de filtro.

- *FTaskSelectorFilter* se utiliza para indicar qué instancias o tipo de tareas se van a filtrar. Para ello, tiene una relación binaria «*AllowedTask*» con las tareas. Se puede indicar tantas instancias de tareas como se quiera. El campo *ByType* es muy útil, pues permite filtrar las tareas por tipo. De esta forma, si se desea aplicar los filtros a tareas de tipo desplazarse, solo tendrá que poner una instancia de ejemplo y marcar este campo como verdadero. Fíjese que otros filtros no se pueden conectar a éste, ya que solo juega el rol *SourceFilter*.
- *FUnableFilter* suprime la tarea directamente, es decir, que no devuelve ninguna tarea por lo que a continuación de este filtro no puede ir ningún otro. Por ello, solo juega el rol *TargetFilter*. Este filtro se puede aplicar a tareas que no son decisivas para completar la actividad, e.g. decir algo.
- *FReplaceTaskFilter* reemplaza la tarea que se le pasa por una secuencia de tareas definidas en el campo *SeqTaskDiagramField*. Por ejemplo, el filtro podría tener la condición de probabilidad y la tarea de andar se podría reemplazar por la de caer-se y luego levantarse. Para facilitar la reutilización de diagramas, posteriormente, se definió el campo *RepTypeField*, para indicar el tipo de filtro de reemplazo. Este puede tomar tres valores: REPLACE (valor por defecto), BEFORE y AFTER. BEFORE indica que las tareas definidas en *SeqTaskDiagramField* se realizarán antes de la tarea filtrada. En cambio, si el campo toma el valor AFTER, dichas tareas se realizarán después. BEFORE es muy útil para especificar algún problema y, posteriormente, reanudar la tarea filtrada. Por ejemplo, para cualquier tipo de tarea de tipo desplazamiento, ésta se puede filtrar anteponiendo la tarea de caerse y levantarse con una pequeña probabilidad. Sin esta mejora, es necesario definir tantos filtros de reemplazo como tareas de desplazamiento haya definidas.
- *FDelayFilter* tiene un campo *DelayPercentageField* que se emplea para indicar en qué medida la tarea filtrada se realiza más rápida o más lenta. Este filtro se aplica a las tareas de tipo duración (*BSimpleDurationTask*) de forma que su campo duración se modifica según la fórmula $d * v/100$, donde d es la duración original de la tarea y v es el valor del campo *DelayPercentageField*. Este filtro también se puede aplicar a las tareas de desplazamiento, pero afectando al campo de la velocidad de forma inversa, es decir, $s * (100/v)$, donde s es la velocidad.
- *FModifyPlaceFilter* modifica el lugar donde el sujeto se va a desplazar. Este filtro tiene efecto en las tareas de desplazamiento.
- *FChangTargetObjFilter* modifica el objeto de la tarea. Por ejemplo, para la tarea sentarse se puede cambiar la silla donde se vaya a sentar.
- *FChangToolFilter* modifica la herramienta utilizada para realizar una tarea. La herramienta necesaria (meta-objeto *ETool*) para realizar la tarea se asocia con la relación «*tool*». Este filtro está pensado para caracterizar las tareas indicando la ayuda que se necesita para realizar la tarea. Esta ayuda puede ser un sujeto, un

objeto o herramienta. Por ejemplo, dependiendo de la rigidez, el enfermo podría necesitar usar un bastón para desplazarse.

3.3.6.3. Notación y modelos

Los nuevos elementos de la notación para especificar modelos de la enfermedad se muestran en la Tabla A.11 del Apéndice A. Destacar sobre la notación cómo se diferencian entre ellos tanto por el color como por la forma: los síntomas tienen forma de estrella azul y los filtros forma de embudo con tres posibles colores (amarillo, naranja y rojo) dependiendo del nivel de intensidad del síntoma a filtrar.

Los elementos de la notación para especificar la toma de medicamentos se muestra en la Tabla A.12 del Apéndice A.

La notación para las entidades que forman los diagramas para mostrar las limitaciones se muestran en la Tabla A.14. Fíjese que los filtros tienen una forma nueva, no empleada hasta ahora, son de color azul y todos comparten el mismo símbolo de ruedas dentadas indicando un proceso de modificación de las tareas. También se destaca dos tareas nuevas no deseadas (la caída al suelo del sujeto y la de caerse cosas de las manos al suelo) y, por ello, se diferencian por el borde de color rojo.

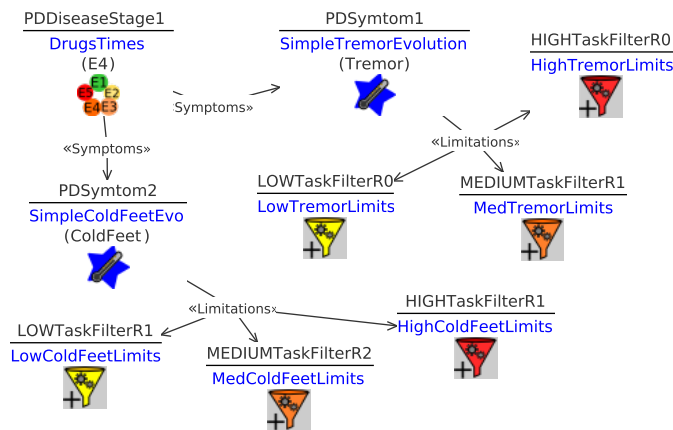


Figura 3.28: Ejemplo de modelo de la enfermedad de Parkinson.

Para ilustrar el uso del meta-modelo, supóngase un enfermo de párkinson que se encuentra en el estadio E4 y que se está interesado en el síntoma de pies congelados y en el de los temblores. Dependiendo de los niveles de intensidad de los síntomas, el paciente experimenta unas limitaciones u otras. La Figura 3.28 contiene el modelo equivalente a esta descripción. Se puede apreciar cómo se muestra la información general sobre la enfermedad y cómo todas las entidades poseen enlaces para acceder a otros modelos. La entidad *PDDiseaseStage1* contiene el nombre del diagrama con la información sobre las tomas de la medicación. Los síntomas, *PDSymptom2* y *PDSymptom1*, tienen enlaces a modelos que describen cómo evolucionan sus niveles de intensidad, *SimpleColdFeetEvo* y *SimpleTremorEvolution*, respectivamente. Por último, cada filtro hace referencia a su descriptor, los cuales definen las limitaciones para el nivel de intensidad del síntoma correspondiente.

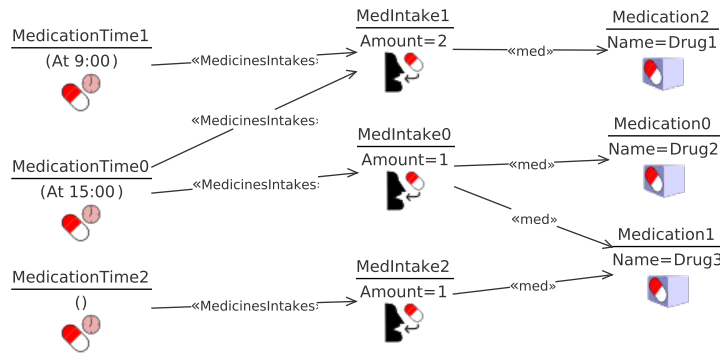


Figura 3.29: Ejemplo de modelo donde se especifica la toma de medicamentos del enfermo de párkinson.

En la Figura 3.29 se muestra el modelo con la información para las tomas de los medicamentos. Este modelo define las horas a las que el paciente debe tomarse sus medicinas, en este caso a las 9:00, a las 15:00 y a las 23:00. A las 9:00, el paciente se tiene que tomar dos pastillas de la medicación con nombre *Drug1*. A las 15:00 el paciente se tiene que tomar las dos mismas pastillas de la mañana, más una del medicamento *Drug2* y otra del medicamento *Drug3*. Por último, a las 23:00, el paciente solo tiene asignado una pastilla del medicamento *Drug3*.

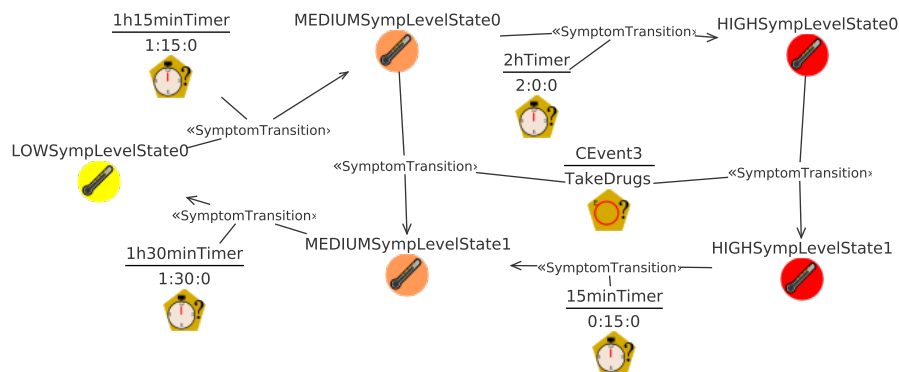


Figura 3.30: Ejemplo de modelo para describir la evolución del síntoma de pies congelados.

Supóngase que el modelo de la Figura 3.30 describe cómo evolucionan los niveles de intensidad del síndrome de pies congelados del enfermo. Por un lado, se puede comprobar que el sujeto siempre manifiesta el síntoma, pues no existe ningún estado del tipo *NONESymptomLevelState*. Si se presta atención a los periodos de recuperación, se puede comprobar que si se encuentra en el nivel de intensidad alto del síntoma (estado *HIGHSymptomLevelState0*) y se toma la medicación (condición *CEvent3*) se pasa al estado *HIGHSymptomLevelState1* donde el nivel del síntoma es alto pero, tras 15 minutos, el nivel se reduce a intensidad media (estado *MEDIUMSymptomLevelState1*) y tras una hora y media pasa al nivel bajo (estado *LOWSymptomLevelState0*). Si se toma la medicación (condición *CEvent3*) cuando está en el nivel medio (estado *MEDIUMSym-*

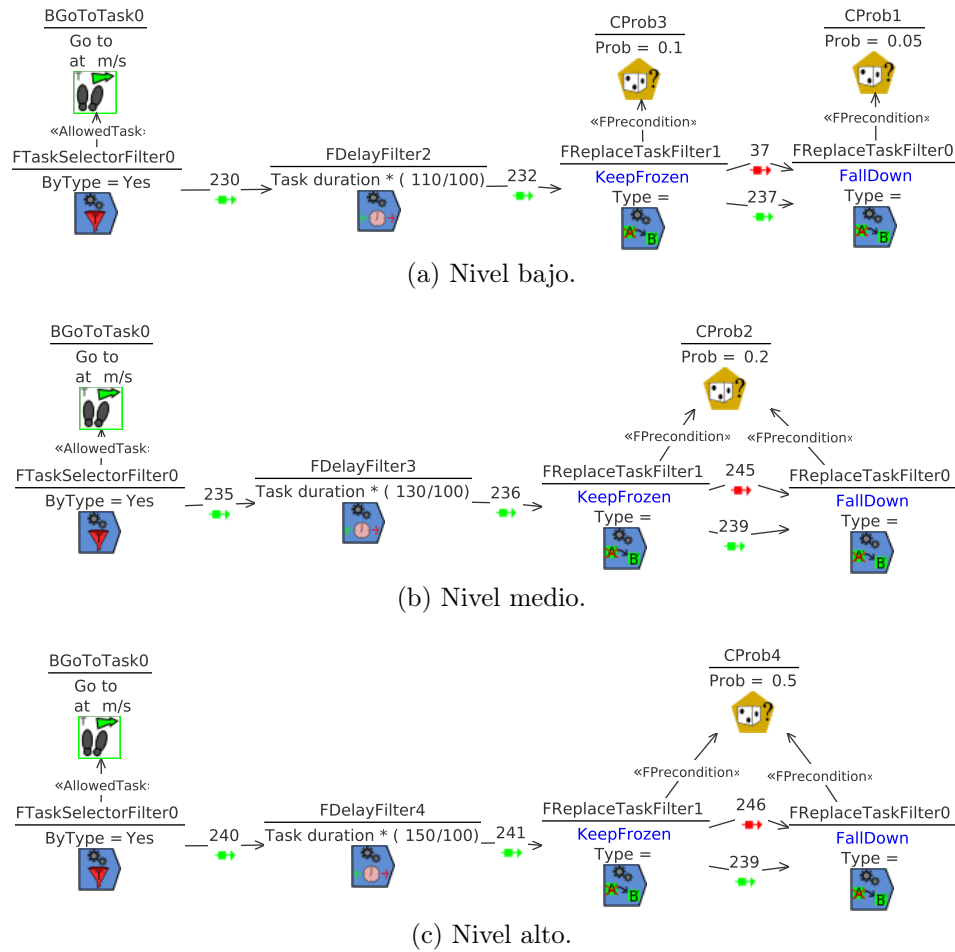


Figura 3.31: Limitaciones según los diferentes niveles de intensidad del síntoma de pies congelados.

$pLevelState$) pasa a tener niveles de intensidad bajos (estado $LOWSympLevelState0$) tras hora y media. Por tanto, tras tomar la medicación, en una hora y media el paciente se encuentra en niveles bajos del síntoma y si se encontraba en niveles altos solo le supone 15 minutos más. Por otro lado, si se presta atención a cómo incrementa el síntoma, se puede comprobar que estando en niveles bajos del síntoma el paciente alcanza el nivel medio en una hora y cuarto y el nivel alto a las 2 horas de iniciarse el nivel medio.

Por último, siguiendo con el síntoma de pies congelados, considérese el caso de que el síntoma limita al paciente al andar de tres formas: (1) la velocidad con la que se desplaza el paciente es menor, (2) hay posibilidad de que se quede congelado durante un tiempo sin poder andar y (3) también puede ocurrir que se caiga. Dependiendo del nivel de intensidad del síntoma estas afecciones pueden ocurrir con más intensidad o más probabilidad. Los modelos para cada uno de los niveles se encuentran en la Figura 3.31. Todos tienen en común el filtro que selecciona aquellas tareas del tipo desplazarse. A continuación, la tarea de desplazarse se modifica por el filtro de retraso,

en este caso, hacer que el paciente vaya más lento un 10 %, 30 % o 50 % dependiendo del nivel de intensidad. El siguiente filtro con probabilidad cada vez mayor (0.1, 0.2 y 0.5) reemplaza la tarea de andar por la de no hacer nada simulando que se ha quedado con los pies congelados, sin poder andar. Por último, tanto si se queda parado como no, se aplica el filtro que provoca la caída del paciente con probabilidades cada vez mayores según sea el nivel del síntoma (0.05, 0.02 y 0.5).

3.3.6.4. Evaluación

Esta parte del meta-modelo propuesto para modelar a los enfermos de párkinson consigue los objetivos deseados. Permite recoger la información más importante sobre el enfermo de párkinson como su estadio, toma de medicamentos y los síntomas (su evolución y las limitaciones que provocan). Haciendo uso de los filtros se modela cómo se ven alteradas las tareas dependiendo de los niveles de los síntomas. Esta solución es una extensión del lenguaje y dependiente de los conceptos explicados en el perfil de las actividades diarias, lo cual permite modificar cualquier comportamiento descrito en estos tipos de perfiles sin tener que hacer modificaciones. Es decir, el perfil de la enfermedad de Parkinson se podría asociar a cualquier sujeto y, sin ningún otro cambio, dicho sujeto sufriría las limitaciones de la enfermedad.

A continuación se verá cómo los árboles de comportamiento con parámetros son engorrosos, poco modulares y presentan dificultades para su reutilización a la hora de modelar estas limitaciones. Supóngase las limitaciones de pies congelados descritas en los filtros de la Figura 3.31 y que el mecanismo para modelar la evolución de los síntomas está solucionado. Para aplicar las limitaciones con los árboles de comportamiento habría que buscar en todos los árboles de comportamiento la tarea de andar y reemplazarla por un subárbol andar donde se considere las limitaciones. En la Figura 3.32 se muestra dicho subárbol con los parámetros de la tarea ir a un lugar (l para el destino y s para la velocidad de desplazamiento). El primer elemento es un selector que considera cada subrama dependiendo del nivel del síntoma. Si el síntoma no se manifiesta, el paciente realiza la tarea de andar normalmente (sin modificar los parámetros de entrada) como se indica en la última opción ($WalkTo(l,s)$). Las ramas del árbol para alterar el comportamiento del paciente según el síntoma siguen el mismo patrón. Primero se comprueba si el paciente padece la intensidad de la rama, si es así pueden ocurrir dos cosas: (1) caerse si la probabilidad lo indica o (2) andar más despacio con una cierta probabilidad de que antes se quede un rato congelado.

A primera vista, se puede comprobar que modelar estas limitaciones es más complejo en el árbol de comportamiento basándose en el número de entidades necesarias, 38 frente a 19. Además, como ya se ha comentado, esta solución requiere modificar todos los modelos que contengan la tarea de andar. Imagínese que un síntoma afecta a todas las tareas haciendo que éstas se realicen un 20 % más despacio. Con SociAALML simplemente habría que crear un filtro de retraso. En cambio, con los árboles de comportamiento resultaría muy costoso pues habría que volver a definir todas las acciones con sub-arboles donde se modelase la modificación de la acción en función de los síntomas. La otra ventaja de SociAALML frente a los árboles de comportamiento es su mayor

eficiencia cognitiva. La evaluación cualitativa de la notación basada en los principios de Moody se encuentran en la Tabla 3.4.

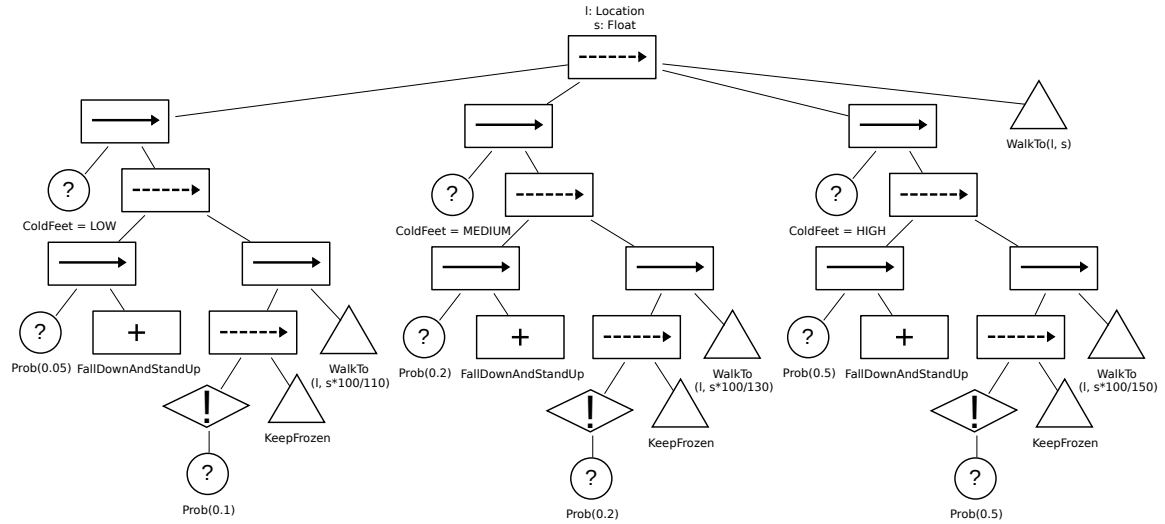


Figura 3.32: Especificación para modelar las limitaciones en la tarea de andar para el síntoma de pies congelados empleando los árboles de comportamiento.

Este mecanismo de filtros para alterar el comportamiento de las tareas de los pacientes en función de los síntomas ha resultado ser bastante apropiado. Sin embargo, se considera necesario incluir mecanismos para gestionar el conflicto entre filtros. Por ejemplo, asignarles prioridades o niveles.

3.3.7. El escenario

3.3.7.1. Introducción

Una vez se dispone de la descripción del entorno, la identificación de los sujetos y el modelado de sus perfiles, inmediatamente después se hace necesario un componente que aúne dichos conceptos, los instancie e inicialice para que se produzcan las simulaciones de interés. En este sentido, tras la experiencia del uso del lenguaje SociAALML, se han ido identificando necesidades y conceptos que el lenguaje debería recoger para componer los escenarios, los más importantes son:

- Especificar la semilla, es decir, permitir indicar un valor que afecta a la generación de números aleatorios de forma que siempre se generan los mismos para un valor de la semilla dado. De esta forma, se consigue escenarios no deterministas pero repetibles.
- Seleccionar el modelo de la vivienda 3D donde se desea que se desarrolle el escenario.

Principios de Moody	Comentario
Claridad simbólica	Se cumple totalmente al existir una representación por cada meta-objeto.
Discriminabilidad perceptual	Se cumple bastante bien al emplear iconos que ofrecen distancia visual. También se utilizan formas y colores para distinguir tipos de entidades. Donde se acusa algo de dificultad en diferenciar objetos es en la evolución de los síntomas, ya que el dorado de las condiciones y el naranja de los estados medios del síntoma son parecidos, ver Figura 3.30.
Transparencia semántica	Se cumple totalmente al incluir iconos que evocan el significado semántico de lo que representan.
Gestión de la complejidad	Se gestiona la complejidad al incluir la jerarquía de elementos más generales a más concretos, es decir, descripción general de la enfermedad, toma de medicamentos, evolución de los síntomas y filtros para las limitaciones.
Integración cognitiva	Como ya se ha comentado, INGENME no incluye elementos de integración de varios diagramas en uno para tener vistas generales. Sin embargo, sí facilita elementos de navegación a través de los enlaces, tal y como muestran los elementos de la vista general de la enfermedad, ver Figura 3.28.
Expresividad visual	Tiene expresividad visual sin llegar a saturar al emplear 3 elementos visuales (los iconos, la forma y el color en algunos casos).
Codificación dual	Se cumple al incluir texto junto con los elementos gráficos (identificadores y algunos valores de los atributos) y también al permitir incluir notas en los diagramas.
Economía gráfica	Hay 20 elementos gráficos nuevos, uno por cada nuevo concepto.
Ajuste cognitivo	No lo cumple al no proporcionar dialectos.

Tabla 3.4: Evaluación cualitativa de la notación para los elementos del perfil de enfermedad de Parkinson según los principios de Moody.

- Indicar la fecha y la hora de inicio de la simulación. Sobre todo, para modelar escenarios cortos resulta ventajoso poder especificar la hora de inicio.
- Especificar los sujetos que formarán parte de la simulación. Puede ser necesario definir escenarios donde no tomen parte determinados sujetos que se hayan modelado, e.g. los hijos o el asistente.
- Por cada individuo que intervenga en la simulación, se hace necesario indicar su posición inicial, ya sea en una habitación de la vivienda o fuera de esta (no se muestre inicialmente en el escenario).
- Especificar el valor inicial de cada síntoma de la los enfermos de párkinson.
- Al igual que en el caso de los sujetos, también es necesario indicar el despliegue de los dispositivos, es decir, qué dispositivos intervendrán en la simulación, dónde se situarán y qué software ejecutarán. Esto resulta de utilidad para diseñar y evaluar los diseños de las soluciones AAL, pues se puede definir varios escenarios donde solo se diferencien en la configuración de los dispositivos (su número, localización y/o software instalado).

De los simuladores analizados, solo SimCon [88] aborda la composición e inicialización de las simulaciones o escenarios. Esta composición se realiza con dos modelos y dos herramientas. Con el editor Hammer crean escenarios donde situar en el entorno 3D a los avatares y los bots con sus comportamientos rudimentarios. Y con la herramienta SimConfig modelan y despliegan los sensores en un escenario, combinando los lenguajes de sensores (SensorML) y el de modelos de edificios IFC. Este simulador presta mayor atención al despliegue de los dispositivos que al de los sujetos virtuales. El mayor problema de esta propuesta es que la posición inicial de los sujetos no está separada del modelo del entorno 3D, lo que dificulta su reutilización y mantenimiento. Si se tiene varias configuraciones iniciales de sujetos virtuales con un entorno 3D dado y se desea modificar dicho entorno, en vez de alterar solo el modelo 3D del entorno (IFC) es necesario cambiar también todas las configuraciones realizadas con el editor Hammer.

3.3.7.2. Propuesta de Meta-modelo

Para conseguir las características previamente expuestas y conseguir definir los escenarios, se ha ampliado el meta-modelo de SociAALML con los siguientes elementos, ver Figura 3.33:

- *WorldInitialization* es el objeto definido para establecer los valores de inicio del entorno. Por ello, tiene un campo para indicar el tipo de vivienda que se cargará (*HouseTypeField*), otro para indicar la semilla (*SimulationSeedField*) y, por último, una relación con el meta-objeto *SDate* que contiene la hora y fecha de inicio de la simulación.

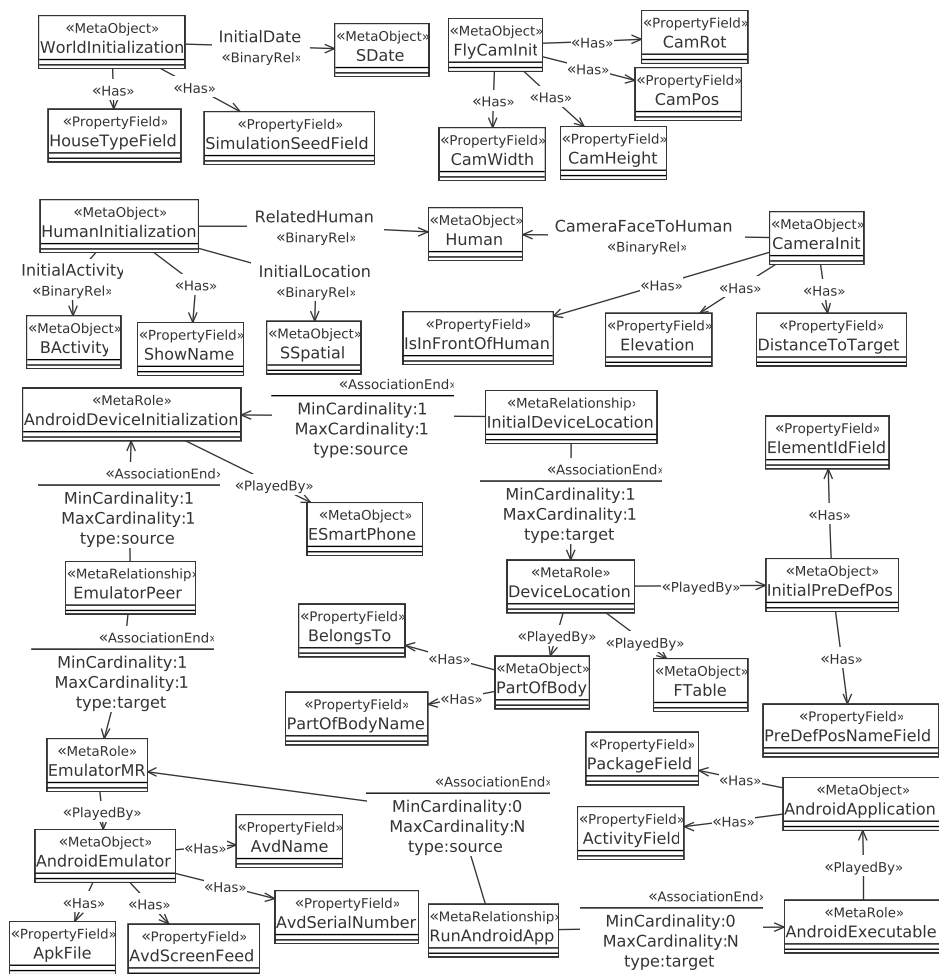


Figura 3.33: Vista del meta-modelo de un escenario de simulación.

- *FlyCamInit* permite configurar la resolución de la ventana del simulador e indicar la posición y orientación de la cámara para ver la simulación desde una perspectiva adecuada. No obstante, durante la simulación, la cámara se puede mover libremente utilizando el ratón y el teclado.
- *HumanInitialization* es el objeto a partir del cual se configuran todos los aspectos relevantes de un sujeto para el escenario. La relación «*RelatedHuman*» es obligatoria para saber a qué sujeto se hace referencia. Con la relación «*InitialLocation*» se indica el objeto espacial donde se situará inicialmente el sujeto virtual. Si no se indica nada, el sujeto no se verá en la simulación pero aparecerá en el exterior de la vivienda cuando se inicie la primera tarea que tenga que ver con ir a algún sitio. Normalmente, los sujetos realizarán las actividades y tareas especificadas en el perfil ADL, pero en casos de micro escenarios donde los sujetos virtuales realizan solo una actividad, ésta se puede indicar directamente con la relación «*InitialActivity*». Si se emplea esta relación, se ignorará el perfil ADL.
- *CameraInit* es alternativo al uso de *FlyCamInit*. Resulta conveniente cuando en dicho escenario, se está especialmente interesado en visualizar el comportamiento de un solo individuo. Ya que este objeto se asocia a un sujeto (*Human*) y durante la simulación la cámara seguirá a dicho sujeto sin poder moverla. Se puede configurar la elevación de la cámara con el campo *Elevation*, la distancia al sujeto con el campo *DistanceToTarget* y si dicha distancia es por delante o por detrás del sujeto virtual con el campo *IsInFrontOfHuman*.
- *AndroidDeviceInitialization* es un meta-rol asociado al teléfono inteligente (*ESmartPhone*) que tiene las relaciones *InitialDeviceLocation* y *EmulatorPeer* para indicar la posición inicial del dispositivo y su emulador, respectivamente. Un dispositivo se puede situar en una mesa, en la parte del cuerpo de un individuo (e.g. muñeca, pecho o cadera) o en cualquier objeto con identificador que tenga posiciones predefinidas (*InitialPreDefPos*).
- *EmulatorMR* representa la parte real del dispositivo virtual. Durante la simulación, por cada móvil virtual debe existir un emulador. Los emuladores permiten conectar a las aplicaciones al mundo virtual a través de los sensores y actuadores del dispositivo virtual. Un emulador Android se caracteriza por su nombre AVD (*AvdName*) y un puerto (*AvdSerialNumber*). El campo *ApkFile* indica la aplicación a instalar en el emulador. El campo *AvdScreenFeed* permite que se pueda visualizar en la pantalla del móvil virtual lo mismo que se visualiza en el emulador, dando más realismo a la simulación y facilitando la identificación de cada dispositivo. Por último, la relación *RunAndroidApp* permite especificar la aplicación que se arrancará en el emulador cuando se inicie la simulación. En Android, la información necesaria para indicar que se inicie una aplicación es el nombre del paquete (*PackageField*) y el nombre del Activity (*ActivityField*).

3.3.7.3. Notación y modelos

La notación para los elementos del meta-modelo dedicados a la definición de los escenarios se encuentran en la Tabla A.15 del Apéndice A.

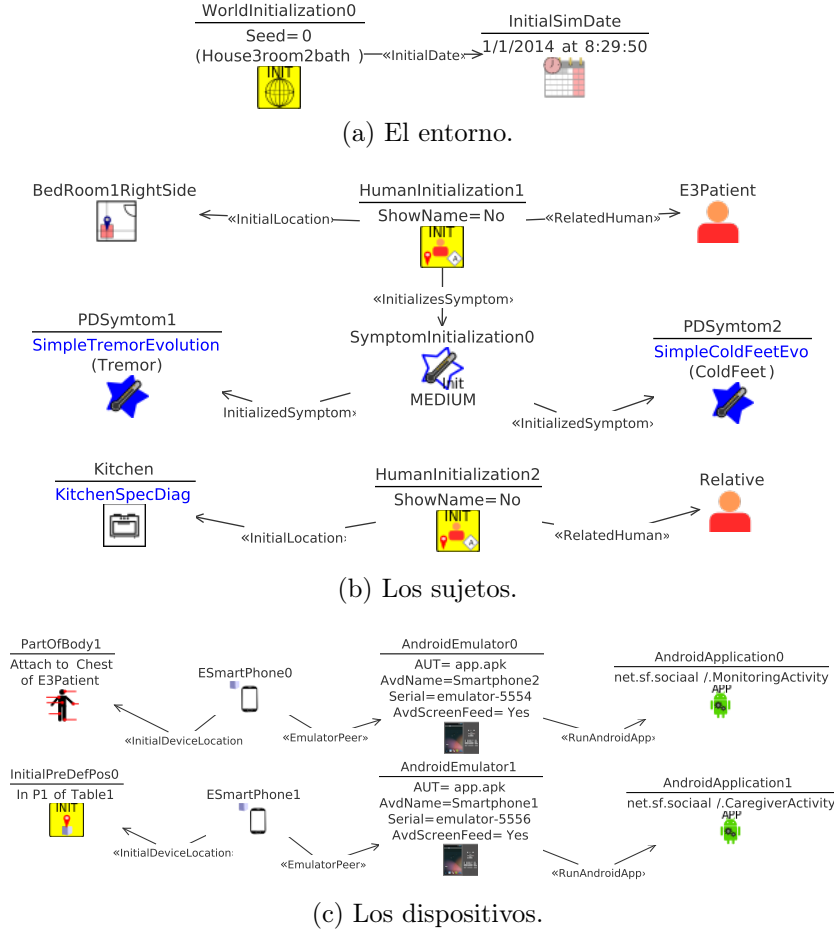


Figura 3.34: Ejemplo de modelo para definir un escenario de simulación.

Para ilustrar cómo se emplea esta parte del lenguaje para definir escenarios, se va a explicar el modelo de la Figura 3.34. Este se encuentra dividido en tres figuras para facilitar su explicación, aunque todos los elementos pertenecen a una misma vista. La primera parte (3.34a) especifica que el escenario se desarrollará en la vivienda *House3room2bath* el 1 de enero de 2014 a las 8:29:50. La semilla tendrá el valor 0. Si se fijan en la parte dedicada a los sujetos (3.34b), se puede comprobar que el escenario se compondrá de dos sujetos, el paciente (*E3Patient*) y un familiar (*Relative*). La posición inicial del paciente será en la zona situada en el lado derecho de la cama (*BedRoom1RightSide*) y la del familiar será en la cocina. El enfermo empezará sufriendo niveles de intensidad medios en los síntomas de pies congelados (*PDSymptom2*) y en el de temblores (*PDSymptom1*). Por último, la parte dedicada al despliegue de los dispositivos (3.34c) especifica que habrá dos dispositivos con identificadores *ESmartPhone0* y *ESmartPhone1* en el escenario. El primer dispositivo se instalará en el pecho del

paciente (*PartOfBody1*), mientras que el segundo se situará inicialmente en la posición (“P1”) de la mesa *Table1*. A cada dispositivo se le conectará un emulador con la aplicación “app.apk” instalada. En ambos dispositivos virtuales se mostrará en su pantalla las imágenes del emulador (“AvdScreenFeed=Yes”). Finalmente, la especificación indica que en el teléfono asociado al pecho del paciente correrá el Activity que monitoriza al paciente (*MonitoringActivity*), mientras que el que está en la mesa ejecutará el Activity para el cuidador (*CaregiverActivity*).

3.3.7.4. Evaluación

La evaluación cualitativa basada en los principios de Moody de la notación referente a la definición de los escenarios se encuentran en la Tabla 3.5.

Principios de Moody	Comentario
Claridad simbólica	Se cumple totalmente al existir una representación por cada meta-objeto.
Discriminabilidad perceptual	Se cumple bastante bien al usar iconos que ofrecen distancia visual. También se usan formas y colores para distinguir tipos de entidades. Se destacan en color amarillo, forma cuadrada y con las letras “INIT” aquellos elementos para la inicialización.
Transparencia semántica	Se cumple totalmente al usar iconos que evocan el significado semántico de lo que representan.
Gestión de la complejidad	No se gestiona al no incluir ningún tipo de jerarquía.
Integración cognitiva	Como solo hay una vista no se necesita para consultar los conceptos relativos al escenario.
Expresividad visual	Tiene expresividad visual sin llegar a saturar al usar 3 elementos visuales (los iconos, la forma y el color en algunos casos).
Codificación dual	Se cumple al incluir texto junto con los elementos gráficos (identificadores y algunos valores de los atributos) y también al permitir incluir notas en los diagramas.
Economía gráfica	Hay en total 12 elementos gráficos, uno por cada meta-objeto. Si hay menos se viola el principio de claridad simbólica
Ajuste cognitivo	No lo cumple al no proporcionar dialectos.

Tabla 3.5: Evaluación cualitativa de la notación para los elementos que definen los escenarios según los principios de Moody.

Si bien esta parte del lenguaje cumple con los objetivos planteados inicialmente, también sufre de algunas deficiencias que pueden mejorarse. Por un lado, se generan diagramas complejos, tal y como puede verse en el ejemplo de la Figura 3.34. En este ejemplo, para definir un escenario pequeño de dos sujetos y dos dispositivos se necesitan

19 objetos. Además, su lectura resulta algo molesta con tantos elementos inconexos. Como trabajo futuro se considera interesante definir nuevas vistas que recojan los despliegues de cada tipo de entidad: configuración del entorno, sujetos y dispositivos. De esta forma, se podrían reutilizar despliegues. Supóngase que se quieren definir varios escenarios donde se va variando el número de dispositivos, su posición y software a ejecutar. Con la solución actual es necesario replicar todas las entidades de la simulación en cada nuevo escenario y reemplazar aquellas entidades que vayan a ser diferentes. Si existen vistas por cada tipo de despliegue y la vista de simulación las agrega, por cada nuevo escenario solo habría que crear una vista de simulación nueva y una de despliegue de dispositivos. Así, se reduciría significativamente el número de diagramas en los que aparece una misma entidad y se facilita el mantenimiento de los modelos.

3.4. Conclusiones

La finalidad principal de este capítulo ha sido definir un lenguaje gráfico que sea cognitivamente eficiente y que permita generar especificaciones que modelen el día a día de los elementos no software (entorno, dispositivos y sujetos) relacionados con el sistema AAL. Además, el lenguaje permite, a partir de dichas especificaciones, generar simulaciones 3D que reflejen la semántica de ese día a día.

Para ello, se ha propuesto el lenguaje llamado SociAALML definido a partir de otro lenguaje de meta-meta-modelado basado en GOPRR con la herramienta INGENME. La experiencia de usar dicha herramienta ha sido satisfactoria para la generación del lenguaje y el editor. No obstante, carece del mecanismo de visualización de sub-diagramas para la gestión de la complejidad requerido por los principios de Moore.

SociAALML es un lenguaje gráfico específico de dominio que supone una novedad en el desarrollo de simulaciones dirigidas por modelos donde se recrea la vida diaria de los individuos en sus viviendas y, en particular, la de los enfermos de párkinson. Se ha demostrado que aunque faltan conceptos por modelar, como los dispositivos, y otros por mejorar, como las interacciones, este lenguaje permite generar gran cantidad de escenarios interesantes para el desarrollo e investigación de sistemas AAL. Además, el lenguaje cumple bastante bien los principios de eficiencia cognitiva de Moody de forma que dichas especificaciones pueden ser entendidas y más fácilmente creadas por usuarios no expertos (e.g. familiares, trabajadores sociales o médicos). Así, SociAALML también permite ser un lenguaje formal de comunicación entre los actores que intervienen en el ciclo de vida del sistema. No obstante, estas últimas afirmaciones deben ser confirmadas evaluando la experiencia de uso del lenguaje por parte de dichos usuarios no expertos.

Capítulo 4

El marco de dominio: PHAT

*Ignoramos nuestra propia estatura hasta que
nos ponemos en pie.*

Emily Dickinson

RESUMEN: Este capítulo cubre tanto el simulador PHAT como el proceso de generación de código. Primero se identifican los requisitos deseables para el simulador. Posteriormente, se explica el diseño e implementación de cada módulo para alcanzar dichos objetivos. Una vez presentado el simulador y sus interfaces se muestra el proceso de generación de simulaciones a partir de modelos del lenguaje SociAALML. Finalmente, se evalúa brevemente el código generado a través de unas métricas y se presentan las conclusiones.

4.1. Requisitos

El *Marco del dominio* proporciona interfaces entre el código generado y la plataforma subyacente. Este elemento es conveniente para facilitar la generación del código. Para ello, el marco de dominio debe implementar los conceptos que recogen las especificaciones generadas con el lenguaje. Como se vio en el apartado 2.3 sobre el estado del arte de los simuladores de entornos de Inteligencia Ambiental, no hay ningún simulador que se ajuste a nuestros requerimientos. Aquellos que más se aproximan son las distintas versiones de TATUS [109, 108, 87]. Sin embargo, estos no están disponibles en la web para poder adaptarlos y así emplearlos como marco de dominio. Por ello, en este capítulo se presenta nuestra propuesta de plataforma de simulación para que haga las veces de marco de dominio. Esta consistirá en un conjunto de módulos que ofrezcan interfaces para la creación de elementos virtuales que son relevantes para recrear escenarios del día a día de los enfermos de párkinson y sus familiares: la vivienda, los sujetos con sus comportamientos y los dispositivos (sensores y actuadores).

Como ya se discutió en el Capítulo 2, para crear una simulación 3D es necesario especificar por cada elemento virtual un comportamiento y una representación. El comportamiento viene dado por el código de programación o los scripts y la representación por los modelos 3D y recursos multimedia. En el desarrollo de videojuegos, una parte importante del presupuesto se invierte en su programación. Pero también, en muchos casos, la generación de los recursos multimedia (llamados *assets*) es incluso más costosa [118]. Estos recursos incluyen modelos 3D de los sujetos virtuales, los objetos y la vivienda, materiales, texturas, terrenos, música o sonidos ambientales. La plataforma de simulación debe facilitar la generación de los comportamientos de los elementos virtuales. Como por ejemplo, las actividades de un sujeto. Sin embargo, esta no se encargará de la generación de recursos multimedia o modelos 3D, por ejemplo, describir la geometría de una vivienda o la apariencia física de un sujeto virtual. Como ya avanzamos, realizar este tipo de descripción con un editor de lenguaje tipo entidad relación resultaría muy costoso e inadecuado. En su lugar, la plataforma debería ofrecer un repositorio de recursos multimedia los cuales se pudieran seleccionar y/o configurar a partir del lenguaje. Por ejemplo, dependiendo de la edad y el sexo de un sujeto, se podría seleccionar unos modelos 3D u otros.

PHAT (Physical Human Activity Tester) es el nombre de la plataforma de simulación propuesta en esta tesis y persigue ofrecer un entorno de simulación convincente, gratuito, fácil de utilizar y accesible a todo aquel que disponga de un ordenador. PHAT, junto con su documentación y tutoriales, se encuentra disponible en la web del proyecto SociAAL¹ o directamente en GitHub². La idea es que PHAT sea un *Living Lab Virtual* donde tanto investigadores como desarrolladores puedan experimentar, evaluar conceptos de Inteligencia Ambiental e incluso realizar pruebas y depurar los sistemas a través de la simulación. Además de que la plataforma implemente los conceptos propios del dominio, esta debe generar simulaciones con las siguientes propiedades:

1. **Repetibles.** No es deseable que dados unos parámetros iniciales, el resultado de la simulación varíe en cada nueva ejecución. En tal caso, la simulación no serviría como herramienta para validar escenarios ante el usuario, ni para depurar el sistema AAL ante situaciones específicas donde el sistema no responde como se espera. Esto no significa que los comportamientos tengan que ser deterministas. A partir de una configuración inicial, no tiene porqué saberse a priori lo que sucederá, pero sí debe ocurrir siempre lo mismo si la configuración inicial no se modifica. Esto se consigue con el concepto de semilla, tal y como se verá más adelante.
2. **Convincentes.** El comportamiento de los elementos de la simulación debe ser lo más próximo al que se esperaría en el mundo real. Principalmente, debe reproducir el comportamiento de sujetos realizando las actividades diarias del día a día en sus viviendas, permitiendo interaccionar con otros sujetos y con los dispositivos.

¹Sitio web de SociAAL: <http://grasia.fdi.ucm.es/sociaal/>

²Código fuente de PHAT: <https://github.com/Grasia/phatsim>

3. **Visuales.** Además de que las simulaciones se comporten de forma realista, también es importante que la visualización de las mismas sean expresivas. Esto es importante para permitir que el observador de la simulación pueda interpretar fácilmente lo que está ocurriendo en la misma. De esta forma, las simulaciones pueden ser útiles para validar los modelos creados y, también, para comprobar que el sistema AAL se comporta como se espera.
4. **Configurables.** Debe ser sencillo configurar y crear los elementos que componen la simulación. Por ejemplo, añadir una vivienda, indicar la hora de inicio de la simulación, especificar los sujetos virtuales que intervendrán y sus comportamientos asociados, y, también, el despliegue de dispositivos con el software a instalar en cada uno.
5. **Híbridas.** Las simulaciones híbridas son aquellas que combinan elementos reales y virtuales. En nuestro caso, el elemento real es el software que corre en los dispositivos y el cual compone el sistema AAL que se desea probar en el mundo virtual. Además, la plataforma de simulación debe permitir que los cambios en el código del software sean los mínimos posibles para que no se produzcan nuevos errores en el proceso de traducción del código al mundo real. De esta forma, la plataforma da soporte para que se utilice el enfoque de *software-in-the-loop* [121].

4.2. La arquitectura de PHAT

Como se vio en la sección 2.4, PHAT se construye a partir del motor de juegos jME3. En la Figura 4.1 se muestra el diseño de la arquitectura del simulador PHAT por capas, donde cada capa es un módulo. Aquellas capas que tienen el fondo gris son las propias del simulador y las de fondo blanco corresponden a las tecnologías más importantes en las cuales se basa. En la capa más baja se encuentran las tecnologías más importantes utilizadas por el motor de juegos jME3: *OpenGL* para el renderizado 3D, *OpenAL* para el sonido y *jBullet* para el motor físico. Estas mismas tecnologías se utilizan también en el desarrollo de videojuegos comerciales, por lo que son una buena base para recrear mundos virtuales realistas. *jME3* facilita el uso de dichas tecnologías ofreciendo objetos e interfaces para crear entornos con física e iluminación, crear fuentes de audio y cargar modelos 3D con animaciones. En el apartado 4.3 se da una pequeña descripción básica de los elementos de jME3 que se han empleado para implementar los módulos de PHAT. A continuación, se describen las responsabilidades de cada uno de los módulos, tanto a nivel de lógica como de recursos multimedia:

- El módulo *core* debe ofrecer las clases e interfaces básicas que implementan las funcionalidades comunes al resto de módulos, tanto para el momento de la simulación como para su depuración. Una de las funcionalidades más importantes se refiere a la del audio. Este debe facilitar: la generación de audio a partir de texto para que los sujetos virtuales puedan hablar y un sensor de audio (micrófono) que permita redirigir el audio a distintas salidas (por ejemplo, los emuladores de

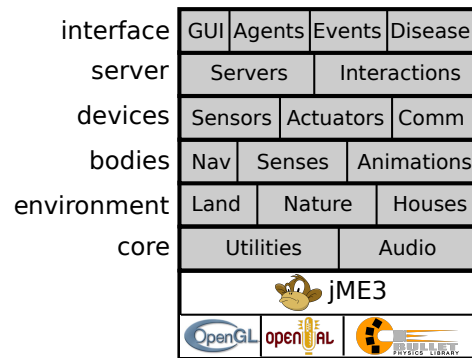


Figura 4.1: Los módulos que componen la arquitectura de PHAT.

los dispositivos virtuales). Además, se debe permitir incluir varios micrófonos en una misma simulación. Esto último haría posible que si un escenario requiere de varios dispositivos que hagan uso del micrófono, cada uno percibiría el sonido del mundo virtual dependiendo de su posición. Esta es una propiedad muy importante que no se encuentra en ningún simulador de Inteligencia Ambiental, ni en jME3, ni en el resto de motores estudiados. Estos solo permiten obtener el sonido desde una perspectiva, la del jugador. PHAT incorporará esta funcionalidad gracias al trabajo de McIntyre [89] quien la implementó en una librería para su simulador de criaturas llamado CORTEX³, el cual está basado también en jME3.

- El módulo *environment* se debe encargar de gestionar diferentes modelos 3D de los terrenos y las viviendas donde se desarrollará el Living Lab Virtual. En este módulo también se debe simular el paso del tiempo (reloj y calendario) y el sol. En PHAT, la iluminación variará dependiendo de la hora del día. Los detalles de este módulo se presentan en el apartado 4.4.
- El módulo *bodies* se debe encargar de la parte física de los sujetos virtuales, es decir, la representación 3D del cuerpo y la simulación de sus sentidos, aparato locomotor y la capacidad de desplazarse por el entorno. También debe ofrecer animaciones que representen las tareas del día a día de los sujetos, los síntomas del paciente (e.g. la rigidez y los temblores en manos y cabeza) con diferentes intensidades, y la simulación de caídas convincentes haciendo uso de la física. En el apartado 4.5 se abordan los detalles de este módulo.
- El módulo *devices* sería el equivalente al *bodies*, es decir, debe ofrecer las funcionalidades para crear y configurar los componentes hardware de los dispositivos. Este debe permitir identificar un dispositivo, especificar los sensores y actuadores con los que estará equipado, y los mecanismos de comunicación (como WiFi y Bluetooth). Además, los dispositivos se deben poder desplegar por el entorno, tanto en zonas de la vivienda como en los sujetos virtuales. En el caso de los sujetos, se debe poder instalar en partes del cuerpo, por ejemplo, el pecho, la

³Sitio web de CORTEX: <http://aurellem.org/cortex/html/>

espalda, las muñecas o los tobillos. De esta forma, se permitiría la evaluación de distintos diseños de soluciones AAL en los que se variaría el número y posición de los dispositivos. En la Figura 4.15 se puede ver un dispositivo asociado al pecho de un sujeto virtual. Esta tesis se centra en los dispositivos Android. Se considera suficiente que la representación de los mismos corresponda a un cubo plano donde, en una de las caras, se encuentre la pantalla. En cuanto al comportamiento, estos deben seguir las leyes de la física como cualquier objeto. En este trabajo se considera prioritario abordar primero la incorporación de los sensores y, posteriormente, los actuadores. Simular las comunicaciones y propagaciones de las señales WiFi se considera interesante, pero no se aborda en esta tesis. En el apartado 4.6 se detalla cómo el módulo implementa estas funcionalidades.

- El módulo *server* debe ofrecer funcionalidades para permitir que aplicaciones externas, como los emuladores, puedan acceder a los sensores, actuadores y el módulo de comunicación de los dispositivos virtuales. Así, el sistema AAL instalado en el emulador debe percibir y actuar sobre el entorno virtual de forma transparente. Esto permite emplear el enfoque *software-in-the-loop* para el desarrollo del sistema. Para ello, un sujeto virtual debe poder interactuar con el emulador, es decir, realizar acciones como instalar aplicaciones, arrancarlas y pulsar sobre puntos de la pantalla. Estas funcionalidades están detalladas en el apartado 4.7.
- El módulo *interface* debe dar soporte a varias funciones relevantes. Por un lado, debe implementar el comportamiento de los sujetos, los cuales pueden variar en función de las limitaciones dadas por la enfermedad y de los eventos. Por tanto, debe incluir (1) la enfermedad, (2) las limitaciones asociadas a los síntomas y cómo evolucionan los mismos, y (3) un mecanismo para generar y gestionar los eventos. Por otro lado, es necesario un mecanismo que facilite la definición de los escenarios haciendo uso de los módulos anteriores, permitiendo: elegir el terreno, la vivienda, día y hora de la simulación, cuerpos a crear y su posición inicial, dispositivos a crear y su posición inicial, los sujetos virtuales con sus comportamientos y cuerpos asociados. Muchas de estas necesidades son consecuencia directa de los conceptos que recoge el diagrama de simulación de SociAALML. Por último, debe ofrecer una GUI con un menú con herramientas para interactuar en tiempo de ejecución con la simulación de forma que permita pausar o avanzar la simulación, hacer capturas de pantalla, mostrar el rendimiento de la simulación (FPS, Frames Per Second) y mostrar el log de los comportamientos de los sujetos virtuales.

4.3. Conceptos básicos de jME3

Para entender el diseño e implementación de PHAT, en este apartado se describen las clases e interfaces principales que ofrece jME3 para la creación de videojuegos o, como es el caso, la creación de simulaciones. En la Figura 4.2 se muestra un diagrama

UML de estas clases principales. *SimpleApplication* es la clase básica que extiende *Application* para la creación de la simulación. Los elementos que forman la escena de una simulación son del tipo *Spatial*. Estos pueden ser de dos tipos: *Geometry* y *Node*. Los primeros son los que realmente contienen la información sobre la representación en forma de mallas y materiales. Los segundos se utilizan para organizar los elementos en forma de árbol y siguen un patrón de composición. Además, las transformaciones, por ejemplo las translaciones o las rotaciones, que se realizan en un nodo afectan a todos sus hijos. Una aplicación hace referencia a un nodo raíz (de tipo *Node*) que contiene todos los elementos visuales organizados en forma de árbol. Para visualizar los elementos en una simulación hay que añadirlos al *rootNode*.

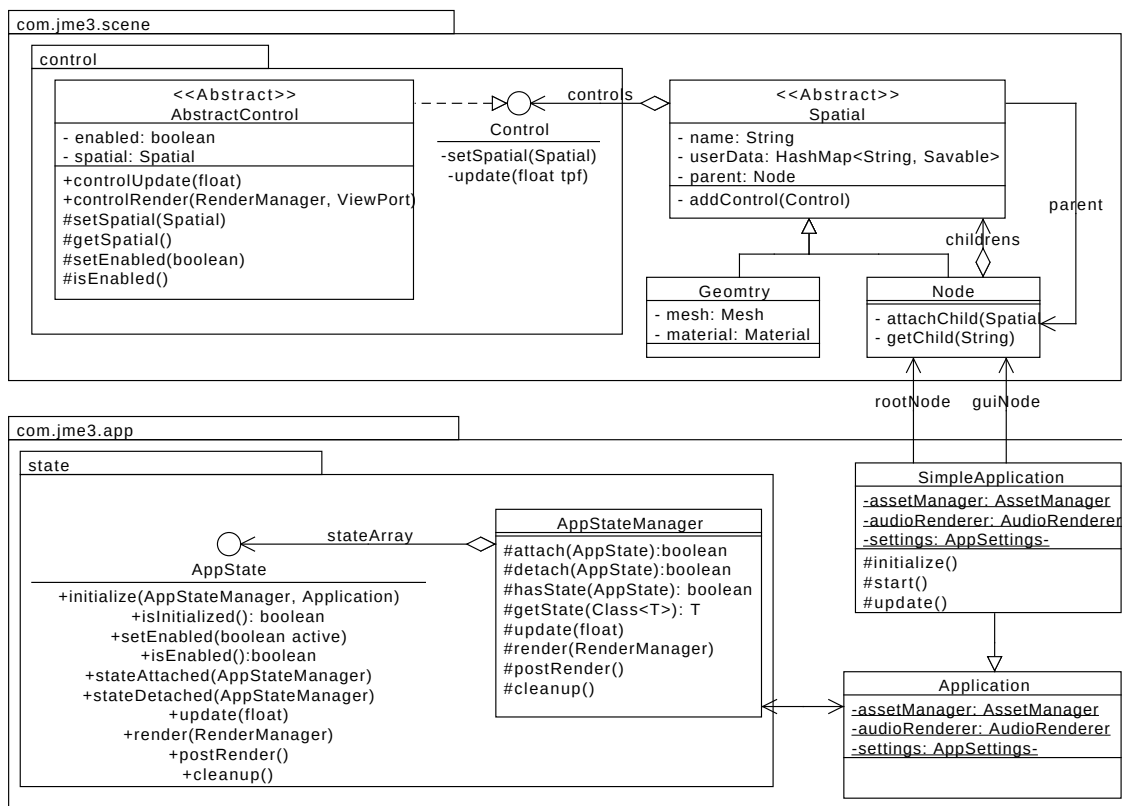


Figura 4.2: Clases principales de jME3 para crear la simulación.

Para especificar la lógica, es decir, para controlar el comportamiento de la simulación jME3 ofrece dos tipos de elementos: *Control* y *AppState*. En el diagrama se puede ver que los *Controls* están asociados a los elementos de la escena. Esto es así porque se encargan de controlar el comportamiento de un elemento concreto. Disponen de una referencia al *Spatial* a controlar y un método *update()* donde se implementa el comportamiento del controlador. En cada iteración de la simulación, la aplicación llama a este método indicando el tiempo que ha pasado desde la última llamada. Por ejemplo, se verá que se creará un controlador asociado a un sujeto virtual para simular

los temblores de las manos. Por otro lado, los *AppState* se utilizan para implementar comportamientos más globales. Por ejemplo, jME3 ofrece un *BulletAppState* que se encarga de controlar el motor físico. A diferencia de los controladores anteriores, este puede almacenar referencias a elementos globales de la simulación a través del método de inicialización (*initialize(...)*). Ejemplo de ellos son todos los elementos de la escena a través del *rootNode* o el resto *AppStates* a través del *StateManager*.

En la Figura 4.3 se encuentra el diagrama con las clases más importantes que definen la arquitectura de PHAT. Las clases en gris son las propias de PHAT y las blancas las de jME3. La clase principal de la simulación es *PHATApplication* que extiende la clase de jME3 *SimpleApplication* para establecer los valores iniciales de la simulación y controlarla. El resto de clases implementan la interfaz *AppState* y estas se encargan de implementar gran parte de las funcionalidades requeridas para los módulos descritos en el apartado anterior. Para ello, cada una incorpora un conjunto de comandos para acceder a las funcionalidades del módulo. Estos se ejecutan de forma asíncrona, es decir, los comandos se encolan en los módulos y estos se ejecutan la próxima vez que el módulo tenga el control. La entidad que crea un comando, si lo desea, puede ser notificada indicando que el comando se completó satisfactoriamente, que hubo un error o que fue interrumpido. Este mecanismo de comandos está inspirado en la plataforma ACE [66]. En los siguientes apartados se verá estos módulos, sus requerimientos y el tipo de comandos que ofrecen. El conjunto completo de comandos que ofrece PHAT se encuentran recogidos en el Apéndice B. Estos componen la API para acceder a las funcionalidades de cada módulo.

Además de dar soporte para crear la lógica de la simulación, jME3 permite importar recursos multimedia, configurarlos y utilizarlos. A lo largo de los siguientes secciones se verá también qué recursos son necesarios para cada módulo.

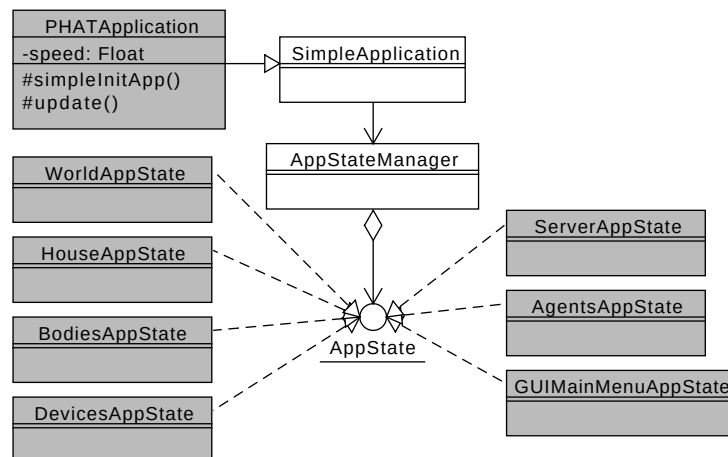


Figura 4.3: AppStates que forman los módulos de PHAT.

4.4. El módulo del entorno

El trabajo se centra en el desarrollo de sistemas AAL en entornos domésticos. Por ello, el principal elemento del entorno es el relativo a los modelos 3D de la vivienda y sus muebles. Pero también es importante el entorno exterior en el cuál se encuentra la vivienda, es decir, el modelo 3D del terreno, el cielo, el sol, las leyes de la naturaleza como la física y el clima. Este módulo debe ofrecer un entorno virtual de forma que cumpla con los requisitos de simulación convincente, visual y configurable. Pero además, el entorno debe incorporar información para que los sujetos virtuales se puedan desenvolver en el mismo.

4.4.1. El entorno exterior de la vivienda

Las actividades de los sujetos virtuales se van a desarrollar principalmente en el interior de la vivienda. Aun así, hace falta un entorno exterior que aporte realismo y permita mostrar, por ejemplo, cómo los sujetos entran y salen de la vivienda. Los elementos más importantes y necesarios son:

1. El modelo 3D del terreno donde se asentará la vivienda. En principio, se considera que una superficie plana con césped o alguna textura uniforme es suficiente. No se busca simular, por ejemplo, todo el entorno exterior de la vivienda, ya sea urbano o rural.
2. La textura del cielo y la simulación del comportamiento del sol para que el paso del tiempo sea realista y la iluminación vaya variando conforme va pasando el día. Esto es importante porque podría afectar al comportamiento de los sujetos virtuales (y del sistema AAL) en cuanto a que tendrían que encender las luces para ver y permitiría simular situaciones problemáticas debido a poca iluminación.

La simulación del clima también sería interesante, es decir, si es un día soleado o lluvioso, si hace una determinada temperatura o humedad. Pero debido a que la actividad se centra en el interior de la vivienda y durante las entrevistas no se hace prácticamente referencia al clima, este aspecto no se ha considerado.

La clase principal que implementa y ofrece las funcionalidades requeridas es *WorldAppState*, ver Figura 4.3. Esta ofrece una interfaz muy sencilla para crear el entorno exterior. Por un lado, ofrece un método para seleccionar el tipo de terreno. Por otro lado, ofrece un método para seleccionar la fecha y hora de simulación. PHAT implementa un calendario virtual (*PHATCalendar*) que una vez iniciado, avanza en función de la simulación y actualiza la iluminación de la escena simulando el movimiento del sol. Para más detalles consultar el Apéndice B.

En la Figura 4.4a se muestra una captura de pantalla del escenario resultado de seleccionar el terreno *TwoHouses*. En la Figura 4.4b se muestra el árbol de los elementos espaciales. Fijese, que este contiene información visual como el cielo (*Sky*) y el terreno (*Terrain*) pero también meta-información como la posición de las viviendas (posición de los nodos *House1* y *House2*) y el comportamiento de la física del terreno (*Physics*).

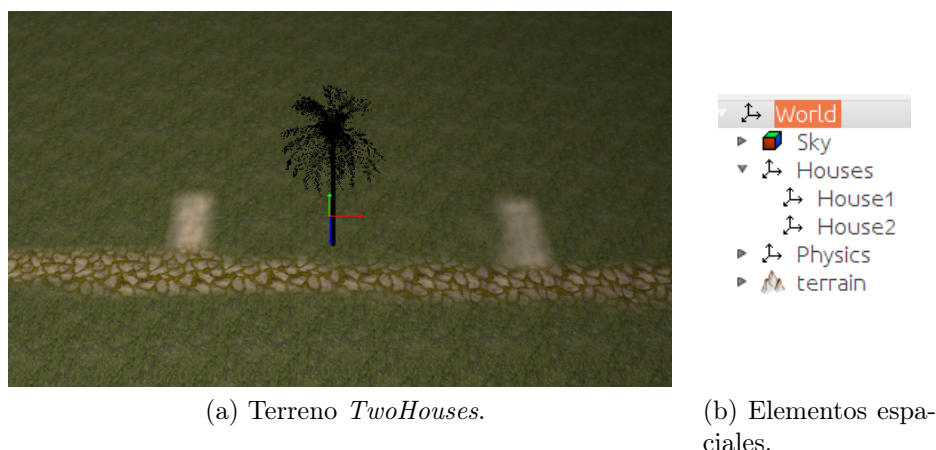


Figura 4.4: Terreno TwoHouses para simular el entorno exterior.

4.4.2. El interior de la vivienda

La parte del entorno relacionada con el interior de la vivienda es la de más interés ya que es donde se realizará la mayor parte de la simulación. Este entorno comprende tanto los elementos estructurales de la vivienda, como sus muebles y electrodomésticos principales. En un proyecto de desarrollo de un sistema AAL a medida para un cliente, la vivienda deberá corresponderse con la del cliente. Por ello, PHAT debe dar la posibilidad de seleccionar entre diferentes viviendas con los siguientes elementos:

1. *La estructura y distribución de la vivienda.* Esta debe tener suelo, paredes, techos y puertas lo más parecidos al entorno real que se pretende simular. Además, deben seguir las leyes de la física de forma que el sujeto virtual no pueda, por ejemplo, atravesar los muros.
2. *Muebles, sanitarios y electrodomésticos principales.* La vivienda también debe estar equipada con muebles, sanitarios y los electrodomésticos cotidianos de un hogar. Estos son importantes ya que el sujeto virtual los empleará cuando realiza sus actividades diarias. Además, deben reflejar de forma convincente su comportamiento de forma que el observador pueda discernir qué está sucediendo en cada momento. Por ejemplo, si el sujeto virtual abre el grifo del fregadero, debería verse cómo sale el agua y oírse. Estos también deben seguir las leyes de la física. Dependiendo del tipo de objeto, serán fijos o móviles. Para estos últimos habrá que especificar un peso, por ejemplo, una silla podría pesar 5 kilogramos.
3. *Iluminación.* La vivienda debe incluir los interruptores y las luces de cada habitación. Si un sujeto virtual quiere encender la luz de una habitación debe ir a uno de sus interruptores y encenderlo.

4. *Soporte para la navegación de los sujetos virtuales.* El escenario debe incorporar información espacial relativa al nombre de las habitaciones y que facilite que los sujetos virtuales se puedan desplazar por el entorno.

Modelar la vivienda desde el lenguaje SociAALML resultaría muy complejo. Es decir, realizar la descripción de la representación y especificar la distribución exacta de cada elemento (como los muros o los muebles). Por ello, la vivienda ha de crearse previamente, al igual que el resto de recursos multimedia, con las herramientas adecuadas. Pero a diferencia del terreno, el SDK de jME3 no ofrece una herramienta para crear viviendas 3D. Para modelar una vivienda, se podrían emplear las mismas herramientas profesionales que utilizan los arquitectos (e.g. AUTOCAD de Autodesk⁴), pero estas son complejas y costosas. En su lugar, se ha elegido una herramienta mucho más intuitiva y productiva llamada SweetHome3D⁵. Por ejemplo, la vivienda de la Figura 4.5 se creó en una 1 hora. Esto permite hacerse una idea de la productividad de la herramienta una vez se tiene cierta experiencia en su uso. Esta herramienta es bien conocida por el autor, pues ya se utilizó previamente para el simulador UbikMobile [29] con resultados satisfactorios. Diewald también la propone en sus trabajos sobre simulación [39, 122].

SweetHome3D es de código libre, gratuita y programada en Java. Está concebida para que cualquier usuario sea capaz de diseñar el interior de una vivienda, por lo que es muy sencilla de utilizar. Permite modelar la planta, las paredes, las puertas y las ventanas de la vivienda. Además, ofrece una paleta con muebles y sanitarios más comunes. Para el diseño de los interiores, la interfaz gráfica de SweetHome3D se divide en 4 componentes, ver Figura 4.5: (1) Un panel para la edición que muestra una vista 2D de la planta de la vivienda (arriba a la derecha). Este permite introducir una imagen del plano de la vivienda de forma que se pueden crear los suelos y los muros de forma muy rápida. (2) Una paleta de muebles y sanitarios que se pueden añadir simplemente arrastrándolos al panel de edición. Además, dicha paleta es ampliable importando nuevos muebles. (3) Una vista 3D del modelo de la vivienda (abajo a la derecha). (4) Una lista de los elementos añadidos (abajo a la izquierda).

Una vez creado el modelo de la vivienda, la aplicación permite exportar dicho modelo en un formato muy común para modelos 3D llamado OBJ, el cual es también soportado por jME3. El problema es que se importa como un bloque, es decir, como un solo objeto. Por tanto los objetos individuales y móviles como una silla no son posibles. Debido a esto, la creación se realiza en dos fases. En una primera, se modela los elementos fijos de la vivienda, por ejemplo, las paredes, el suelo, los sanitarios y los muebles de la cocina. El resultado se carga en el SDK de jME3 para añadir los objetos móviles y la meta-información necesaria. Las tareas necesarias son:

- Añadir las coordenadas de cada habitación. Se crea un nodo con el nombre de la habitación y se coloca en la posición que corresponda. De esa forma, cuando el sujeto virtual quiera ir a la habitación irá a dicho punto.

⁴Sitio web de Autodesk: <http://www.autodesk.com/>

⁵Sitio web de SweetHome3D: <http://www.sweethome3d.com/>

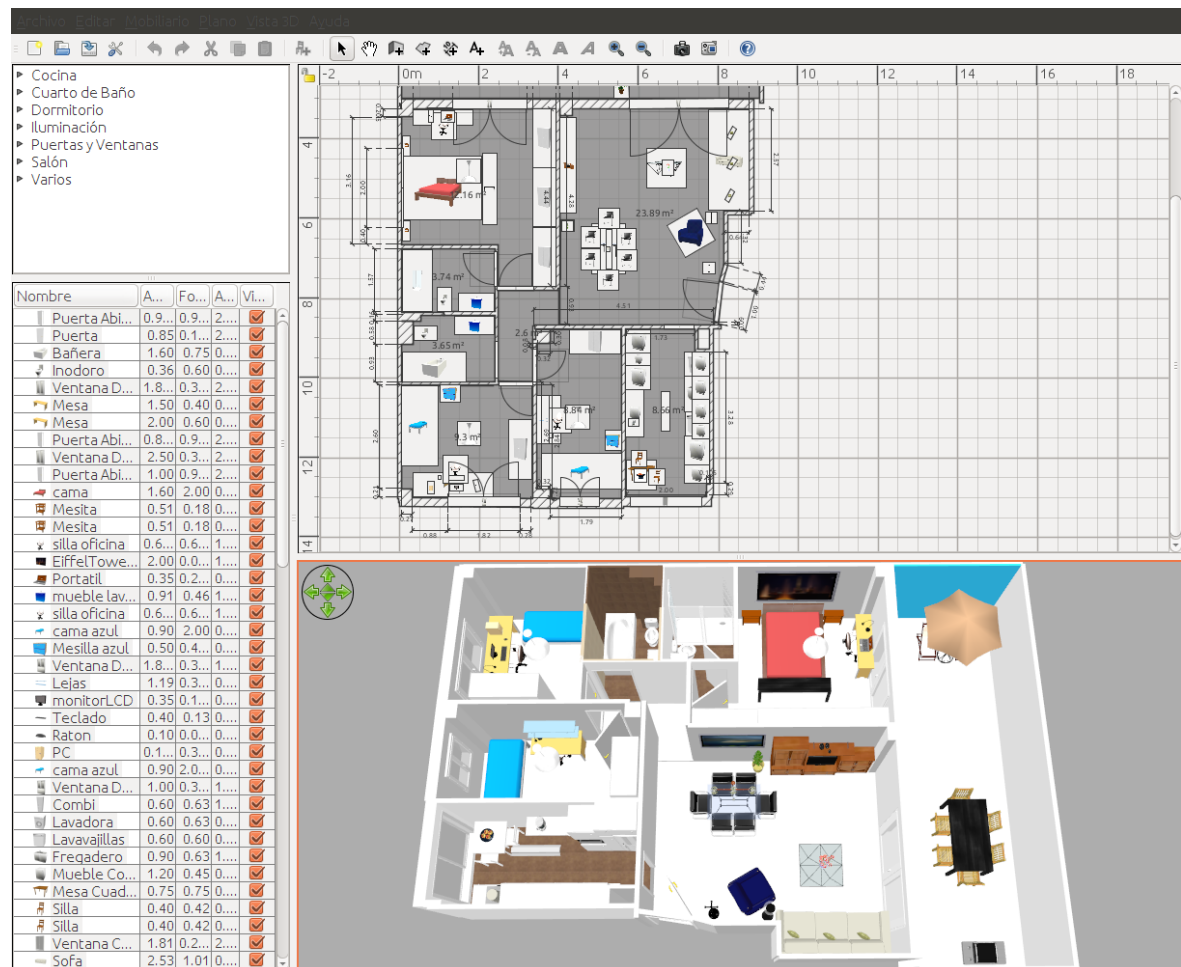


Figura 4.5: Captura de pantalla de la herramienta de diseño de interiores SweetHome3D. Los modelos 3D y las texturas empleadas están bajo licencia libre.

- Cada vivienda debe tener su propia malla de navegación para que los sujetos virtuales puedan desplazarse de un punto a otro. Es importante crear una buena malla de navegación para que no haya luego problemas, ver Figura 4.6. Como se verá más adelante, se ha considerado oportuno ofrecer un comando para que se muestre dicha maya y poder comprobar que es correcta.
- Cada objeto que se añade a la vivienda, necesita un Node que contenga su posición y se le asocie información sobre su rol y su identificador. De esta forma, se puede localizar objetos en la vivienda tanto a partir de un rol como de un id. Además, dependiendo del tipo de objeto, este debe aportar información adicional, ver Figura 4.7. Por ejemplo, la ducha (4.7a) y el grifo del fregadero (4.7b) tienen asociados sonidos y emisores de partículas que se reproducen cuando se están utilizando dichos objetos. En este caso, los emisores de partículas son útiles para simular el agua. En cambio, la mesa (4.7c) tiene un controlador de física (PhysicsControl) porque se puede mover y coordenadas donde el sujeto virtual puede dejar objetos. Como último ejemplo, el sofá (4.7d) contiene información de los sitios donde el sujeto virtual puede sentarse.

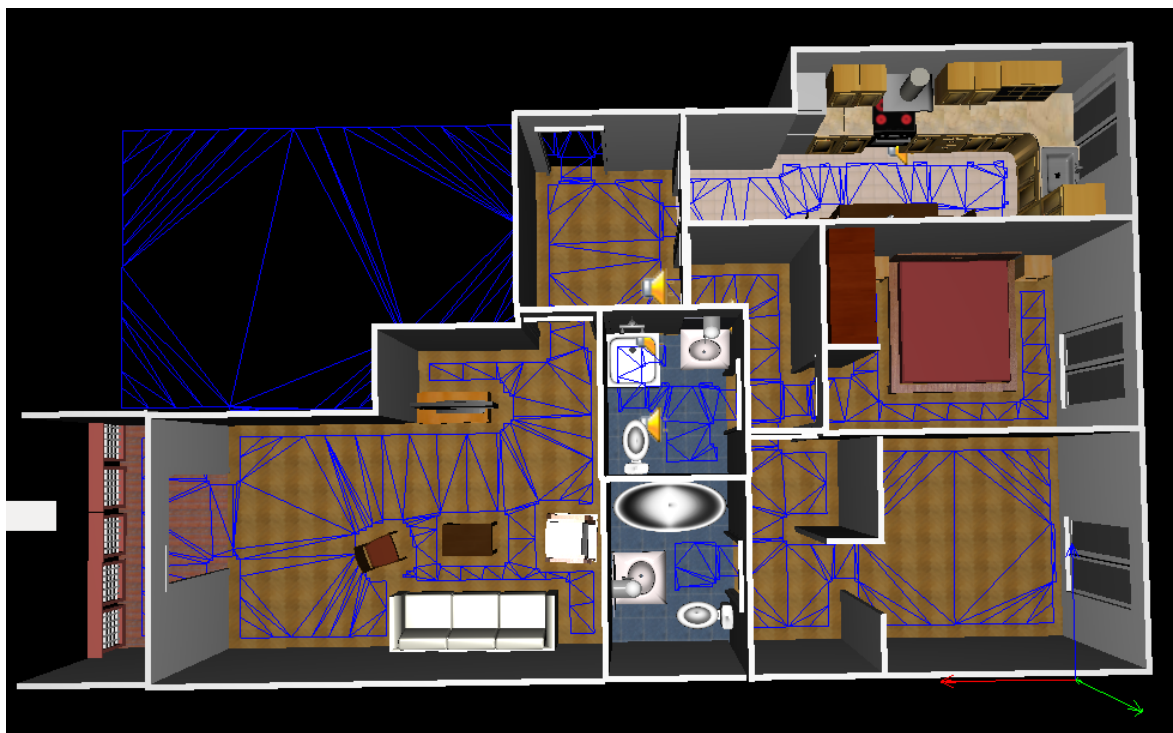


Figura 4.6: Captura de pantalla donde se muestra la malla de navegación generado con el SDK de jME3 para una vivienda.

HouseAppState es la clase de PHAT encargada de gestionar las viviendas virtuales. Si se consulta el Apéndice B, se puede comprobar que se han desarrollado comandos para satisfacer las necesidades. Por ejemplo, existe un comando para seleccionar una

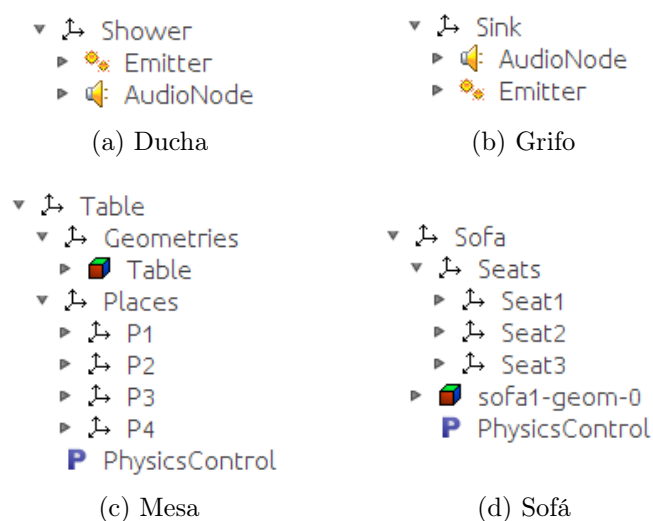


Figura 4.7: Objetos de la vivienda con información, sonidos y/o emisores de partículas.

vivienda. Esta contiene todos los elementos necesarios: sus objetos, maya de navegación y localización de las habitaciones. También se ofrece otro comando para encender y apagar las luces de una habitación. Este comando podría permitir que tanto actuadores como sujetos virtuales puedan encender y apagar las luces de la vivienda.

En este apartado se puede comprobar la carga de trabajo que lleva generar los recursos multimedia, y que incluso el mayor esfuerzo del módulo corresponde a crear dichos recursos, más que el de implementar la lógica o el comportamiento de los elementos. Por ello, como trabajo futuro es importante facilitar el proceso de creación de las viviendas en dos sentidos: (1) ofreciendo una amplia librería de objetos domésticos (muebles, electrodomésticos, sanitarios, etcétera) y (2) modificar SweetHome3D para permitir añadir información semántica a los elementos del entorno (identificador y rol, propiedades físicas, entre otros) al igual que se hizo en UbikSim y Diewald para su simulador. Una vez que se da soporte para crear el entorno (tanto el interior como el exterior de la vivienda), el siguiente apartado está dedicado al módulo encargado de simular el cuerpo de los sujetos virtuales.

4.5. El módulo dedicado a los cuerpos de los sujetos virtuales

La simulación de los cuerpos de los sujetos virtuales es una de las partes más importantes. La enfermedad de Parkinson afecta al sistema nervioso encargado de controlar los músculos, por lo que los pacientes experimentan problemas relacionados con el aparato locomotor. El síntoma más conocido es el del temblor de manos, pero también hay otros como el de la cabeza, el de rigidez y el de pies congelados. Por ello, PHAT debe permitir simular aspectos comunes de los individuos y los específicos de

los del párkinson. Las funcionalidades que el simulador debe ofrecer para los cuerpos de los sujetos virtuales son:

1. *Apariencia*. Debería poder configurarse la apariencia de los sujetos virtuales (como el color de piel, la altura o la complexión) para poder identificar fácilmente qué sujeto virtual se corresponde con el paciente o el familiar.
2. *Movimientos naturales*. El sujeto virtual debe moverse acorde a la actividad que está realizando, por ejemplo, andar, beber y comer, o sentarse en una silla. Los movimientos específicos identificados para el enfermo de párkinson, son los temblores de manos y cabeza y las caídas. Esta propiedad es muy importante para conseguir que los valores generados por los sensores virtuales que se asocian al cuerpo (e.g. un acelerómetro) sean realistas.
3. *Navegación*. El sujeto virtual debe ser capaz de desplazarse de un punto a otro evitando objetos.
4. *Hablar*. Los sujetos virtuales deben ser capaces de hablar como medio para comunicarse. Requisito importante para simular situaciones donde el paciente pide ayuda y para interaccionar con las aplicaciones a través de comandos de voz.
5. *Percepción*. Cada sujeto virtual debe percibir también el entorno de forma realista. Por ejemplo, debería ver solo aquellas cosas que están a su vista y oír las que están próximas.

Con estos requerimientos, puede comprobarse que, además de disponer del modelo 3D del sujeto virtual, es necesario una gran cantidad de funcionalidades. Aun así, la creación de sujetos virtuales es de los recursos más costosos. Un sujeto virtual se compone de una piel, un esqueleto y unas animaciones. La piel corresponde al modelo 3D formado por una malla y unas texturas. El esqueleto se compone de huesos y articulaciones que se asocian a puntos de la malla que modelan el sujeto virtual de forma que si se mueve un hueso se mueven dichos puntos asociados. Esto permite que las animaciones sean secuencias de desplazamientos y rotaciones de los huesos.

A la hora de obtener un modelo 3D del sujeto virtual para el simulador, se plantearon varias opciones:

- Crearlo uno mismo desde cero con una herramienta de modelado genérica como, por ejemplo, Blender⁶. Esta opción requiere conocimientos avanzados en modelado 3D y animación.
- Utilizar una herramienta específica para la creación de modelos 3D con forma humana. MakeHuman⁷ es una herramienta de código abierto para crear modelos 3D de este tipo. Es un proyecto muy activo y actualmente está muy evolucionada permitiendo crear todo tipo de modelos humanos 3D fácilmente variando

⁶Sitio web de Blender: <https://www.blender.org/>

⁷Sitio web de MakeHuman: <http://www.makehuman.org/>

parámetros como la edad, el sexo, la altura, el peso o la raza. También se puede dotar al cuerpo 3D de esqueleto, pero no facilita la creación o incorporación de animaciones.

- Comprar u obtener gratuitamente un modelo 3D ya creado. Existen gran variedad de sitios web para la compra de recursos multimedia 3D y algunos pocos gratuitos. La dificultad de esta opción radica en encontrar el modelo 3D que encaje con las características deseadas.

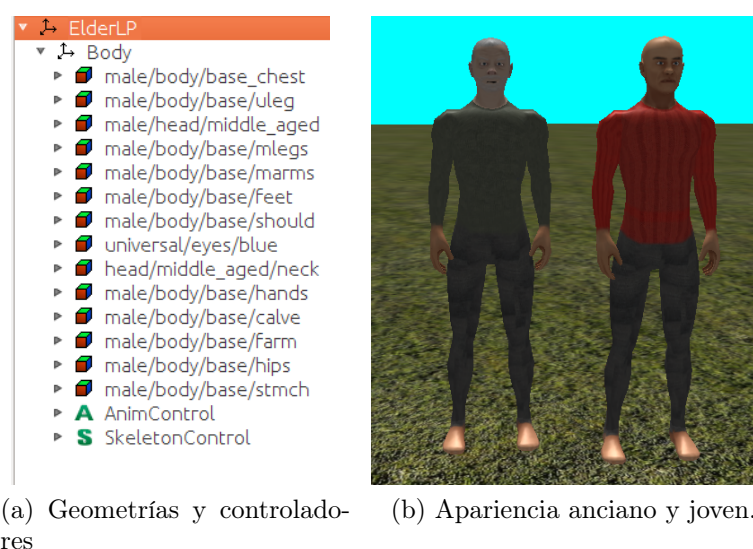


Figura 4.8: Metamodelo y modelo equivalente de las entidades de condición.

En el momento de la elección, MakeHuman no estaba muy maduro. No permitía la exportación a formato Ogre3D preferido de jME3, ni la inclusión de esqueleto o animaciones. Finalmente, se obtuvo un modelo 3D gratuito de la web [WORLDFORGE](http://worldforge.org/)⁸ dedicada al desarrollo de aplicaciones y recursos de código libre para la creación de mundos virtuales. Modificando las texturas del modelo 3D se obtuvo dos sujetos virtuales, uno más anciano para representar al paciente y otro más joven para representar a un familiar o cuidador. En la Figura 4.8a pueden verse las geometrías que crea jME3 al cargar el modelo 3D y los controladores, uno para el esqueleto (`SkeletonControl`) y otro para las animaciones (`AnimControl`). En la Figura 4.9a se puede ver la relación de huesos y cómo esta es bastante detallada, ya que incluye los dedos de la mano. El modelo también incluía un conjunto bastante amplio de animaciones interesantes para nuestro dominio como: andar (`WalkForward`), saludar (`WaveAttention`), estar parado de pie (`IdleStanding`), levantarse del suelo (`StandUpGround`), bostezar para indicar sueño (`Yawn`), comer (`EatStanding`), beber (`DrinkStanding`), llevarse la mano a la barriga para indicar que tiene hambre o que le duele (`Hand2Belly`) y barrer (`Sweeping` y `Sweeping1`) para simular que está haciendo tareas del hogar.

⁸Sitio web de WORLDFORGE: <http://worldforge.org/>



Figura 4.9: Huesos y animaciones que ofrece el modelo 3D del sujeto virtual.

Partiendo de una apariencia básica, un esqueleto y unas cuantas animaciones, es necesario enriquecer el sujeto virtual. Para ello, fue necesario crear más de 20 controladores encargados de coordinar y generar movimientos, de la navegación por el entorno y de la percepción. Dichos controladores se asocian al sujeto virtual para variar su comportamiento. A través de los comandos que ofrece la clase principal del módulo *BodiesAppState* se activan y desactivan dichos controladores. Los comandos disponibles son numerosos y variados. En el Apéndice B se encuentran todos los comandos disponibles.

Se han creado comandos de inicialización. Por ejemplo, para crear tipos de cuerpos o para situar el cuerpo en el lugar de la casa indicada. En su versión actual, los modelos 3D de los sujetos virtuales corresponden a los de la Figura 4.8b.

También se ofrecen comandos relacionados con los desplazamientos de los sujetos virtuales: quitar un sujeto virtual del escenario, establecer la velocidad de desplazamiento, desplazarse a una habitación o junto a un objeto. Destacar el comando que simula las caídas del sujeto virtual. Para que la caída resulte convincente, el comando utiliza el controlador *KinematicRagdollControl* que hace uso de la física, ver Figura 4.10.

Otros comandos están dedicados a interaccionar con el entorno. Por ejemplo, coger un objeto con la mano (izquierda o derecha) y dejarlo en un algún objeto (e.g. una mesa), sentarse y levantarse de una silla, decir una frase y abrir o cerrar un objeto. Los comandos para abrir y cerrar un objeto son genéricos y su funcionalidad depende del



Figura 4.10: Controlador KinematicRagdollControl para simular las caídas según las leyes de la física. Este genera automáticamente objetos rígidos entorno a los huesos para simular el efecto de la física sobre los mismos (colisiones y gravedad).

objeto. Por ejemplo, abrir/cerrar una puerta, abrir/cerrar un grifo o abrir/cerrar una luz (encenderla/apagarla).

También se han creado comandos muy necesarios para que el sujeto virtual realice gestos. El más sencillo, simplemente reproduce una de las animaciones de la Figura 4.9b. Otros más complejos e interesantes se encargan de simular características del enfermo de párkinson. Por ejemplo, acortar los pasos, encorvar la postura del cuerpo, limitar el balanceo de un brazo, simular temblores en las manos y la cabeza.

Por último, se han implementado comandos para facilitar la depuración de las simulaciones. Por ejemplo, un comando para mostrar y ocultar los iconos que representan las tareas de los sujetos virtuales, o un comando para que se visualice el nombre del sujeto, o un comando para hacer que la cámara siga a un sujeto dado. Esta funcionalidad última es útil cuando se desea mantener el foco todo el tiempo en un solo sujeto virtual.

4.6. El módulo dedicado a los dispositivos

Los dispositivos son los elementos fundamentales que forman la infraestructura de red de los sistemas AAL. Estos pueden estar formados por cero o varios sensores, por cero o varios actuadores y al menos un mecanismo de comunicación. A través de los sensores, el sistema AAL percibe el contexto en el que se encuentran los usuarios y les ofrece servicios acordes a sus necesidades. Dichos servicios pueden modificar el entorno a través de los actuadores. La comunicación es fundamental para interconectar los dispositivos y que estos puedan tomar decisiones, no solo basándose en la información que ellos recogen, sino también en la del resto. Por ejemplo, a través de un acelerómetro (sensor) el sistema puede detectar una caída y hacer sonar una alarma a través de un altavoz (actuador). El sensor podría encontrarse instalado en la muñeca del sujeto. En cambio, el altavoz podría corresponder al del teléfono personal del cuidador. Por

tanto, es necesario la interconexión y comunicación de ambos dispositivos. Como ya se discutió en el apartado 2.1, los dispositivos inteligentes son un tipo de dispositivo que incluye tanto sensores, como actuadores, varios mecanismos de comunicación y buena capacidad de cómputo para albergar el sistema AAL, o al menos parte del mismo. Algunos de los sensores que pueden disponer estos dispositivos son: micrófono, cámara, acelerómetro, giroscopio, proximidad, presión atmosférica, pulsómetro y luminosidad. Los actuadores comunes son: altavoces, pantalla, vibrador y el flash de la cámara. Y las tecnologías de comunicación: Wifi, NFC, 3/4G y Bluetooth. El objetivo de este módulo es crear lo que correspondería a la parte hardware de los dispositivos y simularlos, es decir, los sensores, los actuadores virtuales y los componentes de comunicación.

Si se presta atención a los simuladores AmI vistos en el apartado 2.3, se puede comprobar que existen diferencias entre ellos. La mayoría se centra en sensores y, algo menos en actuadores. Hay que tener en cuenta que algunos de ellos, como por ejemplo SIMACT [17] o UbiSim [76], están enfocados al reconocimiento de actividades. Por tanto, no se precisa de actuadores. Sin embargo, la simulación de mecanismos de comunicación es todavía menos común. De todas las propuestas analizadas, solo TATUS [109] y UbiREAL [103] se integran con simuladores de redes. Esta integración permite simular la propagación de las ondas de radio de las antenas y así poder simular más fielmente determinadas situaciones. Por ejemplo, aquellas donde los dispositivos pierden la señal. Por otro lado, resulta extraño que las propuestas no enfatizen lo ventajoso que resultaría disponer de un repositorio básico de sensores, actuadores y mecanismos de comunicación virtuales genéricos y configurables con los que componer los dispositivos, similar a las ideas de UbiREAL [103]. Por ejemplo, crear un dispositivo con forma de pulsera y equipado con conexión WiFi, un acelerómetro y un altavoz que sirva para detectar caídas y recordar las tomas de medicamentos. De todas las propuestas, solo la propuesta Persim3D [57] hace especial énfasis en la importancia del diseño e implementación de un conjunto de sensores.

En el caso de PHAT, los sensores se han implementado utilizando un patrón *listener*, ver Figura 4.11. Un sensor extiende a un controlador pues va a necesitar periódicamente tiempo de ejecución para percibir el entorno. Además, tiene una lista de objetos interesados (*SensorListener*) en sus valores (*SensorData*). En la figura se puede ver el ejemplo de un acelerómetro cuyo *Listener* es una clase que muestra una gráfica de las aceleraciones recibidas (*AccelerationData*).

PHAT ofrece actualmente tres sensores virtuales: acelerómetro, cámara y micrófono, con los cuales se puede dotar un dispositivo. Estos son los que se han considerado más relevantes para los casos de estudio planteados durante el desarrollo del proyecto y la tesis. El acelerómetro es un instrumento muy apropiado para detectar las temidas caídas. Son varios los autores [74, 18, 67, 81] que han diseñado y evaluado algoritmos para detectar caídas en sujetos que realizan actividades cotidianas. En estos trabajos se evalúan los algoritmos instalando los sensores en diferentes partes del cuerpo (muslos, cintura, pecho o cabeza). Estos trabajos emplean a sujetos reales para que simulen las caídas o toman los datos de repositorios. PHAT podría ser una alternativa para evaluar este tipo de algoritmos a través de simulaciones. Así, además de ahorrar costes,

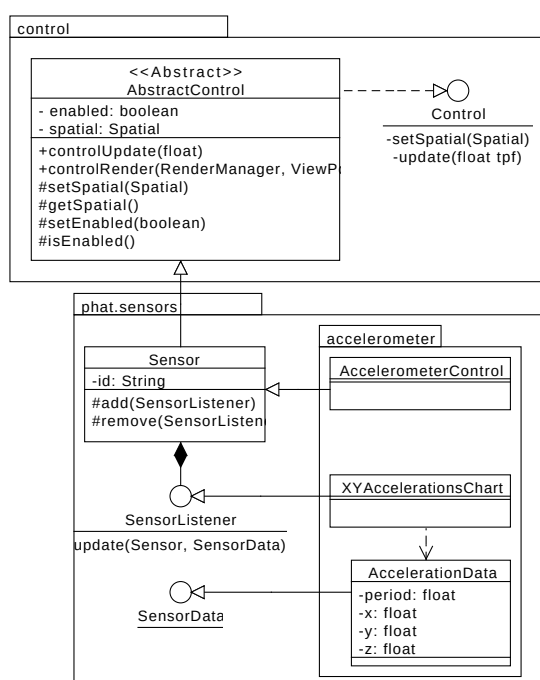


Figura 4.11: Clases controladores de PHAT para crear sensores con ejemplo del acelerómetro.

se permitiría la evaluación de estos algoritmos en condiciones más peligrosas como, por ejemplo, en caídas por las escaleras. De hecho, se está trabajando en esta dirección ya que se han iniciado los primeros contactos con los responsables del concurso de reconocimiento de actividades EvAAL⁹ para que PHAT tenga un papel en dicho concurso, bien como plataforma evaluadora o de entrenamiento de los algoritmos. La implementación del acelerómetro no resulta complicado gracias al motor físico y al controlador *KinematicRagdollControl* vistos en el apartado anterior. Ya que el cuerpo se cae según las leyes de la física, el acelerómetro solo tiene que generar las aceleraciones en función de las variaciones de velocidad respecto al tiempo. En la Figura 4.12 puede verse, a la izquierda, la simulación de un sujeto virtual con un dispositivo con sensor de aceleración asociado al pecho y, a la derecha, la gráfica de las aceleraciones. Nótese que en la parte central de la gráfica hubo una caída. A pesar de la utilidad de este sensor para el reconocimiento de actividades, ninguno de los simuladores estudiados lo ofrecen.

La cámara es un sensor que puede ser muy útil para el desarrollo de este tipo de aplicaciones de AAL. Se puede emplear para detectar situaciones de alarma automáticamente a través de algoritmos de visión por computador o para que un familiar pueda descartar falsos positivos tras una alarma. Estudios como [112] realizan reconocimiento de actividades cotidianas a partir de imágenes subjetivas procedentes de una cámara.

⁹Sitio web de EvAAL: <http://evaal.aaloo.org/>

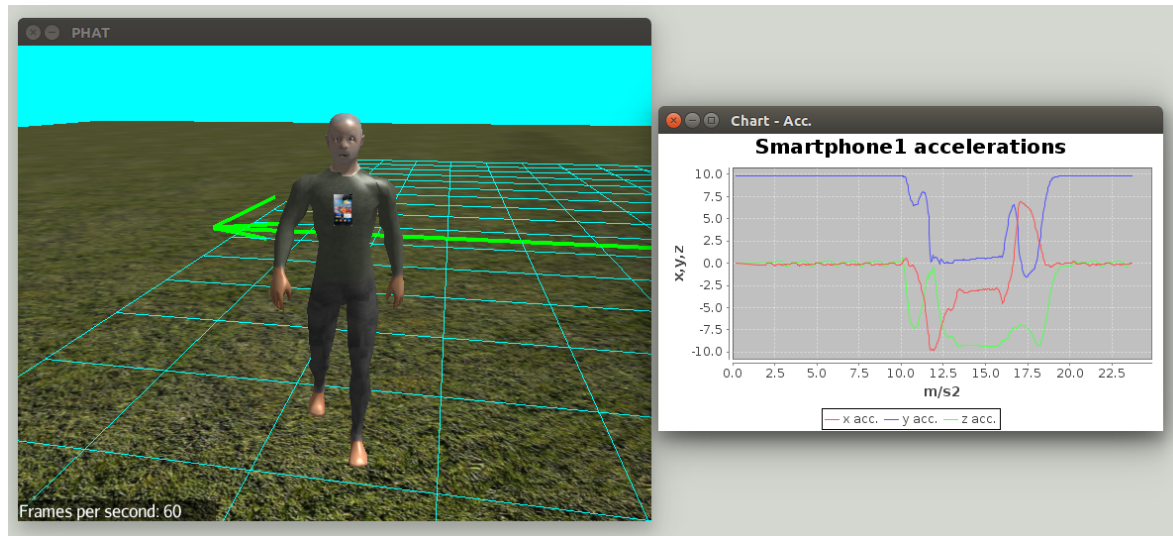


Figura 4.12: Simulación para probar el acelerómetro de un dispositivo asociado al pecho de un sujeto virtual que realiza repetidamente las acciones de andar, caer y levantarse. A la derecha se muestra la gráfica de los valores de las aceleraciones generadas en x, y, z.

Según sus estudios, la clave para reconocer actividades es identificar aquellos objetos con los que está interaccionando el sujeto. Por ello, para probar este tipo de aplicaciones en el mundo virtual, es fundamental gran nivel de detalle tanto en el entorno, como en los objetos y en los movimientos de los sujetos virtuales. La cámara también es un sensor fácil de simular pues en los motores de juegos ya está creado este elemento. En la Figura 4.13 puede verse una simulación donde hay tres ventanas. La ventana central contiene dos dispositivos representados mediante cubos con una cámara cada uno mirando a la izquierda (flecha verde). En la ventana de la izquierda se muestran las imágenes que observa el dispositivo azul y en la de la derecha las del rosa. Nótese cómo el dispositivo rosa sí que ve al dispositivo azul. No obstante, este tipo de sensores son polémicos. Las entrevistas, al igual que estudios como [92], reflejaron que las cámaras no son bien acogidas por los ancianos aunque sí despiertan el interés de los familiares. Se ofrece este tipo de sensor porque los autores lo consideran interesante (al igual que los familiares), siempre y cuando se garantice la privacidad.

Los micrófonos son sensores que perciben el audio generado por el entorno. Estos son interesantes tanto para el reconocimiento de actividades como para emplear interfaces de voz gracias a técnicas de reconocimiento de voz [116], es decir, traducir las palabras habladas a texto. La creación de los micrófonos, como sensores virtuales en PHAT, se basa en el trabajo de McIntyre [89]. Este modificó la librería OpenAL para permitir percibir el audio desde distintos puntos de la escena. Gracias a esta modificación, el desarrollo de los micrófonos ha sido sencillo. En la Figura 4.14 se puede ver una demostración basada en una previa realizada por McIntyre donde hay 3 dispositivos (SP1, SP2 y SP3) y una fuente de audio que sigue el camino azul. Cada dispositivo posee su propio sensor de micrófono. Además, se ha implementado un *SensorListener*

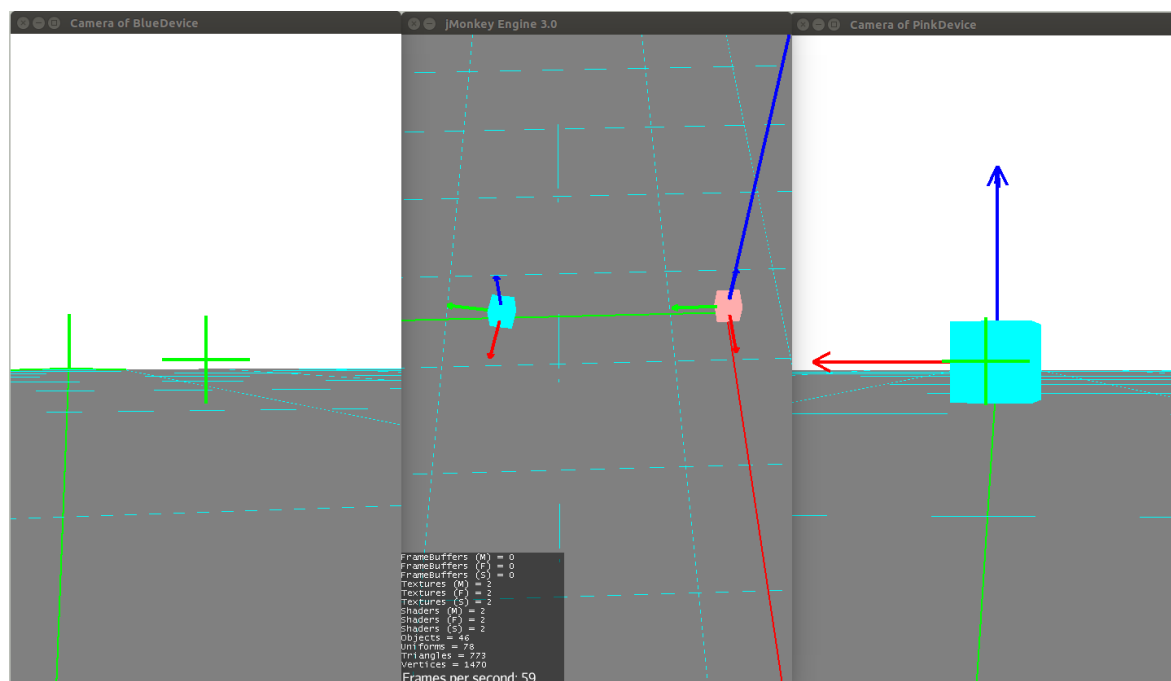


Figura 4.13: Dos dispositivos con el sensor de cámara. En el centro la ventana de la simulación, a la izquierda la vista de la cámara del dispositivo azul y a la derecha la vista del dispositivo de color rosa.

que obtiene los datos de audio y dibuja una gráfica con la intensidad del audio a lo largo del tiempo. Observando las gráficas se puede comprobar cómo varía el sonido dependiendo de la posición de la fuente de audio. En primer lugar, la fuente visitó el dispositivo SP1 (instante del 0 al 30), luego el SP3 (instante del 35 al 65) y, actualmente, esta se encuentra entorno al SP2 (instante 80).

Los actuadores son elementos para modificar el entorno. Un altavoz puede hacer sonar una alarma debido a una caída o reproducir un mensaje que recuerde al paciente que debe tomar la medicación. Un flash del teléfono inteligente puede utilizarlo el sistema AAL para iluminar cuando sea necesario o para llamar la atención del paciente si este no puede oírle. Para este último objetivo, el vibrador también puede resultar útil si el dispositivo se encuentra en contacto con el paciente. Por último, la pantalla es un actuador que principalmente servirá para mostrar información y para interactuar con el sujeto. El actuador que incorpora PHAT, por el momento, es el de la pantalla. Esta consiste en la proyección de un flujo de imágenes sobre la superficie de una geometría. Esto nos permitirá, como se verá más adelante, que a base de capturas de pantalla del emulador, estas se puedan proyectar en el dispositivo virtual. De esta forma, se dota de realismo a la escena y facilita la identificación de los dispositivos en función de lo que muestren sus pantallas.

En lo que se refiere a la simulación de la comunicación, aunque puede resultar interesante, no se ha abordado en este trabajo. Eso no significa que las aplicaciones del sistema AAL no se puedan comunicar, sino que lo harán directamente a través de sus

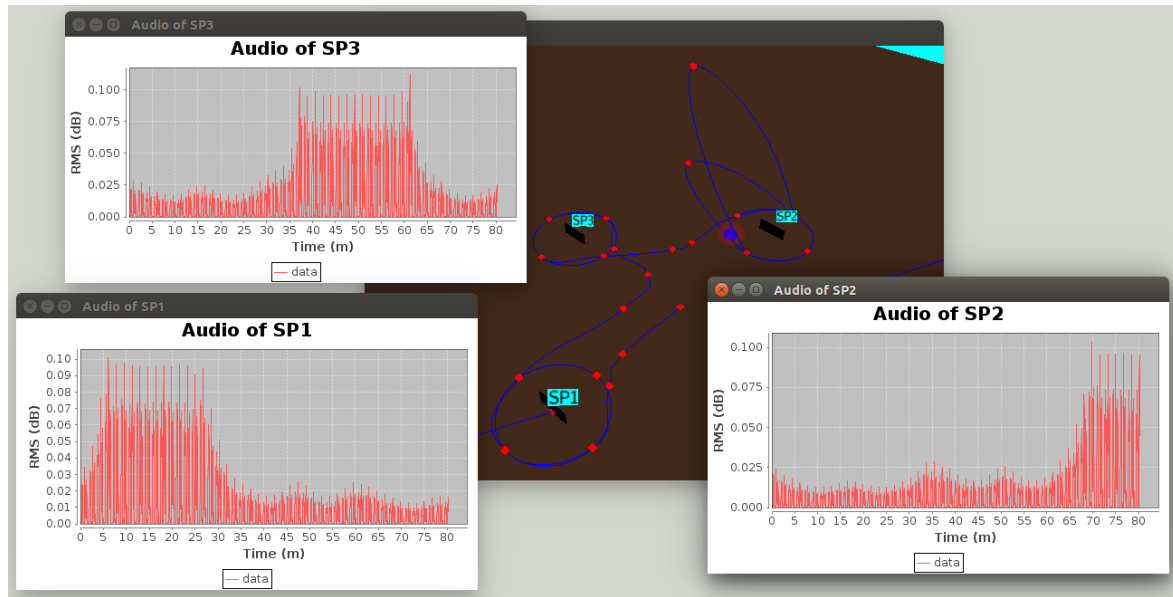


Figura 4.14: Simulación basada en [89] donde se comprueba cómo varía la intensidad del audio percibida por cada micrófono (en dispositivos SP1, SP2 y SP3) dependiendo de la fuente de audio (esfera azul).

emuladores sin tener en cuenta las condiciones de la simulación. Esta parte se verá más claramente en el siguiente apartado dedicado a la sincronización entre los dispositivos virtuales y los reales (emuladores).

Al igual que con el resto de elementos, los dispositivos los gestiona un `AppState`, ver Figura 4.3. Los comandos que ofrece esta clase para dar soporte a crear simulaciones con dispositivos virtuales se encuentran en la Apéndice B. Estos se encargan de crear un dispositivo Android con unas dimensiones dadas (incluyendo a los sensores implementados), situar al dispositivo encima de un mueble o fijarlo a una parte del cuerpo del sujeto virtual. Las partes del cuerpo que se permiten son: “LeftHand”, “RightHand”, “LeftWrist”, “RightWrist”, “LeftUnkle”, “RightUnkle”, “Head” y “Chest”. En la Figura 4.15 puede verse un dispositivo en la muñeca izquierda haciendo de reloj y otro más grande en el pecho.



Figura 4.15: Ejemplo de dispositivos asociados al sujeto virtual. Uno se encuentra fijado al pecho y el otro a la muñeca izquierda.

4.7. El módulo para acceder a los dispositivos virtuales

Se ha visto que el módulo de los dispositivos permite crear los componentes hardware de los dispositivos virtuales (sensores y actuadores). Este módulo tiene dos objetivos principalmente. Por un lado, debe permitir que software externo a la simulación pueda tener acceso a las funcionalidades de dichos componentes hardware virtuales que se encuentren en el simulador. En nuestro caso, el sistema AAL es el software externo que debe percibir y actuar en el mundo virtual creado en PHAT. Por otro lado, se debe permitir que un sujeto virtual interactúe con la interfaz de la aplicación instalada en el dispositivo a través de la pantalla. La pantalla es el mecanismo común de entrada y salida de los teléfonos inteligentes. Por tanto, para ello, el sujeto virtual debería ser capaz de localizar y entender aquellos elementos de la interfaz gráfica que se están visualizando en la pantalla del dispositivo. Y, además, ser capaz de interactuar con dichos componentes pulsando sobre la pantalla.

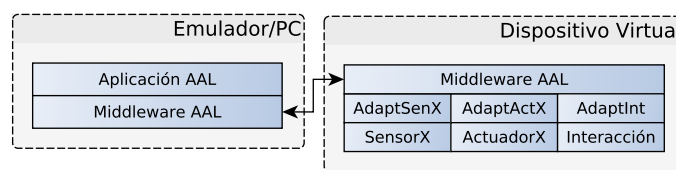
Idealmente, conseguir estos dos objetivos debe ser transparente para la aplicación AAL. Es decir, que el código de la aplicación sea el mismo tanto si está usando hardware virtual como real. Esto permite, por un lado, minimizar los errores, pues la aplicación, una vez depurada en el mundo virtual, no tiene que ser modificada [28], y por otro lado, emplear el enfoque *software-in-the-loop* para el desarrollo del sistema, donde el sistema se desarrolla y prueba en la simulación de forma iterativa. Los simuladores vistos en el Capítulo 2 se centran en la fase de diseño y son empleados para realizar

pruebas conceptuales. PHAT permitiría ir más allá y dar soporte en todas las fases del desarrollo.

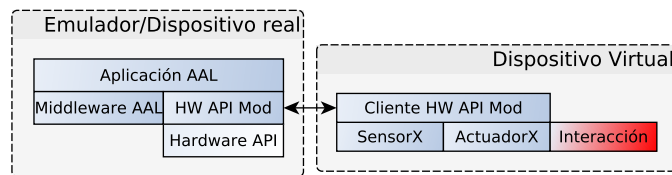
A partir de la literatura estudiada, se han identificado tres técnicas para que el software del sistema AAL pueda acceder al mundo virtual, ver Figura 4.16:

1. *A través del Middleware.* Como muestra la Figura 4.16a, la aplicación AAL en el PC o en el Emulador interacciona con el mundo virtual exclusivamente a través del middleware. En la parte del dispositivo virtual es necesario disponer de adaptadores por cada tipo de sensor, de actuador e de interacción. Estos generarán contexto o lo consumirán según el caso. Por ejemplo, un acelerómetro generará contexto a partir de las aceleraciones registradas en x, y, z. En cambio, un actuador, consumirá contexto relacionado con su estado. Si el estado proporcionado por el contexto difiere del que posee el actuador, este cambiará al indicado por el contexto. Por ejemplo, un actuador virtual que represente al flash del dispositivo recibirá notificaciones del middleware con cambios en su estado (encendido/apagado). Si el flash se encuentra apagado y recibe que su estado es encendido, este se encenderá en el simulador generando iluminación. En el caso de la interacción sería más complejo y la ontología que modela el contexto debería recoger los elementos de la aplicación y las acciones que se puedan realizar sobre los mismos. En la parte del emulador o PC, la aplicación AAL se lanzaría sobre el middleware tal y como se haría en un dispositivo real. Tanto el middleware de la aplicación AAL como aquel que se encuentra en el simulador deben poder configurarse para que se comuniquen y sincronicen su contexto.
2. *A través de la API de acceso al hardware.* Esta alternativa consiste en ofrecer una librería con la misma interfaz del SDK del dispositivo que da acceso a los sensores y actuadores, pero modificada para que acceda a los virtuales. En la Figura 4.16b se puede ver esta técnica. La librería *HW API Mod* se sitúa entre la aplicación y el HW del dispositivo para acceder a los dispositivos virtuales en vez de a los reales. En este caso, la aplicación AAL del dispositivo real podría registrar datos de los sensores y publicarlos en forma de contexto tal cual en el middleware o infiriendo nuevo contexto (e.g. detectando que el paciente se ha caído). Los dispositivos virtuales publican un servidor para que la capa *HW API Mod* pueda acceder a sus sensores y actuadores. Es necesario algún mecanismo para que la capa *HW API Mod* sepa dónde se localiza cada servidor de los sensores y actuadores virtuales que le corresponden. En este enfoque no se considera el código relacionado con la interacción. Pero podría realizarse de la misma forma modificando aquellas clases del SDK del dispositivo dedicadas a la interacción (entrada y salida).
3. *A través del emulador modificado.* Consiste en modificar el emulador del dispositivo donde irá instalado el software del sistema AAL. El emulador es una herramienta que está ideada para probar software cuando no se dispone del dispositivo real. Por ello, este suele disponer de herramientas para interaccionar con

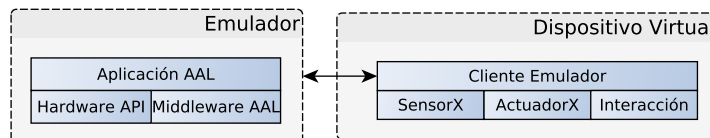
el mismo, permitiendo realizar tareas como instalar aplicaciones, simular interacción con la pantalla y eventos del sistema. Además, un aspecto muy conveniente es que permiten ejecutar exactamente la misma aplicación que luego se instalará en el dispositivo real. Como muestra la Figura 4.16c, el dispositivo virtual dispone de un cliente para comunicarse con el emulador y así enviar y recibir los datos de los sensores, actuadores e interacciones con el dispositivo. Para ello, el emulador se debe modificar para reemplazar el comportamiento de aquellas clases del sistema que se encarguen de los sensores, los actuadores y la interacción (métodos de entrada y salida).



(a) A través del Middleware.



(b) API de acceso al Hardware modificada.



(c) Emulador modificado

Figura 4.16: Diferentes arquitecturas para acceder al hardware virtual de la simulación.

Cada propuesta tiene sus ventajas e inconvenientes. La primera tiene la ventaja de que la aplicación puede correr en toda máquina que soporte el middleware (Emulador/PC). Sin embargo, las desventajas son varias. Por un lado, obliga a llevar parte de la lógica de la aplicación AAL (los adaptadores) al simulador. Esta parte del código, en el momento de su despliegue en el mundo real, debe ser traducida e integrada en el dispositivo. Por otro lado, también obliga a utilizar un middleware y/u ontología determinada y a tener que realizar la interacción sujeto-dispositivo a través del contexto, lo cual puede no resultar adecuado. La segunda propuesta tiene la ventaja de que no impone un middleware determinado y las modificaciones en la aplicación AAL son mínimas [29]. Sin embargo, solo se pueden conectar al simulador aquellas aplicaciones compiladas con la librería. Otra desventaja radica en que no permite interaccionar

con la interfaz gráfica de la aplicación. La última propuesta, si es factible, es la más recomendada. Una vez modificado el emulador y creado el cliente, se puede conectar todo tipo de aplicaciones incluso aquellas propias del sistema operativo. Por ejemplo, el servicio Android de reconocimiento de voz. Por contra, la limitación radica en que el emulador del dispositivo debe ser de código abierto para poder ser modificado y, al ser un artefacto complejo, su modificación no resulta sencilla.

Por tanto, queda de manifiesto que la modificación del emulador es la mejor opción para conectar el software del sistema AAL a la simulación. Este trabajo de tesis se ha centrado en dispositivos Android. Estos disponen de emulador que, además, es de código abierto. Pero debido a la complejidad de su código, su modificación se ha dejado como trabajo futuro. En su lugar, se ha utilizado la segunda opción pero empleando herramientas del emulador para permitir interaccionar con la aplicación. La elección se debe a que ya se disponía de una versión preliminar con UbikMobile [29]. Esta versión se ha depurado, ampliado, publicado como artefacto Maven. Su código es abierto y está accesible en la web bajo el nombre de jALI (java Android Layer for Interaction)¹⁰.

La propuesta para conectar el sistema AAL a la simulación se muestra en la Figura 4.17. Los 3 elementos principales son el simulador PHAT, el Emulador 1 y el servidor jDOS. En el simulador se puede ver cómo el *Dispositivo1* dispone de sensores y actuadores que interaccionan con el mundo virtual. Sobre esta capa, la capa *Server* crea servidores de sensores (*Sensor1Server* y *Sensor1Server*), actuadores (*Actuator1Server*) y mecanismos de interacción con el emulador Android (*jALI-adm*). Estos servicios se publican en el servidor jDOS (java Directory Of Services)¹¹. En la figura puede verse cómo, por cada identificador del dispositivo, se registra el tipo de sensor/actuador, la ip (la de PHAT x.x.x.x) y el puerto donde están accesibles. Una vez publicados, gracias a *jALI-adm* se lanza el emulador correspondiente (*Emulador1*), se transfiere el fichero de configuración donde se indica el identificador del dispositivo virtual (*Dispositivo1*), la ip y puerto de jDOS (y.y.y.y:60000) para que la librería jALI pueda interaccionar con los servidores de PHAT, y por último, *jALI-adm* lanza la aplicación del sistema AAL en el emulador.

Al igual que con el resto de elementos, los servicios y la interacción con el emulador los gestiona un AppState llamado *ServerAppState*, ver Figura 4.3. Estas funcionalidades se ofrecen de forma muy sencilla a través de comandos. Por ejemplo, existe un comando para asociar un dispositivo virtual con un emulador Android. De esta forma, las aplicaciones en el Emulador recibirán los valores de los sensores del dispositivo virtual dado tal y como se ha descrito en la Figura 4.17. Otro comando permite que en la pantalla del dispositivo virtual se visualice lo mismo que en la pantalla del emulador. Esta funcionalidad aporta realismo a la simulación y facilita identificar la correspondencia entre dispositivos virtuales y emuladores. Existen comandos para instalar aplicaciones en el emulador e iniciarlas. Así se facilita que se puedan simular escenarios donde varíe el software instalado en los dispositivos. Por último, se ofrece un comando para simular pulsaciones sobre las coordenadas x, y de la pantalla del dispositivo. Así se posibilita

¹⁰Sitio web de jALI: <https://sourceforge.net/projects/jali/>

¹¹Sitio web de jDOS: <https://sourceforge.net/projects/jdos/>

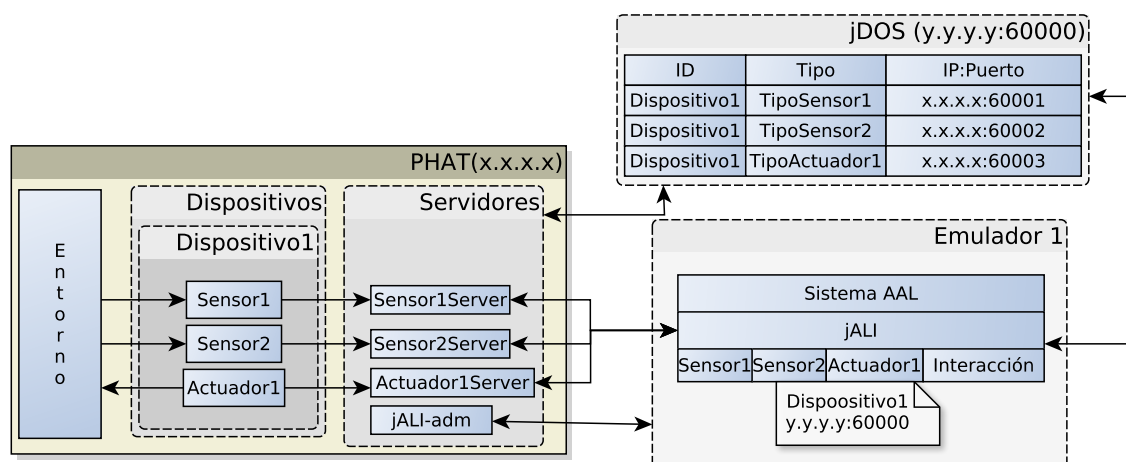


Figura 4.17: Esquema del mecanismo implementado en PHAT para permitir que las aplicaciones AAL accedan a los sensores y actuadores de su dispositivo virtual.

que los sujetos virtuales interaccionen con las interfaces gráficas de los dispositivos. Para más detalles sobre los comandos, consultar el Apéndice B.

Puede comprobarse que estos simples comandos permiten que una aplicación AAL (compilada con la librería jALI) pueda interaccionar con el entorno virtual. Aunque la comunicación con los sensores y los actuadores está bien resuelta, los mecanismos para que un agente autónomo interaccione con la interfaz gráfica son rudimentarios. Por un lado, se debe saber la posición (x,y) de cada componente gráfico de la aplicación (e.g. los botones) en cada Activity. Esta tarea resulta engorrosa y puede variar dependiendo del tamaño y resolución de la pantalla. Por otro lado, se asume que se sabe que la aplicación se encuentra en el Activity adecuado. No se ha desarrollado mecanismos para que el agente pueda averiguar qué Activity se está ejecutando. Este problema se solucionaría modificando el emulador y permitiendo publicar el Activity y los componentes visuales que este contiene. De esta forma, se podría ofrecer un comando más abstracto como, por ejemplo, para pulsar el botón con un nombre dado. La modificación del emulador junto con esta funcionalidad se deja como trabajo futuro.

4.8. El módulo dedicado a la interfaz

Los módulos que se han explicado hasta ahora permiten crear un mundo virtual donde se dispone del entorno, la vivienda, los dispositivos y los cuerpos de los sujetos virtuales. El comportamiento del entorno viene dado por el motor físico y algunas funcionalidades extra (e.g. la simulación del sol), y el comportamiento de los dispositivos lo implementa el software AAL externo que se conecta a PHAT a través de los servidores. En esta sección, se verá cómo se le dotará de comportamiento a los cuerpos de los sujetos virtuales para que interaccionen con el entorno y reproduzcan las actividades diarias que se especifiquen en el lenguaje, incluyendo las limitaciones por la enfermedad

de Parkinson. También se verá el mecanismo para facilitar la definición de escenarios y, por último, se mostrará la interfaz de usuario de PHAT, la cual ofrece herramientas durante la visualización de la simulación en tiempo de ejecución.

4.8.1. Los agentes

En el apartado 3.3.2 dedicado a la definición de la parte del lenguaje referente a los sujetos, se identificaron qué aspectos se deseaban especificar y, por tanto, simular. En resumen, estos son: recrear el comportamiento diario de los sujetos en sus hogares con diferente nivel de detalle, de forma determinista y no determinista, y permitiendo que este varíe dependiendo de eventualidades y de limitaciones causadas por la enfermedad de Parkinson.

En simulación, se suelen emplear *agentes autónomos* para implementar el comportamiento de los sujetos virtuales. Según Franklin y Graesser, la descripción de un agente autónomo viene dado por su entorno, sus sentidos, las acciones, las preferencias u objetivos, y un sistema para seleccionar las acciones a realizar en cada momento [44]. Puede comprobarse que gran parte de los elementos encajan con las necesidades. El entorno corresponde con la vivienda, los sentidos a la vista y el oído, las acciones equivalen a los comandos definidos en el módulo *body*, las preferencias u objetivos de momento no se indican y el mecanismo para seleccionar la acción a ejecutar en cada momento viene dado por el perfil de las actividades diarias del lenguaje SociAALML. Recordar que este perfil organiza el comportamiento en varios niveles: momentos del día, actividades y tareas que, a su vez, se pueden subdividir en subtarefas tantas veces como se necesite. Por ello, el mecanismo de selección de las acciones debería ser jerárquico, tal y como las máquinas de estados finitas jerárquicas o los árboles de comportamiento. Además, el tipo de agente debe ser *reactivo*, pues varía su comportamiento inmediatamente ante cambios en el entorno, es decir, ante eventos como, por ejemplo, que suene el timbre o el paciente necesite asistencia.

Los simuladores analizados enfocados a entornos de Inteligencia Ambiental como TATUS [109], UbiREAL [103], SitCom [35], SHSim [77] e InSitu [108], implementan comportamientos simples que se limitan a describir una secuencia de acciones. Cada acción consiste en especificar la habitación a la que el sujeto virtual debe ir y el tiempo que permanecerá en la misma, asumiendo que realizan una actividad concreta. UbikSim es el único simulador que implementa un módulo de comportamientos llamado HAB [30] (Hierarchical Automaton Behaviours). Como se vio en el apartado 2.6.2, este permite realizar comportamientos jerárquicos más complejos, deterministas y reactivos. Por ello, se ha empleado una versión modificada de la misma, ya que está ideada para la plataforma de simulación multi-agente MASON¹². En un principio, se empleó dicha plataforma para implementar los agentes, pero se desechó por la sobrecarga (ya que es necesario replicar los elementos del entorno de PHAT en MASON) y porque, a parte de HAB, esta aportaba poco más.

¹²Sitio web de MASON: <http://cs.gmu.edu/~eclab/projects/mason/>

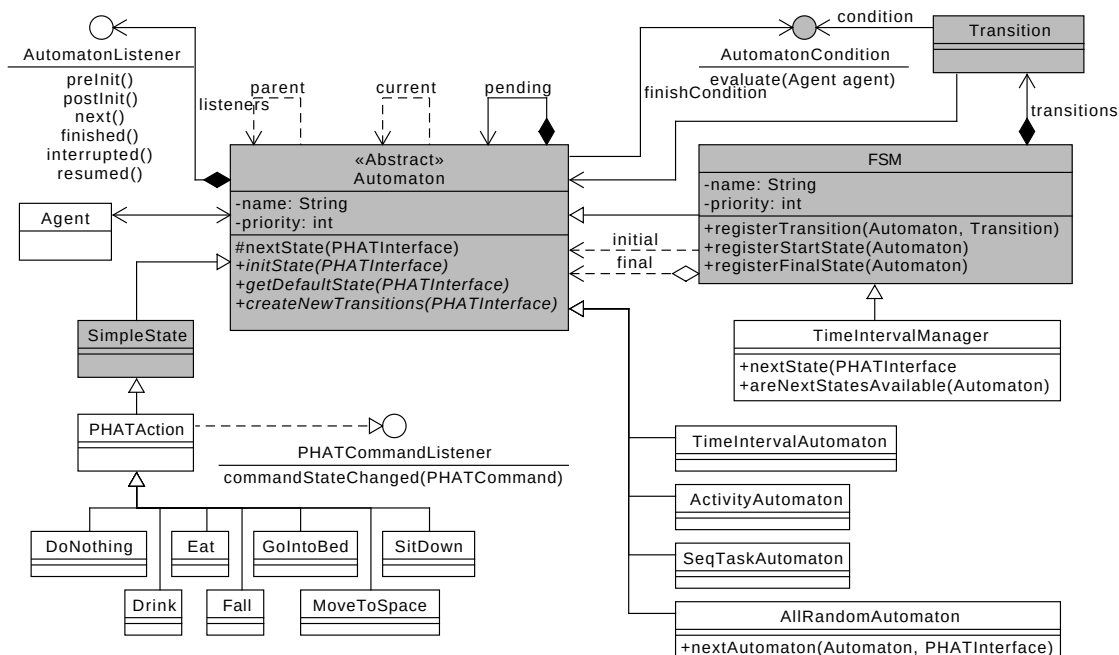


Figura 4.18: Diagrama UML de las clases principales que implementan los comportamientos.

Como alternativa, se modificó el código de HAB para funcionar con el entorno de PHAT en vez de en el de MASON. En la Figura 4.18 puede verse un diagrama UML con las clases principales del módulo. Aquellas que tienen el fondo gris corresponden a las clases que provienen del módulo HAB y las que lo tienen blanco son las nuevas que se han implementado. La clase abstracta general de comportamientos que ofrece HAB es **Automaton**. Esta contiene un nombre identificativo y una prioridad. Además, tiene una referencia a su agente y una lista ordenada por prioridad de forma que se van ejecutando primero los autómatas de mayor prioridad. Esto permite que, de forma dinámica, se añadan nuevos autómatas, reordenando la lista e interrumpiendo el autómata actual (**current**) si alguno de los nuevos tiene mayor prioridad. Un autómata termina cuando su lista de autómatas pendientes está vacía o la condición de terminación es cierta (**AutomatonCondition**). Esta clase abstracta permite el anidamiento de comportamientos de forma que en cada nivel se puede refinar el comportamiento de nivel superior. Además, este anidamiento permite la reutilización de comportamientos. Puede comprobarse cómo se extiende esta clase para crear los diferentes niveles de abstracción de los comportamientos diarios de los sujetos definidos en SociAALML: **TimeIntervalAutomaton**, **ActivityAutomaton**, **SeqTaskAutomaton** y **AllRandomAutomaton**. Este último tipo de **Automaton** es el único que redefine el método **nextAutomaton(...)** para que devuelva de forma aleatoria el siguiente comportamiento, en vez de hacerlo por orden de prioridad. El resto no añaden comportamiento nuevo

pero resulta útil poder diferenciarlos de otro tipo de Automats tal y como se verá más adelante.

La forma de gestionar y especificar los comportamientos de `Automaton` no corresponde con una máquina de estados finitos. Pero HAB ofrece una clase hija llamada `FSM` que sí la implementa. Cada estado corresponde a un autómata, se puede indicar el estado inicial con `registerStartState()` y los finales con `registerFinalState()`. Por cada autómata se puede añadir cero o más transiciones (`Transition`). Una transición tiene una condición y un autómata destino. Cuando un autómata termina se evalúan las condiciones de las transiciones registradas para dicho autómata. De todas las que resulten válidas, se elige una aleatoriamente y se toma su autómata destino como el siguiente. Como se verá en la parte dedicada a la generación de código, esto encaja, por ejemplo, con la especificación de actividades para un momento del día. En cambio, si se recuerda cómo se producían las transiciones entre intervalos de tiempo, estas se podían producir en el momento en el que se cumplía la condición para empezar el siguiente momento del día, es decir, si era la hora. Para permitir dicho comportamiento se implementa la clase `TimeIntervalManager` que redefine los métodos necesarios de la clase `FSM`.

Además de los comportamientos compuestos, la librería incluye estados simples (`SimpleState`), es decir, que no incluyen a su vez comportamientos. Por tanto, esta clase debe implementar los comportamientos de grano más fino. PHAT extiende esta clase a través de `PHATAction` para realizar las acciones del sujeto virtual, por ejemplo, la acción de no hacer nada (`DoNothing`), beber (`Drink`) o caerse (`Fall`). Estas acciones consisten en invocar a los comandos que ofrece la interfaz del cuerpo, a la cual se puede acceder a través de la clase `Agent`. Es por ello, que `PHATAction` implementa la interfaz `PHATCommandListener` para así estar informado acerca de cómo evolucionan las acciones del cuerpo. Una acción terminará, por lo general, cuando el comando le notifique que ha terminado. En el apartado dedicado a la generación del código se mostrará más en detalle qué otras clases se generan para implementar la semántica de los elementos de las actividades diarias de `SociAALML`.

Otro aspecto relacionado con el comportamiento es el de los eventos. Estos son abstracciones con significado que los sujetos virtuales pueden percibir con los diferentes sentidos. Por ejemplo, a través de la vista el familiar ve al enfermo de párkinson que está en el suelo y, en ese momento, podría generarse el evento “ayudar paciente”. Otro ejemplo podría corresponder con el momento en el que suena el timbre de la vivienda, en dicho instante se podría generar el evento “llaman a la puerta”. Una percepción del entorno ideal sería aquella en la que el agente, haciendo exclusivamente uso de los datos procedentes de los sentidos virtuales, reconociese el entorno. Por ejemplo, a través de analizar las imágenes procedentes de los ojos virtuales (sensor de cámara) reconoce los objetos del mundo virtual, o es capaz de reconocer los distintos sonidos o lo que hablan otros sujetos virtuales a partir de analizar el audio de su oído virtual. Oijen y Dignum [139] sostienen que implementar la percepción y comunicación de agentes virtuales de esta forma es complejo y no asumible computacionalmente para simulaciones en tiempo real. Por ello, sugieren emplear mecanismos que faciliten la

percepción. Los eventos son este mecanismo empleado por PHAT, los cuales facilitan la percepción y la comunicación. En PHAT, las acciones pueden lanzar eventos (al igual que en SociAALML) y, a su vez, los agentes pueden percibir los eventos. Un evento tiene asociado el sentido necesario para ser percibido. Por ejemplo, un evento producido por el timbre de un teléfono será percibido dependiendo de la distancia. Otro evento que se genere porque el paciente está en el suelo sin conocimiento será de tipo visual y solo lo procesará el agente si ve al paciente. Lo mismo ocurre con el resto de sentidos. Aunque en principio, PHAT solo tiene implementados los eventos relacionados con el sentido del oído. De esta forma, se aprovecha el entorno para decidir aquello que se puede percibir, pero dicha percepción va anotada con un evento que la identifica y le da un significado.

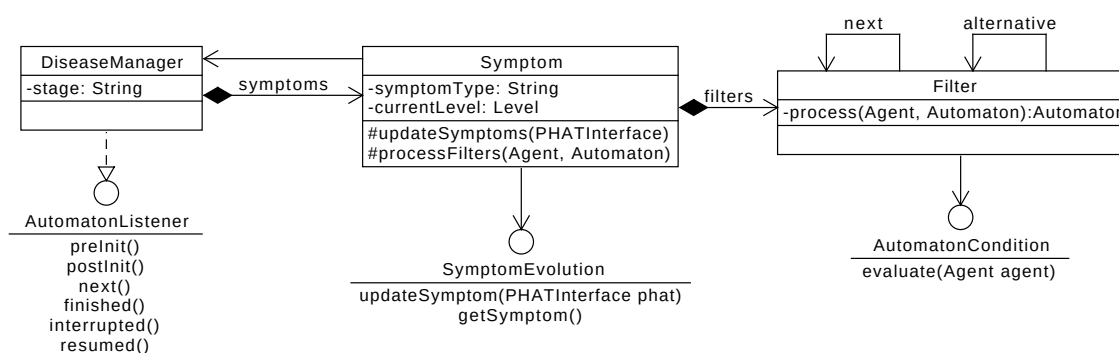


Figura 4.19: Diagrama UML de las clases PHAT que implementan la enfermedad, los síntomas y las limitaciones.

El último elemento que afecta al comportamiento de los sujetos virtuales es la enfermedad. El modelado de la enfermedad se ha implementado desde cero por lo que se ajusta mucho al perfil de enfermedad del lenguaje SociAALML descrito en el apartado 3.3.6. En la Figura 4.19 puede verse un diagrama UML de las clases principales que modelan la enfermedad. Cada agente dispone de un **DiseaseManager** que contiene a su vez la lista de síntomas (**Symptom**). El manejador tiene dos funcionalidades principales relacionadas con los síntomas, estos corresponden a los métodos `updateSymptoms(...)` y `processFilters(...)`. El primer método se encarga de controlar el nivel del síntoma delegando al objeto **SymptomEvolution** el cual es invocado en cada iteración de la simulación. En cambio, el segundo método solo se invoca cuando se cambia de comportamiento desde el método `next()` de la interfaz **AutomatonListener**. En ese momento, este método itera sobre todos los filtros llamando a su vez al método `process(...)` (si se cumple la condición) pasando como parámetro el nuevo autómata devuelto en la iteración anterior. Si finalmente se obtiene un autómata diferente al inicial, entonces se suspende el autómata original y se inicia el indicado por los filtros.

4.8.2. Iniciar las simulaciones

Una vez que se dispone de toda esta serie de módulos con sus comandos para crear mundos virtuales, es fundamental crear cada elemento virtual en el orden adecuado. Para facilitar esta tarea, se ha definido la interfaz `PHATInitializer`, ver Código 4.1, que contiene un método de inicialización por cada módulo donde crear los elementos virtuales relativos a dicho módulo. La clase principal de PHAT llama a esos métodos en el orden adecuado.

```

1 public interface PHATInitializer {
2     public void initWorld(WorldConfigurator worldConfig);
3     public void initHouse(HouseConfigurator houseConfig);
4     public void initBodies(BodyConfigurator bodyConfig);
5     public void initDevices(DeviceConfigurator deviceConfig);
6     public void initServer(ServerConfigurator deviceConfig);
7     public void initAgents(AgentConfigurator agentsConfig);
8     public String getTittle();
9 }

```

Código 4.1: Interfaz `PHATInitializer` para facilitar la creación de simulaciones en PHAT

Para ilustrar lo configurable que es PHAT y la facilidad con la que se crean los mundos virtuales, supóngase que se desea simular el siguiente escenario sencillo:

“El escenario ocurre a las dos del medio día del 3 de febrero del 2014 en una vivienda de 3 habitaciones y dos baños. Un anciano de 1.7 metros de altura sufre de la enfermedad de Parkinson y le tiembla la mano izquierda, la cabeza y está encorvado. Tiene un dispositivo asociado al pecho que controla la posición de su cuerpo. El paciente se encuentra cansado y se va a la cama a dormir durante 8 horas.”

En el Código 4.2 se puede ver cómo crear dicha simulación con PHAT. La clase `MainPHATSimulation` implementa la interfaz anteriormente mencionada (`PHATInitializer`). El método principal (línea 3) crea esta clase de inicialización y se pasa como parámetro a la principal de PHAT (`PHATInterface`). Nótese que en la línea 6 se puede establecer la semilla para que las simulaciones sean repetibles. En la línea 17, con el método `initWorld()`, se indica la fecha y la hora de la simulación y el tipo de terreno. La vivienda de 3 habitaciones y 2 baños se crea en la línea 18 con el método `initHouse()`. El cuerpo del paciente se crea con los comandos correspondientes en el método `initBodies()`, en la línea 22. El dispositivo asociado al pecho se crea y se coloca en el pecho en el método `initDevices()` de la línea 33. En el método `initServer()` de la línea 39, se hace la asociación del dispositivo virtual con el emulador Android (línea 40), además de instalar la aplicación *app.apk* (línea 41) e iniciar la aplicación que controla la posición del cuerpo (línea 42). Por último, activa el actuador de la pantalla para que en el móvil virtual se pueda ver lo mismo que se ve en el emulador Android. El comportamiento del paciente se especifica en el método `initAgents()` de la línea 50. Lo primero que hace es crear un agente con identificador “Patient” (línea 51). Fíjese que el identificador es el mismo que el del cuerpo. De esta forma se asocia cada agente con su cuerpo. Luego se crean los autómatas de ir a la cama y dormir (líneas 53 a 55) para añadirlos al autómata de estados finitos (líneas 57 a 60). La línea 62 añade al sujeto virtual la opción de mostrar los iconos sobre su cabeza en función de las actividades que este realiza. Por último, el autómata de estados finitos se añade al agente (línea 64) y, a su vez, este se añade al módulo de agentes (línea 65). Por

último, se utiliza el método `getTitle()` para especificar el título de la simulación que se mostrará en la ventana de PHAT. En el apartado de generación de código, se verá cómo esta interfaz facilita la generación de código a partir de las especificaciones creadas a partir del lenguaje SociAALML.

```

1 public class MainPHATSimulation implements PHATInitializer {
2
3     public static void main(String[] args) {
4         MainPHATSimulation sim = new MainPHATSimulation();
5         PHATInterface phat = new PHATInterface(sim);
6         phat.setSeed(0);
7         phat.start();
8     }
9
10    @Override
11    public void initWorld(WorldConfigurator worldConfig) {
12        worldConfig.setTime(2014, 2, 3, 14, 0, 0);
13        worldConfig.setLandType(WorldAppState.LandType.Grass);
14    }
15
16    @Override
17    public void initHouse(HouseConfigurator houseConfig) {
18        houseConfig.addHouseType("House1", HouseFactory.HouseType.House3room2bath);
19    }
20
21    @Override
22    public void initBodies(BodyConfigurator bodyConfig) {
23        bodyConfig.createBody(BodiesAppState.BodyType.ElderLP, "Patient");
24        bodyConfig.runCommand(new SetBodyInHouseSpaceCommand("Patient", "House1", "BedRoom1"));
25        bodyConfig.runCommand(new TremblingHeadCommand("Patient", true));
26        bodyConfig.runCommand(new SetStoopedBodyCommand("Patient", true));
27        bodyConfig.runCommand(new TremblingHandCommand("Patient", true, true));
28        bodyConfig.runCommand(new SetPCListenerToBodyCommand("Patient"));
29        bodyConfig.runCommand(new SetBodyHeightCommand("Patient", 1.7f));
30    }
31
32    @Override
33    public void initDevices(DeviceConfigurator deviceConfig) {
34        deviceConfig.runCommand(new CreateSmartphoneCommand("Smartphone1"));
35        deviceConfig.runCommand(new SetDeviceOnPartOfBodyCommand("Patient", "Smartphone1",
36            SetDeviceOnPartOfBodyCommand.PartOfBody.Chest));
37    }
38
39    @Override
40    public void initServer(ServerConfigurator deviceConfig) {
41        deviceConfig.runCommand(new SetAndroidEmulatorCommand("Smartphone1", "Smartphone1",
42            "emulator-5554"));
43        deviceConfig.runCommand(new InstallApkCommand("Smartphone1", "app.apk"));
44        deviceConfig.runCommand(new StartActivityCommand("Smartphone1", "phat.android.apps",
45            "BodyPositionMonitoring"));
46
47        DisplayAVDScreenCommand displayCommand = new DisplayAVDScreenCommand("Smartphone1",
48            "Smartphone1");
49        displayCommand.setFrequency(0.5f);
50        deviceConfig.runCommand(displayCommand);
51    }
52
53    @Override
54    public void initAgents(AgentConfigurator agentsConfig) {
55        Agent patient = new AgentImpl("Patient");
56
57        GoIntoBedAutomaton goIntoBed = new GoIntoBedAutomaton(patient, "Bed1");
58        DoNothing sleep = new DoNothing(patient, "Sleep");
59        sleep.setFinishCondition(new TimerFinishedCondition(8, 0, 0));
60
61        FSM fsm = new FSM(goIntoBed);
62        fsm.registerStartState(goIntoBed);
63        fsm.registerTransition(goIntoBed, sleep);
64        fsm.registerFinalState(sleep);
65
66        fsm.addListener(new AutomatonIcon());
67
68        patient.setAutomaton(fsm);
69        agentsConfig.add(patient);
70    }
71
72    @Override
73    public String getTitle() {
74        return "PHAT-" + getClass().getSimpleName();
75    }
76 }

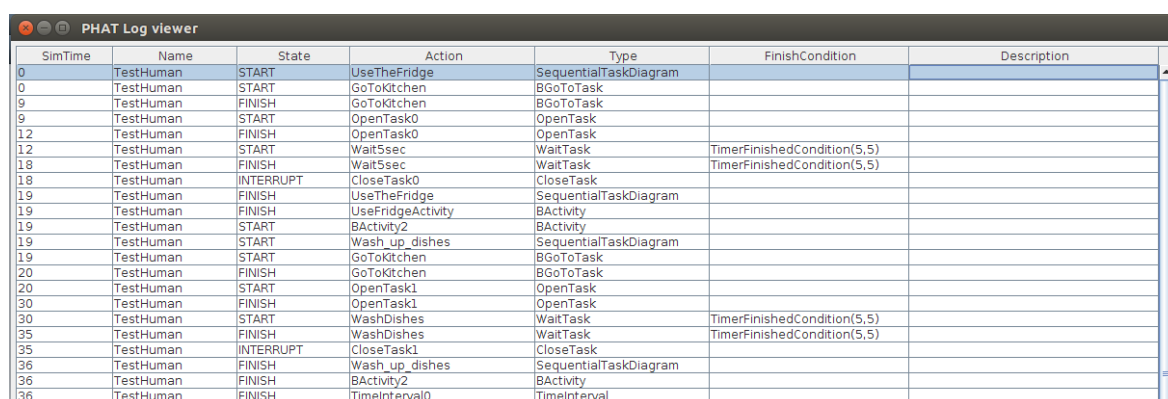
```

Código 4.2: Ejemplo de clase principal para crear un escenario en PHAT

4.8.3. Interfaz gráfica

A partir de nuestra experiencia con la generación y visualización de simulaciones, se han identificado ciertas funcionalidades que resultan de utilidad en tiempo de ejecución. Dichas funcionalidades son accesibles a partir de una interfaz gráfica la cual se muestra al acercar el ratón a la parte superior de la ventana de PHAT. Estas se describen a continuación:

- Modificar la velocidad del paso del tiempo de la simulación. Esto es de gran utilidad para ahorrar tiempo cuando, durante la simulación de un escenario, se desea avanzar hasta una situación que resulte de interés.
- Pausar la simulación durante su visualización. Resulta interesante en muchas situaciones donde se desee analizar un determinado estado de la simulación (e.g. a través del Log) o el estado del sistema AAL. Al estar pausada la simulación, se dispone de todo el tiempo que se necesite para realizar las comprobaciones oportunas. Esta funcionalidad también es conveniente cuando se muestran y explican las simulaciones a otras personas.
- Mostrar u ocultar el log de comportamientos de los sujetos virtuales, ver Figura 4.20. Por cada entrada, muestra el instante del tiempo de simulación en el que se produjo el cambio de estado (*SimTime*), el nombre del agente con el que está relacionado el comportamiento (*Name*), el nuevo estado del comportamiento (*State*), nombre del comportamiento (*Action*), indica el tipo de comportamiento, es decir, tarea secuencial, aleatoria, actividad o intervalo de tiempo (*Type*), condición de terminación del comportamiento (*FinishCondition*) y, por último, descripción del comportamiento (*Description*). Además de emplear la visualización 3D de la simulación para validar que la especificación de los comportamientos fue adecuada, el log permite analizar el comportamiento de cada sujeto virtual y comprobar si es correcta tanto durante la simulación como a posteriori. En el Capítulo 5 dedicado a los casos de estudio, quedará más clara su utilidad.
- Mostrar y ocultar información relativa al tiempo y la fecha de simulación, los fotogramas por segundo e información de rendimiento. Desde el punto del usuario de PHAT, el tiempo de simulación es la funcionalidad más interesante para poder tener una referencia y así comprobar si la simulación se desarrolla en los tiempos especificados. Los fotogramas por segundo son un indicador para comprobar si la máquina donde se ejecuta la simulación es suficientemente potente o no. Considerando que la simulación es aceptable a partir de 15 fotogramas por segundo y que es buena a partir de 30.
- Realizar capturas de pantalla de la simulación es útil para divulgar resultados o realizar documentación.



SimTime	Name	State	Action	Type	FinishCondition	Description
0	TestHuman	START	UseTheFridge	SequentialTaskDiagram		
0	TestHuman	START	GoToKitchen	BGoToTask		
9	TestHuman	FINISH	GoToKitchen	BGoToTask		
9	TestHuman	START	OpenTask0	OpenTask		
12	TestHuman	FINISH	OpenTask0	OpenTask		
12	TestHuman	START	Wait5sec	WaitTask	TimerFinishedCondition(5,5)	
18	TestHuman	FINISH	Wait5sec	WaitTask	TimerFinishedCondition(5,5)	
18	TestHuman	INTERRUPT	CloseTask0	CloseTask		
19	TestHuman	FINISH	UseTheFridge	SequentialTaskDiagram		
19	TestHuman	FINISH	UseFridgeActivity	BActivity		
19	TestHuman	START	BActivity2	BActivity		
19	TestHuman	START	Wash_up_dishes	SequentialTaskDiagram		
19	TestHuman	START	GoToKitchen	BGoToTask		
20	TestHuman	FINISH	GoToKitchen	BGoToTask		
20	TestHuman	START	OpenTask1	OpenTask		
30	TestHuman	FINISH	OpenTask1	OpenTask		
30	TestHuman	START	WashDishes	WaitTask	TimerFinishedCondition(5,5)	
35	TestHuman	FINISH	WashDishes	WaitTask	TimerFinishedCondition(5,5)	
35	TestHuman	INTERRUPT	CloseTask1	CloseTask		
36	TestHuman	FINISH	Wash_up_dishes	SequentialTaskDiagram		
36	TestHuman	FINISH	BActivity2	BActivity		
36	TestHuman	FINISH	TimeInterval0	TimeInterval		

Figura 4.20: Log de PHAT que muestra las tareas y las actividades de los sujetos virtuales.

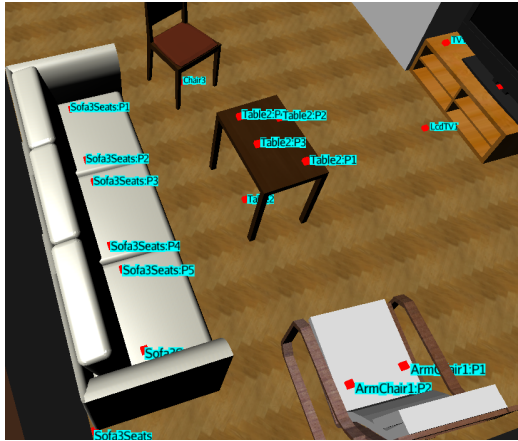
4.8.4. Herramientas

A la hora de modelar con el lenguaje puede resultar difícil saber qué identificador tiene un determinado objeto del escenario o, al contrario, dado el identificador saber a qué objeto se corresponde. Por ejemplo, en una vivienda se pueden tener 10 sillas, con identificadores *Chair1* a *Chair10*, y resulta difícil acordarse si por ejemplo *Chair1* corresponde a una silla de la cocina o a una del salón. Por ello, se ofrece una herramienta que, dado el identificador de una vivienda, lanza una visualización 3D donde se muestran los identificadores de todos los objetos. Una buena práctica consiste en emplear esta herramienta para hacer capturas de pantalla a modo documentación, por ejemplo, por cada habitación tal y como muestra la Figura 4.21. De esta forma, el modelador puede comprobar rápidamente el identificador del objeto al que quiere hacer referencia.

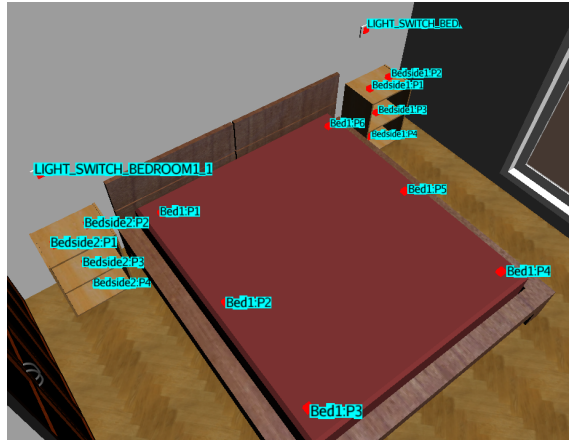
4.9. El generador de código

Una vez se dispone del lenguaje SociAALML y de la plataforma de simulación PHAT, falta el tercer elemento fundamental, el generador de código. Este debe permitir, a partir de las especificaciones creadas con SociAALML, generar simulaciones automáticamente extendiendo y empleando el código de PHAT. Al ser un proceso automático, no es necesario que el usuario de la plataforma sea un experto en simulación, sino simplemente conocer el lenguaje de modelado SociAALML. Esta es una facilidad única que no posee el resto de los simuladores vistos, pues en todos ellos es necesario programar para generar las simulaciones. Así, se permite que personal involucrado en el dominio del Parkinson, pero que no tenga conocimientos de programación, pueda generar especificaciones y visualizarlas a través de una simulación 3D.

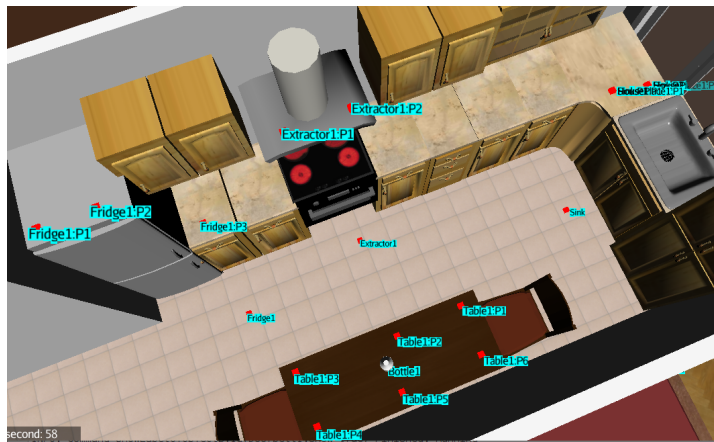
El proceso para la generación del código se basa en plantillas. Esta técnica es probablemente la más antigua y más empleada [71, p.272]. Cada plantilla corresponde a una nueva clase que se crea extendiendo la funcionalidad de PHAT. Las plantillas



(a) El salón.



(b) El dormitorio.



(c) La cocina.

Figura 4.21: Capturas de pantalla donde se visualizan los identificadores de los objetos de la vivienda “House3room2bath”.

contienen etiquetas especiales para facilitar su generación. Las dos etiquetas más importantes son las repeticiones (`<repeat id = "repeatId">...</repeat>`) y las variables (`<varId>`). Las repeticiones abarcan bloques de texto de la plantilla que pueden aparecer cero o más veces. Esto permite crear sentencias que sean opcionales o repetir varias veces un patrón. Las variables son etiquetas a las que se les puede asignar un valor. Durante la generación de código, la variable es reemplazada por dicho valor. Las variables se añaden a las repeticiones reduciendo su ámbito. Las repeticiones también se pueden anidar, es decir, añadir repeticiones a repeticiones.

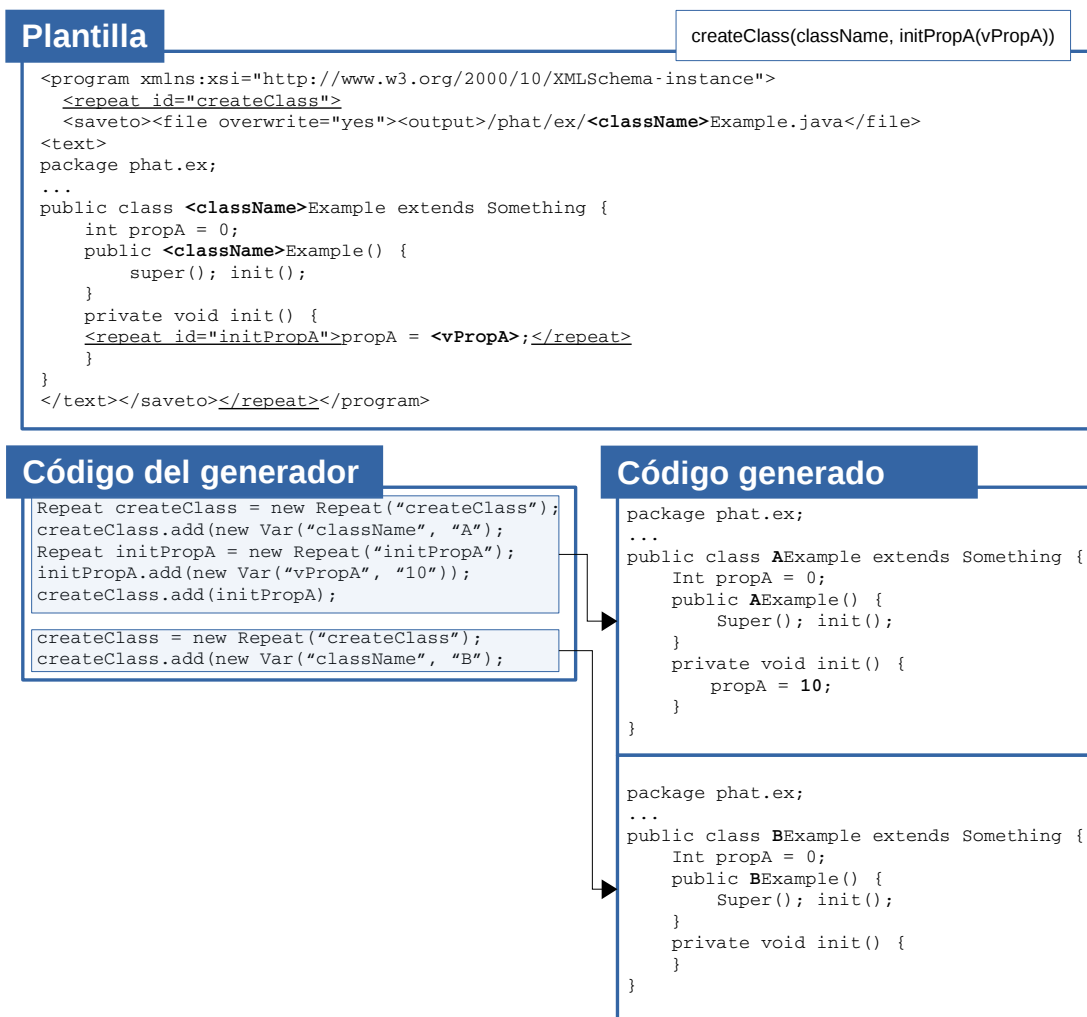


Figura 4.22: Ejemplo sencillo de generación de código a partir de una plantilla.

A fin de entender mejor cómo se traduce cada modelo de SociAALML, primero se va a explicar el proceso de generación de código a través del pequeño ejemplo de la Figura 4.22. Esta contiene una plantilla, el código del generador y las clases que genera. La plantilla consta de código Java y etiquetas para generar las clases específicas según sea necesario. En un recuadro arriba a la derecha, se muestra una simplificación de

las repeticiones y las variables que contiene la plantilla y cómo estas están anidadas. Tener esto presente, nos ha facilitado, en nuestro caso, a la hora de crear el código del generador. Las repeticiones son como funciones y pueden recibir variables o más funciones (repeticiones). La plantilla del ejemplo contiene dos repeticiones. La primera, con identificador *createClass*, crea el fichero de la clase. Para ello, se emplea la variable *className* la cual contiene el nombre de la clase y la del constructor. La segunda repetición tiene identificador *initPropA* y se emplea para iniciar el atributo *propA* a través de la variable *vPropA*. Si se observa la parte del código del generador, puede verse cómo se crean estas repeticiones y variables. En función del número de repeticiones (con sus ids) y de los valores que se asignen a las variables se crearán unas clases u otras. En el ejemplo, se remarcan los dos bloques de código que crean dos clases a partir de la repetición con id “createClass”. El código resultante de cada repetición se muestra en la parte de “Código generado”. Fíjese que en el caso de la clase B, no se crea la repetición *initPropA* y, por tanto, su contenido (la sentencia de inicialización) no se crea.

No obstante, el código del generador no se considera relevante. En su lugar, para ilustrar el proceso de traducción, se han creado unas figuras esquemáticas que muestran el proceso. Estas van ilustrando cómo a partir de fragmentos de modelos de SociAAML y plantillas se generan clases de PHAT. Para una mejor comprensión, dichas clases se representan con notación UML. Estos esquemas se encuentran en el Apéndice C.

Las plantillas creadas para la generación del código se describen a continuación:

- *Plantilla para generar las clases principales de la simulación.* Esta equivale, principalmente, a completar los métodos para iniciar la simulación de la interfaz **PHATInitialiter** que se vio en el apartado 4.8.2. Esta plantilla se aplica a todos los diagramas de simulación tal y como lo describen las Figuras C.1, C.2 y C.3.
- *Plantilla para generar los agentes.* Por cada diagrama *HumanProfileSpecDiagram* y entidad *Human*, se aplica esta plantilla tal y como muestra la Figura C.4. Su cometido es crear las clases de los agentes y enlazarlos a su ADL, a su enfermedad (si procede) y registrar las reacciones ante los eventos analizando su perfil de interacciones.
- *Plantilla para generar la clase que gestiona los momentos del día.* La plantilla se aplica a cada diagrama del tipo *ADLSpecDiagram* para generar clases del tipo **TimeIntervalManager** encargada de instanciar los momentos del día y establecer sus dependencias. En la Figura C.5 se describe dicho proceso.
- *Plantilla para generar los momentos del día con sus actividades.* Esta plantilla se aplica por cada diagrama del tipo *ActivityDiagram* y genera las clases que extienden la clase **TimeIntervalAutomaton** como muestran las Figuras C.6 y C.7.
- *Plantilla para la generación de las actividades.* Esta plantilla es sencilla pues consiste en añadir la secuencia de tareas que la define. La plantilla se aplica a toda entidad de SociAAML de tipo *Activity* tal y como muestra la Figura C.8.

- *Plantilla para generar las secuencias de tareas.* Esta plantilla se aplica a todos los diagramas del tipo *SequentialTaskDiagram* siguiendo el esquema de las Figuras C.9 y C.10.
- *Plantilla para generar las clases encargadas de implementar la enfermedad.* Esta plantilla se nutre de la información que se encuentra en varios diagramas. Primero, tal y como ilustra la Figura C.11, se recorre todos los diagramas del tipo *ParkinsonSpecDiagram* para recabar la información general de la enfermedad, es decir, estadio y síntomas. También explora los diagramas de simulación para obtener el valor inicial de cada síntoma dependiendo del nombre de la simulación. Finalmente, a partir de los filtros, analiza los diagramas de tipo *FilterDiagram* para generar las cadenas de filtros de tareas que implementarán las limitaciones en el comportamiento del paciente. Este proceso se muestra en la Figura C.12.
- *Plantilla para generar las clases que implementan la evolución de los niveles de los síntomas.* Por cada diagrama del tipo *SymptomEvolutionDiagram*, se aplica la plantilla y se crean las clases que extienden la clase `FSMSymptomEvolution`, tal y como muestra la Figura C.13.

4.10. Evaluación de los componentes software

Durante el desarrollo de esta tesis, la implementación del software necesario para llevar a cabo las ideas planteadas ha supuesto una carga considerable de trabajo. Ha sido necesario crear varios componentes: el servidor de servicios `jDOS`, la librería para conectar los dispositivos al simulador `jALI`, el simulador `PHAT`, el generador de código y el lenguaje `SociAALML`. En este apartado se ilustra el esfuerzo que ha supuesto este trabajo y la calidad del mismo.

Para medir el tamaño de un componente software, o el esfuerzo que ha supuesto su desarrollo, es común utilizar una métrica que consiste en contar el número de líneas efectivas del código fuente, esto es, ignorando los comentarios y líneas en blanco. `SLOCCount`¹³ es la herramienta que se ha empleado para estos fines. También se ha incluido el número de tipos creados (clases e interfaces) como un segundo indicador del tamaño del software. La Tabla 4.1 recoge, por cada módulo de los componentes desarrollados, el número de líneas efectivas de código Java y el número de tipos. Para el caso del generador de código, también resulta relevante el número de líneas de los ficheros XML, ya que corresponden a las plantillas. En dicha tabla se puede comprobar que el esfuerzo ha sido considerable. En total, se han contabilizado 36542 SLOCs y 634 tipos. De las cuales, `jDOS` representa entorno a 1000 SLOCs y 27 tipos, `jALI` 3300 SLOCs y 61 tipos, `PHAT` 29000 SLOCs y 531 tipos, y el generador 2839 SLOCs y 15 tipos. Como es de esperar `PHAT` ha consumido el mayor porcentaje del esfuerzo (cerca del 80 % de los SLOCs). Y de sus módulos, el más complejo es el que se encarga de la lógica de los cuerpos (*phat-bodies*), tal y como se avanzó en el apartado 4.5. Fíjese que

¹³Sitio web de `SLOCCount`: <http://www.dwheeler.com/sloccount/>

Componente	Módulo (Artefacto Maven)	SLOC (Java/XML)	Número de tipos
jDOS	jdoss-sm	441/0	8
	jdoss-common	279/0	17
	jdoss-client	192/0	2
jALI	jali-adm	619/0	11
	jali-vapi-common	460/0	10
	jali-vapi	2301/0	40
PHAT	phat-core	1609/0	34
	phat-audio	1979/0	30
	phat-env	1484/0	24
	phat-bodies	11636/0	173
	phat-devices	3224/0	68
	phat-client	141/0	4
	phat-server	2956/0	46
	phat-interface	6382/0	152
Generador	phat-generator	2839/777	15
	TOTAL	36542/777	634

Tabla 4.1: Resumen de los números de líneas efectivas (SLOC) y número de tipos (clases e interfaces) por módulo de los componentes generados.

casi dobla a aquel dedicado a los agentes, la enfermedad y los eventos (*phat-interface*). Por otro lado, destacar que el generador de código solo tiene 2839 SLOCs frente a las 29000 de PHAT. Además, la diferencia en cuanto al número de tipos es mucho mayor, 15 frente a 531. De esto se deducen al menos dos cosas. Primero, que las clases del generador contienen muchas más líneas de código por término medio que las de PHAT y, por tanto, estas pueden ser algo más complejas. Segundo, que PHAT se ha desarrollado de tal forma que ha elevado significativamente el nivel de abstracción acercándolo al dominio de aplicación. Por ello, la traducción entre el lenguaje SociAALML y el código de PHAT no ha resultado muy complejo.

Cunningham introdujo el concepto de deuda técnica (Techincal debt) [34]. Este se refiere al hecho de descuidar la calidad del software para incrementar la productividad. Las herramientas desarrolladas han seguido esta filosofía en muchas ocasiones debido a las presiones típicas derivadas de cumplimiento de plazos que se encuentran en cualquier proyecto de desarrollo de software (y esta tesis tiene asociado un desarrollo de software, como es evidente). Cunningham sostiene que este descuido no tiene por qué ser un problema si, posteriormente, se soluciona. Para medir la calidad del código generado, se ha empleado la plataforma de código abierto SonarQube¹⁴. Esta es una herramienta de gestión de la calidad del código y cubre sus 7 ejes: diseño y arquitectura, comentarios, duplicidades, pruebas unitarias, complejidad, errores potenciales y reglas de codificación. Además, se puede aplicar directamente a proyectos Maven. SonarQube incluye este concepto de deuda técnica en forma de métrica y otros más. En concreto, por cada componente creado, se comprobará si pasa los niveles de calidad (Quality Gates) y en la nota obtenida en la evaluación de la mantenibilidad. Disponer de un

¹⁴Sitio web de SonarQube: <http://www.sonarqube.org/>

código fácil de mantener supone que no es costoso, en parte, la mejora del código y añadir funcionalidades nuevas. Estas características son importantes en proyectos de código abierto, como los aquí creados, donde varios desarrolladores pueden contribuir a su mejora.

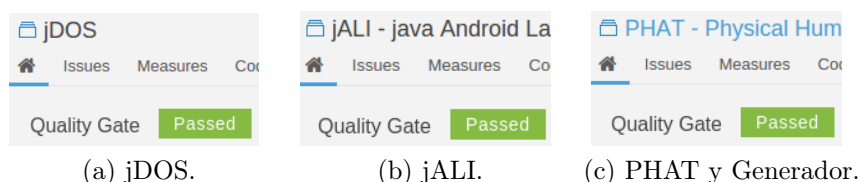


Figura 4.23: Comprobación de los niveles de calidad del código realizado por SonarQube.

SonarQube evalúa el Quality Gates a partir de comprobar que las métricas definidas para los 7 ejes de la calidad del software se mantienen en unos límites adecuados. Consultar la web para obtener más información sobre las métricas¹⁵. Como muestra la Figura 4.23, todos los componentes desarrollados pasan los niveles de calidad establecidos por SonarQube. Tener en cuenta que en esta ocasión no se evalúa el generador independientemente, sino dentro de PHAT ya que pertenece a dicho proyecto y SonarQube no diferencia.

SonarQube valora la mantenibilidad del código en función del ratio de deuda técnica. Esta la definen como cien veces la relación entre el coste de reparación del código y el coste completo del desarrollo del código del componente. Las violaciones o errores totales en el código SonarQube las denomina *code smells*. El tiempo para solucionar cada error se establece de forma fija según el tipo. En cambio, el coste de desarrollo de una línea de código se establece como 0.06 días. Donde se asume que un día tiene 8 horas. La calificación de la evaluación de la mantenibilidad se representa por una letra en función del ratio de deuda técnica obtenido: A = 0-5 %, B = 6-10 %, C= 11-20 %, D=21-50 %, E=51-100 %. En la Figura 4.24 se muestra una captura de pantalla del resultado obtenido en la evaluación de la mantenibilidad de cada componente. Fíjese cómo se obtiene en todos ellos la máxima puntuación ya que el ratio de deuda técnica (Technical Debt Ratio) es menor del 5 % en todos ellos. Las figuras también manifiestan el carácter relativo de la valoración. En jDOS (Figura 4.24a) se han detectado 120 violaciones las cuales se estima que pueden ser solucionadas en 2 días. En cambio, en jALI (Figura 4.24b), el número de violaciones es 582 y precisa de 7 días y 5 horas para su reparación. Por último, PHAT ((Figura 4.24c)) acumula 3251 violaciones las cuales se pueden solventar en 58 días. Fíjese, que aunque el número de violaciones incrementa, el ratio de deuda técnica disminuye, de 4.2 % a 2.9 %. Esto se debe a que el desarrollo se ha centrado en optimizar PHAT. En cambio, jDOS y jALI son proyectos necesarios pero en los que se ha trabajado menos.

¹⁵Definición de las métricas en SonarQube: <http://docs.sonarqube.org/display/SONAR/Metric+Definitions>

Maintainability			
120	A	Technical Debt	2d
Code Smells	Maintainability Rating	Technical Debt Ratio	4.2%
		Effort to Reach Maintainability Rating A	0

(a) Resultado del análisis de mantenibilidad para el componente jDOS.

Maintainability			
582	A	Technical Debt	7d 5h
Code Smells	Maintainability Rating	Technical Debt Ratio	3.7%
		Effort to Reach Maintainability Rating A	0

(b) Resultado del análisis de mantenibilidad para el componente jALI.

Maintainability			
3,251	A	Technical Debt	58d
Code Smells	Maintainability Rating	Technical Debt Ratio	2.9%
		Effort to Reach Maintainability Rating A	0

(c) Resultado del análisis de mantenibilidad para el componente PHAT.

Figura 4.24: Resultados del análisis de la mantenibilidad del código realizado por SonarQube.

4.11. Conclusiones

En este capítulo se ha abordado tanto el simulador como el generador de código para hacer posible que, a partir de las especificaciones del lenguaje SociAALML, se generen automáticamente las simulaciones definidas. Para ello, se han establecido unos requisitos generales para el simulador y, posteriormente, se ha ido identificando funcionalidades necesarias por cada módulo. Los experimentos son repetibles gracias a la semilla. Son convincentes gracias al motor de física (e.g. para las caídas), a las animaciones, a los sonidos y a la iluminación. En esta parte, queda trabajo en cuanto a conseguir movimientos más realistas que sean sensibles al entorno. Las simulaciones son visuales gracias al renderizado 3D y se facilita la comprensión de la escena con, por ejemplo, los iconos sobre las cabezas de los sujetos virtuales o el log. Gracias al diseño de la arquitectura de PHAT y a sus comandos, PHAT es flexible y altamente configurable tanto en el inicio de la simulación como en tiempo de ejecución. PHAT da gran facilidad para conectar software AAL externo (en la versión actual, aquel basado en Android) a los dispositivos virtuales permitiendo así simulaciones híbridas donde la parte real es el software que se prueba en el entorno virtual. Destacar la incorporación del acelerómetro virtual el cual puede ser muy útil para el reconocimiento de actividades, en concreto, la detección de caídas. Además, a pesar de su relevancia, ninguno de los simuladores AmI estudiados lo ofrecen.

Toda la funcionalidad de PHAT se puede emplear gracias al generador de código basado en plantillas. Este genera las simulaciones a partir de los modelos creados con SociAALML. Así se consigue generar simulaciones sin necesidad de ser experto en

motores de juegos o tener conocimientos de programación. Además, se ha dejado de manifiesto cómo la creación del generador no ha resultado ser compleja gracias a que se han implementado la mayoría de los conceptos del dominio en PHAT.

La evaluación del código ha reflejado la cantidad de trabajo desarrollado y su calidad. En concreto, su buena nota en la evaluación de la mantenibilidad facilita a otros desarrolladores a colaborar y extender las funcionalidades de la plataforma desarrollada.

Capítulo 5

Casos de estudio

Una experiencia nunca es un fracaso, pues siempre viene a demostrar algo.

Thomas Alva Edison

RESUMEN: En este capítulo se presenta el uso y la evaluación de las herramientas propuestas en esta tesis a través de dos casos de estudio. El primero consiste en el modelado completo de la entrevista al paciente E9, y el segundo, se centra en la actividad específica de ver la televisión. Para este último caso, además del modelado, se diseña y evalúa una solución AAL sencilla. Después se evalúa la propuesta en base a los dos casos de estudio. Finalmente, se valida la verosimilitud de los valores de los sensores simulados.

Con las herramientas presentadas, se sugiere una filosofía de desarrollo dirigida por modelos donde se persiguen cuatro objetivos:

1. *Definir escenarios validados mediante su simulación.* Estos escenarios requieren determinar el comportamiento de los sujetos en la vivienda, cómo les afecta posibles enfermedades que se den en alguno de sus ocupantes y determinar qué tipo de ayuda necesitan. Estos escenarios validados por el experto o por el propio paciente, son útiles para fases posteriores. Estos pueden hacer las funciones de casos de uso o escenarios de prueba a partir de los cuales se podrían evaluar los prototipos de los diseños.
2. *Definir el despliegue de la solución AAL en un escenario para mejorar las condiciones de vida de los sujetos para dicho escenario.* Esta definición implica determinar qué dispositivos se integrarán en el entorno, dónde se situarán y qué componente software del sistema AAL ejecutarán. La integración de este sistema y los escenarios de prueba se consigue a nivel de especificación. Es la especificación la que declara qué dispositivos existirán y cómo aparecen en la simulación.

3. *Suministrar, de forma transparente, un flujo de datos procedentes de los sensores de la simulación a la aplicación AAL que sean verosímiles.* Gracias a este enfoque propuesto, donde el sistema AAL se conecta de forma transparente a la simulación, se permite un desarrollo *software-in-the-loop*. Es decir, diseñar y evaluar distintos prototipos funcionales sin necesidad de disponer del hardware real.
4. Permitir ejecuciones de los diferentes escenarios definidos donde se evalúe el comportamiento del sistema AAL diseñado. Estas ejecuciones pueden ser de varios minutos, horas o incluso días. Los escenarios cortos son adecuados para las primeras fases del diseño de la solución donde se trata de solucionar situaciones concretas. En cambio, los escenarios de larga duración resultan de utilidad para comprobar la robustez y la respuesta del sistema ante situaciones no consideradas previamente.

Este capítulo trata de demostrar que las herramientas propuestas en esta tesis alcanzan los objetivos planteados. Para validar el primer objetivo es necesario comprobar que la simulación se comporta como esperamos. Para ello, recuérdese que PHAT dispone de dos métodos fundamentalmente. El primero consiste en la visualización 3D de la simulación. Durante la simulación, el experto puede acelerarla para llegar a momentos que resulten más interesantes, o ralentizarla o pararla para analizar un instante determinado. El segundo método consiste en analizar el log. Esta tarea es más compleja y es conveniente disponer de herramientas para extraer la información y analizarla. Pero, en cambio, el log permite obtener más información y más precisa sobre las simulaciones que el método de visualización. Por otro lado, realizar determinados análisis mediante visualización resulta muy tedioso. Por ejemplo, determinar cuántas veces se ha caído el paciente a lo largo del día y durante cuánto tiempo ha permanecido en el suelo. También costaría obtener el listado de las actividades realizadas por cada sujeto y el tiempo empleado, o registrar el número de veces que el cuidador tiene que ofrecer asistencia al paciente. Este último podría ser un buen indicador para mejorar la calidad de vida de los individuos y cuyo número debería reducirse al implantar la solución AAL. Obtener este tipo de información, sobre todo para simulaciones largas, se considera más apropiado emplear el log. Por tanto, el primer método resulta más adecuado para escenarios cortos y el segundo para realizar análisis más minuciosos de simulaciones largas. Sin embargo, aunque se ha implementado una versión preliminar del log, no se ha hecho para las herramientas de análisis. No obstante, siempre puede cada investigador emplear el tipo de herramientas a las que esté acostumbrado. Por ejemplo, R¹ es un lenguaje muy adecuado para este tipo de tareas ya que está enfocado a realizar cálculos estadísticos y representarlos a través de gráficas.

Estas dos técnicas se van a aplicar a dos casos de estudio. El primero caso se verá en el apartado 5.1 y consiste en extraer y modelar toda la información de la transcripción de la entrevista al paciente E9. El caso refleja la complejidad de modelar una semana

¹Sitio web de R: www.r-project.org

completa y cómo es posible validar su modelado a través de las herramientas de visualización y de log (objetivo 1). También demuestra cómo es posible simular la vida diaria de la pareja, sin entrar en detalle, durante una semana (objetivo 4). El segundo caso se verá en el apartado 5.2 y se centra en la actividad de ver la televisión. Este caso, a diferencia del anterior, se centra en una actividad concreta. Se profundizará más en las tareas del sujeto, en identificar situaciones problemáticas y en validarlas mediante su simulación (objetivo 1). Además, se discute brevemente el diseño de una posible solución AAL sencilla para resolver un problema concreto definiendo su despliegue (objetivo 2) y evaluando el comportamiento del prototipo frente al escenario problemático identificado (objetivo 4). El primer caso de estudio no se ha publicado todavía y el segundo está basado en trabajos previos. Los casos de uso publicados hasta ahora en congresos se han centrado en las caídas y en la actividad de ver la televisión:

- El primer caso de uso publicado corresponde a la situación más preocupante y recurrente en el dominio de los enfermos de párkinson, la de las caídas. En el artículo *PHAT: Physical Human Activity Tester* [28] se presenta el simulador y las posibilidades que presenta en este tipo de escenarios permitiendo jugar con la física, la iluminación y el sonido.
- En el artículo demostración *Agent Based Simulation for Creating Ambient Assisted Living Solutions* [25], se enriquece el escenario anterior de la caída incluyendo 3 dispositivos que hacen seguimiento del paciente a través del micrófono y la cámara. Esta demostración contribuye a demostrar el potencial de la simulación para el desarrollo de sistemas AAL y la cuál obtuvo el segundo premio patrocinado por IBM.
- En otro artículo de demostración titulado: *A Framework for Developing Multi-Agent Systems in Ambient Intelligence Scenarios* [27], y presentado en el congreso AAMAS 2015, se introduce un nuevo escenario que se centra en la actividad de ver la televisión por parte del paciente. En esta actividad, aparentemente simple, el paciente puede tener que enfrentarse a varios problemas: que se le caiga el mando a distancia, que se caiga el mismo, o que no sea capaz de usar el mando a distancia por los temblores. Se discute la posibilidad de emplear una televisión y mando inteligentes que asista al paciente ante este tipo de situaciones.
- En el artículo *Achieving Parkinson's Disease Patient Autonomy through Regulatory Norms* [46], se continúa con el escenario anterior y presenta una línea futura de investigación para la prueba automática del sistema AAL. Esta prueba se realiza a través de un sistema normativo que conectado a la simulación y capturando los mensajes JADE de las aplicaciones las penaliza o premia en función de sus respuestas ante cada situación.

Tras presentar los dos casos de estudio, en el apartado 5.3, se aportan más datos de los casos de uso en términos de métricas y estadísticas de los modelos creados y el código generado, y se realiza una valoración crítica de dichos datos y las experiencias

obtenidas durante el desarrollo de ambos casos de uso. Posteriormente, en el apartado 5.4, se realiza una validación de los tres sensores virtuales desarrollados a través de aplicaciones sencillas con el fin de demostrar que si una solución AAL resuelve un problema en el mundo real también lo hará en el mundo virtual (objetivo 3). Por último, el apartado 5.5 expone las conclusiones que se extraen de este capítulo.

5.1. Caso de estudio 1: Modelado de la entrevista al paciente E9

El primer caso de estudio corresponde al modelado completo de una entrevista, la realizada al paciente E9, y su simulación. Se ha elegido el paciente E9 por ser una de las entrevistas más completas donde más información se recoge. Por tanto, la primera tarea consiste en crear un modelo con SociAAML que contenga toda la información relevante a partir de analizar la transcripción de la entrevista. Debido a que las entrevistas se realizan a dos personas, primero al enfermo y, posteriormente, a la pareja del mismo, la información sobre un aspecto en concreto se puede encontrar en varias partes de la entrevista. Por lo que durante el modelado se necesita estar saltando de un diagrama a otro y realizar varias lecturas, pues se pueden escapar detalles. No obstante, como ya se destacó, las entrevistas no profundizan en cómo realizan cada actividad, es decir, no describen la secuencia de tareas que siguen para llevar a cabo las actividades. Sin esta descripción, no es posible generar simulaciones, o dichas simulaciones serían muy pobres modelando cada actividad como permanecer en una habitación durante un determinado tiempo. Lo adecuado hubiese sido completar los modelos realizando una segunda entrevista con el editor y los sujetos presentes. Al no ser así, se han modelado las actividades sin mucho detalle pero intentando especificar aquellas tareas más significativas para realizar cada actividad. El modelado de cada actividad se realiza de forma iterativa, modelando y validando la actividad mediante la visualización de la simulación.

Este caso trata de un paciente varón de 71 años, al cual se le detectó la enfermedad con 50 años y, actualmente, se encuentra en un estadio E3 avanzado de la enfermedad de Parkinson con un nivel de dependencia medio. Formó parte de los cuerpos de seguridad del estado pero desde hace unos años se tuvo que jubilar. El enfermo vive con su compañera de 67 años la cual también padece problemas de salud. Ambos pertenecen a una clase social media y por la pensión del marido no tienen derecho a ayuda para un asistente que acuda a su vivienda, por lo que la compañera se encarga de las tareas domésticas. En cambio, sí acude un voluntario de la asociación de Parkinson Madrid para acompañar al paciente a dar un paseo, los lunes y los miércoles durante una hora. El matrimonio tiene un hijo y una hija. La hija vive en la misma ciudad que los padres pero no dispone de tiempo. El hijo vive en otra ciudad a bastante distancia.

En este caso, los sujetos que se incluirán en la simulación son el paciente, su compañera y el voluntario de la asociación. No se modela ninguno de los hijos pues su presencia en la vivienda es algo esporádico. En los siguientes subapartados se va a

presentar: (1) un extracto representativo de cómo se ha modelado cada perfil de los sujetos a partir de la información de las entrevistas; (2) las simulaciones definidas; y, por último, (3) el análisis y validación del modelo generado (a través de la visualización 3D de las simulaciones y el log de PHAT). Si se desea acceder al modelo completo del paciente, este se encuentra como ejemplo dentro del proyecto SociAALML en GitHub².

5.1.1. El perfil social

El perfil social no es importante desde el punto de vista de la simulación, pero resulta de utilidad para recabar la información de este tipo. A modo de ejemplo, se muestra el modelo correspondiente al perfil social del enfermo E9 que se deduce de la contextualización dada anteriormente, ver Figura 5.1.

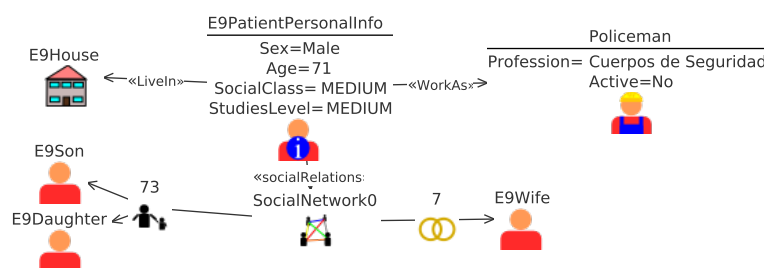


Figura 5.1: Perfil social del paciente E9.

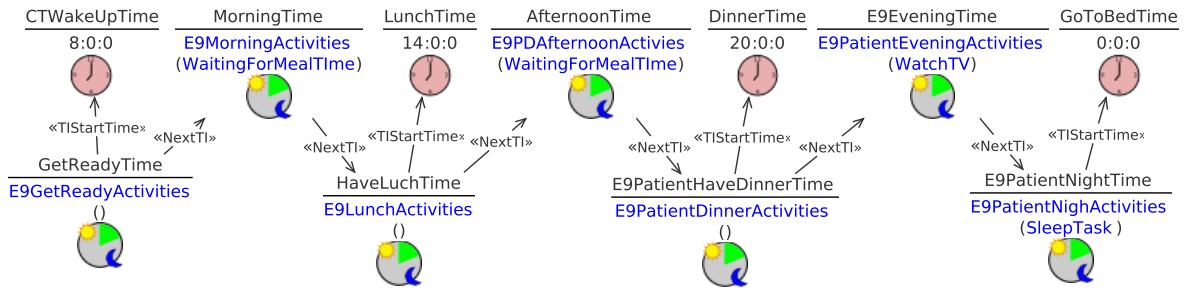
5.1.2. El comportamiento de los sujetos

Las actividades tanto del paciente como las de su compañera se han agrupado de forma muy parecida en el mismo número de periodos del día. Dichos periodos se diferencian, principalmente, por conseguir objetivos menos concretos que las actividades: levantarse y arreglarse, pasar la mañana, comer, pasar la tarde, cenar, pasar la noche y dormir. Destacar que muchos de ellos, la mitad aproximadamente, tienen una hora de inicio asociada, ver Figura 5.2a y 5.2b. En cambio, los que les preceden tienen una actividad por defecto, la cual realizan si ya han terminado todas las actividades y todavía no es el momento de iniciar el siguiente periodo. Por ejemplo, si han terminado las actividades de la mañana y todavía no es la hora de comer (las 14:00), realizan la tarea de esperar a que llegue dicho momento, la cual consiste en leer el periódico. Así, como muestran los diagramas y recoge la entrevista, la pareja se levanta a las 8:00, come a las 14:00, cena a las 20:00 y se acuesta a las 00:00. Fíjese que la mayor diferencia entre ambos es que después de comer, la compañera tiene una hora fija (las 15:00) para iniciar la tarde, la cual corresponde con la actividad de dar un paseo como se verá posteriormente.

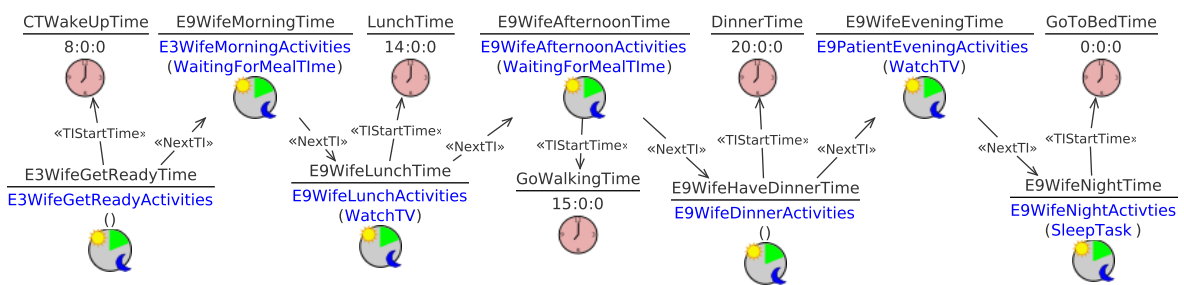
El perfil ADL del voluntario es muy sencillo ya que solo se modelan dos momentos del día: (1) el que corresponde con ir a la vivienda del paciente para acompañarlo en

²Código fuente del proyecto SociAALML: <https://github.com/Grasia/sociaalml>

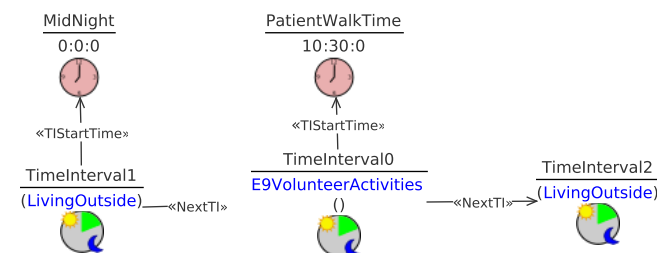
su paseo de las 10:30, y (2) el que equivale al resto del tiempo que no influye en las vidas del matrimonio, ver Figura 5.2c.



(a) ADL del enfermo E9.



(b) ADL de la compañera.



(c) ADL del voluntario.

Figura 5.2: Los ADLs del matrimonio y el voluntario.

Según los diagramas ADLs de las Figuras 5.2 vistas, se han creado 15 diagramas de actividades para definir los momentos del día. Los cuales contienen en total en torno a 70 actividades. Por motivos de espacio, se muestran solo dos de los 15 diagramas de actividades. El primero corresponde a las actividades que realiza la compañera durante la mañana (*E9WifeMorningActivities*) y el segundo a las que realiza el paciente, también por la mañana (*E9MorningActivities*).

La compañera, durante las mañanas, se dedica a realizar tareas domésticas poco a poco tal y como expresa ella misma en el siguiente fragmento de la entrevista:

Fragmento 5.1.1 *CompañeraE9: ... luego yo ya me pongo a hacer mis camas y hacer mis cosas de la casa y ahora ya así me lleva yo ya... -porque lo tengo que hacer muy despacio- me lleva toda la mañana, me lleva toda la mañana porque tengo que hacer*

las cosas muy despacito y a mí me gusta hacer las cosas muy bien, y entonces tengo que hacerlo así.

En la entrevista no se dispone de más detalles más allá de lo que la compañera indica. En este caso, se ha asumido que esta realiza las siguientes actividades: fregar las cosas del desayuno, hacer las camas, y posteriormente, durante dos horas realiza las siguientes actividades de forma iterativa: limpia un rato, luego hace un descanso, tras el mismo puede que vaya al aseo y, después, si no es la hora de ponerse a cocinar (las 12:00) vuelve a seguir limpiando. Se asume también que en función del día se dedica a una actividad doméstica diferente: plancha los lunes, limpia la habitación los martes, limpia el salón los miércoles, limpia la cocina los jueves y limpia el baño los viernes. El fin de semana se asume que descansa de realizar tareas de limpieza y, en su lugar, ve la televisión. La última tarea de la mañana es siempre hacer la comida. El modelo de esta descripción se encuentra en la Figura 5.3. Nótese la complejidad del mismo, por lo que no puede convenir combinar este momento del día con el de comer, pues el diagrama resultaría más complejo todavía.

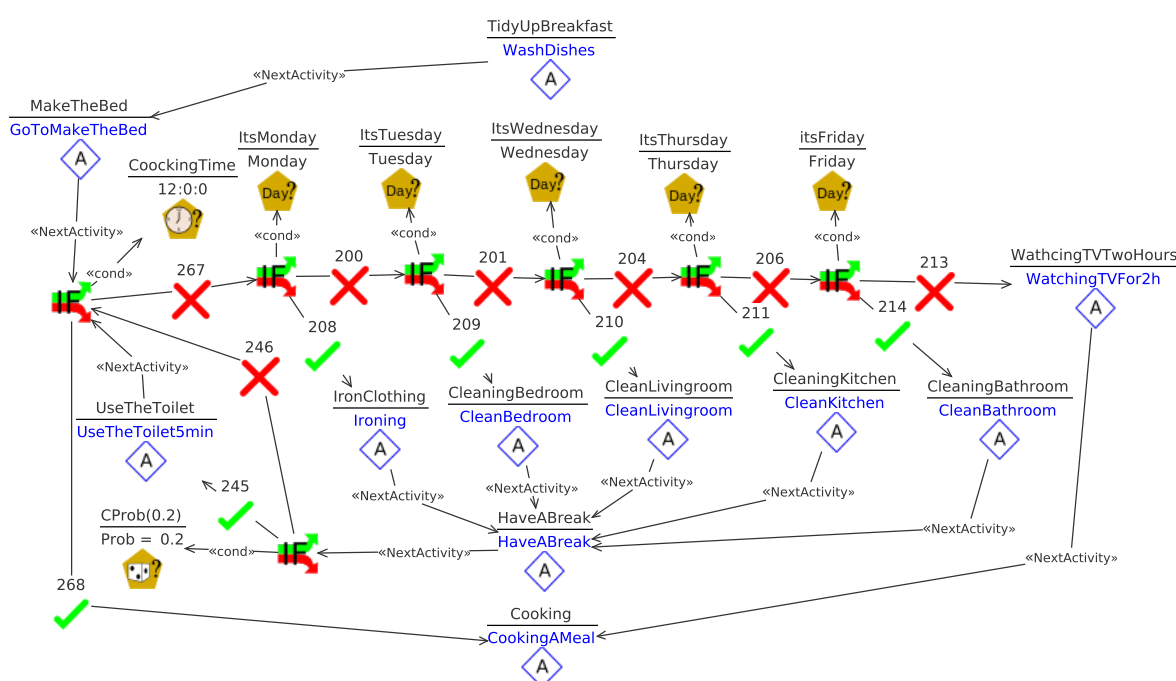


Figura 5.3: Actividades que realiza la compañera del paciente E9 por las mañanas después de asearse y vestirse.

En cambio, el paciente durante las mañanas se dedica a realizar actividades de ocio en el trastero, ya sea tallando la madera o viendo vídeos en internet:

Fragmento 5.1.2 PACIENTE9: *Me entretengo en el trastero haciendo tonterías. Una vez si tengo alguna idea para hacer algún instrumento o alguna cosa de madera pues la hago si no me pongo ahí con lo que pille o a ordenar o a limpiar o a... chorradas para pasar el rato y luego me pongo otro rato en internet y así voy pasando el día.*

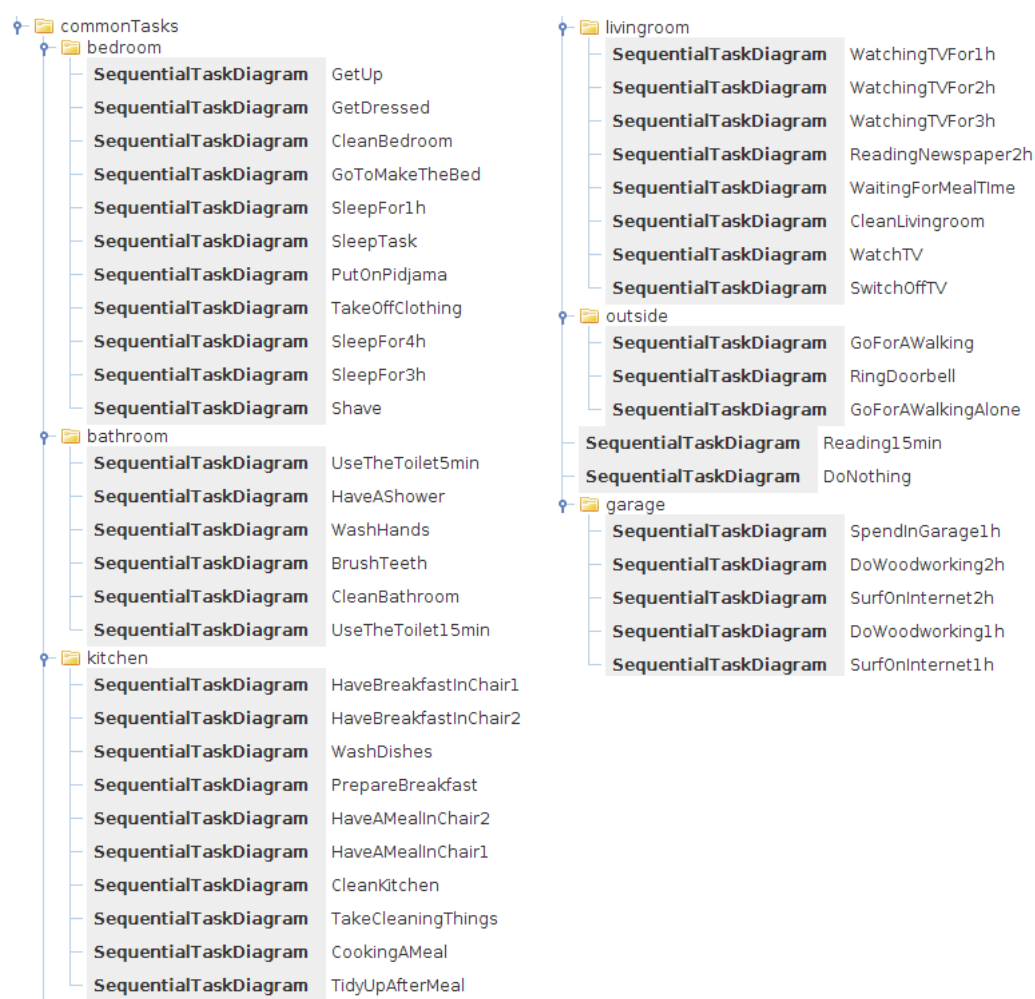


Figura 5.5: Diagramas de secuencia de tareas genéricas organizados por habitación.

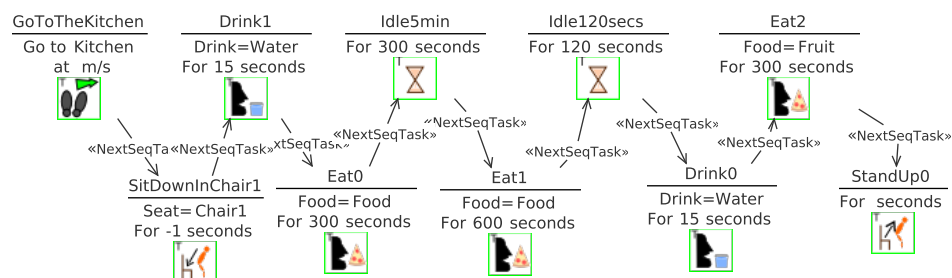


Figura 5.6: Diagramas de secuencia de las tareas que corresponden a realizar una comida sentado en la silla con id *Chair1*.

realiza tareas domésticas como limpiar o cocinar, pero es capaz de llevar a cabo sus actividades básicas lentamente (e.g. bañarse, vestirse o asearse) debido a los temblores. Sufre de depresión por la enfermedad, hecho que se agudiza cuando tiene días malos.

En este apartado se van a mostrar dos limitaciones que manifiesta el paciente, una relacionada con la actividad de vestirse y, la otra, relacionada con las caídas. La primera limitación la expresa el paciente en la entrevista a través del siguiente fragmento:

Fragmento 5.1.4 *PACIENTE9: Me visto solo, tardo mucho, me tengo que sentar en la cama para meterme los pantalones, por ejemplo, porque el equilibrio lo pierdo. No puedo, me cuesta abrocharme los botones de la camisa. Lo hago despacio pero lo voy haciendo... ahí no tengo limitaciones todavía.*

El conjunto de filtros propuesto que modela las afirmaciones anteriores es el que se muestra en la Figura 5.7. La primera línea de filtros establece que todas las tareas del tipo ponerse la ropa se realizan en el doble de tiempo. La segunda cadena de filtros es específica para la tarea de ponerse la camisa, siendo esta especialmente lenta (el cuádruple) debido a la dificultad que le supone al enfermo abrocharse los botones. Obsérvese que al paciente le llevará cuatro veces más de tiempo porque a dicha tarea se le aplicará dos veces un filtro de retraso (*FDelayFilter0* y *FDelayFilter1*). La última cadena detalla la forma en la que se pone los pantalones. Dicha tarea se reemplaza por la secuencia de tareas definidas en el diagrama *PutOnTrousers* que se visualiza en el recuadrado negro. La tarea de ponerse los pantalones se reemplaza por la de sentarse, ponerse los pantalones y levantarse finalmente. Fíjese que el identificador de ponerse los pantalones es diferente, porque si no, se volvería a filtrar y filtrar dicha tarea indefinidamente.

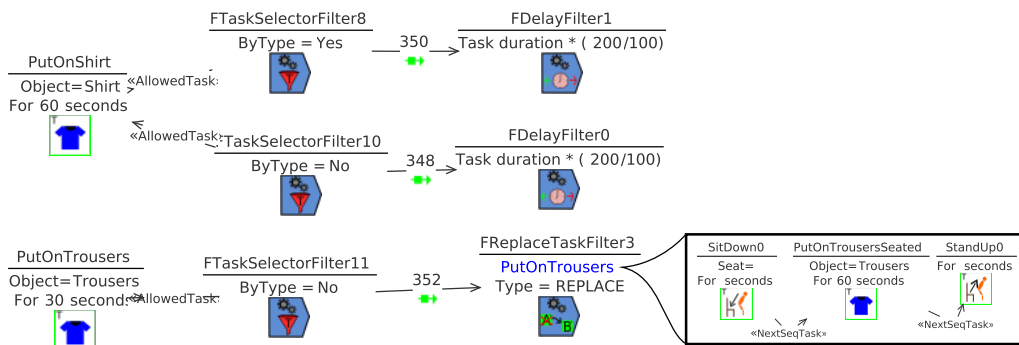


Figura 5.7: Filtros del paciente E9 relacionados con la actividad de vestirse.

La segunda limitación relacionada con las caídas la describe la compañera en el siguiente fragmento:

Fragmento 5.1.5 *CompañeraE9: ...Porque que me hiciera caso un poquito... hay días que se cae dos veces, tres veces... me han llamado que se ha caído en el paso de cebra, aquí en la casa se ha caído muchísimas veces en la calle muchísimas veces y también se ha caído muchas veces en la calle que lo han tenido que llamar ...*

De esta declaración, se extrae que el paciente se cae bastante, tanto fuera de la vivienda como dentro. En este fragmento no se expresa, pero el paciente se desplaza despacio, se asume que es 3 veces más lento que un sujeto sano. No obstante, esta asunción habría que confirmarla. Por tanto, cada vez que el paciente se desplazase, este debería disminuir su velocidad a un tercio. Por otro lado, cada vez que anda existe también la posibilidad de que se caiga. Lo difícil en este caso es establecer qué probabilidad es la adecuada para que el paciente se caiga de una a tres veces al día. Como muestra la Figura 5.8, se estima que dicha probabilidad sea de 0.1. Posteriormente, durante la simulación se puede comprobar cuántas veces el paciente se cae al cabo de un día y reajustar dicho valor, tal y como se verá más adelante. Cuando el paciente se cae, la compañera acude a su encuentro para ver cómo se encuentra, sin embargo, se asume que el paciente se levanta siempre solo tras un tiempo. Como muestra la figura, la tarea de caerse tiene asociado un evento *FallDownEvent*. Tras la caída, el paciente se levanta a los 60 segundos.

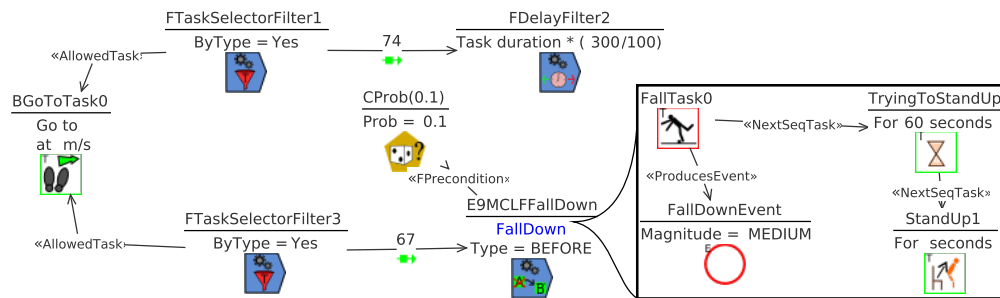


Figura 5.8: Filtros del paciente E9 para modelar velocidad de desplazamiento y las caídas.

La toma de medicamentos y la descripción de la evolución de los síntomas lo describe el paciente a través del siguiente fragmento de la entrevista:

Fragmento 5.1.6 *PACIENTE9: Y ahora en la actualidad estoy tomando una cada cuatro horas, a las 8 a las 12, a las 4. Después a las 8 de la tarde me tomo nada más que media.*

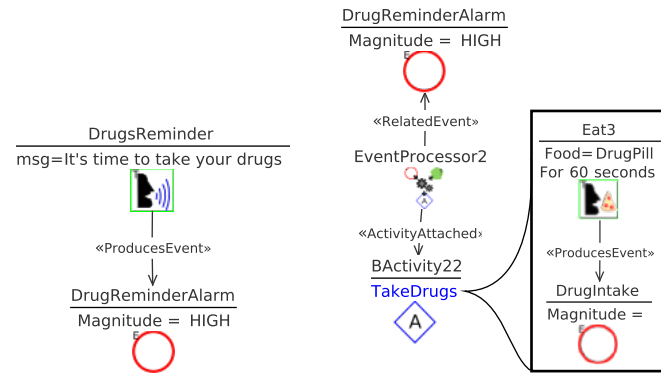
...

Se me olvidan Las tomas estas de... porque llevo el móvil lo tengo en la alarma a la hora que me tengo que tomar las pastillas y me las tomo bien.

...

la pastillas de la tarde la de las 4 me hace efecto desde las 17:30 hasta las 18.30 o siete menos cuarto

De este fragmento se extrae que el paciente toma medicamentos cada 4 horas a partir de las 8:00 y que se le olvida tomarlos, por lo que tiene puesta la alarma del móvil a dichas horas para que este se las recuerde. También especifica cómo evoluciona el efecto de la pastilla que se toma a las 16:00. En cuanto a la evolución de los síntomas, a lo largo de la entrevista, el paciente no diferencia cuándo está mejor de un síntoma



(a) Tarea para recordar al paciente que tome la medicación. (b) Respuesta del paciente ante el recordatorio.

Figura 5.10: Extractos de los diagramas con las entidades para recordar al paciente que debe tomar la medicación y la respuesta del paciente realizando la toma del mismo.

respuesta del paciente ante dicho evento es tomar la medicación de forma que lanza el evento *DrugIntake* relacionado con la evolución de los síntomas.

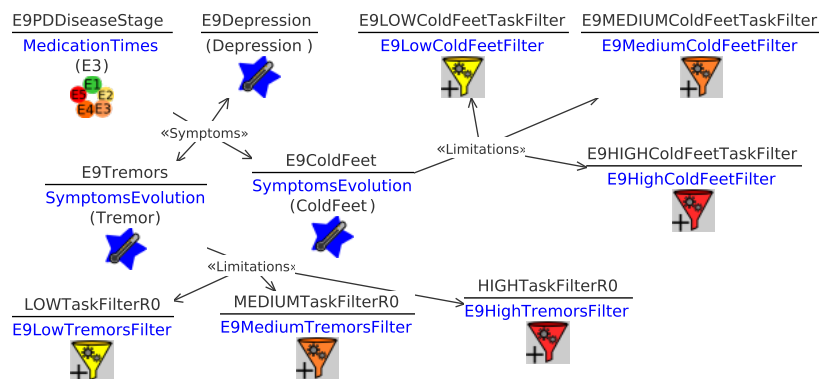


Figura 5.11: Diagrama del perfil de enfermedad de Parkinson del sujeto E9.

Resumiendo, el paciente E9 sufre de tres síntomas principalmente, depresión, temblores y pies congelados. Los efectos de la depresión no se modelan y algunas de las consecuencias de los otros dos síntomas han sido ya explicadas. En la Figura 5.11 se muestra el perfil de la enfermedad del paciente. Fíjese que el síntoma de temblores (Tremor) y el de pies congelados (ColdFeet) tienen asociados 3 tipos de filtros cada uno, los cuales corresponden a las variaciones del comportamiento según el nivel del síntoma. Como se ha comentado anteriormente, se diferencian entre nivel bajo, medio y alto. Los filtros en cada nivel son prácticamente los mismos pero variando los efectos y las probabilidades con las que ocurren. Por ejemplo, cuanto mayor es el nivel del síntoma de pies congelados mayor es la probabilidad de que el paciente se caiga.

5.1.4. Validación del modelo

La validación del modelo consiste en comprobar que la simulación se comporta como esperamos. Para ello, es necesario identificar aquellos hechos que se consideren más importantes comprobar. Estos variarán dependiendo del escenario o caso de uso a tratar. En nuestro caso, la situación más importante y preocupante para los enfermos y sus familiares es la de las caídas. Comprobar que el paciente se toma los medicamentos es otro hecho importante que influirá en los niveles de sus síntomas y en la probabilidad de que este se caiga. Cuando el paciente se cae es importante que sea atendido, en este caso por su compañera. Otros hechos a comprobar que se consideran interesantes para este caso son el número de veces que los sujetos acuden al aseo y las visitas del voluntario. En el primer hecho se considera relevante para dar realismo a la simulación. El segundo segundo hecho también se va a comprobar porque es una actividad destacada que rompe la rutina de la pareja. Además, es la única actividad que realiza el voluntario y con esta comprobación se tendría validado el comportamiento del voluntario. Por tanto, los hechos que se desean comprobar a fin de validar la simulación generada son:

- **H1.** *El número de veces que el paciente se cae a lo largo del día.*
- **H2.** *Las horas a las que se toma el paciente los medicamentos y el número de veces.*
- **H3.** *El número de veces que la compañera va a atender al paciente como consecuencia de una caída.*
- **H4.** *El número de veces que van al aseo cada uno.*
- **H5.** *Acude el voluntario a por el paciente los lunes y los miércoles para dar el paseo de forma puntual.*

Para generar las simulaciones es necesario especificar un diagrama de simulación donde se configure la misma, es decir, se indiquen los sujetos que intervendrán, su posición inicial, el nivel de los síntomas, fecha y hora iniciales, entre otros datos. En un caso grande como este donde se simulan varios días, según la experiencia obtenida, conviene ir incrementando cada vez los escenarios en términos de número de sujetos y tiempo simulado. Así, por ejemplo, se define una simulación donde solo se incluye al paciente, otra donde solo se incluye a la compañera y otra donde solo se incluye al voluntario. No obstante, como los comportamientos dependen unos de otros estas simulaciones no son muy representativas. El siguiente paso es definir simulaciones donde se incluyen a todos los sujetos, pero donde varía el día de inicio de la simulación, por tanto, se definen 7 diagramas de simulación, uno por cada día de la semana. Se asume que el día empieza a las ocho de la mañana, momento que coincide con la hora de levantarse. La Figura 5.12 corresponde al diagrama de simulación para el lunes. Fíjese cómo el voluntario no tiene una posición inicial ya que este se encuentra fuera del escenario inicialmente. También se ha activado que se muestre el identificador de cada

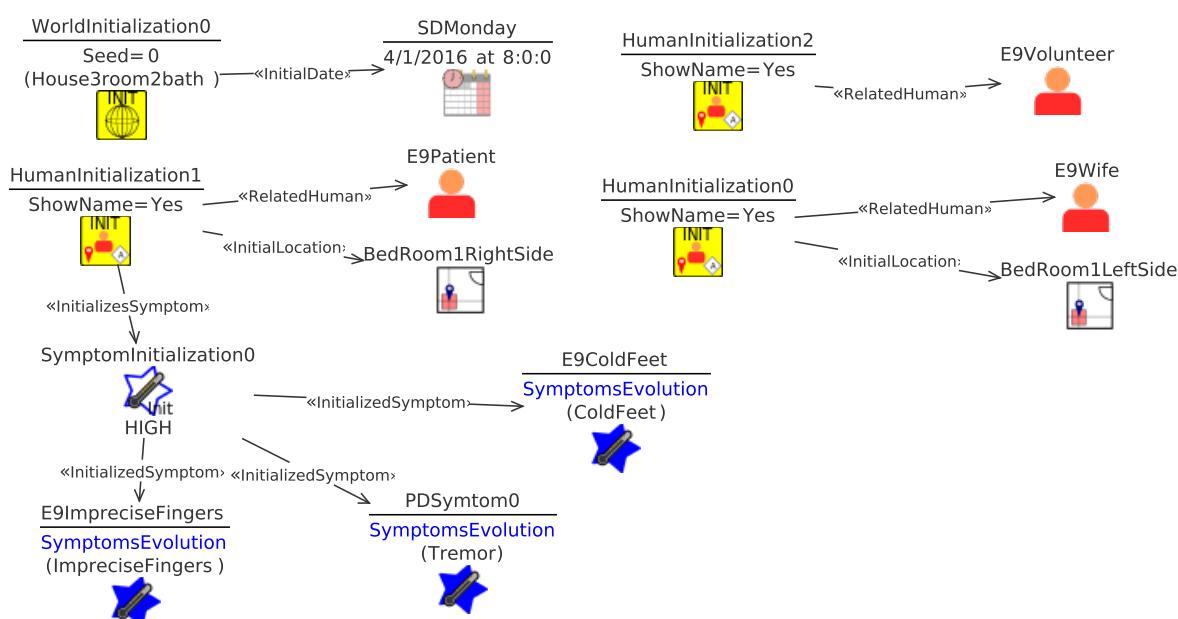


Figura 5.12: Diagrama de simulación donde participan todos los sujetos y la simulación se inicia el lunes 4 de enero a las 8 de la mañana.

sujeto con el fin identificar a los sujetos virtuales más fácilmente durante la visualización de la simulación.

Para comprobar que se cumplen los hechos anteriores se ha simulado el escenario definido en la Figura 5.12 hasta que transcurriese una semana en la simulación. La simulación se ha realizado con avance rápido (x128), mostrando la hora simulada y el log. Simultáneamente, se ha realizado una grabación de la misma para poder consultarla tantas veces como sean necesarias sin tener que volver a realizar la simulación. Por último, se ha guardado el log generado por PHAT para su análisis. Tanto el vídeo como el log están disponibles en la web³.

Para demostrar los hechos identificados anteriormente, se ha empleado en primera instancia el log para extraer las acciones de interés y la simulación para aclarar situaciones no esperadas. La Tabla 5.1 recoge todos los datos obtenidos del log por día, indicando la hora simulada en la que se produjo. Para obtener la información se ha hecho uso de los comandos *cat* y *grep* del bash de Linux. A continuación se irá mostrando, para cada hecho, la consulta empleada para extraer los datos relevantes del log y una discusión de los mismos.

H1. El número de caídas del paciente se obtiene consultando el número de veces que se inicia la tarea *FallTask0*:

```
cat AWeek.log | grep E9Patient | grep STARTED | grep FallTask0 | grep 01/04/2016
```

Modificando la fecha, se obtienen las líneas del log correspondientes al inicio de la caída para la fecha indicada. Obsérvese que el número de caídas por día es proba-

³Sitio web donde se aloja el vídeo de la simulación y el log del caso de estudio 1: <http://grasia.fdi.ucm.es/aide/demos/2016/02/23/simulatingweekparkinson.html>

Hecho a comprobar	L	M	Mi.	J	V	S	D
H1. Núm. caídas	2	3	3	5	4	2	2
	11:52:54	14:32:18	10:38:44	00:01:25	08:36:55	09:11:33	08:23:06
	14:32:29	15:41:55	11:41:12	00:15:19	08:57:32	15:40:19	10:18:29
		16:55:56	13:18:20	08:37:09	09:20:25		
				17:04:02	20:39:30		
				20:01:00			
H2. Horas toma medicamentos	08:00:03	00:04:59	00:05:00	00:07:55	00:04:59	00:04:58	00:04:51
	13:02:39	08:05:02	08:05:07	08:05:04	08:05:09	08:05:04	08:05:07
	14:05:40	12:11:13	12:10:38	12:19:23	12:34:37	11:28:13	11:32:22
	20:05:45	14:05:45	14:05:42	14:05:45	14:05:46	14:05:39	14:05:40
		16:06:51	16:06:50	16:06:51	16:06:52	16:06:56	16:06:59
		20:05:50	20:05:45	?	20:05:43	20:05:48	20:06:26
H3. Núm. veces esposa asiste paciente	2	2	3	5	4	1	0
	11:53:12	14:32:26	10:40:43	00:01:34	08:37:04	09:11:42	08:23:14
	14:32:38	16:56:04	11:41:21	00:15:27	08:57:40		10:18:38
			13:18:28	08:37:18	09:20:34		
				17:04:11	20:39:39		
				20:01:09			
H4.Paciente. Núm. visitas al aseo	7	8	10	9	9	10	9
	08:11:48	00:01:40	00:01:47	00:03:05	00:01:39	00:01:40	00:01:48
	11:41:45	00:06:21	00:06:22	00:09:18	00:06:23	00:06:21	00:06:14
	12:56:20	08:11:27	03:39:33	03:43:48	03:39:23	03:39:22	03:39:33
	13:07:06	10:13:38	08:11:31	08:11:30	08:11:33	08:11:28	08:11:31
	16:43:56	11:26:37	11:43:48	10:16:09	10:37:36	10:15:05	10:20:11
	16:54:42	16:45:03	12:56:37	11:29:04	11:50:25	11:27:48	11:34:36
	18:07:18	16:57:36	13:07:23	16:42:19	12:01:13	11:29:36	11:45:22
		18:10:14	16:42:27	16:53:05	13:13:45	11:38:36	12:59:08
			16:53:15	18:07:17	16:42:18	12:51:18	16:45:32
			18:05:53			16:43:36	
H4.Compañera. Núm. visitas al aseo	2	5	4	4	3	3	4
	08:18:25	00:15:21	00:15:22	00:20:04	00:15:24	00:15:19	00:15:16
	19:11:57	03:43:27	08:23:44	03:48:15	08:23:40	08:23:39	03:43:26
		08:23:39	11:03:20	08:23:39	19:11:50	19:11:51	08:25:31
		11:03:48	19:11:48	19:12:25			19:11:55
		19:11:49					
H5. Hora que llega el voluntario a la casa	10:32:45	0	10:32:46	0	0	0	0

Tabla 5.1: Tabla resumen con los hechos a verificar acontecidos cada día en la simulación a lo largo de una semana.

blemente elevando, especialmente el jueves. Sería interesante comprobar el nivel del síntoma cuando se produjo la caída, pero dicha información no está disponible. Como trabajo futuro en la mejora del log, debería registrarse esta información.

H2. Las horas a las que el paciente se toma la medicación se obtienen consultando cuándo se inicia la tarea *Eat3* que corresponde a la tarea de tomar la pastilla:

```
cat AWeek.log | grep E9Patient | grep STARTED | grep Eat3 | grep 01/04/2016
```

Si recordamos, el paciente debería tomar la medicación entorno a las 00:00, 8:00, 16:00 y 20:00. Sin embargo, se aprecian dos resultados no esperados. El primero consiste en que el paciente está ingiriendo una toma de más (en torno a las 14:00). Esto se debe ha un error en el diseño del modelo con SociAALML. La compañera es la que avisa al paciente de las tomas. Antes de cenar (a las 20:00), esta debe recordarle una de ellas. Para ello, en la secuencia de tareas para comer se ha añadido la tarea de recordar, pero dicha secuencia de actividades se utiliza tanto en la actividad para la comida como en la cena. Por tanto, se identifica que sería necesario sacar dicha acción de la secuencia de tareas. Si se observa, el segundo resultado no esperado es que el paciente no se tomó la pastilla de la hora de la cena el jueves. Observando el vídeo (a partir del instante 43:07) y el log, se puede comprobar que hay una caída justo en el momento en el que finaliza la tarea previa a la del recordatorio. Resulta extraño que se interrumpa la secuencia de tareas de recordar la medicación (*BSequentialTask9*), cuando no aparece en el log que esta se haya iniciado. Al no haber sido iniciada, es decir, no se han añadido las subtareas esta finaliza tras comprobar que no hay tarea por defecto (instante de

simulación 302567). Por tanto, se ha detectado un error de implementación de PHAT cuando se interrumpen secuencias de tareas justo antes de su iniciación.

H3. Los instantes en que la esposa va a asistir al paciente se inician con la actividad *AssistPatientFalling*:

```
cat AWeek.log | grep E9Wife | grep STARTED | grep AssistPatientFalling | grep 01/04/2016
```

Estos instantes deberían coincidir con el número de caídas, pero no ocurre así en dos ocasiones: el martes a las 15:41:55 y el sábado a las 15:40:19. Comprobando el vídeo de la simulación en dichos instantes (0:16:25 y 1:05:23 respectivamente) se comprueba que la compañera se encuentra fuera de la vivienda en su paseo diario de 15:00 a 16:00. Por tanto, el comportamiento de la simulación es correcto.

H4. La tarea que corresponde al uso del WC es *UseWC5min*, la cual comparten ambos sujetos a través de la secuencia de tareas *UseTheToilet5min*. Por tanto, la consulta es la misma para ambos cambiando el id del sujeto. Por ejemplo, para el paciente, el número de veces que acude al aseo el lunes día 4 es:

```
cat AWeek.log | grep E9Patient | grep STARTED | grep UseWC5min | grep 01/04/2016
```

Si se observa la Tabla 5.1, los datos registrados de la compañera parecen razonables y los del paciente muy elevados, produciéndose algunos de ellos muy próximos. El más llamativo se produce de forma regular antes de ir a la cama, entre las 00:01:00 y las 00:07:00. Si se analiza el resto de la tabla se puede comprobar que ese momento coincide con la toma de los medicamentos. No parece muy adecuado que el paciente interrumpa dicha tarea para la toma de los medicamentos. Comprobando el modelo de SociAALML se confirma que la tarea de usar el aseo no está marcada como que no se puede interrumpir. También se extrae de esta consulta que resulta más conveniente consultar el estado de finalización (FINISHED) ya que una misma actividad o tarea puede pasar más de una vez por el estado de STARTED si esta es interrumpida y reanudada. Por tanto, en este caso, no se obtiene el comportamiento esperado en los sujetos virtuales debido a errores en el modelado.

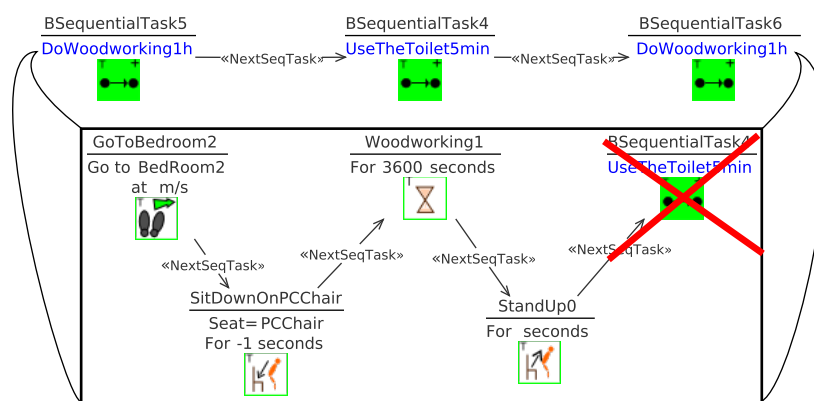


Figura 5.13: Error al modelar la tarea *DoWoodworking1h* al incluir la tarea de ir al aseo.

Otro patrón no deseado es el que ocurre de lunes a jueves, ya que acude al aseo dos veces muy seguidas, entorno a las 16:43:00 y las 16:54:00. Revisando el modelo

SociAALML correspondiente al comportamiento del paciente, se descubre un error en la definición de la actividad de tallar la madera durante dos horas. Esta se define como tallar la madera durante una hora, luego ir al aseo y, posteriormente, vuelve a tallar la madera durante otra hora. El error está en que se incluyó la acción de ir al aseo dentro de la tarea de tallar la madera, por lo que va dos veces seguidas tal y como muestra la Figura 5.13.

H5. La hora a la que llega el voluntario a la vivienda del paciente se podría considerar el momento en el que este toca el timbre, es decir, la tarea *BUseTask5*:

```
cat AWeek.log | grep E9Volunteer | grep STARTED | grep BUseTask5
```

En este caso, no es necesario especificar fecha pues son pocas las salidas que se deben obtener. Además, así es posible asegurarse de que el voluntario no haya visitado al paciente en otro momento que no correspondiese. Observando la Tabla 5.1 se comprueba que este solo acude los lunes y miércoles en torno a las 10:32:45. Por lo que el comportamiento es el que se esperaba.

5.2. Caso de estudio 2: Ver la televisión

En este caso de estudio se trata un escenario que corresponde a una actividad concreta a la que podrían enfrentarse muchos de los pacientes entrevistados. A diferencia del caso anterior, este será mucho más corto pero se analizarán más en profundidad posibles problemas que puedan surgir y se indicará cómo podría ser una posible solución AAL a dichos problemas identificados. Este caso es una extensión del trabajo realizado en la demostración [27] presentada en el AAMAS 2015.

En primer lugar, se describirá el caso de estudio y, posteriormente, se mostrará el modelado de los escenarios, centrándose en aquellos diagramas más relevantes como la secuencia de tareas para ver la televisión y los filtros que modelan las variaciones del comportamiento debido a los síntomas. Una vez se dispone de unos escenarios donde se producen situaciones no deseadas por el paciente y se han validado, se procede a diseñar una solución (prototipo del sistema AAL) que alivie o resuelva una de estas situaciones. Por último, se amplían los escenarios con el despliegue de los dispositivos y el sistema AAL donde se puede volver a simular los escenarios y comprobar cómo se comporta el prototipo en el mundo virtual.

5.2.1. Descripción

La mayor preocupación de los familiares y de los cuidadores son las numerosas caídas que sufren los pacientes de párkinson. A parte de que el paciente sufra algún daño, puede que este no sea capaz de volver a ponerse de pie e incluso cause que su pareja caiga con él cuando esta trata de ayudarlo. En el mejor de los casos, el paciente simplemente se queda de pie parado sin poder seguir andando durante un tiempo. Un caso interesante es el de la paciente E12, una mujer de 64 años que afirma que cuando sufre de bloqueos, el escuchar música le ayuda a reiniciar la marcha. Otro problema

común al que se enfrentan los enfermos es el de pulsar teclas pequeñas como las de los mandos de la televisión debido al agarrotamiento de los dedos.

En este apartado se va a abordar un escenario donde pueden surgir todos estos problemas. Dicho escenario corresponde a la actividad de ver la televisión. Una descripción del mismo es la siguiente:

El paciente se encuentra en la cocina y no tiene un buen día. Se podría decir que tiene un nivel medio en los síntomas de pies congelados (lo que hace que no se siente seguro al andar) y en el de temblores y agarrotamiento. Debido a su estado, decide ir a ver la televisión al salón. En dicho estado, el paciente se desplaza más despacio y durante el recorrido puede ocurrir que se caiga o se quede bloqueado. Una vez llega al salón, coge el mando de la televisión y se sienta. Entonces, intenta pulsar el botón de encendido, este le resulta más fácil porque se encuentra en un extremo del mando y no tiene casi botones a su alrededor. Lo siguiente que quiere el paciente es pulsar el botón número 5 para ver su canal favorito. No obstante, debido a los temblores y a la rigidez de los dedos pulsa varios botones a la vez u otro que no es el que quiere.

5.2.2. Modelado de los escenarios

La primera tarea consiste en modelar esta situación problemática. Para ello, se empieza por modelar el comportamiento del sujeto sin incidentes. La actividad de ver la televisión consiste en ir al salón, coger el mando, sentarse en el sofá, encender la televisión, poner el canal favorito (el número 5), dejar el mando en la mesa, ver la televisión durante media hora y, finalmente, volver a coger el mando para después apagarla. Esta secuencia de tareas se encuentra especificada en el diagrama de la Figura 5.14.

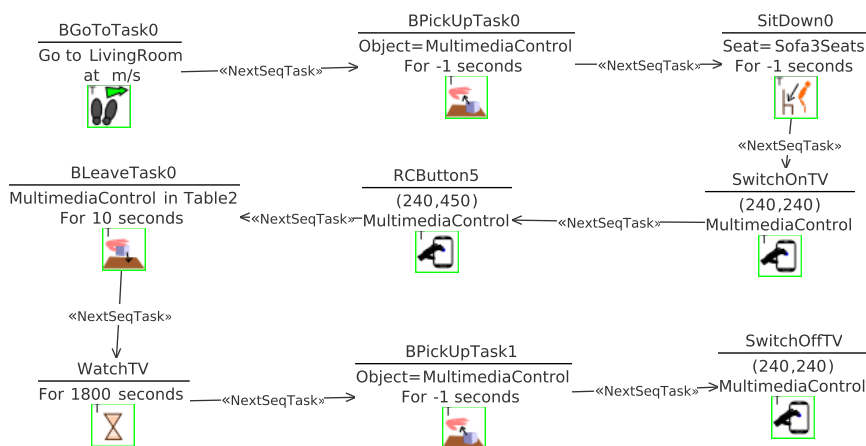


Figura 5.14: Secuencia de tareas que definen la actividad de ver la televisión.

Una vez modeladas las tareas que componen la actividad de ver la televisión, el siguiente paso consiste en modelar los problemas que pueda sufrir el paciente a través

el 50 % de las veces pulsará otro botón (el 8) y luego vuelve a intentar pulsar el botón 5. La Figura 5.17 muestra la especificación que corresponde a estas afirmaciones.

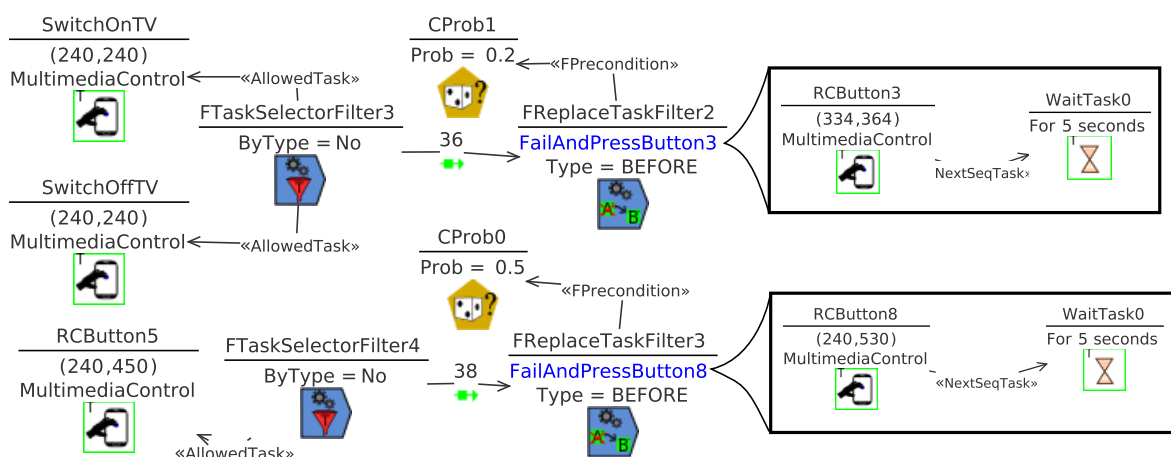


Figura 5.17: Diagrama de filtros que alteran el comportamiento del paciente cuando el nivel del síntoma de temblores es medio.

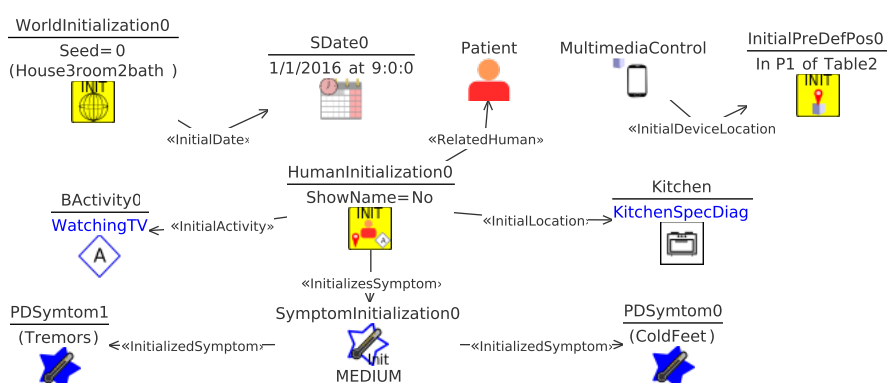


Figura 5.18: Diagrama inicial de simulación del escenario con el paciente con niveles medios en sus síntomas de pies congelados y temblores.

Por último, es necesario crear, al menos, un diagrama de simulación que defina una configuración inicial del escenario. En este caso, solo se simula un sujeto virtual, el paciente, cuya posición inicial será la cocina. Este sufre de nivel medio en los síntomas de pies congelados y temblores. El mando a distancia se encuentra en la mesa del salón. La fecha y la hora inicial no es relevante, supóngase que son, por ejemplo, las 9:00 de la mañana. Se empleará la vivienda “House3room2bath”. El diagrama con esta configuración se encuentra en la Figura 5.18. Fíjese, que al tratarse de un escenario donde el sujeto solo realiza una actividad, esta se especifica en el diagrama de simulación. De esta forma, no es necesario crear el perfil ADL, ni el diagrama de momentos del día y ni el de actividades. Destacar que la mesa del salón corresponde a la que tiene el identificador “Table2” y la posición “P1” es una de las cuatro de las que se dispone.

Esto se puede averiguar fácilmente gracias a las capturas de pantalla de la Figura 4.21a. Debido a las condiciones de probabilidad empleadas en los filtros, no se sabe a priori cuáles de ellos se aplicarán. El único parámetro que afecta a dichas probabilidades es el de la semilla, cambios en su valor puede afectar a dichas condiciones de los filtros y que se produzcan escenarios diferentes. Por ello, para identificarlos y validarlos, se crean varios diagramas de simulación donde solo se altera el valor de la semilla. Posteriormente, tras simularlos, se comprueba qué situaciones se han producido, es decir, qué filtros se han activado.

PHAT Log viewer					
SimTime	Name	State	Action	Type	FinishCondition
0	Patient	START	SimpleADL	ADLSpecDiagram	
0	Patient	START	SimpleADL	ADLSpecDiagram	
0	Patient	START	WatchingTV	SequentialTaskDiagram	
0	Patient	START	BGoToTask0	BGoToTask	
21	Patient	FINISH	BGoToTask0	BGoToTask	
21	Patient	START	BPickUpTask0	BPickUpTask	
27	Patient	FINISH	BPickUpTask0	BPickUpTask	
27	Patient	START	SitDown0	SitDown	
37	Patient	FINISH	SitDown0	SitDown	
37	Patient	START	SwitchOnTV	TapXYTask	TimerFinishedCondition(1,1)
39	Patient	FINISH	SwitchOnTV	TapXYTask	TimerFinishedCondition(1,1)
39	Patient	INTERRUPT	RCButton5	TapXYTask	
39	Patient	START	FailPressingButton5	SequentialTaskDiagram	
39	Patient	START	RCButton8	TapXYTask	Time FinishedCondition(1,1)
40	Patient	FINISH	RCButton8	TapXYTask	Time FinishedCondition(1,1)
40	Patient	START	WaitTask0	WaitTask	Time FinishedCondition(5,5)
45	Patient	FINISH	WaitTask0	WaitTask	Time FinishedCondition(5,5)
45	Patient	INTERRUPT	RCButton5	TapXYTask	
45	Patient	START	FailPressingButton5	SequentialTaskDiagram	
45	Patient	START	RCButton8	TapXYTask	Time FinishedCondition(1,1)
46	Patient	FINISH	RCButton8	TapXYTask	Time FinishedCondition(1,1)
46	Patient	START	WaitTask0	WaitTask	Time FinishedCondition(5,5)
51	Patient	FINISH	WaitTask0	WaitTask	Time FinishedCondition(5,5)
51	Patient	START	RCButton5	TapXYTask	Time FinishedCondition(1,1)
52	Patient	FINISH	RCButton5	TapXYTask	Time FinishedCondition(1,1)
52	Patient	FINISH	FailPressingButton5	SequentialTaskDiagram	
52	Patient	FINISH	FailPressingButton5	SequentialTaskDiagram	
52	Patient	START	BLeaveTask0	BLeaveTask	TimerFinishedCondition(10,10)
62	Patient	FINISH	BLeaveTask0	BLeaveTask	TimerFinishedCondition(10,10)
62	Patient	START	WatchTV	WaitTask	TimerFinishedCondition(1800,1801)
1863	Patient	FINISH	WatchTV	WaitTask	TimerFinishedCondition(1800,1801)
1863	Patient	START	BPickUpTask1	BPickUpTask	
1868	Patient	FINISH	BPickUpTask1	BPickUpTask	
1868	Patient	START	SwitchOffTV	TapXYTask	TimerFinishedCondition(1,2)
1871	Patient	FINISH	SwitchOffTV	TapXYTask	TimerFinishedCondition(1,2)
1873	Patient	FINISH	WatchingTV	SequentialTaskDiagram	
1875	Patient	FINISH	BActivity0	BActivity	
1875	Patient	START	BActivity1	BActivity	
1875	Patient	START	WatchingTV	SequentialTaskDiagram	

Figura 5.19: Captura de pantalla correspondiente al log del escenario con valor de semilla 2.

Para este caso se han realizado simulaciones con valores de semilla desde 0 hasta 9 y se ha anotado en una tabla el número de veces que se ha aplicado cada filtro para cada valor de la semilla. Visualizando las simulaciones 3D se puede llegar a averiguar qué filtros son los que se han disparado. Sin embargo, como ya se vio en el caso anterior, la forma más efectiva de realizar esta identificación es a través del log. Por ejemplo, en la Figura 5.19 se muestra una captura de pantalla del log para la simulación con semilla 2. En esta se puede comprobar cómo la tarea de pulsar el botón 5 (*RCButton5*) se interrumpe dos veces y se reemplaza por la secuencia de tareas *FailPressingButton5* que incluye las tareas: pulsar el botón 8 (*RCButton8*), esperar 5 segundos (*WaitTask0*)

y volver a pulsar el botón 5 (*RCButton5*). En la figura se resalta el anidamiento de las tareas debido a los filtros. En el instante de simulación 39, la tarea *RCButton5* se reemplaza por la secuencia *FailPressingButton5* que a su vez se vuelve a interrumpir por el mismo tipo de tarea (*RCButton5*). En el instante 52 se puede comprobar el anidamiento, ya que tras finalizar la última secuencia, también lo hace la primera. La tabla que resume los disparos de los filtros por cada valor de semilla corresponde a la Tabla 5.2. Esta refleja las siguientes situaciones:

- Semilla 0, 4 y 9: no se activa ningún filtro y el paciente realiza la actividad normalmente.
- Semilla 1: el paciente se cae.
- Semilla 2: falla dos veces a la hora de pulsar el botón 5.
- Semilla 3: falla una vez a la hora de pulsar el botón de encendido y, posteriormente, el de apagado.
- Semilla 5: el paciente no puede andar y luego se cae.
- Semilla 6: falla una vez a la hora de pulsar el botón 5.
- Semilla 7 y 8: falla tres veces a la hora de pulsar el botón 5.

Se puede comprobar que para 10 valores diferentes de semilla, se producen 7 situaciones diferentes. Estas situaciones identificadas, las debería validar el paciente, sus familiares y/o un experto. Y si no fuesen verosímiles, sería necesario realizar alguna modificación en el modelo. En este caso, se asume que son válidas y suficientes. Por tanto, se pueden crear 7 diagramas de simulación iguales, donde solo se diferencien entre ellos en la entidad que contiene el valor de la semilla. Esta replicación de entidades en varios diagramas no es adecuada ya que, entre otras cosas, dificulta el mantenimiento de los modelos. Por ello, se considera conveniente rediseñar esta parte del lenguaje para una gestión más adecuada de las semillas. En cualquier caso, una buena práctica sería, por cada diagrama, anotar en el campo de descripción algo que sea representativo de lo que con dicha configuración sucede en la simulación. Una vez se dispone del conjunto de escenarios problemáticos, el siguiente paso consiste en el diseño de una solución, tal y como se verá en el apartado siguiente.

5.2.3. Diseño de la solución y evaluación

El fin de este apartado es ilustrar el concepto de uso de la plataforma y su potencial utilidad como entorno de validación de diseños AAL. Para ello, de los problemas que le pueden surgir al paciente identificados en el apartado anterior, se va a poner el foco en uno de ellos para diseñar una aplicación sencilla que lo solvete. El problema seleccionado es aquel donde el paciente tiene dificultades para pulsar el botón número 5 del mando a distancia.

Semilla	No anda	Cae	Botón On	Botón 5	Botón Off
0	0	0	0	0	0
1	0	1	0	0	0
2	0	0	0	2	0
3	0	0	1	0	1
4	0	0	0	0	0
5	1	1	0	0	0
6	0	0	0	1	0
7	0	0	0	3	0
8	0	0	0	3	0
9	0	0	0	0	0

Tabla 5.2: Tabla resumen que contabiliza las veces que se producen las diferentes situaciones en función del valor de la semilla. “No anda” corresponde al número de bloqueos donde el paciente no puede andar. “Cae” indica el número de veces que el paciente se cae. “Botón On”, “Botón 5” y “Botón Off” registran el número de veces que el paciente falla a la hora de pulsar el botón correspondiente.

En un escenario sin el sistema AAL, cuando el paciente no puede presionar el botón 5, suele pedir a su pareja (el cuidador) que realice dicha tarea. Dependiendo de qué actividad se encuentre realizando el cuidador, a este le supondrá más o menos molestia atenderle. La solución AAL debe tener como objetivo favorecer la autonomía del paciente y no depender del cuidador. Pero, por otro lado, en este tipo de enfermedades, es conveniente que los enfermos se esfuercen por realizar ellos mismos las tareas y no deleguen a la primera oportunidad. Por ello, una solución en la que el mando se anticipa al paciente encendiéndose en su canal favorito, podría no ser adecuada. En su lugar, se va a desarrollar una solución donde el paciente pueda interaccionar de dos formas con el mando a distancia: pulsando los botones de forma tradicional o mediante comandos de voz.

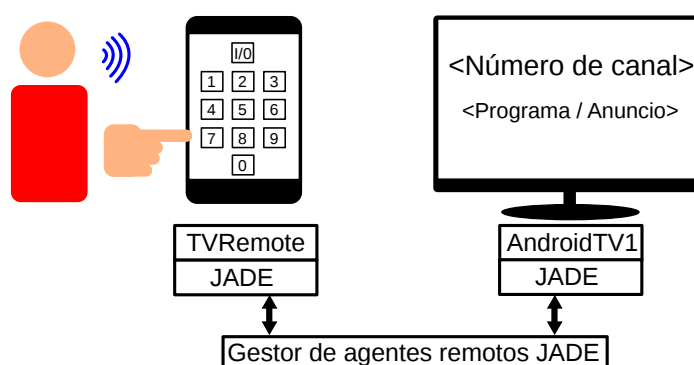


Figura 5.20: Esquema del diseño de la solución AAL para dotar al paciente de más autonomía.

Por tanto, como se vio en el apartado 2.1, un sistema AAL basado en AmI consta del usuario, los dispositivos, el middleware y los servicios y aplicaciones. En la Figura 5.20 se puede ver un esquema del sistema diseñado. Los dispositivos que intervienen son el

mando a distancia y la televisión, ambos dispositivos dotados con el sistema operativo Android. El tipo de dispositivo elegido, aunque lo imponga las limitaciones de nuestra plataforma, se considera totalmente plausible. Existen en el mercado varias marcas de televisiones con este sistema operativo instalado y el mando a distancia podría ser el propio teléfono del paciente. Las aplicaciones son dos, una para el teléfono inteligente y otra para la televisión. El teléfono inteligente tiene instalada una aplicación (TVRemote) a través de la cual el paciente puede acceder a los servicios de la televisión. Como se ha establecido, esta interacción puede ser pulsando en la pantalla sobre el botón del canal deseado o través de comandos de voz. La televisión lleva instalada una aplicación (AndroidTV1) cuya finalidad es doble. Por un lado, emular los canales y la programación y, por el otro, ofrecer los servicios para cambiar de canal. Ambos prototipos se desarrollan basados en JADE⁴, un framework que facilita el desarrollo de aplicaciones basadas en agentes. Se ha empleado este framework porque los autores tienen experiencia en su uso y facilita la comunicación entre aplicaciones (agentes) y la compartición de servicios. Este framework podría hacer las veces de middleware AAL. No se ha empleado un middleware AAL para el caso de estudio porque son elementos complejos que necesitan una fase de aprendizaje previa considerable. Además, para este caso tan sencillo el middleware AAL no aportaría nada relevante.

Véase más en detalle cómo se han desarrollado los prototipos. Por un lado, como se ha avanzado, el prototipo de la televisión emula la propia televisión, ofreciendo 10 canales (del 0 al 9) con una programación y espacios publicitarios. Además, dispone de servicios para encenderse, apagarse y cambiar a un número de canal dado. Estos servicios se publican en el gestor de agentes remotos JADE para que la aplicación TVRemote del mando a distancia pueda hacer uso de los mismos. Por otro lado, el prototipo del mando a distancia ofrece una interfaz gráfica que incluye el botón de encender y apagar, y los botones del 0 al 9 para cambiar de canal. Dependiendo del botón que se pulse, se lanza la petición correspondiente a la televisión. Además de la interfaz gráfica, al prototipo se le ha añadido una interfaz de voz utilizando la librería de reconocimiento de voz CMU Sphinx⁵. Esta es la interfaz que se quiere evaluar y validar. La interacción con el mando a distancia a través de la interfaz de voz es sencilla. Primero, se dice la palabra clave “oh mighty computer” y, posteriormente, el número del canal que se desea, en este caso, el 5 (“five”). Al reconocer el número, el prototipo realiza exactamente la misma acción que si el sujeto pulsase el botón 5. Recordar que es necesario compilar las aplicaciones con la librería jALI para que estas se puedan comunicar con la simulación.

Una vez implementados los prototipos, es necesario definir un diagrama de simulación donde se especifique el despliegue de la solución. Para ello, se parte del diagrama de Figura 5.18, pero con valor de semilla 2, pues es cuando se produce la situación objetivo a resolver: el paciente no es capaz de pulsar el botón 5. A este diagrama es necesario añadirle el despliegue de la solución. Este consiste en dos dispositivos: *Smartphone1* y *AndroidTV1*. El primero corresponde al teléfono personal del paciente, su posición

⁴Sitio web de JADE: <http://jade.tilab.com/>

⁵Sitio web de CMU Sphinx: <http://cmusphinx.sourceforge.net/>

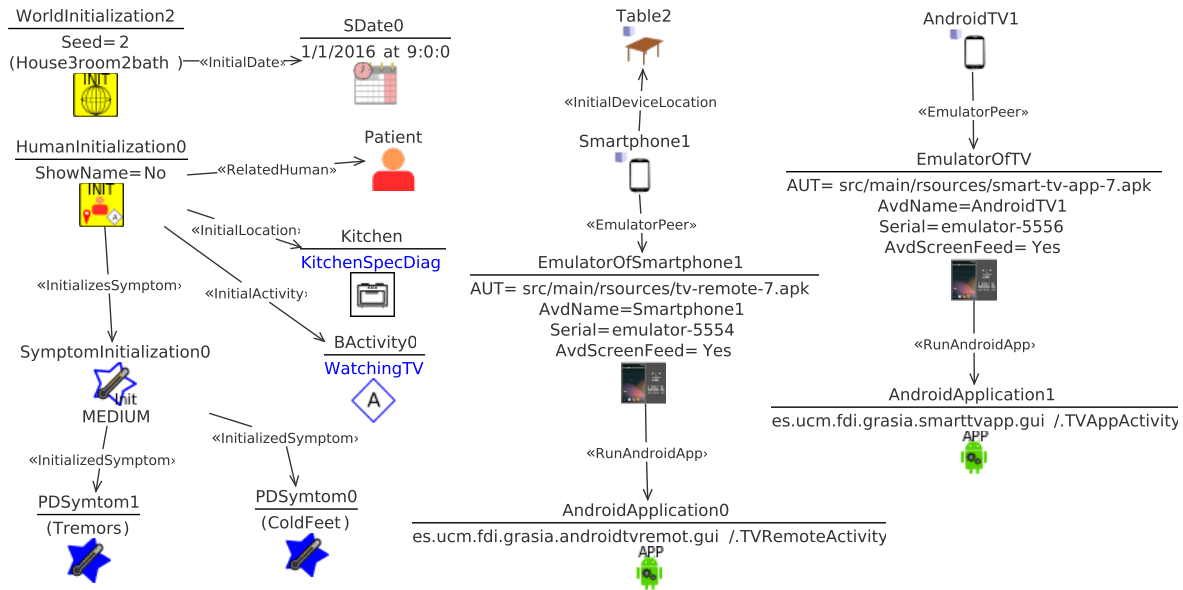


Figura 5.21: Diagrama de simulación para evaluar el prototipo que asiste al paciente para cambiar de canal mediante comandos de voz.

inicial es la mesa del salón (*Table2*), se instalará la aplicación de control remoto *tv-remot-7.apk*, su emulador equivalente será el *emulator-5554* y arrancará la aplicación de mando a distancia (con nombre de Activity “TVRemoteActivity”). El otro dispositivo corresponde a la televisión y su posición no se especifica porque viene ya definida en el escenario. Su emulador será el *emulator-5556* donde se instalará la aplicación que emula la programación (“smart-tv-app-7.apk”) y se iniciará con la simulación llamando al Activity “TVAppActivity”. El diagrama resultante se muestra en la Figura 5.21.

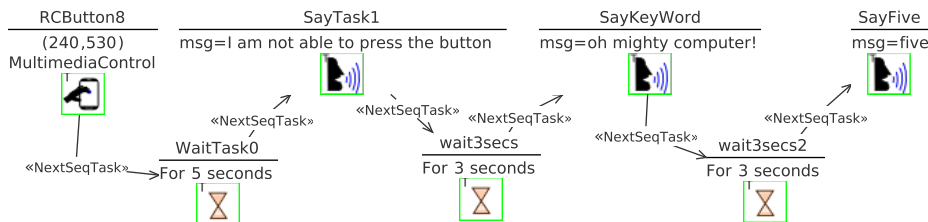


Figura 5.22: Diagrama con la secuencia de tareas para simular el fallo al pulsar el botón 5 (pulsando el 8), su frustración, y su respuesta realizando la misma tarea pero con la voz.

Una vez definida la simulación, es necesario incluir las tareas de voz para interactuar con el prototipo. Con la solución AAL diseñada, la intención es que el paciente, tras fracasar al intentar pulsar el botón 5 y pulse el 8 por error, en vez de volver a intentar pulsar de nuevo el botón 5, utilice la interfaz de voz. Esto supone que diga las palabras claves “oh mighty computer” y, a continuación, el número 5 (“five”). Por ello, hay que cambiar el filtro de forma que sea de tipo reemplazo y realice las tareas especificadas,

ver Figura 5.22. Nótese que esto último es un inconveniente, pues hay que cambiar la definición del filtro dependiendo de si se ejecuta una simulación u otra. Un trabajo futuro debe incluir solventar este inconveniente. En cualquier caso, se puede comprobar ejecutando el escenario con las soluciones diseñadas que estas consiguen su objetivo: ayudar al paciente a cambiar de canal. En la Figura 5.23, se muestra una captura de pantalla del vídeo del escenario definido. El vídeo se puede visualizar en la web⁶. Este contiene la simulación 3D completa del escenario definido y la respuesta del sistema AAL. La figura capta el momento justo después de que el paciente haya dicho el número 5, solicitando el teléfono a la televisión que cambie a dicho canal. La figura muestra (1) los dos emuladores (la televisión y el teléfono), (2) el gestor de agentes JADE donde se comprueba que hay registrados dos agentes, uno por cada aplicación, (3) la ventana del simulador donde se muestra la televisión en el canal 5 y al paciente en el sofá y, por último, (4) el log de PHAT donde se puede comprobar los últimos comportamientos realizados por el sujeto virtual, entre ellos, decir el número 5 (*SayFive*).

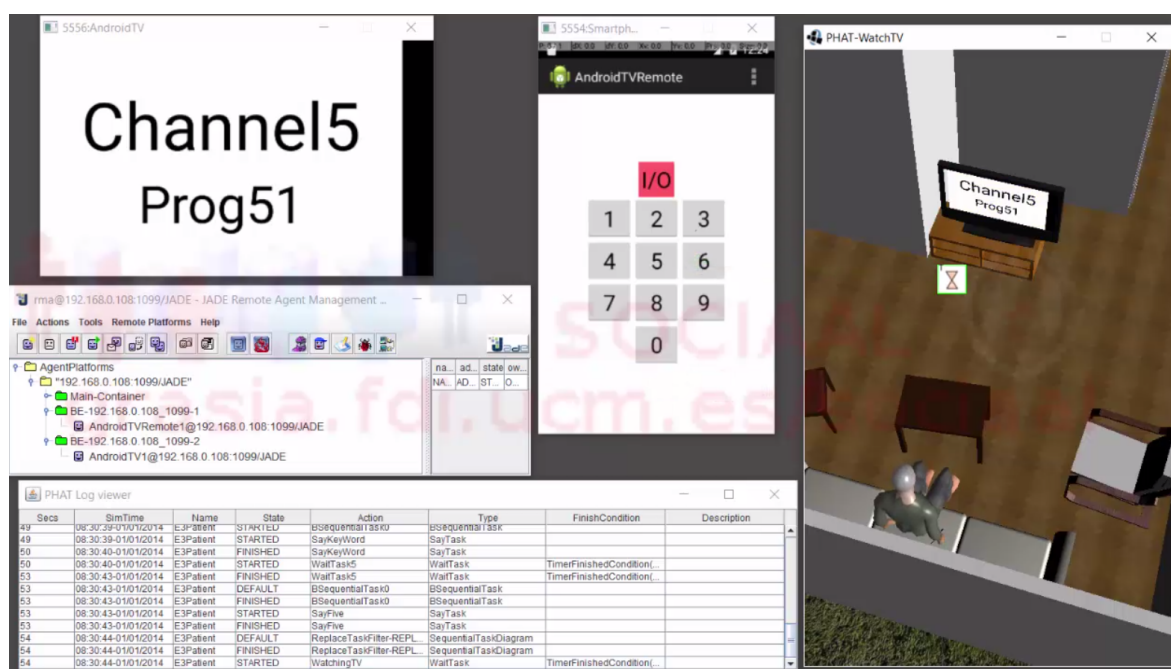


Figura 5.23: Captura de pantalla justo después de que el paciente diga “five” (cinco).

Este caso de estudio nos demuestra cómo es posible, a partir de PHAT y SociAALML, definir escenarios 3D para modelar y validar situaciones problemáticas. Además, ilustra cómo estas especificaciones sirven como casos de uso para la validación de los diseños del sistema AAL.

⁶Url del vídeo del caso de estudio 2: <http://grasia.fdi.ucm.es/aide/demos/2016/03/16/interactingtvtusingvoicecommands.html>

5.3. Valoración crítica de la plataforma a partir de los casos de estudio

A través de los dos casos de estudio presentados, se ha mostrado cómo se pueden modelar los distintos aspectos de los escenarios (principalmente los comportamientos de los sujetos) con el lenguaje específico del dominio SociAALML. Además, se han dado unas herramientas y unas pautas para la validación de dichas especificaciones a través de la simulación. En este apartado se analizarán las estadísticas aplicadas tanto a las especificaciones creadas como al código Java generado para cada caso de uso.

5.3.1. Análisis de los modelos

La finalidad de este análisis es comprobar la cantidad de diagramas y de entidades necesarias para crear los modelos para cada caso de estudio. Además, se comprueba el ratio de entidades únicas por diagrama. Es decir, la relación entre el número total de entidades y aquellas que son únicas. Si este número es elevado, significa que contiene muchas entidades que se encuentran en otros diagramas. Esto, según el caso, puede ser un indicador positivo o negativo. Por ejemplo, en los diagramas de tareas, puede indicar que se produce reutilización de las tareas. Pero, en otros casos como se verá más adelante, puede ser un indicio de que un diagrama repite muchos conceptos que podrían agruparse en un diagrama nuevo. Por último, se muestra el número medio de entidades por diagrama para comprobar que se gestiona la complejidad de acuerdo con los principios de Moody [96].

Por tanto, para cada caso de estudio se mostrarán las estadísticas en forma de tabla. Cada tabla mostrará, por cada tipo de diagrama: (1) el número de diagramas de dicho tipo; (2) el número de entidades que contiene el diagrama (además, del ratio); y (3) la media del número de entidades por diagrama.

5.3.1.1. Caso de estudio 1

En el primer caso de estudio se ha modelado la entrevista al paciente E9, la cual, transcrita, consta de 13.600 palabras en 29 páginas con tipografía Times New Roman y un tamaño de 11 puntos. A partir de este documento ligeramente estructurado, pero con muchos aspectos diseminados a lo largo del mismo, y escrito en lenguaje coloquial, se ha mostrado cómo se puede capturar dicha información en el lenguaje específico de dominio SociAALML.

En la Tabla 5.3 se encuentran los datos descritos previamente y obtenidos del modelo para el primer caso de estudio. De su análisis se extraen las siguientes conclusiones:

- Como ya se avanzó en el caso de uso, una parte importante del esfuerzo de modelado va destinado a modelar detalles del comportamiento de los sujetos. Fíjese que aproximadamente la mitad del total de los diagramas creados corresponden a diagramas de secuencia de tareas. Los cuales contienen algo más de un tercio del

Tipo diagrama	Num.	Num. total de entidades /únicas(ratio)	Media
ADLSpecDiagram	3	28/24(1.2)	9.3
ActivityDiagram	15	111/77(1.4)	7.4
BuildingDiagram	1	2/2(1.0)	2.0
DeviceDiagram	1	4/4(1.0)	4.0
FilterDiagram	6	54/26(2.1)	9.0
FloorSpecDiagram	1	8/8(1.0)	8.0
HumanProfileSpecDiagram	3	12/12(1.0)	4.0
InteractionDiagram	2	9/9(1.0)	4.5
MedicationTimeDiagram	1	10/10(1.0)	10.0
ParkinsonSpecDiagram	1	10/10(1.0)	10.0
RoomSpecDiagram	6	28/28(1.0)	4.7
SequentialTaskDiagram	62	249/129(1.9)	4.0
SimulationDiagram	9	112/20(5.6)	12.4
SocialSpecDiagram	3	16/13(1.2)	5.3
SymptomEvolutionDiagram	1	10/10(1.0)	10.0
TOTAL	115	663/366(1.8)	5.8

Tabla 5.3: Tabla del caso de estudio 1 con estadísticas donde por cada tipo de diagrama se muestra: número de diagramas del tipo, número de entidades totales / no repetidas (ratio), media de aparición de entidades por diagramas.

total de las entidades. Disponer de un repositorio de secuencias de tareas básicas ahorraría una parte considerable de trabajo de modelado.

- Los diagramas resultantes cumplen con el principio de gestión de la complejidad de Moody [96] ya que el número medio de entidades por diagrama no es elevado, siendo la media de 5, el mínimo 2 y el máximo 10.
- Si se atiende al ratio de número de entidades totales entre aquellas únicas, se puede comprobar que destaca ampliamente aquel que corresponde a los diagramas de simulación (**SimulationDiagram**). Esto reafirma el problema de diseño detectado en el Capítulo 3 dedicado al lenguaje. Los diagramas de simulación necesitan ser fragmentados en diferentes aspectos de inicialización: entorno, sujetos y despliegue de dispositivos.
- Se manifiesta escasa interacción entre los sujetos virtuales en general, dos diagramas y nueve entidades en total. Esto se debe a que, al igual que en la transcripción de las entrevistas no detallan las actividades, tampoco se detallan las interacciones o eventualidades que les surgen a los sujetos en la vivienda. Por ello, el modelo recoge pocas interacciones.
- El número de diagramas (6) que modelan las limitaciones de la enfermedad (**FilterDiagram**) también parece reducido. Recuérdese que se han modelado dos síntomas: los temblores y el de pies congelados, y el de depresión se ha omitido por falta de información. De nuevo, se refleja la necesidad de realizar una nueva entrevista con los pacientes para recabar más información y completar los modelos.

5.3.1.2. Caso de estudio 2

El segundo caso de estudio se centraba en las dificultades que podía encontrar el paciente en la actividad de ver la televisión. En este escenario, solo se consideraba al paciente, por lo que no era necesario modelar otros sujetos, ni sus interacciones.

Tipo diagrama	Num.	Num. total de entidades / únicas(ratio)	Media
BuildingDiagram	1	2/2(1.0)	2.0
DeviceDiagram	1	6/6(1.0)	6.0
FilterDiagram	2	17/15(1.1)	8.5
FloorSpecDiagram	1	7/7(1.0)	7.0
HumanProfileSpecDiagram	1	3/3(1.0)	3.0
ParkinsonSpecDiagram	1	5/5(1.0)	5.0
RoomSpecDiagram	5	18/18(1.0)	3.6
SequentialTaskDiagram	6	22/19(1.2)	3.7
SimulationDiagram	11	126/27(4.7)	11.5
SocialSpecDiagram	1	2/2(1.0)	2.0
TOTAL	30	208/93(2.2)	6.9

Tabla 5.4: Tabla del caso de estudio 2 con estadísticas donde por cada tipo de diagrama se muestra: número de diagramas del tipo, número de entidades totales / no repetidas (ratio), media de aparición de entidades por diagramas.

En la Tabla 5.4 se encuentran los datos descritos previamente para el modelo del segundo caso de estudio. De su análisis, y en comparación con los datos del primer caso de estudio, se extraen las siguientes conclusiones:

- En este caso, el esfuerzo ha sido mucho menor. El número total de diagramas ha sido 30. Pero si se tiene en cuenta que aquellos relativos al entorno y a los dispositivos (BuildingDiagram, FloorSpecDiagram, RoomSpecDiagram y DeviceDiagram) ya estaban creados (8 diagramas), y que 10 diagramas de simulación de los 11 creados, son iguales variando la semilla (9 diagramas no costaron nada), el número efectivo de diagramas que se tuvieron que crear fue solo de 13.
- El lenguaje resulta adecuado para escenarios pequeños y requiere de menos diagramas. Por ejemplo, cuando solo es necesario modelar una actividad para un sujeto, no es necesario crear los diagramas ADLSpecDiagram y ActivityDiagram. Además, si no es necesario modelar las interacciones porque solo se simula un sujeto, ni las horas de tomar la medicación, ni la evolución de los síntomas pues el escenario es corto en el tiempo, entonces se pueden omitir aun más diagramas como es el caso.
- Al igual que con el primer caso de estudio, los diagramas resultantes cumplen con el principio de gestión de la complejidad de Moody [96]. El número medio de entidades por diagrama no es elevado, siendo la media de 6.9, el mínimo 2 y el máximo 11.5.
- Si se atiende al ratio de número de entidades totales entre aquellas únicas, hay dos aspectos reseñables. El primero es el reducido número de diagramas creados no da oportunidad de reutilizar entidades en otros diagramas, por lo que en general el

ratio es próximo a uno. El segundo aspecto tiene que ver con el elevado ratio para los diagramas de simulación (próximo a 5). De nuevo, se manifiesta la necesidad de rediseñar los diagramas de simulación.

5.3.2. Análisis del código generado

Por un lado, nos interesa comprobar la cantidad de código generado y, por el otro, su calidad. La cantidad de código generado nos dará una estimación del esfuerzo que supondría su codificación a mano. Así, si se compara con el tiempo que supuso la creación del modelo, se puede estimar la ventaja en cuanto a productividad que supone nuestra propuesta. La calidad del código generado es importante pero tampoco tanto, ya que no es necesario el mantenimiento del mismo. En el enfoque propuesto dirigido por modelos, el mantenimiento se realiza a nivel de modelos. Para analizar el código, se hace uso otra vez de SonarQube.

5.3.2.1. Caso de estudio 1

La Figura 5.24 contiene una captura del informe general creado por SonarQube. Lo primero que se puede comprobar es que el código generado pasa los umbrales de calidad (“Quality Gate Passed”). Los errores y las vulnerabilidades detectadas en el código son cero (es decir, tiene fiabilidad y seguridad). Por otro lado, al igual que se vio en los proyectos de jDOS, jALI y PHAT, hay presentes varias violaciones (code smells). Estas tienen una deuda técnica de 22 días, pero como ya se ha avanzado, esto no es real. La métrica hace la estimación asumiendo que la solución se realiza modificando el código generado. Sin embargo, esto no es realmente así, sino que solo hay que cambiar las plantillas y el generador. Un solo cambio en una plantilla podría resolver de un plumazo gran cantidad de violaciones en el código generado. La siguiente métrica está relacionada con las duplicidades de código. Si se observa la figura, esta refleja que el 40.1 % del código es duplicado. Esto no resulta extraño debido a que a la hora de generar el código no resulta costoso duplicar líneas. No obstante, se considera que esta duplicidad debe ser reducida y, por supuesto, tenerse en cuenta para estimar el esfuerzo del código generado. Por tanto, si el número de líneas de código generadas han sido 7700, tras descontar las duplicidades, estas se quedarían en 4620 líneas. La codificación a mano de esta cantidad de líneas de código supone una cantidad considerable de esfuerzo. SonarCube asume que el tiempo que conlleva generar una línea de código es de 0.06 días. Por tanto, la estimación para generar el código de la simulación sería de 277 días. No obstante, se considera dicha estimación muy superior a la que realmente sería por dos motivos. El primero se debe a que el código a desarrollar no es complejo, ya que la mayor parte consiste en extender y combinar clases ya creadas. Y el segundo se debe a que la estimación de 0.06 días asume el coste de la documentación, pruebas unitarias, entre otras prácticas que se podrían eludir en cierto grado. De todos modos, sí se considera que el tiempo para conseguir desarrollar a mano el código generado sería mucho mayor de una semana. Una semana fue el tiempo aproximado en el que se creo el modelo para el paciente E9. Por tanto, además de permitir a investigadores generar

simulaciones sin conocimientos en programación, el proceso de creación de simulaciones es más productivo y fiable.

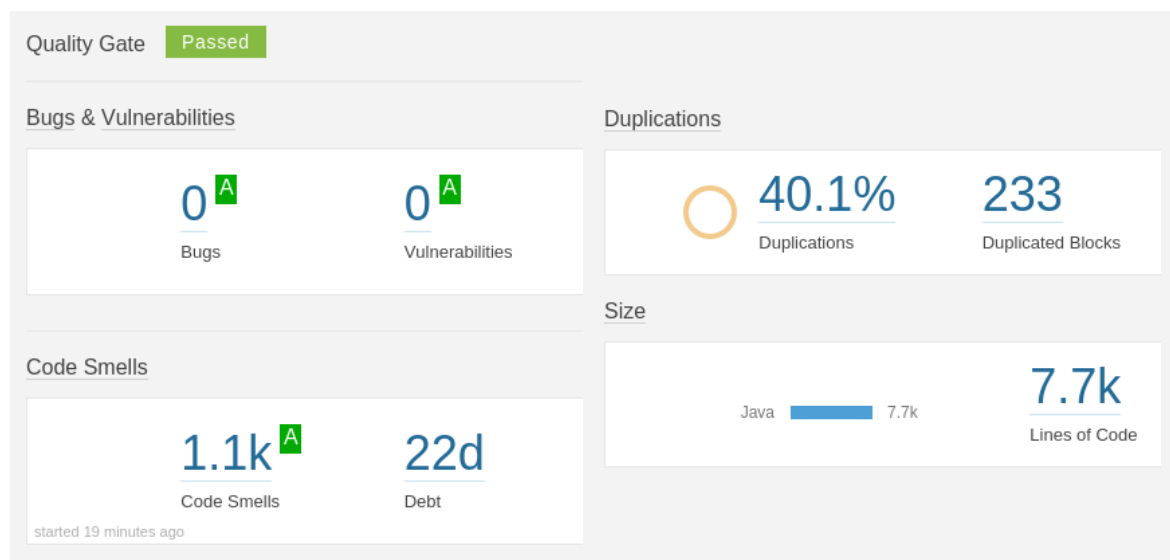


Figura 5.24: Resultado del análisis de SonarQube aplicado al código generado a partir del modelo creado para la entrevista al paciente E9.

5.3.2.2. Caso de estudio 2

Las estadísticas generadas por SonarQube para el segundo caso de estudio se encuentran en la Figura 5.25. Al igual que en el primer caso, el código generado pasa los umbrales de calidad, no contiene ni errores ni vulnerabilidades. Cabe esperar que haya una proporción de estos resultados con aquellos obtenidos en el primer caso de estudio. Esta proporción debe depender del número total de entidades creadas en SociAALML para cada caso de estudio. Esta es de 663 frente a 208, es decir, el segundo caso de estudio contiene 3.19 veces menos entidades. La cantidad de código generado en términos de líneas de código es 3.35 veces menos que las generadas para el primer caso de uso. Sin embargo, el número de violaciones es 4 veces menos (1100 frente a 369). Esto se tiene que deber a que, en proporción, en el segundo caso se han empleado menos plantillas que contienen más violaciones. Lo mismo sucede con las duplicidades de código. En el primer caso hubo 40.1 % de código repetido y en el segundo el 46.7 %. Por tanto, se puede afirmar que la proporción se mantiene.

En cuanto a la ventaja productiva que supone el lenguaje de modelado para este caso de uso se considera menor frente a la del anterior caso. Cuanto más complejo sea el caso, es decir, más sujetos, más comportamientos, síntomas y limitaciones, más valioso se vuelve el lenguaje para manejar dicha complejidad. En el segundo caso de estudio solo se dispone de un sujeto (el paciente) y una actividad (ver la televisión). Solo se definen dos síntomas y no se modela su evolución. Tampoco se crea el perfil de interacciones. Según la estimación de SonarQube y suprimiendo las líneas duplicadas, el esfuerzo que supone

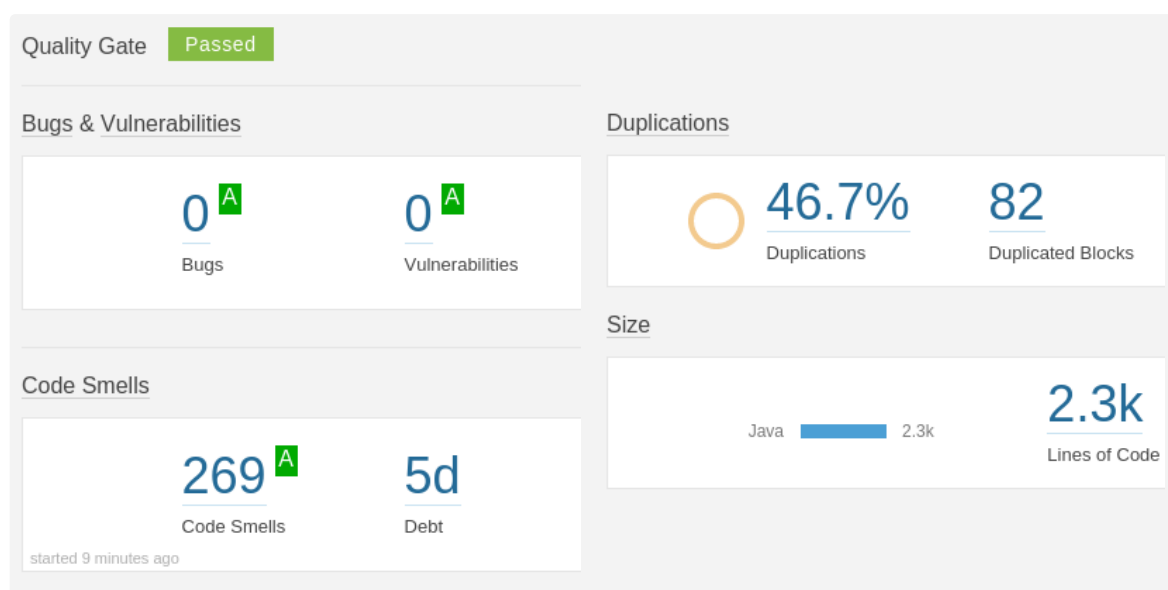


Figura 5.25: Resultado del análisis de SonarQube aplicado al código generado a partir del modelo creado para el caso de uso de ver la televisión.

generar este código es de 73 días y medio (1226 líneas de código x 0.06 días). Como ya se argumentó para el caso anterior esta estimación es bastante elevada. No obstante, seguirá siendo mucho más costoso escribir el código que modelarlo con SociAALML. Para cuantificar con mayor exactitud esta productividad se deja como trabajo futuro realizar un experimento donde se escoja a programadores no familiarizados ni con la API de PHAT ni con el lenguaje SociAALML y se mida el tiempo empleado para crear un escenario con cada aproximación.

5.4. Uso de datos de sensores de la simulación

Las aplicaciones que se pretenden evaluar en el simulador, perciben el entorno simulado a través de los sensores simulados a los que se conectan las aplicaciones compiladas con la librería jALI e instaladas en el emulador. Una aplicación actúa en función de los valores recibidos por los sensores y se persigue que estos valores generados en la simulación sean lo suficientemente cercanos a los reales. El mundo real debería replicar la configuración que se haya simulado para que el desarrollo de una aplicación sea posible. Si un sujeto virtual ejecuta un movimiento y la aplicación reacciona ante ese movimiento, en el mundo real se esperará que, ante un movimiento similar, se obtenga una reacción similar de la aplicación. Esto no quiere decir que a la inversa sea necesariamente cierto. La variabilidad del mundo real (variaciones en los movimientos de la persona por particularidades de la fisionomía del sujeto o por la interferencia debido a movimientos de otras partes del cuerpo) combinado con la casuística del hardware (ruidos en las señales, propiedades específicas de un fabricante, o parecidos), hace que simular todos estos factores no sea trivial, aunque no imposible.

De cualquier forma, esta cuestión requiere un trabajo adicional que se contemplará dentro del trabajo futuro de la tesis.

Esta tesis se ha centrado en dispositivos Android por varias razones. Actualmente existen gran variedad de dispositivos Android: teléfonos, televisiones, relojes, mini-pcs, gafas, coches, entre otros. Además, los autores tienen una experiencia previa en conectar aplicaciones Android a UbikSim, Android ofrece un emulador de código libre que nos permitiría en un futuro modificarlo y hacer totalmente transparente la inmersión de las aplicaciones en la simulación, y por último, Android ofrece herramientas que facilitan que los sujetos virtuales interactúen con el emulador (e.g. simular pulsaciones en la pantalla). No obstante, se considera interesante ampliar el tipo de dispositivo que se puede conectar al simulador. Por ejemplo, incluir emuladores de microcontroladores, como por ejemplo Arduino⁷, o máquinas virtuales Linux.

Esta sección trata de validar los sensores virtuales que ofrece PHAT (acelerómetro, cámara y micrófono), los cuales ya fueron descritos y se justificó su utilidad en el apartado 4.6. Como en todo proceso de este tipo, las fases son: (1) definición de la simulación en SociAALML o mediante programación, (2) generación de datos reales sobre el caso que se quiere simular, (3) desarrollo de la simulación, (4) recogida y análisis de datos de la simulación, (5) evaluación y conclusiones.

Por tanto, por cada tipo de sensor, se diseñan escenarios específicos para ser reproducidos tanto en el mundo real como en el virtual. Durante el experimento se registran los datos generados por los sensores. Para ello, se ha desarrollado una aplicación que registra dichos valores. Este dispone de un menú para seleccionar el tipo de sensor a registrar y un botón para iniciar el almacenamiento de los datos en un fichero. Dicho fichero almacena el instante de la lectura y los valores registrados dependiendo del sensor seleccionado. Una vez se obtienen de los datos del dispositivo real y del simulado, estos se procesan y comparan para ver la verosimilitud como se verá en los próximos subapartados. De forma adicional, para el caso del micrófono y la cámara, también se han comparado las respuestas de aplicaciones que hacen uso de dichos sensores. El dispositivo real empleado para todos los experimentos fue un LG Nexus 4.

5.4.1. Efectividad de la simulación de un acelerómetro

El acelerómetro es un instrumento para medir aceleraciones, es decir, variaciones de velocidad. El acelerómetro simulado es aquel que asume Android que va instalado en sus dispositivos. Este es un acelerómetro de tres ejes (x,y,z), con el sistema de coordenadas que muestra la Figura 5.26. Cada acelerómetro simulado está accesible a través de un servidor socket con el protocolo TCP. Al cual se conecta la aplicación compilada con jALI para obtener el flujo de aceleraciones en x, y, z con el intervalo de tiempo entre medida y medida. Como se vio en el apartado 4.6, el acelerómetro es un instrumento muy apropiado para detectar las temidas caídas [74, 18, 67, 81].

Para comparar las aceleraciones generadas por el dispositivo real y el simulado, se ha elegido el gesto de beber. Esto se debe a que tras visualizar el repertorio de

⁷Sitio web de Arduino: <https://www.arduino.cc/>

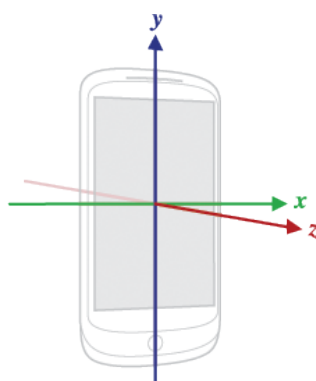


Figura 5.26: Sistema de coordenadas de los acelerómetros en los dispositivos Android.

animaciones disponibles (ver Figura 4.9b), esta es la que nos ha resultado ser una de las más naturales. Se debe tener en cuenta que las aceleraciones que se obtienen de la simulación dependen totalmente del realismo de las animaciones del sujeto virtual. No obstante, conseguir dicho realismo no es objetivo de esta tesis.

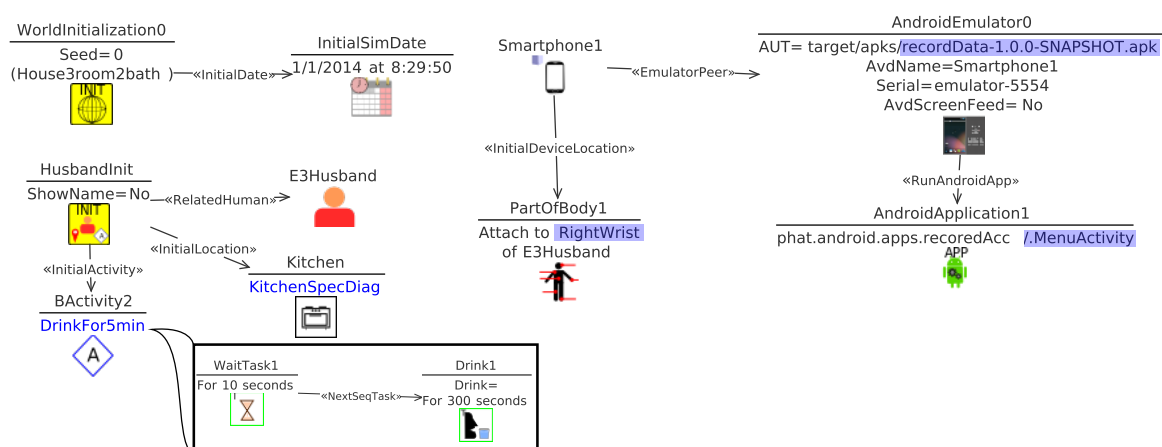


Figura 5.27: Diagrama de simulación para recrear la situación del sujeto virtual bebiendo con el dispositivo sujeto a la muñeca derecha y así poder guardar las aceleraciones recibidas en un fichero.

Por tanto, para llevar a cabo la validación, se necesita un escenario donde el sujeto realice el gesto de beber, el dispositivo se encuentre adherido a la muñeca que realiza la acción (la derecha), y la aplicación que registra las aceleraciones esté instalada y corriendo. Este escenario de prueba se define con SociAALML. Su diagrama de simulación lo muestra la Figura 5.27. Fíjese que tras 10 segundos el sujeto virtual estará realizando el gesto de beber, con el dispositivo en la muñeca derecha y la aplicación arrancada en el menú. El momento de pulsar el botón para registrar y dejar de registrar aceleraciones se ha realizado manualmente sobre el emulador Android.

Para registrar las aceleraciones en el mundo real, en primera instancia el sujeto dedicó un tiempo a imitar los gestos del sujeto virtual. Una vez interiorizadas, se

registraron las aceleraciones fijando el Nexus 4 a la muñeca de la misma forma que en la simulación. El resultado obtenido en esta prueba se ajusta mucho a la de la simulación como muestran las gráficas de la Figura 5.28, sobre todo en los ejes x e y. En el eje z también se ajusta bastante excepto a mitad del gesto donde la aceleración es más acusada. Se debe tener en cuenta que aunque el sujeto practicó durante unos minutos para imitar al sujeto virtual, resulta imposible que este reproduzca exactamente los mismos movimientos que el sujeto virtual.

Por tanto, a la vista de los resultados, se puede afirmar que siempre y cuando se consigan animaciones y movimientos realistas para el sujeto virtual, las aceleraciones generadas en la simulación serán consistentes y suficientes para alimentar una aplicación AAL que haga uso de dichos valores para, por ejemplo, realizar reconocimiento de actividades o caídas. No obstante, destacar que sería conveniente introducir algún modelo de ruido en los valores de las aceleraciones simulados para que se ajuste más a la realidad.

5.4.2. Efectividad de la simulación de la cámara

Una cámara es un dispositivo que obtiene imágenes bien de forma aislada (foto) o bien en forma de secuencia con una frecuencia determinada (vídeo). Una cámara se caracteriza, entre otras cosas, por el foco, el campo de visión y la resolución. La cámara simulada emplea los valores por defecto ofrecidos por jME3. Como trabajo futuro se considera interesante poder configurar el sensor variando los parámetros anteriores. Al igual que el acelerómetro, cada cámara simulada está accesible a través de un servidor socket con el protocolo TCP. Una aplicación compilada con jALI puede obtener el flujo de imágenes.

Como se vio en el apartado 4.6, la cámara es un sensor que puede ser muy útil para el desarrollo de este tipo de aplicaciones AAL. Las imágenes tomadas por la cámara las puede utilizar el sistema AAL simplemente para que un usuario visualice las imágenes o para otros fines más complejos en los que se aplicasen técnicas de visión artificial para, por ejemplo, reconocer situaciones, acciones de los usuarios o lecturas de códigos QR como en [29]. Es en este último caso donde la consistencia de las imágenes generadas es más determinante. Además, también es necesario que el número de fotogramas por segundo (FPS) generados por la simulación sea suficiente para la aplicación. Para ello, se ha creado una aplicación que registra la cantidad de imágenes recibidas por segundo desde la cámara, y se ha ejecutado en el dispositivo real y simulado. El resultado de ambas lecturas se muestran en la gráfica de la Figura 5.29. Por un lado, la gráfica refleja que la cámara real suministra imágenes de forma más regular (a 10 FPS). En cambio, la cámara simulada de media ofrece más imágenes (15 FPS) aunque no de forma constante. Se detecta un instante donde los FPS descienden a 5 y otro donde se superan los 25. Por tanto, por un lado, PHAT puede suministrar una cantidad suficiente de imágenes por segundo, pero por otro, se desvela que este suministro debe ser más constante.

Comparativa aceleraciones del gesto de beber

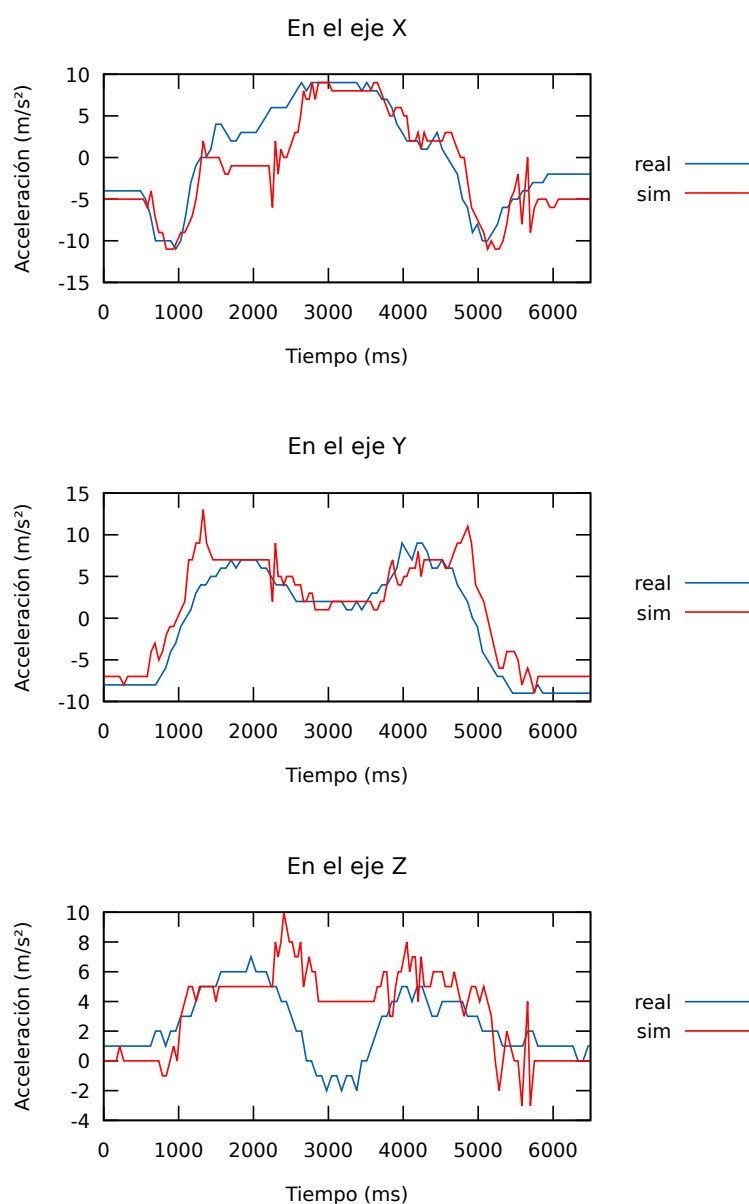


Figura 5.28: Comparativa eje a eje de los datos obtenidos de los acelerómetros virtual y real mientras se realiza el gesto de beber.

Para ilustrar una comparativa entre las imágenes obtenidas por el dispositivo virtual y uno real, se van a realizar dos experimentos: uno subjetivo y otro objetivo. El primero consiste en comprar la similitud entre las imágenes obtenidas en el mundo virtual y el real. El segundo compara las distancias máximas a las que el dispositivo consigue leer códigos QR.

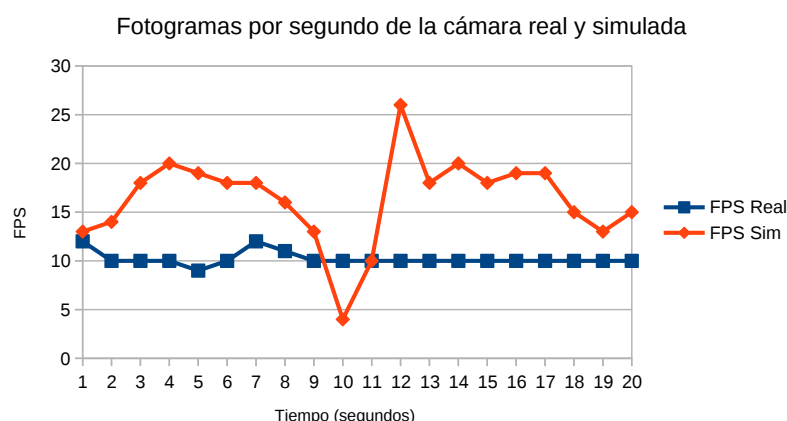


Figura 5.29: Fotogramas por segundo obtenidos por el dispositivo Android real y el simulado.

Para el *experimento subjetivo* se ha elegido el mismo escenario del caso anterior (diagrama de simulación). Pero, en este caso, se ha cambiado la aplicación por otra que lo que hace es visualizar las imágenes tomadas por la cámara. En la Figura 5.30 se puede ver una captura de pantalla del vídeo del experimento⁸. En el mismo, se visualizan tres imágenes. La de la izquierda corresponde al mundo virtual donde se muestra al sujeto virtual con el dispositivo sujeto a la muñeca derecha y reproduciendo la animación de beber. La del centro es la ventana del emulador Android asociado al dispositivo situado en la muñeca del sujeto virtual. En este caso, se ha instalado una aplicación que muestra las imágenes capturadas por la cámara. Por último, a la derecha se reproduce el vídeo generado por el dispositivo real tratando de reproducir la misma situación. De forma subjetiva, es apreciable la similitud aunque los elementos de la cocina no sean los mismos. No obstante, destacar que todavía habría que simular el desenfoque de la cámara cuando esta se mueve, tratar la iluminación (fuentes de luz y sombras) y, sobre todo, mejorar el nivel de detalle de los objetos de la escena (texturas y modelado).

El *experimento objetivo* es el mismo que ya se realizó en [29]. Este consiste en registrar las distancias máximas a las que la aplicación de lectura de códigos QR Zxing⁹ decodifica las etiquetas. Estas distancias se toman variando: el tamaño de la etiqueta del código QR y el ángulo de incidencia. Los tamaños empleados son tres: 1 cm, 2.5 cm y 5 cm, y los ángulos de incidencia son también tres: -22.5° , 0° , $+22.5^\circ$. La iluminación no se considera como variable ya que en [29] se comprobó que dichas variaciones no afectan significativamente al experimento. La Figura 5.31a corresponde a una fotografía del experimento publicado en [29] y realizado por Pedro Fernández. El dispositivo empleado fue un Samsung Nexus S. En cambio, la Figura 5.31b, corresponde

⁸Url del vídeo del experimento de la cámara: <http://grasia.fdi.ucm.es/aide/demos/2016/02/28/comparationbetweenimages.html>

⁹Sitio web de Zxing: <https://github.com/zxing/zxing>

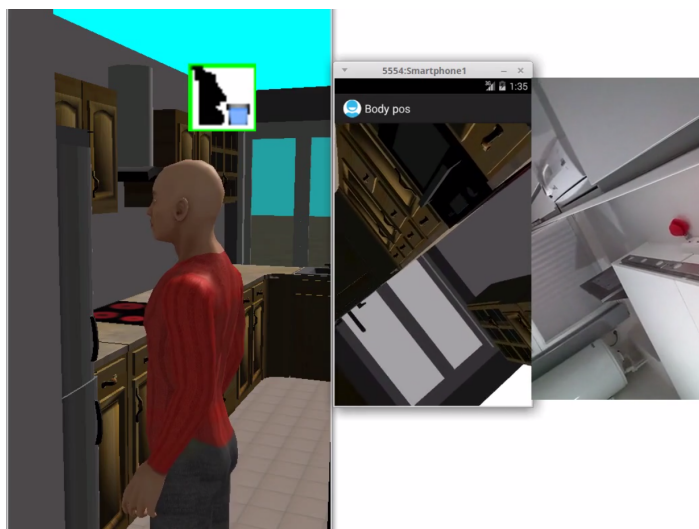


Figura 5.30: Comparativa del vídeo capturado desde la simulación y su equivalente en el mundo real.

al experimento realizado en PHAT. Arriba el emulador Android con la aplicación Zxing justo en el momento de leer el código QR. Abajo el simulador PHAT con el dispositivo virtual. Este envía imágenes al emulador para que las analice la aplicación Zxing. En el centro de la ventana de PHAT se puede ver la etiqueta con el código QR y justo encima un marcador con la distancia a la que se encuentra el dispositivo virtual respecto de la etiqueta (0.9 m). Con las teclas del ordenador se cambia el ángulo de incidencia y se aproxima o se aleja el dispositivo virtual en pasos de 1 cm. El vídeo del experimento se encuentra accesible en la web¹⁰.

Una vez registradas las distancias máximas para todos los tamaños y ángulos de incidencia en PHAT, estos se representan en la gráfica de la Figura 5.31 junto con aquellos obtenidos en el mundo real con el Nexus S y los obtenidos en UbikMobile. Se puede comprobar cómo PHAT detecta los códigos a distancias muy parecidas a las que lo hace el Nexus S. Este obtiene mejores resultados que UbikMobile para tamaños de códigos QR de 5 cm y 2.5 cm y peores para los de 1 cm. En global PHAT obtiene un error cuadrático medio de 0.86 y UbikMobile de 1.46. Por tanto, se considera que PHAT ofrece imágenes para ser procesadas de forma verosímil.

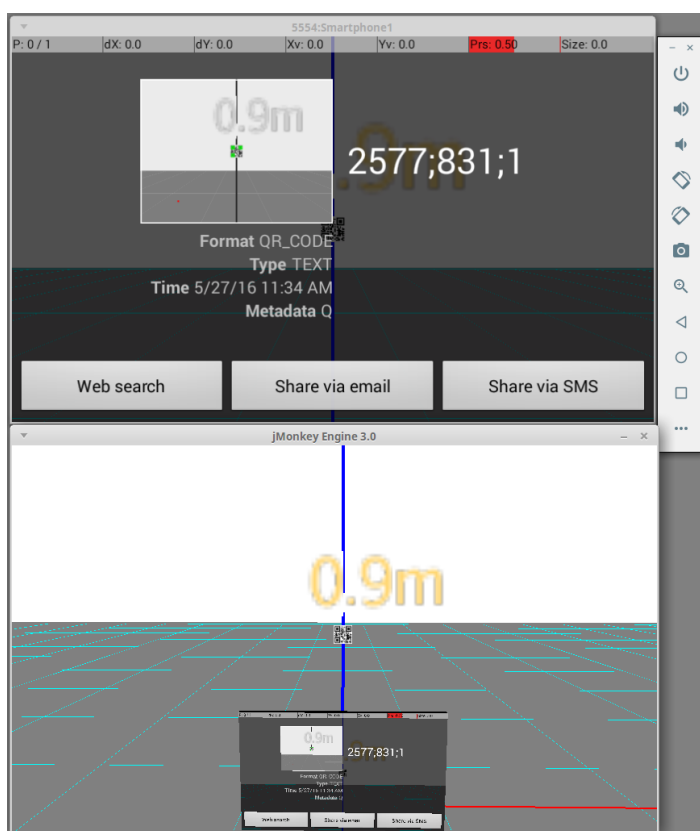
5.4.3. Efectividad de la simulación del audio

Los micrófonos son sensores que perciben el audio generado por el entorno. Estos son interesantes tanto para el reconocimiento de actividades como para emplear interfaces de voz gracias a técnicas de reconocimiento de voz [116], es decir, traducir las palabras habladas a texto. Por tanto, el audio es un elemento que puede aportar bastante información al sistema AAL o un mecanismo interesante para que los usuarios

¹⁰URL del experimento de códigos QR: http://grasia.fdi.ucm.es/sociaal/demos/2016/06/20/distance_check.html



(a) Fotografía del experimento de lecturas de códigos QR en el mundo real con un Samsung Nexus S. Este fue realizado por Pedro Fernández.



(b) Experimento de lecturas de códigos QR en PHAT.

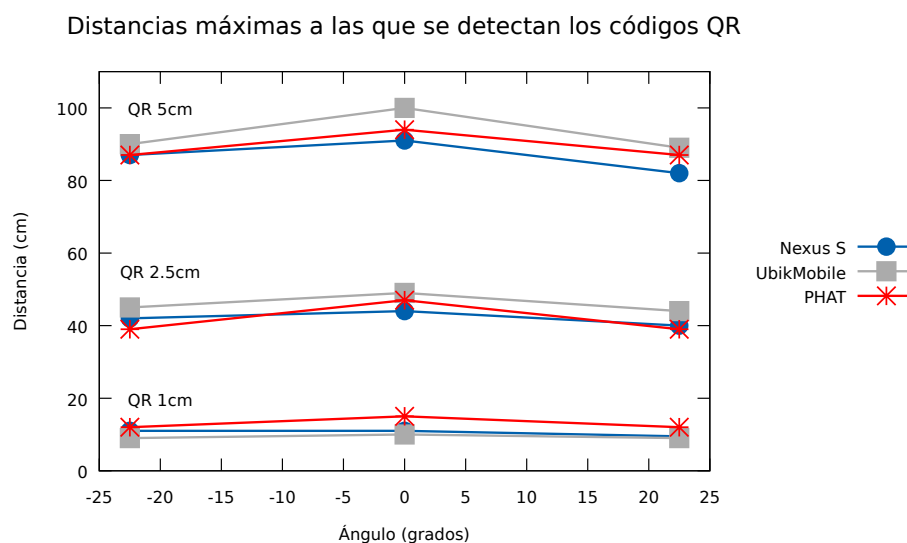


Figura 5.31: Distancias máximas a las que es posible leer diferentes tamaños de códigos QR (5, 2.5 y 1 cm) en UbikMobile, PHAT y el mundo real (Nexus S).

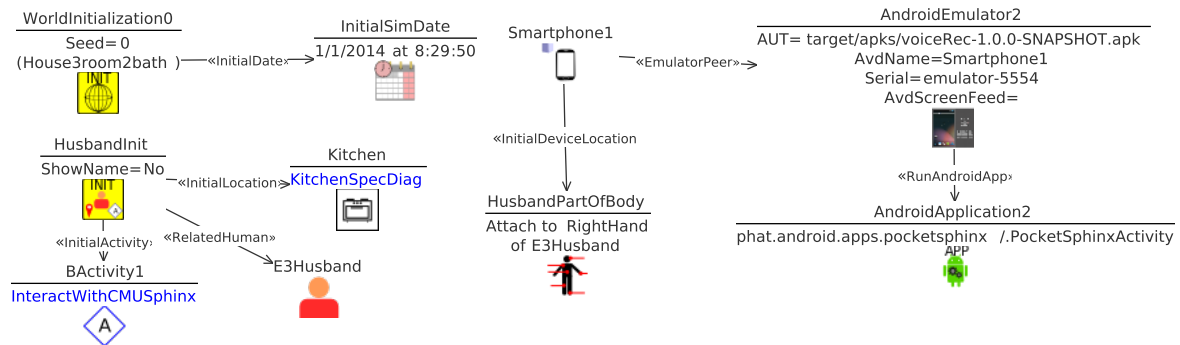
interaccionen con el sistema por comandos de voz. Como ya se comentó en el apartado 4.6, los micrófonos simulados son posibles gracias a los trabajos de McIntyre [89]. Conseguir que sea consistente el audio generado en la simulación con el mundo real depende en gran medida del render de audio, del sintetizador de voz de los sujetos virtuales y, en general, que el volumen de las fuentes de audio se calibren adecuadamente. Al igual que el acelerómetro y la cámara, cada micrófono simulado está accesible a través de un servidor socket con el protocolo TCP. Una aplicación compilada con jALI puede obtener el flujo de audio.

Para validar los sintetizadores de voz y los micrófonos virtuales, se va a realizar una prueba con una aplicación sencilla para Android que ofrece el proyecto CMU Sphinx¹¹ para el reconocimiento de voz. Dicha aplicación se queda a la espera de reconocer la palabra clave “Oh mighty computer” y, posteriormente, se indica qué tipo de palabras se quiere reconocer. En este caso, se va a probar con números (al igual que en el caso de estudio), por ello, se debe decir “digits”. Finalmente, se dice un número, por ejemplo el uno (“one”), y la aplicación muestra con un mensaje de texto lo que ha reconocido.

Para realizar esta prueba en el mundo virtual, se han definido principalmente los diagramas de la Figura 5.32. La Figura 5.33 muestra una captura de pantalla del escenario previamente definido en el momento justo después de que el sujeto virtual dijese “digits”. En el mundo real se ha procedido a realizar la misma secuencia de comandos de voz obteniéndose el mismo resultado. La secuencia completa de ambas pruebas se graba en un vídeo¹². Queda así demostrado que la plataforma puede servir para evaluar diseños de soluciones AAL que utilicen el reconocimiento de voz.

¹¹Sitio web de CMU Sphinx: <http://cmusphinx.sourceforge.net/>

¹²Url del vídeo del experimento de reconocimiento de voz: <http://grasia.fdi.ucm.es/aide/demos/2016/03/17/speechrecognitionwithsphinx.html>



(a) Diagrama de simulación.

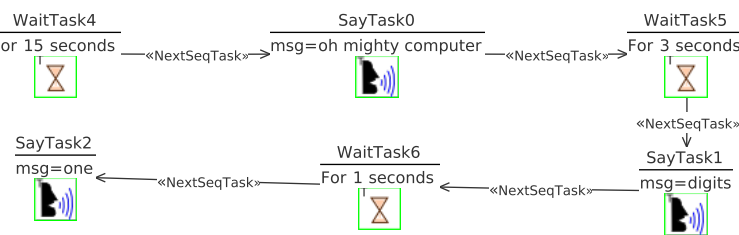
(b) Diagrama *InteractWithCMUSphinx* con la secuencia de comandos de voz y sus pausas.

Figura 5.32: Diagramas principales para probar la interacción por voz del sujeto virtual con la aplicación de demo de CMU Sphinx.

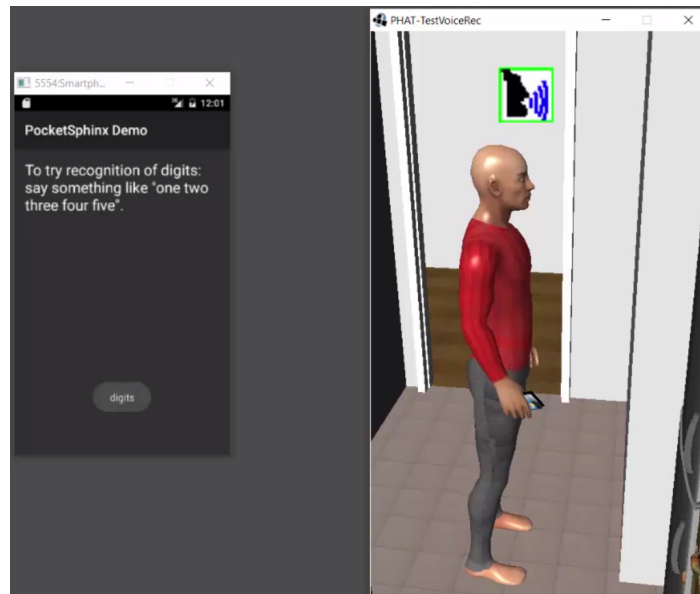


Figura 5.33: Sujeto virtual interactuando con la aplicación a través de comandos de voz. La aplicación emplea la librería CMU Sphinx para su reconocimiento.

Un aspecto interesante que surgió al probar la demo CMU Sphinx fue que esta no acepta silencio absoluto en el audio. Por ello, se tuvo que introducir un pequeño ruido en el audio procedente de los micrófonos virtuales antes de ser enviado al emulador Android. Por ello, se avanzó en la mejora necesaria para que el audio que se genere sea más verosímil: introducir sonido ambiente y ruido en los micrófonos. También resultaría interesante mejorar los sintetizadores de voz para que no fuesen tan robóticos y se pudiese distinguir entre voces, al menos una combinación de: hombre, mujer, anciano y joven. De esta forma, se podría usar aplicaciones AAL que reconociesen las voces de los usuarios.

5.5. Conclusiones

En este capítulo se ha demostrado cómo las herramientas propuestas alcanzan los objetivos planteados a través de dos casos de estudio y una serie de experimentos.

Los dos casos de estudio, apartado 5.1 y 5.2, muestran cómo el lenguaje SociAALML permite modelar el comportamiento de sujetos en un entorno doméstico y, en concreto, las variaciones del comportamiento del paciente en función de los niveles de sus síntomas. Además, se deja de manifiesto cómo es posible validar los modelos generados en PHAT a partir de analizar el log y la visualización de la simulación 3D generada automáticamente. En el primer caso, se ha mostrado cómo a partir de una entrevista se recoge la información más relevante, dejando patente que dicha información es incompleta y resultaría interesante una segunda entrevista con el paciente y sus familiares, esta vez con la herramienta presente. Aun así, se han identificado situaciones esperadas y tras su simulación se ha comprobado que algunas de ellas no se han cumplido por diversos motivos, entre ellos, errores en el modelado, valores de probabilidades en las condiciones y un error en la implementación del simulador. En el segundo caso, se demuestra más el nivel de aleatoriedad y la gran variedad de situaciones que pueden darse al variar la semilla. Para la actividad de ver la televisión se han producido 7 situaciones diferentes para 10 valores de semilla. Con este caso, también se muestra cómo es posible definir el despliegue de la solución en un escenario que mejora las condiciones de vida de los habitantes del entorno para dicho escenario. Un prototipo de mando a distancia con reconocedor de voz, permite al paciente especificar el canal que desea ver mejorando así su autonomía y no molestando al cuidador para que realice dicha tarea. A través del diagrama de simulación se ha podido especificar el despliegue de esta solución, indicando tanto la televisión como el mando, su posición inicial y el software a ejecutar.

A partir de los experimentos del apartado 5.4, se demuestra cómo los sensores virtuales implementados suministran a la aplicación AAL datos verosímiles, siendo conscientes de las limitaciones y que la verosimilitud de dichos datos dependen en gran medida en la calidad de los movimientos del sujeto virtual en el caso del acelerómetro, o de los gráficos y texturas para la cámara. De todos modos, se demuestra que al menos para aplicaciones basadas en reconocimiento de voz, lectura de códigos QR y

reconocimiento de gestos como el de beber agua, la plataforma ofrece las herramientas para realizar una primera aproximación para abordar el problema.

Capítulo 6

Conclusiones

Solo cabe progresar cuando se piensa en grande, solo es posible avanzar cuando se mira lejos.

José Ortega y Gasset

RESUMEN: En este último capítulo se recoge las principales conclusiones del trabajo realizado, las aportaciones y la gran cantidad de trabajo futuro que queda por realizar.

6.1. Aportaciones

Las principales aportaciones en esta tesis han ido encaminadas a demostrar que el modelado realista de escenarios AmI puede ser una herramienta con futuro en el proceso de desarrollo de este tipo de productos, y en concreto para el caso de la asistencia de enfermos de Parkinson. Para ello, se ha seguido un enfoque de desarrollo de simulaciones dirigidas por modelos. Siguiendo este enfoque, las principales contribuciones han sido un lenguaje gráfico específico del dominio (SociAALML), un simulador de mundos domésticos convincentes (PHAT) con su generador de código asociado y casos de uso donde se demuestra el valor que aportan dichos elementos en el proceso de desarrollo de este tipo de sistemas:

- El lenguaje SociAALML, descrito en el Capítulo 3, parte tanto del trabajo del estudio del dominio del Parkinson a partir de la literatura especializada y de las transcripciones de las entrevistas a los enfermos y sus familiares identificando aquellos elementos más relevantes, como de los primeros casos de estudio donde se identificaban conceptos necesarios para la generación automática de la simulación. Dichas entidades y relaciones se especifican en el meta-modelo del lenguaje y se centran en conceptos como individuo, actividad, tarea, enfermedad, síntoma y

limitación, los cuales tienen un mayor nivel de abstracción permitiendo especificar la solución (la simulación de un escenario) en términos del dominio. Destacar el trabajo realizado en la notación (creación de todos los símbolos) para que esta sea cognitivamente eficiente según los principios de Moody[96] de forma que resulte más fácil, a usuarios no expertos, el aprendizaje y comprensión del lenguaje. El lenguaje sirve como herramienta para la captura formal de conocimiento del dominio del problema, para compartir y comunicar dicho conocimiento entre los diferentes miembros involucrados en el desarrollo del sistema AAL. El lenguaje es totalmente abierto y público¹ pudiéndose beneficiar toda persona interesada. Además, también se aporta conocimiento al incluir el modelo de la entrevista E9 del caso de estudio visto en el apartado 5.1. Por último, el lenguaje también se puede reutilizar para otras enfermedades. Por ejemplo, en [26] se modelaron algunos escenarios para el caso de enfermos de Alzheimer.

- El simulador PHAT junto con el generador de código permite la creación automática de simulaciones convincentes a partir de las especificaciones realizadas con SociAALML. A partir del trabajo del dominio, este simulador permite simular la vida diaria de unidades familiares en sus viviendas y en concreto las limitaciones de los enfermos de párkinson. Por ejemplo, simulación de caídas o temblores de manos y cabeza. PHAT también incorpora funcionalidades útiles, como el log, para ayudar a los usuarios a comprobar que la simulación se comporta como se espera y, por tanto, la especificación SociAALML realizada es correcta. Además, gracias a la librería jALI, se permite la inmersión de dispositivos Android en la simulación haciendo posible que la plataforma sirva como entorno para el diseño y la evaluación de prototipos de soluciones AAL. Esta característica se pudo comprobar a través del caso de estudio del apartado 5.2, donde a partir de la identificación de una serie de problemas en la actividad de ver la televisión, se diseñó e implementó una solución AAL sencilla para el problema de pulsar los botones del mando.
- Tanto el lenguaje como el simulador y el generador son totalmente abiertos y públicos. La creación, sobre todo, de PHAT desde cero, ha supuesto una carga adicional considerable pero necesaria pues, como se vio en el Capítulo 2, no hay disponible en la comunidad un simulador de estas características que sea libre y de código abierto. Ahora sí lo hay y está disponible² para todo aquel que lo necesite. La intención es generar una comunidad de usuarios y desarrolladores que promuevan la simulación para la investigación y el desarrollo de sistemas AAL. De hecho, este simulador ha sido ya empleado por alumnos de la Facultad de Informática de la UCM para desarrollar un sistema de detección de caídas para su proyecto final de grado [140]. También, lo está empezando a utilizar un estudiante de doctorado, Marlon Cárdenas, como base para realizar reconocimiento

¹Proyecto SociAALML: <https://github.com/Grasia/sociaalml>

²Código fuente de PHAT: <https://github.com/Grasia/phatsim>

de actividades. El trabajo conjunto logró el primer premio concedido por IBM en el congreso PAAMS 2016 [51].

- El trabajo presentado en PAAMS'16 justifica la viabilidad del modelo mixto de desarrollo simulación/prototipo físico. A parte, se han desarrollado una serie de experimentos en el apartado 5.4 con el fin de demostrar la verosimilitud de los datos generados por los sensores virtuales con los del mundo real. Lo cual implica que si existe una solución AAL para un problema del mundo real esta también será válida para el mundo virtual siempre y cuando se dispongan de los elementos virtuales necesarios como animaciones, objetos 3D, sensores y actuadores, entre otros.

6.2. Líneas de investigación abiertas

A lo largo del trabajo se han encontrado varios aspectos que hubiese sido interesante tratar pero que por su complejidad, se han dejado como trabajo futuro.

Referente al lenguaje, se consideran especialmente relevantes 3 aspectos: las interacciones, la reutilización de las actividades y la validación del lenguaje:

- Resulta costoso definir tareas o actividades conjuntas, donde intervengan más de un sujeto y se deban sincronizar. Investigar mecanismos para ampliar el lenguaje en este sentido resultaría conveniente para poder definir escenarios más complejos.
- Durante los casos de estudio también se ha comprobado que en varias ocasiones resulta complicado reutilizar actividades. Por ejemplo, se definen varios diagramas de secuencias de tareas donde solo varía la duración de la tarea principal como podría ser la actividad de ver la televisión. Una mejora a corto plazo podría consistir en enriquecer el concepto de actividad dando la posibilidad de añadir atributos como la duración, tareas necesarias previas, tareas principales y tareas de finalización de la actividad. Estas ayudarían a dicha reutilización e incluso a facilitar la interrupción y reanudación de las mismas.
- La validación del lenguaje ha sido limitada, reduciéndose su aplicación a unos casos de estudio. Sería necesario conseguir en el futuro que expertos en la materia (tales como médicos, trabajadores sociales) y los propios enfermos y familiares lo utilicen. Esto permitirá mejorar y validar aspectos del lenguaje como idoneidad de los conceptos, expresividad, facilidad de aprendizaje y comprensión.

En cuanto al simulador PHAT, se ha comprobado que la realización de simulaciones realistas es costosa. Esta demanda mucha inversión tanto en desarrollo de contenidos multimedia para aportar la apariencia 3D a los elementos como en la programación para dotarlos de comportamiento. Debido a ello, se puede observar que los casos de estudio se reducen a una vivienda y se dispone de dos tipos de sujetos virtuales. Disponer de herramientas y estándares que faciliten la creación e integración de este tipo de recursos

es muy conveniente para que el empleo de la simulación en los procesos de desarrollo de sistemas AAL sea una realidad. En cuanto a la creación de recursos, se identifican fundamentalmente necesidades en todas las entidades involucradas en el sistema AAL: la vivienda con sus objetos, los dispositivos y los sujetos virtuales:

- Por cada nuevo proyecto, el entorno será diferente, es decir, la vivienda donde diseñar y desplegar el sistema AAL. Facilitar la generación de viviendas virtuales operativas donde poder desplegar los dispositivos y a los sujetos virtuales con sus comportamientos para que interaccionen en el mismo resulta fundamental. El lenguaje IFC es el estándar más popular para representar, modelar y capturar la semántica de un edificio. Existen herramientas que asisten en la creación de estos modelos pero todas ellas profesionales: Autodesk®Revit™, Graphisoft®Constructor™ o Bentley®Architecture™. Parece muy conveniente tratar de estudiar este modelo e integrarlo en SociAALML de forma que los modelos IFC se transformen a SociAALML. Imagínese en un futuro toda casa o edificio tiene su modelo IFC. En ese caso, el coste de desarrollar la vivienda virtual sería cero.
- Como parte del diseño de una solución AAL, se puede contemplar la posibilidad de crear un nuevo dispositivo hardware. Actualmente, PHAT soporta dispositivos Android sencillos, con forma de cubo y una pantalla. Se considera muy interesante ofrecer una herramienta para el prototipado de dispositivos donde se les pueda dotar de una forma específica y equipar con sensores y actuadores. Además, se debería especificar el tipo de interacción disponible con el dispositivo de forma que el sujeto virtual sea capaz de utilizar el nuevo dispositivo. Dicho editor debería generar el emulador correspondiente donde se instalaría su software, lo conectaría a la simulación y permitiría la interacción con el entorno.
- Actualmente existen herramientas para el modelado 3D de objetos con apariencia humana realista adecuados para ser empleados como sujetos en la simulación, por ejemplo, MakeHuman³. Lo que resulta realmente costoso, es conseguir que estos se muevan de forma realista. La investigación y el desarrollo de técnicas para conseguir que cualquier sujeto virtual realice convincentemente movimientos, gestos y acciones dependientes del entorno y parametrizables como, por ejemplo, coger un objeto, abrir una puerta o un cajón, mirar a un sujeto, supondría un gran avance para este tipo de simulaciones.

Por último, otra área a investigar consiste en el estudio de una metodología de uso más exhaustiva para la propuesta presentada en este trabajo. Especialmente, trabajar con un método de desarrollo industrial de sistemas que implique este tipo de desarrollos.

³Sitio web de MakeHuman: <http://www.makehuman.org/>

6.3. Trabajos futuros

Como se ha visto en el apartado anterior, esta tesis deja abiertos una gran cantidad de líneas de investigación. De todas ellas, la línea de trabajo futuro más inmediata iría encaminada a la mejora de los dispositivos y la integración con sistemas AAL reales como UniversAAL⁴. Las acciones serían:

- Ampliar el repertorio de sensores virtuales e incluir actuadores, pudiendo ser ambos configurables, es decir, poder ajustar: frecuencias de muestreo, rangos, modelo del ruido, entre otros.
- Modificar la capa de drivers del emulador Android para que se conecte al simulador y no sea necesario modificar en absoluto las aplicaciones. De esta forma, incluso los servicios de Google o cualquier otra aplicación percibirán el entorno virtual. También sería necesario sincronizar la fecha y la hora del emulador con la del simulador. Esto resulta fundamental para realizar pruebas con aplicaciones dependientes de la hora como, por ejemplo, un planificador para un enfermo de Alzheimer.
- Modificar también el emulador para que publique los componentes gráficos que se encuentran actualmente en la pantalla. Por ejemplo, saber el Activity actual, los layout, botones, entre otros. Esto tiene como objetivo que un sujeto virtual pueda interaccionar de forma mucho más compleja con las aplicaciones Android.
- Mejorar el mecanismo para acceder desde el exterior a los sensores y actuadores para que sea sencillo conectar middlewares como UniversAAL.
- Ampliar el lenguaje todo lo necesario para que recoja los conceptos nuevos y facilite la simulación de distintos diseños donde varíen la configuración y características de los dispositivos.

⁴Sitio web de UniversAAL: www.universaal.org

Parte I

Apéndices

Apéndice A

Símbolos de los elementos del lenguaje SociAAML

RESUMEN: El apéndice recoge las representaciones de todas las meta-entidades del lenguaje SociAAML.

A.1. Símbolos de los elementos del espacio físico

Notación	Meta-objeto	Descripción
 SBuilding0	SBuilding	Texto superior con el identificador de la casa seguido del icono de la casa.
 SFloor0 : 0 Floor0	SFloor	Texto superior con el identificador de la planta más el nivel de la misma (propiedad <i>FloorNumberField</i>) separada por “:”. Tras una línea de separación se encuentra un enlace cuyo texto es el nombre del diagrama que define la planta. Por último icono de una planta.
 Lift1 SCorridor0 SCorridor1	Lift	Texto superior con el identificador del ascensor, tras la línea se encuentran los identificadores de las habitaciones que conecta el ascensor en cada planta y, por último, se muestra el icono de un ascensor.
 Staircase1 SCorridor0 SCorridor1	Staircase	Texto superior con el identificador de las escaleras, tras la línea se encuentran los identificadores de las habitaciones que conecta las escaleras en cada planta y, por último, se muestra el icono de unas escaleras.

Tabla A.1: Tabla de la notación para los objetos correspondientes a la vista del edificio.

Notación	Meta-entidad	Descripción
 SBedroom0 BasicBedroom	SBedroom	Texto superior con el identificador de la habitación. Tras una línea de separación se encuentra un enlace cuyo texto es el nombre del diagrama que define la habitación. Por último icono representativo de la habitación.
 SCorridor0 BasicCorridor	SCorridor	
 SBathroom0 BasicBathroom	SBathroom	
 SKitchen0 BasicKitchen	SKitchen	
 SLivingroom0 BasicLivingroom	SLivingroom	
 Door0	SDoor	Texto superior con el identificador de la relación de puerta seguido de un icono que representa una puerta.

Tabla A.2: Tabla de la notación para los objetos correspondientes a la vista de una planta.

Notación	Meta-entidad	Descripción
 FOWashbasin0	FOWashbasin	Texto superior con el identificador del mueble y debajo del mismo se encuentra el icono representativo del mueble.
 FTable0	FTable	
 FOSink0	FOSink	
 FOBath0	FOBath	
 FChair0	FChair	
 FSofa0	FSofa	
 Shower0	Shower	
 FBed0	FBed	
 FOWater0	FOWater	
 FWardrobe0	FWardrobe	
 SArea0	SArea	Texto superior con el identificador del área y debajo del mismo se encuentra el icono que representa una habitación donde se resalta un área de la misma.

Tabla A.3: Tabla de la notación para los objetos correspondientes a la vista de una habitación.

A.2. Símbolos de los elementos de los sujetos a simular

A.2.1. Los símbolos de los perfiles

Notación	Meta-entidad	Descripción
E3Patient 	Human	Texto superior con el identificador del humano y debajo del mismo se encuentra el icono de un monigote que representa al humano.
E3PatientSocialProfile E3PatientSocialProfile 	SociaProfile	Texto superior con el identificador del perfil social. A continuación enlace a su definición y, por último, el icono de un monigote con una red de monigotes representando una red social y un círculo azul con una "i" representando la información personal.
E3PatientADLProfile AtNightADLDiagram 	ADLProfile	Texto superior con el identificador del perfil de las actividades diarias. Debajo el enlace a su definición. El icono representativo corresponde a un calendario mensual.
InteractionProfile1 E3.Patient.Interaction 	InteractionProfile	Texto superior con el identificador del perfil de interacciones. A continuación, el enlace a su definición y, por último, el icono de un monigote con un conjunto de círculos rojos, que representan de los eventos, y unas flechas apuntando hacia el mismo indicando la percepción de dichos eventos o estímulos.
GenericDisease0 ReumaSpec 	GenericDisease	Texto superior con el identificador del perfil de la enfermedad o dolencias generales de la persona. Debajo el enlace a su definición. El icono representativo corresponde a un monigote y estetoscopio representativo de los médicos.
AParkinsonProfile PD.spec.example 	ParkinsonProfile	Texto superior con el identificador del perfil de la enfermedad de Parkinson. Debajo el enlace a su definición y, por último, el icono representativo corresponde a un monigote y cerebro (origen de la enfermedad) con las siglas en inglés de la enfermedad PD (Parkinson's Disease).

Tabla A.4: Tabla con la notación correspondiente a las entidades que forman los perfiles de un sujeto.

A.2.2. Los símbolos del perfil social

Notación	Meta-entidad	Descripción
<u>E3PatientPersonalInfo</u> Sex=Female Age=75 SocialClass=LOW StudiesLevel=LOW 	PersonalInfo	Texto superior con el identificador de la entidad con la información personal. A continuación muestra los valores de sus cuatro campos: Sex (<i>SexField</i>), Age (<i>AgeField</i>), SocialClass (<i>SocialClassField</i>), y StudiesLevel (<i>StudiesLevelField</i>). Por último, el icono representativo de un humano con un círculo azul y la “i” de información.
<u>HouseWife</u> Profession=housewife Active=No 	Profession	Texto superior con el identificador de la entidad con la información relativa a la profesión. Justo debajo se muestran los valores de cuatro de sus dos atributos: Profession (<i>Profe</i>), Age (<i>AgeField</i>), SocialClass (<i>SocialClassField</i>), y StudiesLevel (<i>StudiesLevelField</i>). Por último, el icono de un monigote con un mono y casco de trabajo representando a un trabajador.
<u>SocialNetwork0</u> 	SocialNetwork	Texto superior con el identificador de la red social y debajo un icono representando personas conectadas por líneas de diferentes colores representando las diferentes relaciones.
	isSpouseOf	Icono de dos alianzas para representar la relación de pareja, ya sean esposos o no.
	isParentOf	Icono de un monigote grande llevando de la mano a uno pequeño para indicar la relación de paternidad.
	isCaregiverOf	Un icono de una cruz roja para indicar una relación de cuidador o asistente.

Tabla A.5: Tabla de la notación para los objetos correspondientes a la vista del perfil social.

A.2.3. Los símbolos del perfil de las actividades diarias

Notación	Meta-entidad	Descripción
<u>Wake up time</u> 8:0:0 	ClockTime	El identificador se muestra en parte superior, tras la línea se muestra los valores de los atributos de la entidad, es decir, la hora, los minutos y los segundos separados por “:”. Por último, se muestra el icono que representa un reloj con el fondo rosa.
<u>GetReadyTime</u> Do.get_ready activities (Rest for a while) 	TimeInterval	El identificador se muestra en la parte superior. Tras la línea se muestran 2 atributos, el primero hace referencia al modelo de actividades que define el periodo de tiempo y el segundo, entre paréntesis, hace referencia a una secuencia de tareas por defecto. Debajo del todo se encuentra el icono donde se muestra el sol y la luna representando un día completo y una cuña verde indicando que se hace referencia a una parte del día.

Tabla A.6: Tabla de la notación para los objetos correspondientes a los periodos del día.

Notación	Meta-entidad	Descripción
<u>Have Breakfast</u> <u>Breakfast in chair1</u> 	BActivity	Muestra el enlace con el nombre del diagrama de tareas que define la actividad.
	IFFlowControl	Icono representando una bifurcación con dos flechas, una roja representando que no se cumple la condición y otra verde cuando sí. No muestra su identificador ya que no se usa.
	TrueFlow	Icono que representa un acierto ya que es la relación que une el flujo cuya condición se cumple con el siguiente elemento.
	FalseFlow	Icono que representa un fallo ya que es la relación que une el flujo cuya condición no se cumple con el siguiente elemento.
<u>DayOfTheWeek0</u> <u>Monday</u> 	CDayOfTheWeek	Muestra el día de la semana (atributo <i>DayOfTheWeekField</i>) que es la clave para evaluar la condición. Finalmente se muestra un icono con forma de pentágono verde (representativo de las condiciones) y la palabra día en inglés (Day) seguido de una interrogación, común también a todas las representaciones de las condiciones.
<u>Half of the time</u> <u>Prob = 0.5</u> 	CProb	Muestra el valor del atributo (<i>ProbVarField</i>) indicando la probabilidad de que la condición sea cierta. El icono representa un dado, símbolo muy utilizado para lo aleatorio.
<u>CInside0</u> <u>Caregiver</u> 	CInside	Muestra el valor del atributo (<i>HumanTarget</i>) indicando la persona por la que se pregunta si está dentro de la casa. El icono representa una casa con una flecha hacia la casa.
<u>COutside0</u> <u>Caregiver</u> 	COutside	Muestra el valor del atributo (<i>HumanTarget</i>) indicando la persona por la que se pregunta si está fuera de la casa. El icono representa una casa con una flecha saliendo de la casa.
<u>CSymptom0</u> <u>is Tremor MEDIUM</u> 	CSymptom	Muestra el valor del atributo <i>PDSymptomTypeField</i> y <i>IntensityLevelField</i> en forma de pregunta. El icono representa un termómetro.
<u>CTimer0</u> <u>0:15:0</u> 	CTimer	Muestra los campos <i>HoursField</i> , <i>MinutesField</i> y <i>SecondsField</i> en formato hora separados por ":". El icono representa un cronómetro.
<u>CTime0</u> <u>12:0:0</u> 	CTime	Muestra los campos <i>HoursField</i> , <i>MinutesField</i> y <i>SecondsField</i> en formato hora separados por ":". El icono representa un reloj.

Tabla A.7: Tabla de la notación para los objetos correspondientes a las actividades del día.

Notación	Meta-entidad	Descripción
Eat2 Food=Sandwitch For 300 seconds 	Eat	Muestra el valor del campo con el identificador del objeto a comer. El icono representa la silueta de una cara con la boca abierta y un trozo de pizza.
Drink4 Drink=Water For 30 seconds 	Drink	Muestra el valor del campo con el identificador del objeto a beber. El icono representa la silueta de una cara con la boca abierta y un vaso de agua.
GoIntoBed0 For 20 seconds 	GoIntoBed	Indica los segundos que tarda en meterse en la cama. El icono representa una cama y una flecha apuntando a ella.
BGetUpFromBed3 For 15 seconds 	BGetUpFromBed	Indica los segundos que tarda en salir de la cama. El icono representa una cama y una flecha saliendo de ella.
SitDown0 Seat=Chair1 For 10 seconds 	SitDown	Muestra la silla donde se sentará (si se ha indicado) y el tiempo que le llevará dicha acción. El icono representa una silla con un monigote encorvado, la flecha hacia abajo indica que se va a sentar.
StandUp0 For 5 seconds 	StandUp	No muestra la silla de la cual se levanta pues en principio no es relevante y el tiempo que le llevará dicha acción. El icono representa una silla con un monigote encorvado, la flecha hacia arriba indica que se va a levantar.
Put on shocks Object=Shocks For 60 seconds 	PutOnTask	Muestra el identificador de la prenda que se va a poner. El icono representa una camiseta.
TakeOffShocks Object=Shocks For 30 seconds 	TakeOffTask	Muestra el identificador de la prenda que se va a quitar. El icono representa una camiseta con una señal de prohibido representando la acción o necesidad de quitarla.
SayTask0 msg=Thank you! 	SayTask	Muestra el mensaje que dirá el personaje. El icono representa el perfil de una persona con ondas de sonido saliendo de su boca.
Have Shower Object=Shower1 For 600 seconds 	BUseTask	Muestra el identificador del objeto que se usará y durante cuánto tiempo. El icono consiste en la caja lila representativa de los objetos con dos engranajes símbolo de la funcionalidad o trabajo.

Tabla A.8: Tabla de la notación para los objetos correspondientes a las tareas, parte 1.

Notación	Meta-entidad	Descripción
<u>BGoToTask2</u> Go to BedRoom1 at 0.5 m/s 	BGoToTask	Muestra el identificador del lugar espacio donde irá y a qué velocidad indicando las unidades (m/s). El icono representa unas pisadas de zapatos y una flecha.
<u>GoToBodyLoc1</u> Go to E3Patient at 1 m/s 	GoToBodyLoc	Muestra el identificador de la persona donde irá y a qué velocidad indicando las unidades (m/s). El icono representa unas pisadas de zapatos y una flecha apuntando a un monigote.
<u>BPickUpTask2</u> Object=Drugs For 10 seconds 	BPickUpTask	Muestra el identificador del objeto a coger y el tiempo que le tomará. El icono consiste la superficie de una mesa, con un cubo lila representando el objeto y una flecha hacia una mano representando que lo va a coger.
<u>BLeaveTask0</u> Drugs in FTable0 For 5 seconds 	BLeaveTask	Muestra el identificador del objeto a dejar y el del objeto destino donde lo dejara. El icono consiste una mano que sostiene un cubo lila representando el objeto y una flecha apuntando hacia el objeto destino representando la superficie de una mesa.
<u>FallSleep0</u> For 600 seconds 	FallSleep	Muestra el tiempo que estará durmiendo la persona. El icono consiste en tres zetas de diferente tamaño.
<u>WaitTask0</u> For 60 seconds 	WaitTask	Muestra el tiempo que durará la tarea. El icono representa un reloj de arena pues la tarea es genérica.
<u>WaitForBodyClose1</u> Waiting Caregiver 	WaitForBodyClose	Muestra el identificador de la persona que se está esperando. El icono representa a dos monigotes con una flecha que va de uno a otro y un reloj de arena representando la espera.
<u>OnOffButton</u> (240,280) Smartphone1 	TapXYTask	Muestra los valores en forma de coordenadas, primero x y luego y, de la pantalla donde pulsará la persona. Debajo muestra el identificador del dispositivo. El icono representa a una mano pulsando en un teléfono inteligente.
<u>BSequentialTask0</u> Sit down on chair with arms 	BSequentialTask	Muestra un enlace con el nombre del diagrama de tareas que se ejecutarán de forma secuencial. Este icono a diferencia de los anteriores ya que está relleno de color y tiene el símbolo más (+), indicando que no es una tarea compuesta. La secuencia se con dos cuadrados unidos por una flecha.
<u>BRandomTask0</u> TaskSet 	BRandomTask	Muestra un enlace con el nombre del diagrama de tareas que se ejecutarán de forma aleatoria. El icono sigue el mismo patrón que el anterior pero se utiliza un dado para representar la aleatoriedad.

Tabla A.9: Tabla de la notación para los objetos correspondientes a las tareas, parte 2.

A.2.4. Los símbolos del perfil de las interacciones

Notación	Meta-entidad	Descripción
<u>EventProcessor4</u> 	EventProcessor	El icono consta de un símbolo de ruedas dentadas al que le llegan dos símbolos, el evento y la condición, y del que sale el símbolo de la actividad.
<u>PatientHasTVRemoteProblemEvent</u> Magnitude = LOW 	BEvent	Muestra la importancia del evento ("Magnitude ="). El icono consiste en una circunferencia roja.
<u>CReceiveHelp0</u> 	CReceiveHelp	El icono consiste en un flotador que representa asistir o ayudar a alguien.

Tabla A.10: Tabla de la notación para los objetos correspondientes a la respuesta ante los eventos.

A.2.5. Los símbolos del perfil de la enfermedad

Notación	Meta-entidad	Descripción
<u>PDDiseaseStage1</u> <u>DrugsTimes</u> (E4) 	PDDiseaseStage	Muestra un enlace con el nombre del diagrama con los tiempos de las tomas de la medicación. Debajo, entre paréntesis, se muestra el estadio de la enfermedad. El icono consiste en cinco círculos representando los cinco estadios de la enfermedad, del E1 al E5, donde el color va variando, del verde (E1) hasta el rojo (E5).
<u>PDSymptom1</u> <u>SimpleTremorEvolution</u> (Tremor) 	PDSymptom	Muestra el enlace con el nombre del diagrama que describe cómo evoluciona el síntoma. Debajo entre paréntesis muestra el nombre del síntoma. El icono consiste en una estrella azul (nunca usada hasta ahora) que contiene un termómetro.
<u>LOWTaskFilterR0</u> <u>LowTremorLimitations</u> 	LOWTaskFilterR	Muestra el enlace con el nombre del diagrama que describe las limitaciones cuando el nivel del síntoma asociado es bajo. El icono consiste en un embudo con las ruedas dentadas representando que filtra las tareas. El color amarillo representa que es para niveles bajos del síntoma.
<u>MEDIUMTaskFilterR1</u> <u>MedTremorLimitations</u> 	MEDIUMTaskFilterR	Muestra el enlace con el nombre del diagrama que describe las limitaciones cuando el nivel del síntoma asociado es medio. El icono consiste en un embudo con las ruedas dentadas representando que filtra las tareas. El color naranja representa que es para niveles medios del síntoma.
<u>HIGHTaskFilterR0</u> <u>HighTremorLimitations</u> 	HIGHTaskFilterR	Muestra el enlace con el nombre del diagrama que describe las limitaciones cuando el nivel del síntoma asociado es alto. El icono consiste en un embudo con las ruedas dentadas representando que filtra las tareas. El color rojo representa que es para niveles altos del síntoma.

Tabla A.11: Tabla de la notación para los objetos que modelan la enfermedad del Parkinson.

Notación	Meta-entidad	Descripción
<u>MedicationTime0</u> (At 13:00) 	MedicationTime	Muestra la hora a la que se debe realizar la toma. El icono consiste en una píldora y un reloj.
<u>MedIntake1</u> Amount=2 	MedIntake	Muestra la dosis o cantidad de pastillas que debe tomar el paciente. El icono consiste en una cara de perfil y una píldora con la flecha hacia la boca.
<u>MedInjection0</u> Amount=1 	MedInjection	Al igual que el anterior, muestra la cantidad a inyectarse. El icono consiste en una píldora y una jeringuilla.
<u>Medication2</u> Name=Sinemet 	Medication	Muestra el nombre del medicamento. El icono consiste en el cubo lila que representa a los objetos con el dibujo de una píldora en una de sus caras.

Tabla A.12: Tabla de la notación para los objetos correspondientes a la toma de medicamentos.

Notación	Meta-entidad	Descripción
 NONESympLevelState0	NONESympLevelState LOWSympLevelState MEDIUMSympLevelState HIGHSympLevelState	El icono consiste en un círculo con un termómetro representando el estado del nivel del síntoma. El color del círculo depende del nivel del síntoma: verde no manifiesta síntoma, amarillo el nivel del síntoma es bajo, el naranja indica que el nivel del síntoma es medio y el rojo que es alto
 LOWSympLevelState0		
 MEDIUMSympLevelState0		
 HIGHSympLevelState2		

Tabla A.13: Tabla de la notación para los objetos correspondientes a describir la evolución de los síntomas.

Notación	Meta-entidad	Descripción
 FTaskSelectorFilter5 ByType = Yes	FTaskSelectorFilter	Muestra el atributo para indicar si se filtran las tareas por tipo o instancia concreta. El icono consiste en un embudo y unas ruedas dentadas representando la selección de tareas.
 FDelayFilter0 Task duration * (200/100)	FDelayFilter	Muestra en qué proporción retrasa o acelera la tarea. El icono consiste en un reloj y dos flechas a izquierda y derecha representando la variación del tiempo. La que apunta a la izquierda de color verde representa acortamiento en la tarea, y la que apunta a la derecha de color rojo representa un retraso en la tarea.
 FReplaceTaskFilter2 Press b6 instead of b3	FReplaceTaskFilter	Muestra el enlace con el nombre del diagrama de tareas que reemplazará a la tarea filtrada. El icono consiste en dos símbolos de tarea. El primero representa a la tarea de entrada y por ello está tachada y el segundo representa la tarea que reemplazará a la primera.
 FModifyPlaceFilter0 Place = BedRoom1	FModifyPlaceFilter	Muestra el identificador del nuevo lugar donde se dirigirá el personaje. El icono representa un campo con una símbolo de posición.
 FChangeTargetObjFilter0 Object = ArmChair1	FChangeTargetObjFilter	Muestra el identificador del objeto objetivo a cambiar. El icono consiste en una "T" de tarea y un cubo representando al objeto.
 FallTask0	FallTask	El icono consiste en un monigote que se va a caer porque tropieza con algo en el suelo.
 DropObj0	DropObj	El icono consiste en una mano de la cual sale una flecha roja que acaba en un objeto representado que antes estaba en la mano y ahora no porque se le ha caído.
 CObjWeight0 lessThan 2000 gr.	CObjWeight	Muestra la condición de comparación con el peso y el peso del objeto en gramos. El icono consiste en una caja lila símbolo de los objetos con una "g" de gramos.

Tabla A.14: Tabla de la notación para los objetos correspondientes a los periodos del día.

A.2.6. Los símbolos del diagrama de simulación

Notación	Meta-entidad	Descripción
<u>WorldInitialization0</u> Seed=0 (House3room2bath) 	WorldInitialization	Muestra el valor de la semilla que permite que los comportamientos sean repetibles y entre paréntesis el nombre de la casa 3D. El icono consiste en una esfera del mundo con la palabra "INIT" encima.
<u>TimeToBed23:00</u> 1/1/2015 at 23:0:0 	SDate	Muestra la fecha y la hora de la simulación. El icono consiste en un reloj representando la hora y un calendario representando la fecha.
<u>FlyCamInit0</u> 720x480 Pos=0,9,0 Rot=90,0,0 	FlyCamInit	Muestra la resolución deseada de la pantalla y la posición (en metros) y rotación de la cámara (en grados). El icono consiste en una cámara y la palabra "INIT" en la parte superior.
<u>HumanInitialization2</u> ShowName=Yes 	HumanInitialization	Muestra el atributo para indicar si se desea ver el nombre o no en la simulación. El icono consiste en la palabra "INIT", después muestra un monigote y los iconos de actividad y posición.
<u>SymInit0</u>  MEDIUM	SymInit	Muestra el nivel inicial del síntoma. El icono se parece a la estrella de los síntomas pero está hueca y muestra la palabra "Init".
<u>CameraInit1</u> Distance= 5 Elevation=1.0 Front=No 	CameraInit	Muestra los valores donde irá posicionada la cámara respecto al personaje, es decir, la distancia en horizontal y la elevación, ambos en metros. El valor <i>Front</i> indica si la cámara se situará enfrente del personaje o detrás. El icono se compone de las palabras "INIT" en la parte superior y debajo el icono de una cámara apuntando a un monigote.
<u>PartOfBody0</u> Attach to Chest of E3Patient 	PartOfBody	Describe en qué parte del cuerpo y de qué persona se situará el dispositivo. El icono consiste en un monigote negro con líneas rojas indicando partes del cuerpo.
<u>InitialPreDefPos0</u> In P1 of Table1 	InitialPreDefPos	Muestra el identificador de la posición relativa a un objeto. El icono consiste en la palabra "INIT" seguido de un cubo lila representando el objeto y un símbolo de localización sobre el objeto.
<u>ESmartPhone0</u> 	ESmartPhone	Muestra el identificador del dispositivo Android. El icono consiste en el dibujo de un teléfono inteligente y un cuadrado lila indicando que es un objeto.
<u>AndroidEmulator0</u> AUT= app.apk AvdName=Smartphone2 Serial=emulator-5554 AvdScreenFeed=Yes 	AndroidEmulator	Muestra el nombre y la ruta relativa donde se encuentra la aplicación a instalar en el emulador ("AUT=", Application Under Test), el nombre del dispositivo Android Virtual o Emulador ("AvdName="), el número serie del emulador ("Serial=") y también muestra si se desea ver o no la pantalla del emulador en su dispositivo virtual correspondiente. El icono consiste en una captura de un emulador Android.
<u>TVRemoteActivity</u> net.sf.sociala/TVRemoteActivity 	AndroidApplication	Muestra el nombre del paquete y el Activity de la aplicación que se desea que se arranque. El icono consta de las palabras "APP" en la parte superior (abreviatura de Application) y del monigote de Android con dos ruedas dentadas representando que es una aplicación funcional.

Tabla A.15: Tabla de la notación para los objetos correspondientes para definir los escenarios.

Apéndice B

API de los comandos de PHAT

RESUMEN: El apéndice contiene los comandos definidos por PHAT para acceder a cada una de las funcionalidades de sus módulos. Estos comandos se presentan agrupados en función del módulo al que va dirigido.

B.1. El módulo del entorno

B.1.1. El entorno exterior de la vivienda

La clase principal que implementa y ofrece las funcionalidades requeridas es *WorldAppState*. Esta ofrece una interfaz muy sencilla para crear el entorno exterior:

- `setLandType(LandType landType)`: método para seleccionar el tipo de terreno a crear. Hay definidos 3 tipos de terrenos: `Basic`, `Grass` y `TwoHouses`. El `Basic` consiste en un terreno plano de un color uniforme, el `Grass` es igual pero con una textura de césped y el último `TwoHouses` es el más elaborado e incluye información para añadir dos viviendas.
- `setCalendar( int year, int month, int dayOfMonth, int hour, int minute, int second)`: método para indicar la hora y la fecha de la simulación. PHAT implementa un calendario virtual (`PHATCalendar`) que una vez iniciado, avanza en función de la simulación y actualiza la iluminación de la escena simulando el movimiento del sol.

B.1.2. El interior de la vivienda

HouseAppState es la clase de PHAT encargada de gestionar las viviendas virtuales. Los comandos que ofrece *HouseAppState* son:

- `CreateHouseCommand(String id, HouseFactory.HouseType houseType)`. Crea una vivienda del tipo *houseType* con identificador *id*. Debido a que es relativamente costoso crear una vivienda para la simulación, de momento solo hay una disponible.
- `SetTransparentLevelOfWallsCommand(String id, float level)` Dado el identificador (*id*) de una vivienda hace que dichos muros sean transparentes según el nivel indicado (*level*). Esto facilita que el observador pueda visualizar en todo momento al sujeto virtual y no deje de verlo a causa de que se encuentra detrás de un muro.
- `SwitchLight(String id, String roomName, boolean on)`. Enciende o apaga (*on*) las luces de una habitación (*roomName*) de una vivienda dada (*id*). Este comando podría permitir que tanto actuadores como sujetos virtuales puedan encender y apagar las luces de la vivienda.
- `DebugShowHouseNavMeshCommand(boolean enable)`. Comando de depuración que muestra la malla de navegación. Resulta útil solo para descartar problemas de que un sujeto virtual no es capaz de llegar a su destino a causa de que no se creó correctamente la malla de navegación, ver Figura 4.6.

B.2. El módulo dedicado a los cuerpos de los sujetos virtuales

Comandos de inicialización:

- `CreateBodyTypeCommand(String bodyId, String urlResource)`. Comando que crea un cuerpo con un identificador y el modelo 3D que se pase como parámetro. De momento, los modelos 3D de sujetos virtuales son los de la Figura 4.8b.
- `SetBodyInCoordinatesCommand(String bodyId, Vector3f location)`. Comando para situar el cuerpo indicado en las coordenadas dadas por el vector.
- `SetBodyInHouseSpaceCommand(String bodyId, String houseId, String spaceId)`. Comando para situar el cuerpo en el lugar de la casa indicada.
- `SetBodyColorCommand(String bodyId, ColorRGBA color)`. Comando para cambiar el color del sujeto virtual. Disminuye realismo pues pasa a ser un muñeco todo del mismo color. Es útil si se desea identificar rápidamente tipos de sujetos virtuales o para representar su estado. Aunque el lenguaje no utiliza este comando.
- `SetBodyHeightCommand(String bodyId, float height)`. Comando para especificar la altura del cuerpo del sujeto virtual en metros.

- `SetPCListenerToBodyCommand(String bodyId)`. Comando para seleccionar el sujeto virtual a partir del cual se escuchará la escena durante la simulación.

Comandos relacionados con los desplazamientos:

- `RemoveBodyFromSpaceCommand(String bodyId)`. Comando para quitar el sujeto virtual del escenario. Este comando es útil para simular el momento en el que un sujeto virtual abandona la casa.
- `SetSpeedDisplacemenetCommand(String bodyId, float speed)`. Comando para modificar la velocidad a la que se desplaza el cuerpo del sujeto virtual (en m/s).
- `GoCloseToBodyCommand(String bodyId, String targetBodyId)`. Comando para que el cuerpo se desplace a la posición donde se encuentra el cuerpo con identificador dado.
- `GoCloseToObjectCommand(String bodyId, String targetObjectId)`. Comando para que el sujeto virtual se desplace hasta una posición próxima a un objeto.
- `GoIntoBedCommand(String bodyId, String placeId)`. Comando para hacer que el cuerpo permanezca en posición horizontal en la cama indicada.
- `GoToSpaceCommand(String bodyId, String spaceId)`. Comando para que el sujeto virtual vaya a la habitación indica en el parámetro.
- `RotateTowardCommand(String bodyId, String entityId)`. Comando para hacer que el cuerpo gire hasta que se encuentre mirando hacia el objeto indicado.
- `AlignWithCommand(String bodyId, String entityId)`. Comando para hacer que el cuerpo del sujeto virtual mire hacia la misma dirección que el objeto dado. Por ejemplo, antes de sentarse el sujeto virtual debe mirar en la misma dirección que la silla.
- `FallDownCommand(String bodyId)` Comando para hacer que el cuerpo indicado se desplome simulando una caída. Para que resulte convincente utilizar el controlador `KinematicRagdolControl` que hace uso de la física, ver Figura 4.10.

Comandos que permiten al sujeto virtual interaccionar:

- `OpenObjectCommand(String bodyId, String objectId)`. Comando genérico para que un cuerpo abra el objeto dado (`objectId`). Dependiendo del objeto, significará una cosa u otra. Por ejemplo, abrir una puerta, abrir un grifo o abrir una luz (encenderla).
- `CloseObjectCommand(String bodyId, String objectId)`. Comando genérico para que un cuerpo cierre el objeto dado (`objectId`). Al igual que el anterior, dependiendo del objeto significará una cosa u otra. Por ejemplo, cerrar una puerta, cerrar un grifo o cerrar una luz (apagarla).

- `PickUpCommand(String bodyId, String entityId, Hand hand)`. Comando para que el cuerpo coja el objeto con la mano izquierda o derecha, según se indique.
- `SayASentenceBodyCommand(String bodyId, String message)`. Comando para que el cuerpo `bodyId` diga la frase que se pasa como parámetro (`message`). Este comando hace uso de la funcionalidad `TestToSpeech` del módulo de audio.
- `SitDownCommand(String bodyId, String placeId)`. Comando para que el cuerpo se siente en la silla indicada.
- `StandUpCommand(String bodyId)` Comando para que el cuerpo se ponga de pie independientemente de si se encuentra tumbado en la cama, sentado o tirado en el suelo.
- `WaitForCloseToBodyCommand(String bodyId, String targetBodyId)`. Comando para indicar al sujeto virtual que no haga nada hasta que otro sujeto virtual (el que tenga el identificador `targetBodyId`) se encuentre próximo al mismo. Este comando puede servir para simular la asistencia del paciente.

Comandos para que el sujeto virtual realice gestos:

- `PlayBodyAnimationCommand(String bodyId, String animationName)`. Comando para que el cuerpo reproduzca una de las animaciones de la Figura 4.9b.
- `LookAtCommand(String bodyId, String targetId)`. Comando para hacer que la cabeza del sujeto virtual mire al objeto indicado. Esta funcionalidad da realismo y ayuda a saber dónde está manteniendo el foco el sujeto virtual.
- `SetShortStepsCommand(String bodyId, Boolean on)`. Comando para hacer que los pasos del sujeto virtual sean más cortos. Esto es un síntoma típico de los enfermos de párkinson, acortan los pasos y arrastran los pies.
- `SetStoopedBodyCommand(String bodyId, Boolean on)`. Comando para cambiar la postura del cuerpo del sujeto virtual. Hace que adopte la postura encorvada típica de algunas personas mayores y de los enfermos de párkinson.
- `SetRigidArmCommand(String bodyId, Boolean on, Boolean left)`. Comando que anula la animación del brazo indicado. Los enfermos de párkinson van perdiendo el balanceo de los brazos al andar.
- `TremblingHandCommand(String bodyId, Boolean on, Boolean left)`. Comando para hacer que tiemble la mano izquierda o derecha del sujeto virtual según se indique.
- `TremblingHeadCommand(String bodyId, Boolean on)`. Comando para hacer mover la cabeza del sujeto virtual. También es un síntoma común a todos los enfermos de párkinson.

Commands para la depuración de las simulaciones:

- `AttachIconCommand(String bodyId, String imagePath, Boolean show)`. Comando que permite mostrar u ocultar un icono. Dicho icono aparecerá encima de la cabeza del sujeto virtual. La idea es que el icono ayude al observador a identificar lo que está haciendo en cada momento el sujeto virtual. Se aprovecharán los iconos del lenguaje SociAALML de forma que se sabrá qué tarea está realizando el sujeto virtual en cada momento.
- `BodyLabelCommand(String bodyId, Boolean show)`. Comando para mostrar u ocultar una etiqueta con el nombre del sujeto virtual sobre su cabeza. Se puede usar para ayudar a identificar quién es cada sujeto virtual, ya que de momento la apariencia de los sujetos virtuales es muy limitada.
- `ShowLabelsOfVisibleObjectsCommand(String bodyId, boolean on)`. Comando que muestra el nombre de los objetos que el sujeto virtual está viendo en cada momento.
- `SetCameraToBodyCommand(String bodyId)`. Comando que hace que la cámara siga al cuerpo indicado. Esta funcionalidad es útil cuando se desea mantener el foco todo el rato en un solo sujeto virtual.

B.3. El módulo dedicado a los dispositivos

Al igual que con el resto de elementos, los dispositivos los gestiona un `AppState`, ver Figura 4.3. Los comandos que ofrece esta clase para dar soporte a crear simulaciones con dispositivos virtuales son:

- `CreateSmartphoneCommand(String id)`. Comando para crear un dispositivo con un identificador dado. Además se puede especificar las dimensiones usando el método `setDimensions()`. Dispone de métodos para indicar con qué sensores se quiere que el dispositivo esté dotado en su creación (de momento micrófono, cámara y acelerómetro).
- `SetDeviceInCoordinatesCommand(String id, Vector3f location)`. Comando para situar el móvil en una posición específica en el mundo virtual.
- `SetDeviceOnFurnitureCommand(String id, String houseId, String furnitureId)`. Comando para situar el móvil encima del mueble (`furnitureId`) de la vivienda (`houseId`).
- `SetDeviceOnPartOfBodyCommand(String bodyId, String deviceId, PartOfBody partOfBody)`. Comando para asociar el dispositivo a una parte del cuerpo del sujeto virtual. Las partes del cuerpo que se permiten son: “LeftHand”, “RightHand”,

“LeftWrist”, “RightWrist”, “LeftUnkle”, “RightUnkle”, “Head” y “Chest”. En la Figura 4.15, puede verse un dispositivo en la muñeca izquierda haciendo de reloj y otro más grande en el pecho.

- `SetImageOnScreenCommand(String id, String imagePath)`. Comando para hacer que en la pantalla del dispositivo virtual se muestre una imagen especificada en el parámetro `imagePath`.
- `CreatePresenceSensorCommand(String presenceSensorId)`. Comando para crear el sensor de presencia con identificador dado. Ofrece métodos para variar gran cantidad de parámetros: frecuencia de muestreo, distancia de detección, ángulo horizontal y vertical de cobertura, modificar la orientación y la precisión (cantidad de rayos). El modelo de la casa debe tener creados un conjunto de Nodos para los sensores de presencia. El comando busca dichas Nodos y crea el sensor en la posición y orientación del Nodo que coincida con el `id` dado. En la Figura ?? puede comprobarse un ejemplo de distribución de los sensores de presencia.
- `CreateDoorSensorCommand(String objectId, String doorSensorId)`. Comando para crear un sensor de puerta con identificador `doorSensorId` asociado a la puerta con identificador `objectId`.

B.4. El módulo dedicado a los servidores

Al igual que con el resto de elementos, los servicios y la interacción con el emulador los gestiona un `AppState` llamado `ServerAppState`, ver Figura 4.3. Estas funcionalidades se ofrecen de forma muy sencilla a través de los siguientes comandos:

- `SetAndroidEmulatorCommand(String id, String avdId, String se)`. Comando para asociar el dispositivo Android virtual con identificador `id` al emulador Android con nombre `avdId` y número serie `se`. De esta forma, las aplicaciones en el Emulador recibirán los valores de los sensores del dispositivo virtual dado (`id`) tal y como se ha descrito en la Figura 4.17.
- `DisplayAVDScreenCommand(String id, String avdId)`. Comando para hacer que se muestre en la pantalla del dispositivo virtual `id` lo que aparece en el emulador `avdId`. Esta funcionalidad aporta realismo a la simulación y permite identificar la correspondencia entre dispositivos virtuales y emuladores.
- `InstallApkCommand(String id, String apkFile)`. Instala la aplicación Android (sistema AAL) especificada en la ruta `apkFile` en el dispositivo `id`. Este comando permite que se puedan instalar diferentes aplicaciones. Así se facilita que se puedan simular escenarios donde varíe el software instalado en los dispositivos.
- `StartActivityCommand(String id, String packageName, String activityName)`. Comando para que el Emulador asociado al dispositivo indicado inicie la aplicación indicada por el nombre del paquete `packageName` y de la clase `activityName`.

En Android, un Activity corresponde a una interfaz gráfica con la cual el usuario puede acceder a servicios. Por ejemplo, una aplicación de mando a distancia universal podría estar compuesta por varios Activities, uno por cada mando. Este comando permite que desde el simulador se inicien aplicaciones, ya sea como parte de la inicialización de una simulación o resultado de una interacción del sujeto virtual.

- `PressOnScreen(String id, int x, int y)`. Comando para simular que se pulsa la pantalla del dispositivo en las coordenadas `x`, `y` dadas. Manda el evento al emulador correspondiente. Permite que los sujetos virtuales interactúen con las interfaces gráficas de los dispositivos.

Apéndice C

Esquemas descriptivos de la generación del código

RESUMEN: El apéndice contiene los esquemas que describen cómo a partir de modelos de SociAALML se genera el código de simulación en PHAT.

El código del generador no se considera relevante. Para ilustrar el proceso de traducción, se va a mostrar, por cada plantilla, cómo a partir de elementos del modelo de SociAALML, se generan las clases Java expresadas en notación UML. Se emplea notación UML ya que esta resulta más compacta y visual. Los valores de los campos de las entidades de SociAALML son la fuente principal de información empleada para completar las plantillas, es decir, dichos valores se asignan a las variables de las plantillas. Para facilitar la visualización de dicha correspondencia, los campos de las entidades de SociAAML que se visualizan en el editor son etiquetadas con los nombres de las variables de la plantilla. Hay que tener en cuenta que existen entidades cuyas propiedades no se visualizan en el editor. En tal caso, las variables hacen referencia a su identificador. Otro caso especial es la generación del código de las entidades condición de SociAALML. Para este caso, el generador posee un método al cual se le pasa una entidad de tipo condición y este devuelve una cadena de texto que corresponde a la sentencia Java que crea dicha condición. En la plantilla, las variables se resaltan en negrita. En el diagrama UML, también se resaltan aquellos valores que provienen de las variables de la plantilla.

C.1. Plantilla para generar las clases principales de la simulación

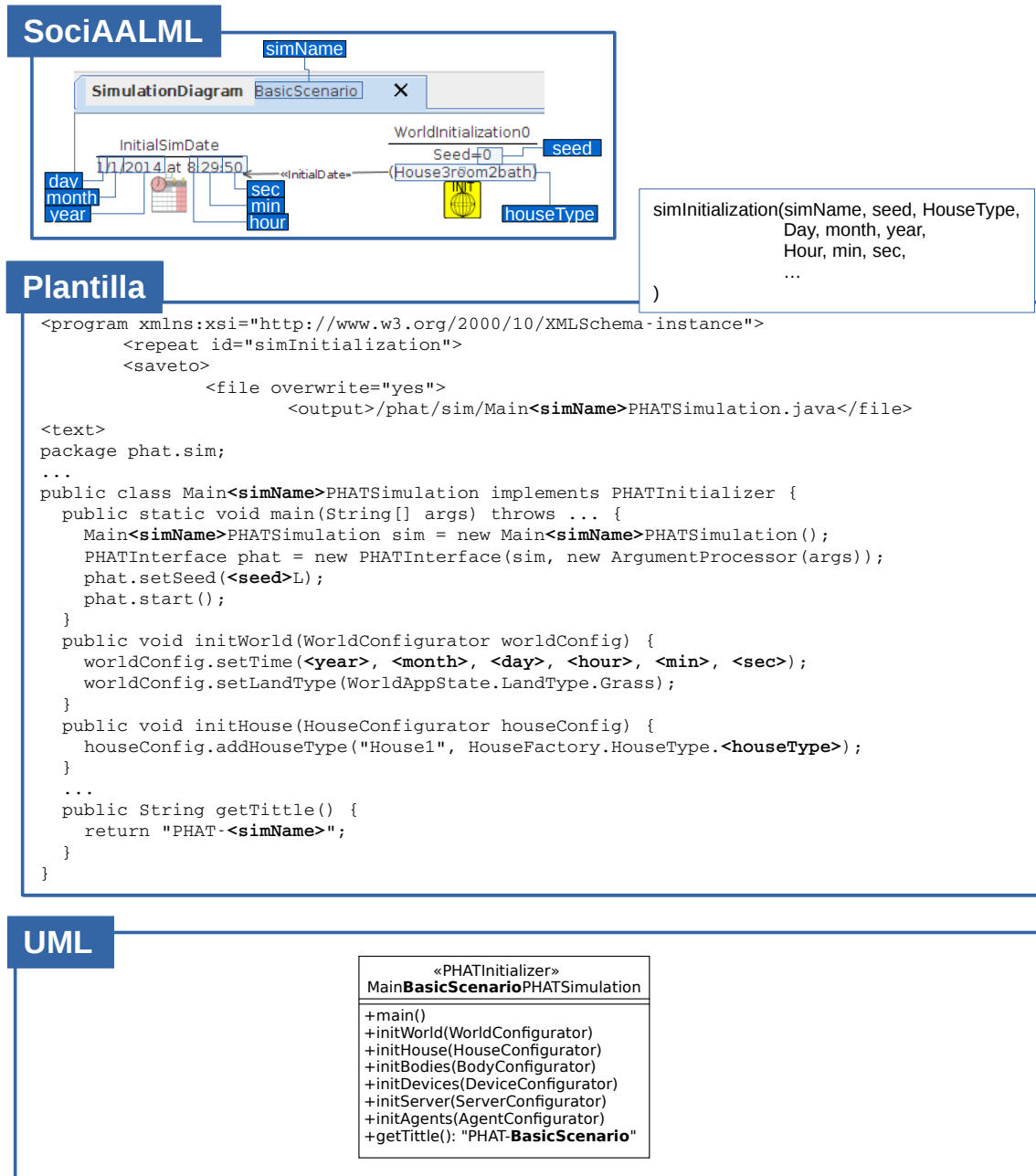


Figura C.1: Parte de la plantilla de las simulaciones que corresponde a la inicialización de la fecha, la hora, la semilla y el modelo 3D de la vivienda.

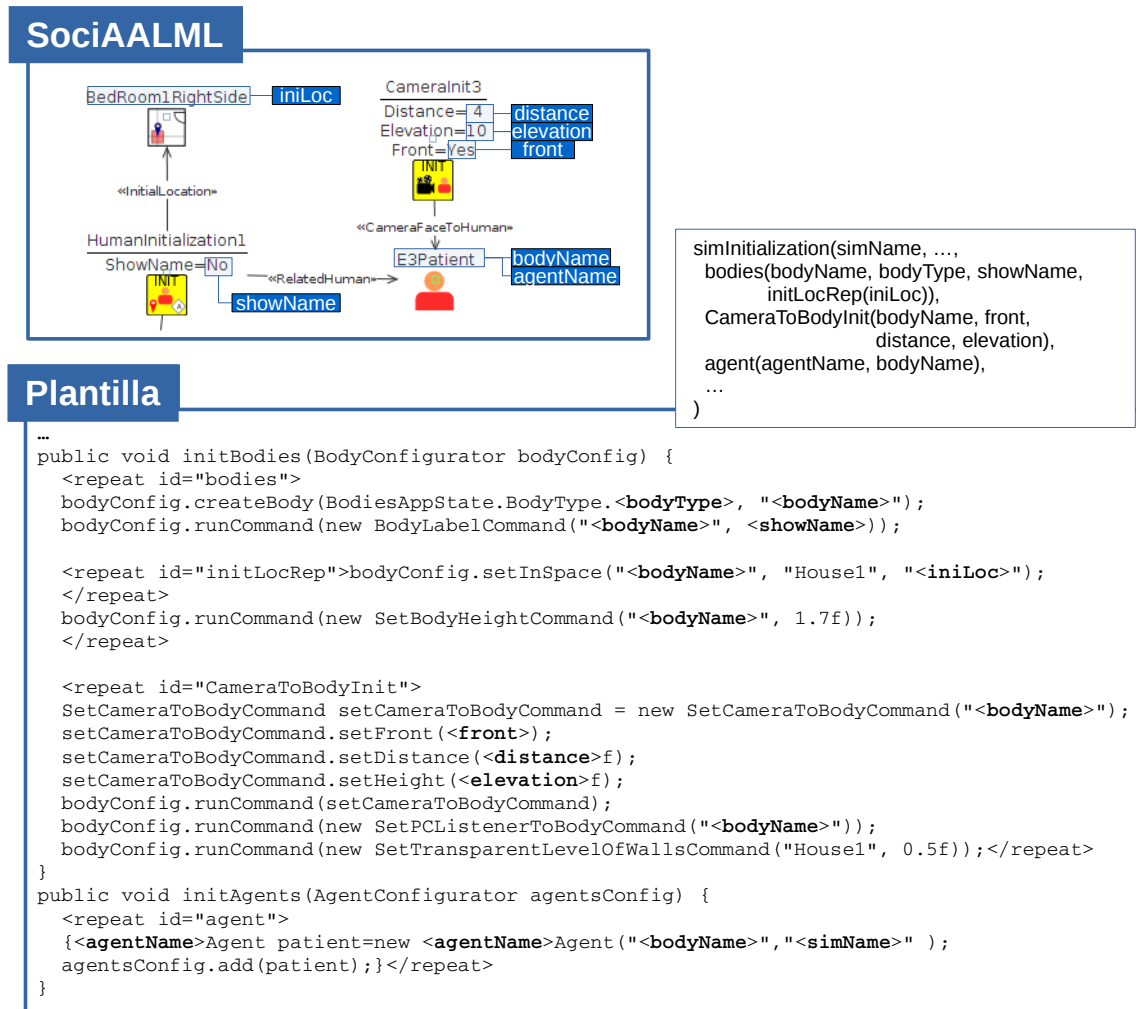


Figura C.2: Parte de la plantilla de las simulaciones que corresponde a la inicialización de los sujetos virtuales.

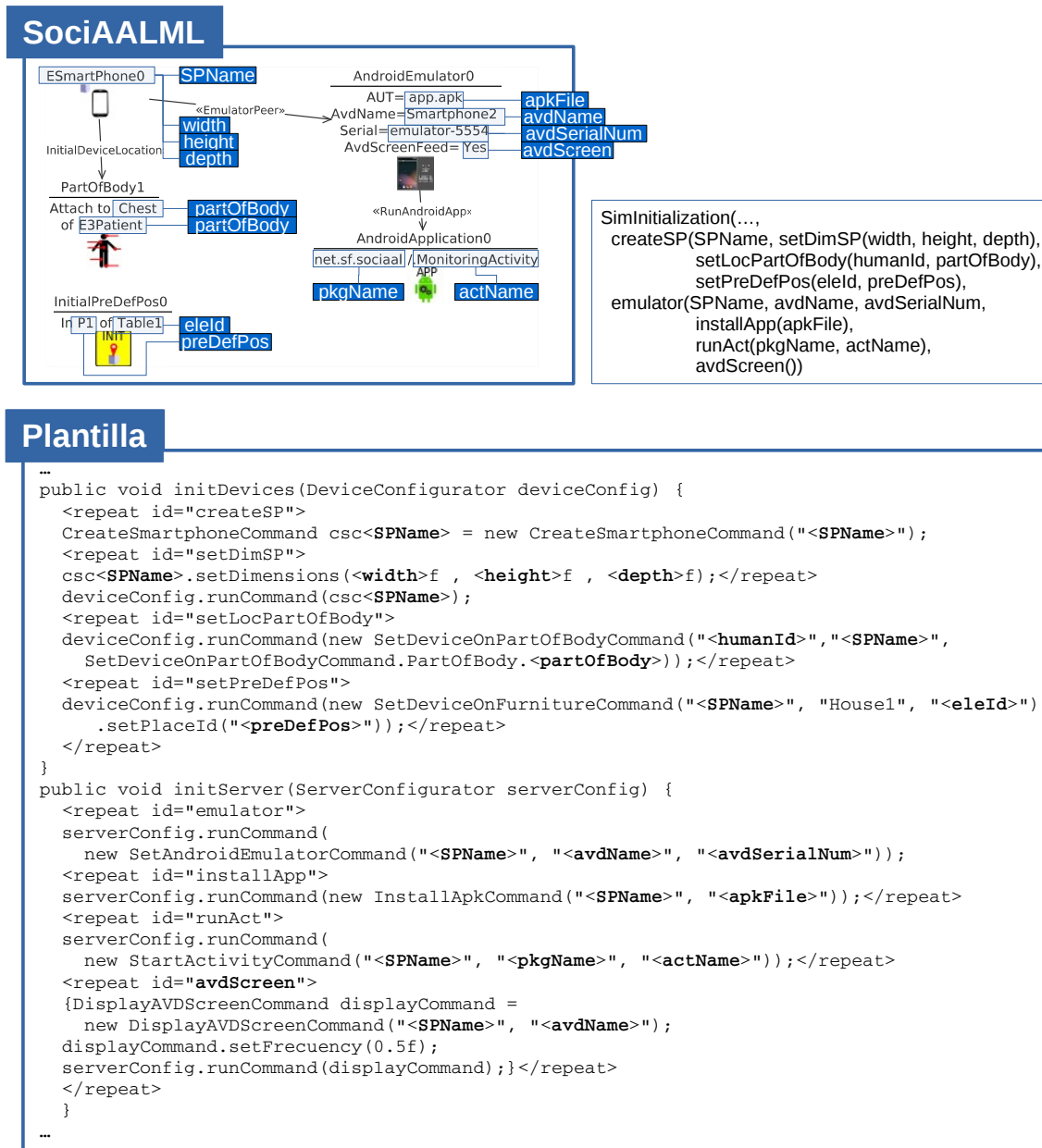


Figura C.3: Parte de la plantilla de las simulaciones que corresponde a la inicialización de los dispositivos.

C.2. Plantilla para generar los agentes

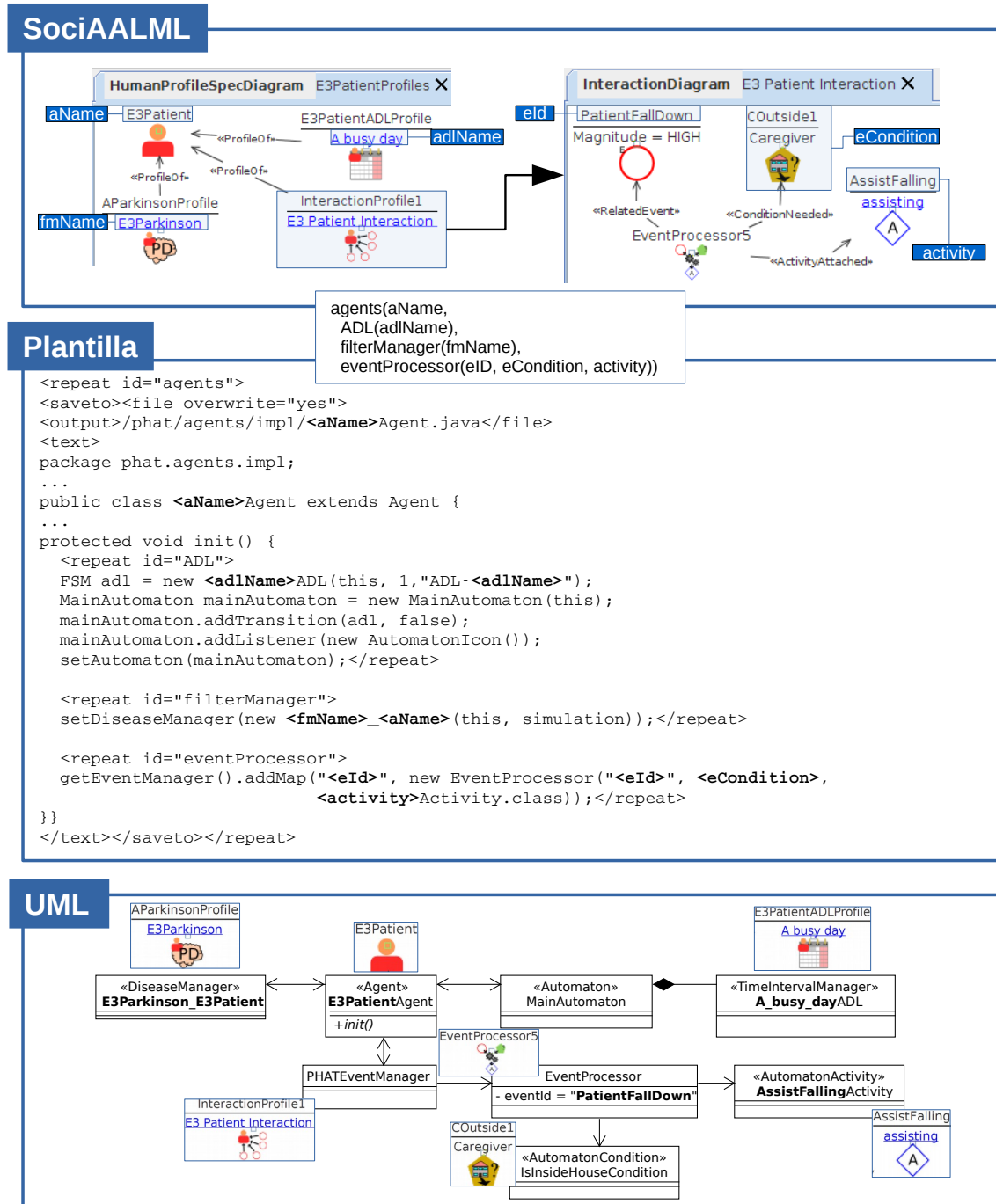


Figura C.4: Plantilla para generar los agentes e instanciar a su ADL, su gestor de eventos y la enfermedad.

C.3. Plantilla para gestionar los momentos del día

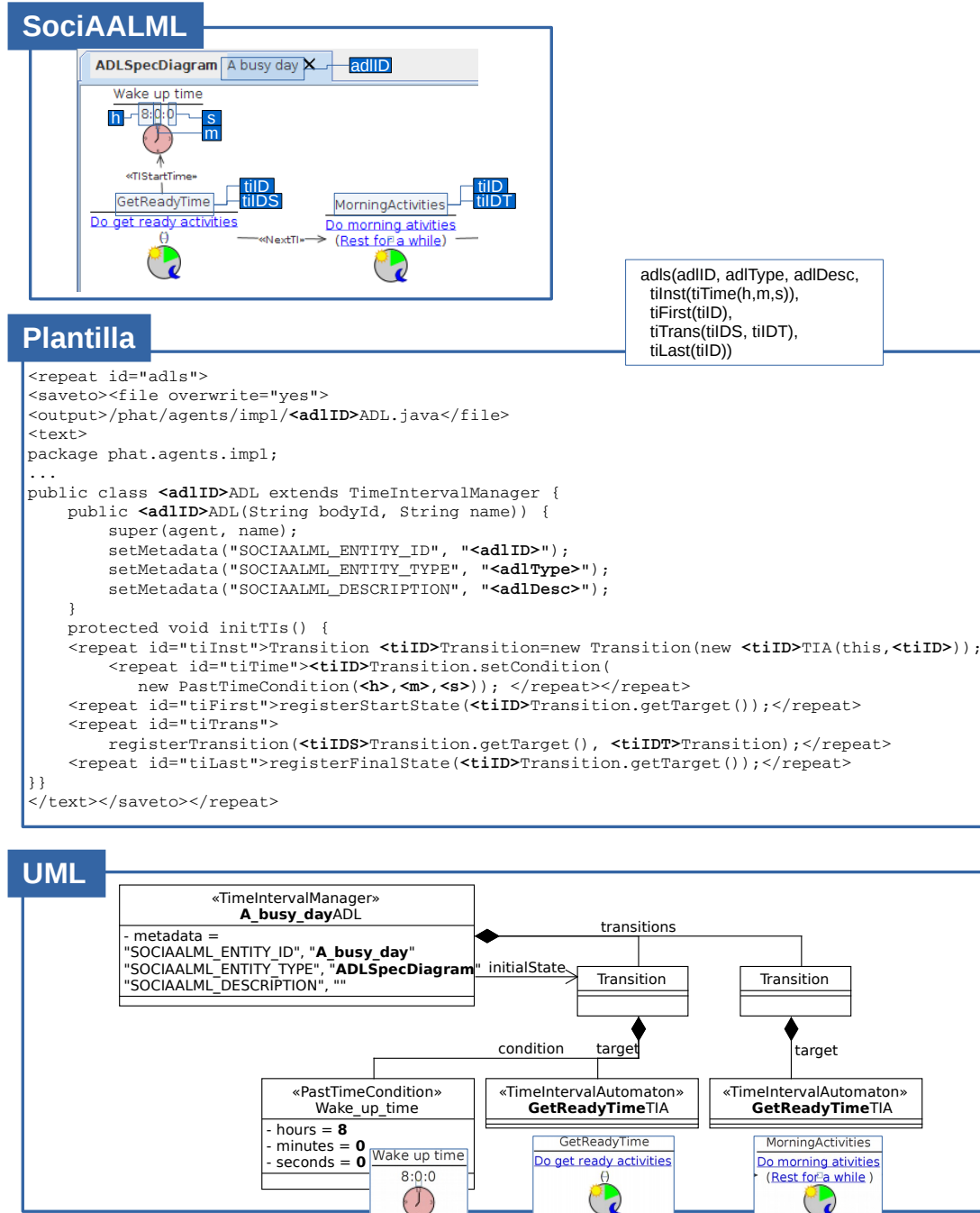


Figura C.5: Plantilla para generar los periodos del día que conforman las actividades diarias de la persona.

C.4. Plantilla para generar los momentos del día con sus actividades

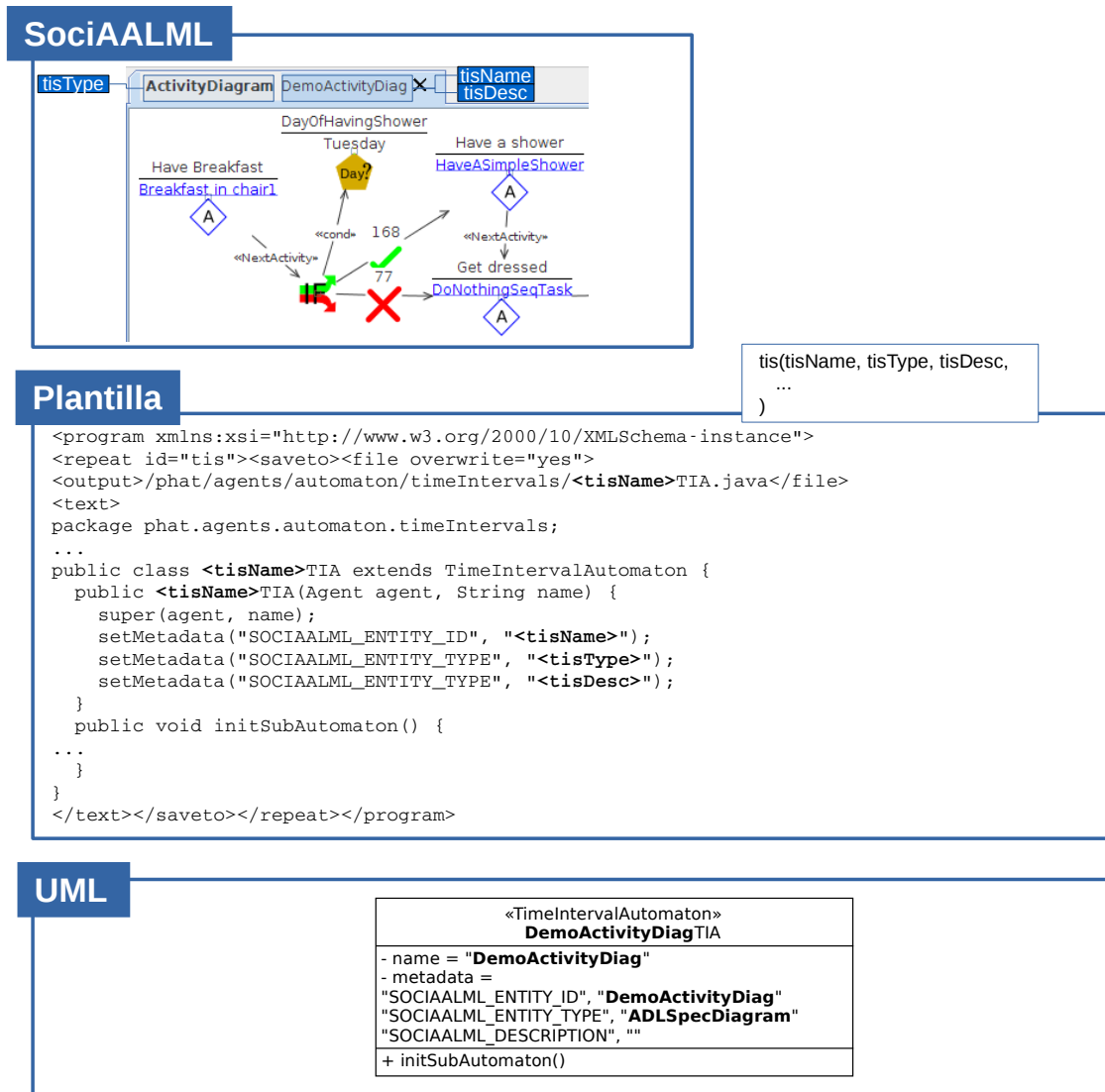


Figura C.6: Parte de la plantilla para definir la clase que definirá las actividades de un periodo de tiempo.

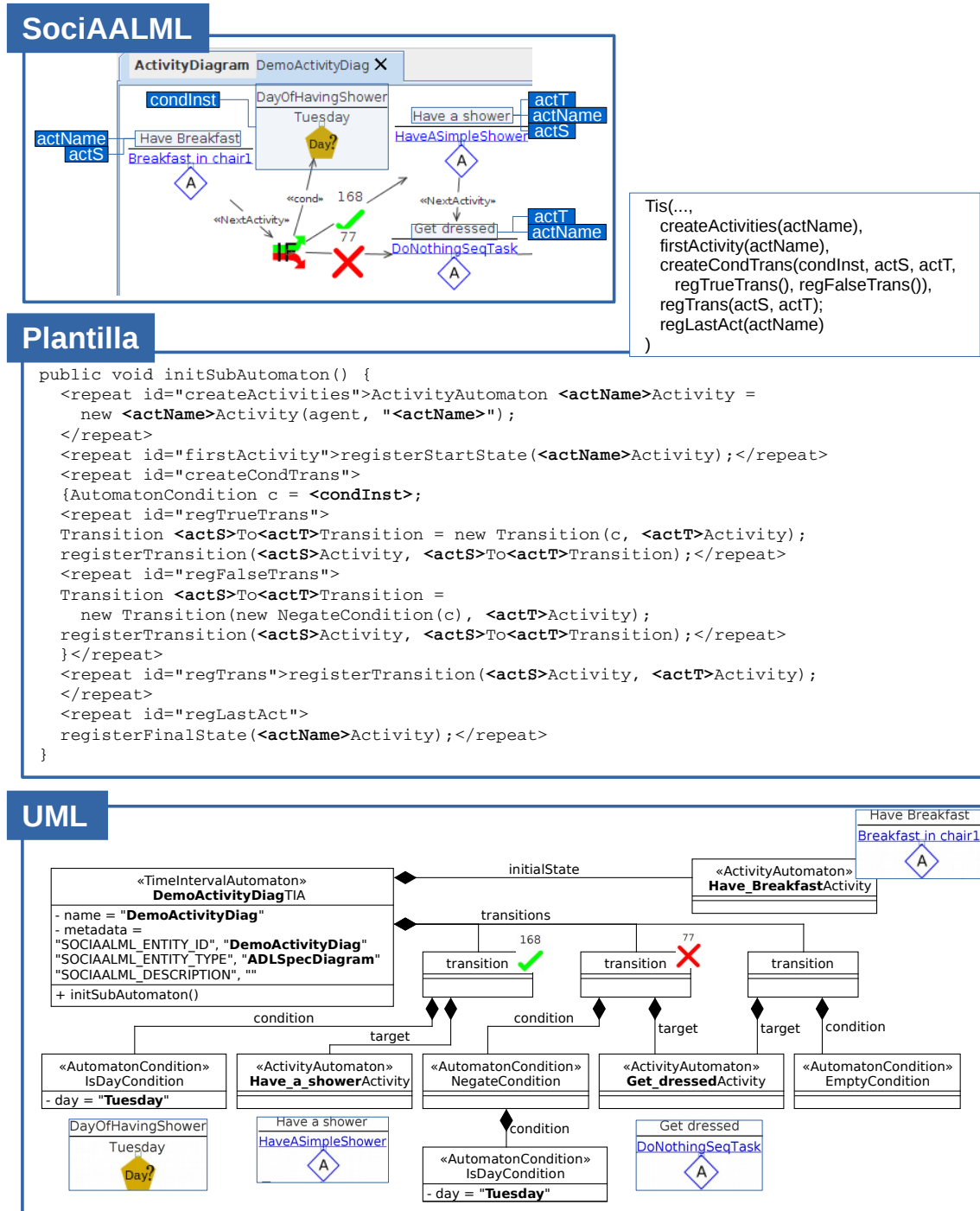


Figura C.7: Parte de la plantilla que crea las actividades y sus relaciones para un periodo del día.

C.5. Plantilla para la generación de las actividades

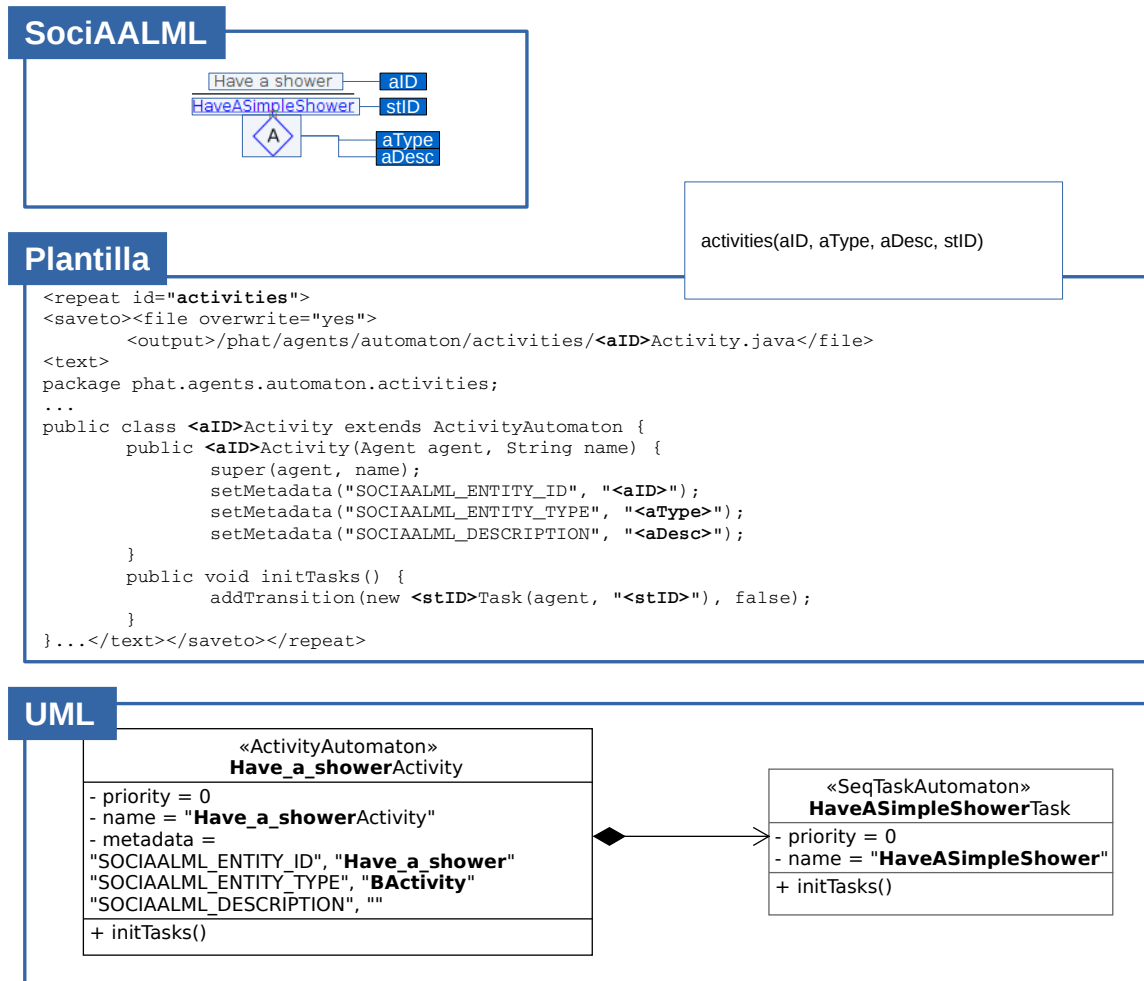


Figura C.8: Plantilla para la generación de las actividades.

C.6. Plantilla para generar las secuencias de tareas

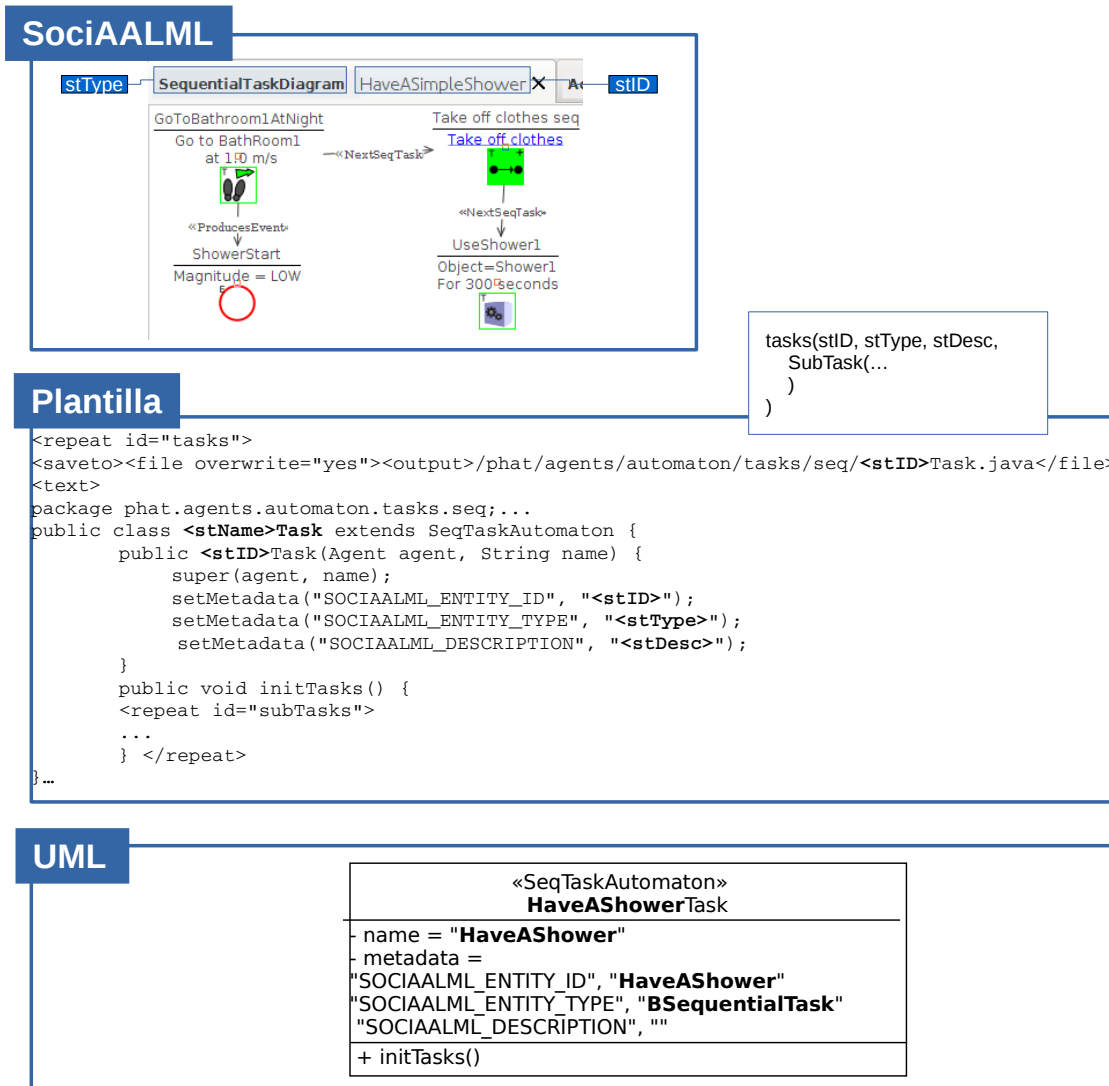
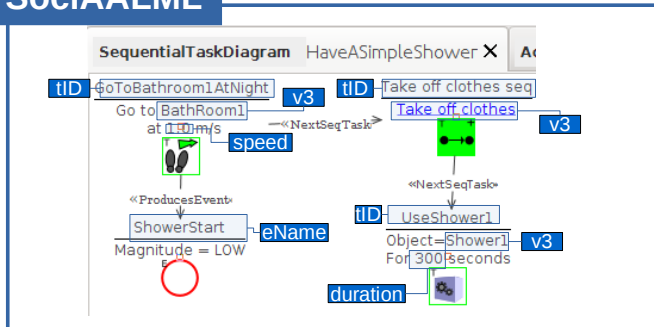


Figura C.9: Parte de la plantilla que genera la clase y el constructor de una secuencia de tareas.

SociAALML



```

Tasks(...,
  subTask(tID, tType, tDesc, cName,
    eParam(v3),
    durRep(duration)
    speedRep(speed)
    setXY(x,y),
    events(eName)
  )
)

```

Plantilla

```

<repeat id="tasks">
...
public void initTasks() {
<repeat id="subTasks">
  {Automaton automaton = new <className>(
    agent, "<className>"
    <repeat id="3params">, "<v3>"<repeat id="xyParams">, <x>, <y></repeat></repeat>
  ).setCanBeInterrupted(<interrup>)
  .setMetadata("SOCIAALML_DESCRIPTION", "<tDesc>")
  .setMetadata("SOCIAALML_ENTITY_ID", "<tID>")
  .setMetadata("SOCIAALML_ENTITY_TYPE", "<tType>");
  addTransition(automaton, false);
  <repeat id="durRep">
    automaton.setFinishCondition(new TimerFinishedCondition(0, 0, <duration>));</repeat>
  <repeat id="speedRep">(<className> automaton).setSpeed(<speed>f);</repeat>
  <repeat id="events">
    PHATAudioEventAutomatonFinishedListener audioEvent =
      new PHATAudioEventAutomatonFinishedListener(
        agent, "<eventName>", new AgentEventSource(agent));
    automaton.addListener(audioEvent);</repeat>
  } </repeat>
} ...

```

UML

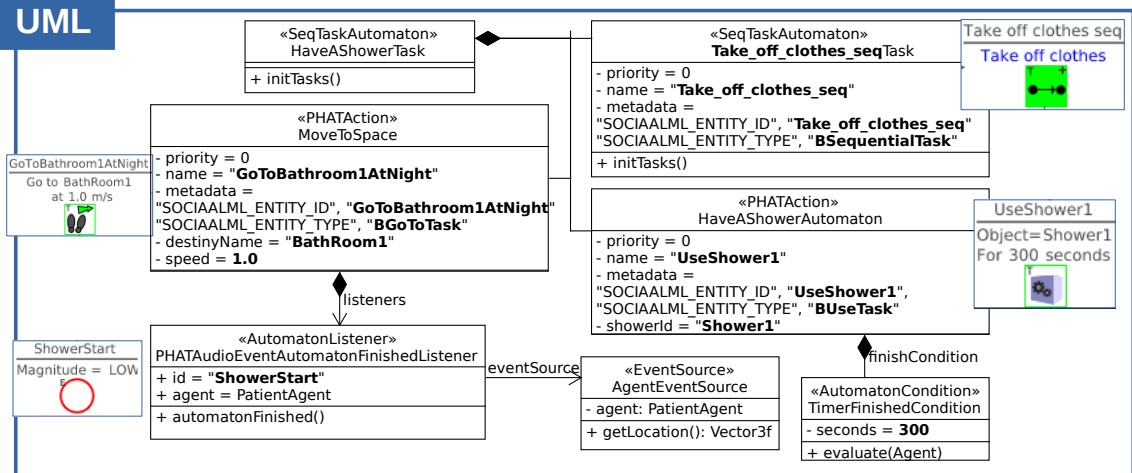


Figura C.10: Parte de la plantilla que genera las subtareas de una secuencia de tareas.

C.7. Plantilla para generar la clase encargada de implementar la enfermedad

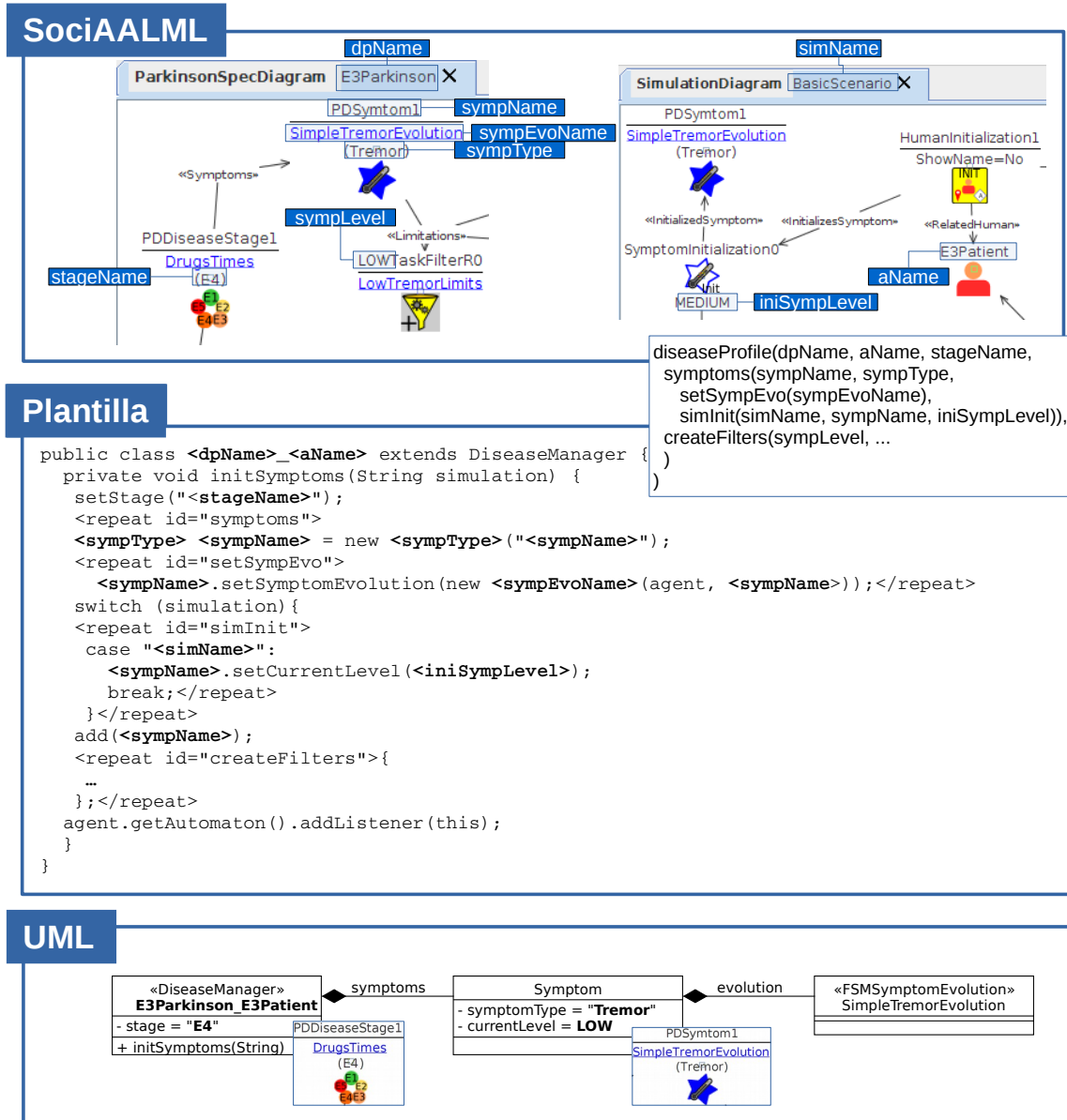


Figura C.11: Parte de la plantilla para crear la clase que gestiona la enfermedad añadiendo los síntomas y estableciendo sus niveles iniciales.

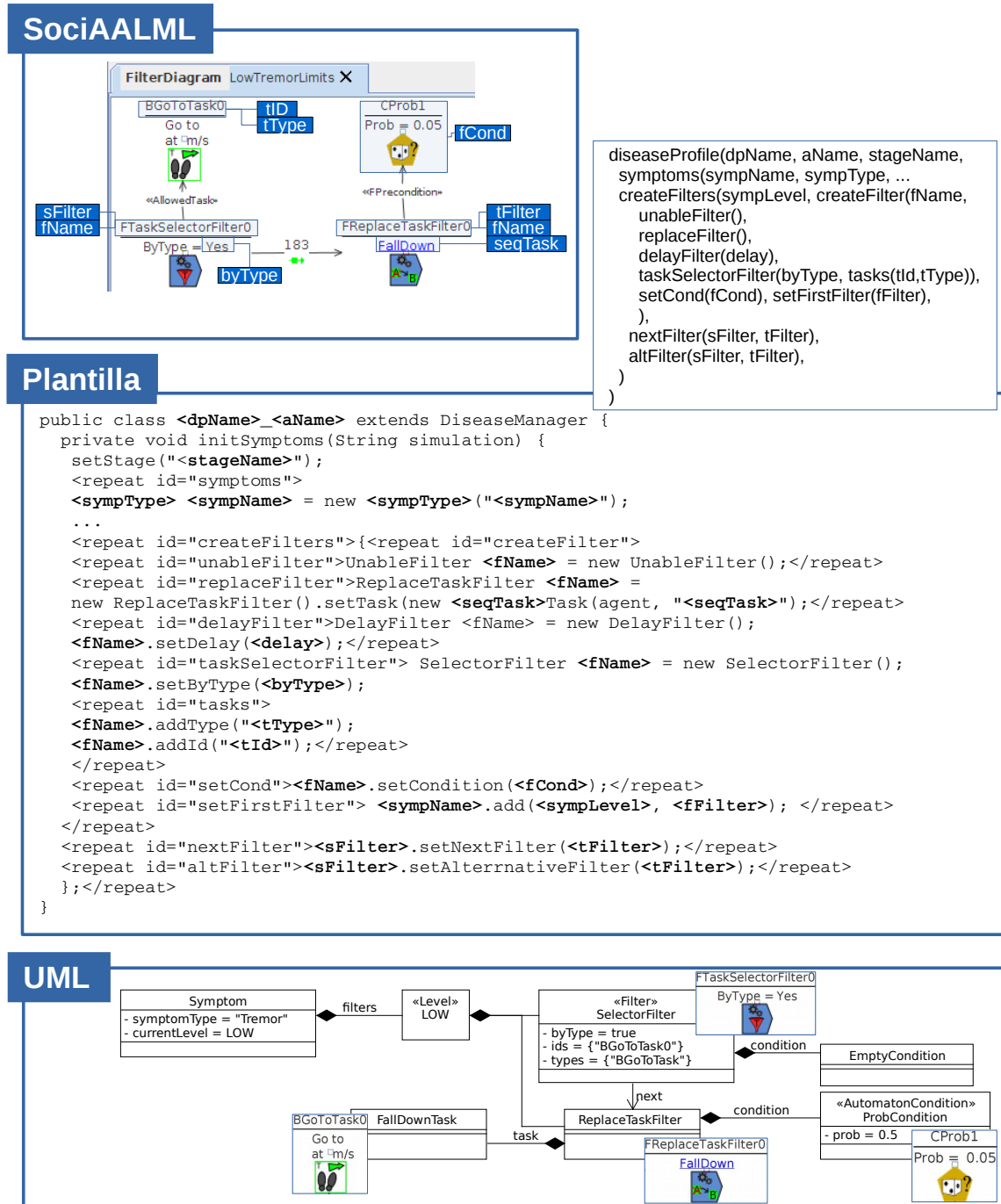


Figura C.12: Parte de la plantilla que crea las cadenas de filtros y las añade a los síntomas según el nivel.

C.8. Plantilla para generar las clases que implementan la evolución de los niveles de los síntomas

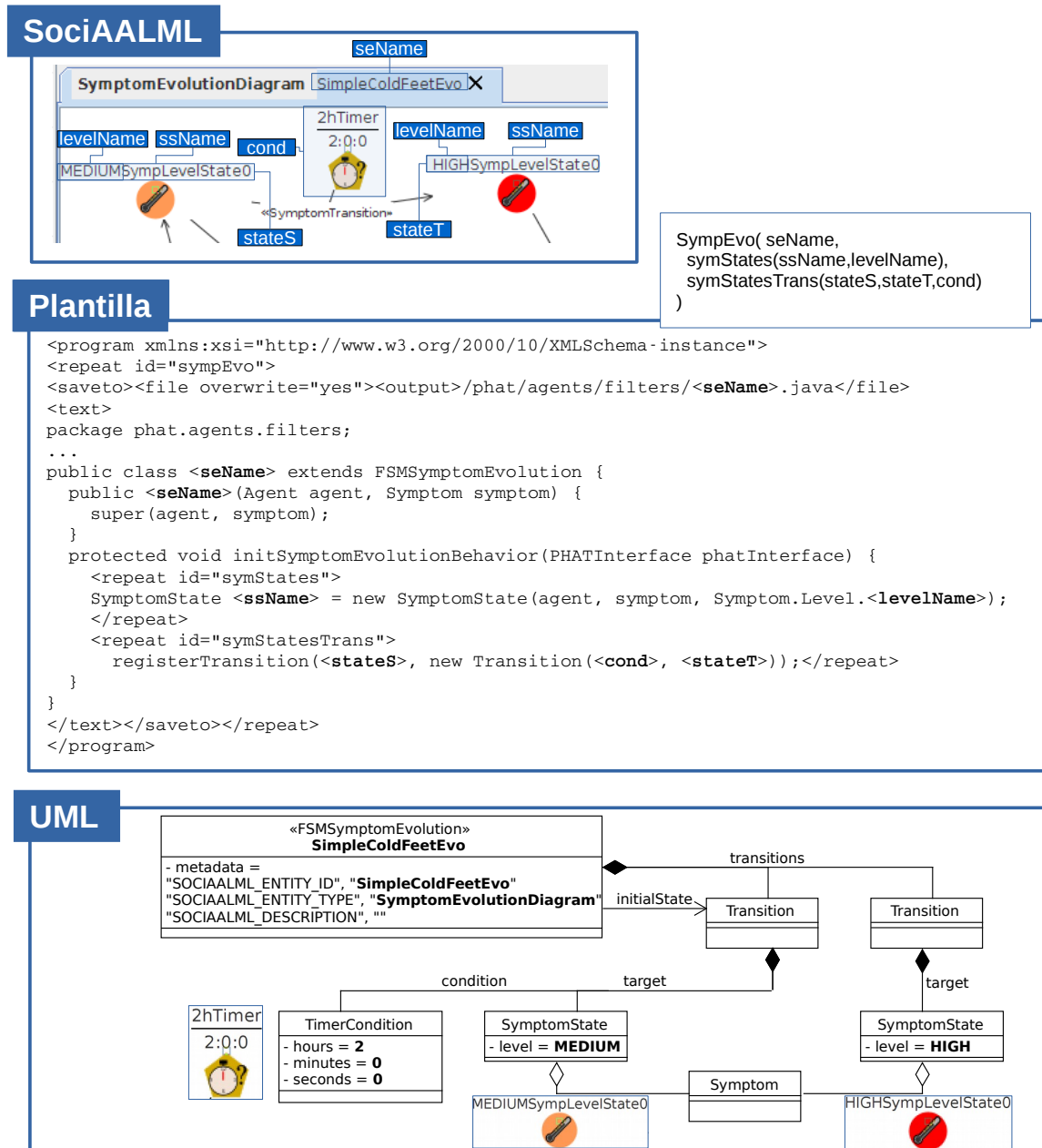


Figura C.13: Plantilla para generar el código relativo a la evolución de los síntomas.

Bibliografía

*Y así, del mucho leer y del poco dormir, se le
secó el cerebro de manera que vino a perder el
juicio.*

Miguel de Cervantes Saavedra

- [1] E. Aarts and B. de Ruyter. New research perspectives on ambient intelligence. *Journal of Ambient Intelligence and Smart Environments*, 1(1):5–14, 2009.
- [2] E. Aarts and R. Wichert. *Ambient intelligence*. Springer, 2009.
- [3] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.
- [4] F. Al Machot, H. C. Mayr, and J. Michael. Behavior modeling and reasoning for ambient support: Hcm-l modeler. In *Modern Advances in Applied Intelligence*, pages 388–397. Springer, 2014.
- [5] M. Arroyo, L. Finkel, and J. Gomez-Sanz. Requirements for an intelligent ambient assisted living application for parkinson patients. In J. Corchado, J. Bajo, J. Kozlak, P. Pawlewski, J. Molina, V. Julian, R. Silveira, R. Unland, and S. Giroux, editors, *Highlights on Practical Applications of Agents and Multi-Agent Systems*, volume 365 of *Communications in Computer and Information Science*, pages 441–452. Springer Berlin Heidelberg, 2013.
- [6] D. Avison and G. Fitzgerald. *Information systems development: methodologies, techniques and tools*. McGraw Hill, 2003.
- [7] S. Azhar. Building information modeling (bim): Trends, benefits, risks, and challenges for the aec industry. *Leadership and Management in Engineering*, 2011.
- [8] S. Azhar, A. Nadeem, J. Y. Mok, and B. H. Leung. Building information modeling (bim): A new paradigm for visual interactive modeling and simulation for construction projects. In *Proc., First International Conference on Construction in Developing Countries*, pages 435–446, 2008.

- [9] W. S. Bainbridge. The scientific research potential of virtual worlds. *science*, 317(5837):472–476, 2007.
- [10] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.
- [11] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *Pervasive computing*, pages 1–17. Springer, 2004.
- [12] J. Barton and V. Vijayaraghavan. Ubiwise: A ubiquitous wireless infrastructure simulation environment. *HP Labs*, 2002.
- [13] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. Base-a micro-broker-based middleware for pervasive computing. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 443–451. IEEE, 2003.
- [14] J. Bertin. *Semiology of graphics: diagrams, networks, maps*. 1983.
- [15] B. Bertran, J. Bruneau, D. Cassou, N. Lorient, E. Balland, and C. Consel. Diasuite: A tool suite to develop sense/compute/control applications. *Science of Computer Programming*, 79:39–51, 2014.
- [16] T. Bosse, M. Hoogendoorn, M. Klein, and J. Treur. A component-based ambient agent model for assessment of driving behaviour. In F. Sandnes, Y. Zhang, C. Rong, L. Yang, and J. Ma, editors, *Ubiquitous Intelligence and Computing*, volume 5061 of *Lecture Notes in Computer Science*, pages 229–243. Springer Berlin Heidelberg, 2008.
- [17] K. Bouchard, A. Ajroud, B. Bouchard, and A. Bouzouane. Simact: A 3d open source smart home simulator for activity recognition. In T.-h. Kim and H. Adeli, editors, *Advances in Computer Science and Information Technology*, volume 6059 of *Lecture Notes in Computer Science*, pages 524–533. Springer Berlin Heidelberg, 2010.
- [18] A. Bourke, J. O’Brien, and G. Lyons. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. *Gait & Posture*, 26(2):194 – 199, 2007.
- [19] H. Braak, E. Ghebremedhin, U. RÄ¼b, H. Bratzke, and K. Del Tredici. Stages in the development of parkinson’s disease-related pathology. *Cell and Tissue Research*, 318(1):121–134, 2004.
- [20] J. Bruneau and C. Consel. DiaSim: a simulator for pervasive computing applications. *Software: Practice and Experience*, 43(8):885–909, Aug. 2013.
- [21] M. Bylund and F. Espinoza. Testing and demonstrating context-aware services with quake iii arena. *Commun. ACM*, 45(1):46–48, Jan. 2002.

- [22] M. Cabo Díez, F. J. Ferrández Pastor, F. Flórez Revuelta, V. Romacho Agud, et al. Dai middleware: plataforma de hogar digital para la provisión de servicios orientados a la vida asistida por el entorno. In *IV International Congress on Design, Research Networks, and Technology for all*. Fundación ONCE, 2011.
- [23] P. Campillo-Sanchez and J. Botia. Simulation based software development for smart phones. In P. Novais, K. Hallenborg, D. I. Tapia, and J. M. C. Rodríguez, editors, *Ambient Intelligence - Software and Applications*, volume 153 of *Advances in Intelligent and Soft Computing*, pages 243–250. Springer Berlin Heidelberg, 2012.
- [24] P. CAMPILLO-SÁNCHEZ, J. A. BOTÍA, and J. GÓMEZ-SANZA. Development of sensor based applications for the android platform: an approach based on realistic simulation. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 2(4):23–28, 2013.
- [25] P. Campillo-Sanchez and J. Gómez-Sanz. Agent based simulation for creating ambient assisted living solutions. In Y. Demazeau, F. Zambonelli, J. Corchado, and J. Bajo, editors, *Advances in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*, volume 8473 of *Lecture Notes in Computer Science*, pages 319–322. Springer International Publishing, 2014.
- [26] P. Campillo-Sánchez and J. Gómez-Sanz. Modelling and simulation of alzheimer’s disease scenarios. *Procedia Computer Science*, 83:353–360, 2016.
- [27] P. Campillo-Sanchez and J. J. Gomez-Sanz. A framework for developing multi-agent systems in ambient intelligence scenarios. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’15, pages 1949–1950, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.
- [28] P. Campillo-Sanchez, J. J. Gómez-Sanz, and J. A. Botía. Phat: Physical human activity tester. In *Hybrid Artificial Intelligent Systems*, pages 41–50. Springer, 2013.
- [29] P. Campillo-Sanchez, E. Serrano, and J. A. Botía. Testing context-aware services based on smartphones by agent based social simulation. *Journal of Ambient Intelligence and Smart Environments*, 5:330, 01/2013 2013.
- [30] F. Campuzano, E. Serrano, and J. A. Botía. Towards socio-chronobiological computational human models. In *Advances in Artificial Intelligence-IBERAMIA 2012*, pages 392–401. Springer, 2012.
- [31] R. Castano, G. Dotto, R. Suma, A. Martina, and A. Bottino. Virtual-me: A library for smart autonomous agents in multiple virtual environments. In *Agent Technology for Intelligent Mobile Services and Smart Societies*, pages 34–45. Springer, 2015.

- [32] A. Champandard. Behavior trees for next-gen game ai. In *Game Developers Conference, Audio Lecture*, 2007.
- [33] L. M. Cunningham, C. D. Nugent, D. D. Finlay, G. Moore, and D. Craig. A review of assistive technologies for people with parkinson’s disease. *Technol. Health Care*, 17(3):269–279, Aug. 2009.
- [34] W. Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1993.
- [35] J. CurĂn and J. Kleindienst. SitCom: Virtual smart-room environment for multi-modal perceptual systems. In N. T. Nguyen, L. Borzinski, A. Grzech, and M. Ali, editors, *New Frontiers in Applied Artificial Intelligence*, number 5027 in Lecture Notes in Computer Science, pages 476–485. Springer Berlin Heidelberg, Jan. 2008.
- [36] A. M. Davis. Operational prototyping: A new development approach. *Software, IEEE*, 9(5):70–78, 1992.
- [37] J. M. F. de Alba, P. Campillo, R. Fuentes-FernĂndez, and J. Pavón. Opportunistic control mechanisms for ambience intelligence worlds. *Expert Systems with Applications*, 41(4, Part 2):1875 – 1884, 2014.
- [38] T. DeMarco. *Structured analysis and system specification*. Yourdon Press, 1979.
- [39] S. Diewald, L. Roalter, A. Möller, and M. Kranz. Towards a holistic approach for mobile application development in intelligent environments. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia*, MUM ’11, pages 73–80, New York, NY, USA, 2011. ACM.
- [40] C. Eastman, C. M. Eastman, P. Teicholz, and R. Sacks. *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons, 2011.
- [41] J. Favela, M. Tentori, L. A. Castro, V. M. Gonzalez, E. B. Moran, and A. I. Martínez-García. Activity recognition for context-aware hospital applications: Issues and opportunities for the deployment of pervasive networks. *Mob. Netw. Appl.*, 12(2-3):155–171, Mar. 2007.
- [42] J. FernĂndez-de Alba, P. Campillo, R. Fuentes-FernĂndez, and J. Pavón. Opportunistic sensor interpretation in a virtual smart environment. In H. Yin, J. Costa, and G. Barreto, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, volume 7435 of *Lecture Notes in Computer Science*, pages 109–116. Springer Berlin Heidelberg, 2012.
- [43] F. FerrĂndez-Pastor, J. GarcĂa-Chamizo, M. Nieto-Hidalgo, and F. Flórez-Revuelta. Dai virtual lab: a virtual laboratory for testing ambient intelligence digital services. In *V Congreso Internacional de DiseĂo, Redes de Investigación y Tecnología para todos, DRT4ALL*, pages 163–169, 2013.

- [44] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent agents III agent theories, architectures, and languages*, pages 21–35. Springer, 1997.
- [45] M. Friedewald, O. Da Costa, Y. Punie, P. Alahuhta, and S. Heinonen. Perspectives of ambient intelligence in the home environment. *Telematics and informatics*, 22(3):221–238, 2005.
- [46] J. J. Gomez-Sanz and P. Campillo-Sanchez. Achieving parkinson’s disease patient autonomy through regulative norms. In J. Bajo, J. Z. Hernández, P. Mathieu, A. Campbell, A. Fernández-Caballero, M. N. Moreno, V. Julián, A. Alonso-Betanzos, M. D. Jiménez-López, and V. Botti, editors, *Trends in Practical Applications of Agents, Multi-Agent Systems and Sustainability*, volume 372 of *Advances in Intelligent Systems and Computing*, pages 97–104. Springer International Publishing, 2015.
- [47] J. J. Gomez-Sanz, P. Campillo-Sanchez, and J. Pavón. Sociaal: Un entorno 3d para la simulación de soluciones aal. In *Congreso internacional turismo para todos*.
- [48] J. J. Gomez-Sanz, P. Campillo-Sanchez, and J. Pavón. Sociaalml: Lenguaje de modelado para escenarios de inteligencia ambiental. In M. Canós, J. H. y González-Harbour, editor, *Actas de las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015)*. Sistedes Biblioteca Digital.
- [49] J. J. Gómez-Sanz, M. Cardenas, R. Pax, and P. Campillo. Building prototypes through 3d simulations. *Advances in Practical Applications of Scalable Multi-agent Systems. The PAAMS Collection*, page 299, 2016.
- [50] J. J. Gomez-Sanz and P. C. Sánchez. Domain independent regulative norms for evaluating performance of assistive solutions. *Pervasive and Mobile Computing*, pages –, 2016.
- [51] J. J. Gómez-Sanz and P. C. Sánchez. A multiple data stream management framework for ambient assisted living emulation. In *Intelligent Interactive Multimedia Systems and Services 2016*, pages 695–703. Springer, 2016.
- [52] P. Goolkasian. Pictures, words, and sounds: From which format are we best able to reason? *The Journal of General Psychology*, 127(4):439–459, 2000.
- [53] J. Gregory. *Game engine architecture*. Ak Peters Series. A K Peters, Limited, 2009.
- [54] T. Gross. Towards a new human-centred computing methodology for cooperative ambient intelligence. *Journal of Ambient Intelligence and Humanized Computing*, 1(1):31–42, 2010.

- [55] M. Heidari, E. Allameh, B. de Vries, H. Timmermans, J. Jessurun, and F. Mozaffar. Smart-bim virtual prototype implementation. *Automation in Construction*, 39:134–144, 2014.
- [56] M. Heidari Jozam, E. Allameh, B. De Vries, H. Timmermans, and M. Masoud. Vr-smart home. In S. Andreev, S. Balandin, and Y. Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networking*, volume 7469 of *Lecture Notes in Computer Science*, pages 107–118. Springer Berlin Heidelberg, 2012.
- [57] A. Helal, K. Cho, W. Lee, Y. Sung, J. Lee, and E. Kim. 3d modeling and simulation of human activities in smart spaces. In *2012 9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC)*, pages 112–119, Sept. 2012.
- [58] K. Henriksen and J. Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and mobile computing*, 2(1):37–64, 2006.
- [59] S. Herrlich, S. Spieth, R. Nouna, R. Zengerle, L. I. Giannola, D. E. Pardo-Ayala, E. Federico, and P. Garino. Ambulatory treatment and telemonitoring of patients with parkinson’s disease. In *Ambient Assisted Living*, pages 295–305. Springer, 2011.
- [60] M. M. Hoehn, M. D. Yahr, et al. Parkinsonism: onset, progression, and mortality. *Neurology*, 50(2):318–318, 1998.
- [61] J.-y. Hong, E.-h. Suh, and S.-J. Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4):8509–8522, 2009.
- [62] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *Networking, IEEE/ACM Transactions on*, 11(1):2–16, 2003.
- [63] D. Isla. Handling complexity in the halo 2 ai. In *Game Developers Conference*, volume 12, 2005.
- [64] J. Jankovic. Parkinson’s disease: clinical features and diagnosis. *Journal of Neurology, Neurosurgery & Psychiatry*, 79(4):368–376, 2008.
- [65] K. Jensen, L. M. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.
- [66] M. Kallmann, J.-S. Monzani, A. Caicedo, and D. Thalmann. *Ace: A platform for the real time simulation of virtual human agents*. Springer, 2000.

- [67] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, and T. Jämsä. Comparison of low-complexity fall detection algorithms for body attached accelerometers. *Gait & Posture*, 28(2):285 – 291, 2008.
- [68] M. Kapadia, A. Shoulson, F. Durupinar, and N. I. Badler. Authoring multi-actor behaviors in crowds with diverse personalities. In *Modeling, Simulation and Visual Analysis of Crowds*, pages 147–180. Springer, 2013.
- [69] A. Katifori, C. Vassilakis, I. Daradimos, G. Lepouras, Y. Ioannidis, A. Dix, A. Poggi, and T. Catarci. Personal ontology creation and visualization for a personal interaction management system. In *Proceedings of PIM Workshop, CHI*, page 15, 2008.
- [70] S. Kelly, K. Lyytinen, and M. Rossi. Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Advanced Information Systems Engineering*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin Heidelberg, 1996.
- [71] S. Kelly and J.-P. Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.
- [72] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner, and W. Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Cooperative buildings. Integrating information, organizations, and architecture*, pages 191–198. Springer, 1999.
- [73] T. Kleinberger, M. Becker, E. Ras, A. Holzinger, and P. Müller. Ambient intelligence in assisted living: enable elderly people to handle future interfaces. In *Universal access in human-computer interaction. Ambient interaction*, pages 103–112. Springer, 2007.
- [74] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, Mar. 2011.
- [75] J. H. Larkin and H. A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11(1):65 – 100, 1987.
- [76] W. Lee, S. Cho, W. Song, K. Um, and K. Cho. UbiSim: Multiple sensors mounted smart house simulator development. In *2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 450–453, Dec. 2013.
- [77] Z. Lei, S. Yue, C. Yu, and S. Yuanchun. SHSim: An OSGI-based smart home simulator. In *2010 3rd IEEE International Conference on Ubi-media Computing (U-Media)*, pages 87–90, July 2010.

- [78] A. N. Leontév. *Activity, consciousness, and personality*. Prentice-Hall, Englewood Cliffs, N.J., 1978.
- [79] J. Lertlakkhanakul, J. W. Choi, and M. Y. Kim. Building data model and simulation platform for spatial interaction management in smart home. *Automation in Construction*, 17(8):948–957, Nov. 2008.
- [80] M. Lewis and J. Jacobson. Game engines in scientific research - introduction. *Commun. ACM*, 45(1):27–31, 2002.
- [81] Q. Li, J. Stankovic, M. Hanson, A. Barth, J. Lach, and G. Zhou. Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information. In *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, pages 138–143, June 2009.
- [82] E. Martignoni, A. Citterio, R. Zangaglia, L. Godi, C. Pacchetti, C. Fundaro, E. Corengia, G. Bono, and G. Nappi. How parkinsonism influences life: the patients’ point of view. *Neurological Sciences*, 32(1):125–131, 2011.
- [83] J. J. Martínez, T. García-Valverde, F. Campuzano, P. Campillo-Sánchez, A. García-Sola, and J. A. Botía. Multi-agent based social simulation applied to validation of location services., 2012.
- [84] S. Martínez, L. Marín, A. P. Madrid, and C. M. O. Social. *Cuidarse para cuidar mejor en la enfermedad de Parkinson*. Biblioteca Parkinson. Asociación Parkinson Madrid, 2007.
- [85] A. H. Maslow. A theory of human motivation. *Psychological review*, 50(4):370, 1943.
- [86] H. C. Mayr and J. Michael. Control pattern based analysis of hcm-1, a language for cognitive modeling. In *Advances in ICT for Emerging Regions (ICTer), 2012 International Conference on*, pages 169–175. IEEE, 2012.
- [87] K. McGlinn, L. Hederman, and D. Lewis. SimCon: A context simulator for supporting evaluation of smart building applications when faced with uncertainty. *Pervasive and Mobile Computing*, 12:139–159, June 2014.
- [88] K. McGlinn, E. O’Neill, A. Gibney, D. O’Sullivan, and D. Lewis. Simcon: A tool to support rapid evaluation of smart building application design using context simulation and virtual reality. *J. UCS*, 16(15):1992–2018, 2010.
- [89] R. L. McIntyre. *Recognizing Actions using Embodiment & Empathy*. PhD thesis, Massachusetts Institute of Technology, 2014.
- [90] J. Michael and H. C. Mayr. Conceptual modeling for ambient assistance. In *Conceptual Modeling*, pages 403–413. Springer, 2013.

- [91] E. Miguel. *Parkinson día a día: antes de decir “no puedo”, ¡¡inténtalo!* Biblioteca Parkinson. Asociación Parkinson Madrid, 2009.
- [92] A. Mihailidis, B. Carmichael, and J. Boger. The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home. *Information Technology in Biomedicine, IEEE Transactions on*, 8(3):238–247, Sept 2004.
- [93] G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.
- [94] I. Millington and J. Funge. *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.
- [95] U. S. D. Modeling and S. Office. Modeling and simulation (m&s) glossary, 1998.
- [96] D. Moody. The “physics” of notations: Toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, 35(6):756–779, Nov 2009.
- [97] D. Moody and J. van Hilleghersberg. Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. In D. Gašević, R. Lämmel, and E. Van Wyk, editors, *Software Language Engineering*, volume 5452 of *Lecture Notes in Computer Science*, pages 16–34. Springer Berlin Heidelberg, 2009.
- [98] R. Morla and N. Davies. Evaluating a location-based application: a hybrid test and simulation environment. *IEEE Pervasive Computing*, 3(3):48–56, July 2004.
- [99] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
- [100] H. Nakashima, H. Aghajan, and J. C. Augusto. *Handbook of ambient intelligence and smart environments*. Springer Science & Business Media, 2009.
- [101] A. A. Nazari Shirehjini and F. Klar. 3dsim: Rapid prototyping ambient intelligence. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies*, sOcEUSAI ’05, pages 303–307, New York, NY, USA, 2005. ACM.
- [102] I. Nieto, J. A. Botía, and A. F. Gómez-Skarmeta. Information and hybrid architecture model of the ocp contextual information management system. *Journal of Universal Computer Science*, 12(3):357–366, 2006.
- [103] H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto, and M. Ito. UbiREAL: Realistic smartspace simulator for systematic testing. *UbiComp 2006: Ubiquitous Computing*, pages 459–476, 2006.

- [104] J. C. Nordbotten and M. E. Crosby. The effect of graphic style on data model interpretation. *Information Systems Journal*, 9(2):139–155, 1999.
- [105] D. A. Norman and S. W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1986.
- [106] Y. Oh, H. Yoon, and W. Woo. Simulating context-aware systems based on personal devices. In *Proceedings of the International Symposium on Ubiquitous Virtual Reality, Gwangju, Korea*, pages 6–10, 2006.
- [107] E. O’Neill, O. Conlan, and D. Lewis. Modeling and simulation to assist context aware system design. *Simulation*, 2010.
- [108] E. O’Neill, O. Conlan, and D. Lewis. Situation-based testing for pervasive computing environments. *Pervasive and Mobile Computing*, 9(1):76 – 97, 2013. Special Section: Pervasive Sustainability.
- [109] E. O’Neill, M. Klepal, D. Lewis, T. O’Donnell, D. O’Sullivan, and D. Pesch. A testbed for evaluating human interaction with ubiquitous computing environments. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, TRIDENTCOM ’05, pages 60–69, Washington, DC, USA, 2005. IEEE Computer Society.
- [110] T. Pazlar and Z. Turk. Interoperability in practice: geometric data exchange using the ifc standard, 2008.
- [111] L. Penserini, P. Bresciani, T. Kuflik, and P. Busetta. Using tropos to model agent based architectures for adaptive systems: a case study in ambient intelligence. In *Software - Science, Technology and Engineering, 2005. Proceedings. IEEE International Conference on*, pages 37–46, Feb 2005.
- [112] H. Pirsiavash and D. Ramanan. Detecting activities of daily living in first-person camera views. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2847–2854, June 2012.
- [113] J. Poushter. Smartphone ownership and internet usage continues to climb in emerging economies. <http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/>, 2016. Último acceso: Abril, 2016.
- [114] H. Prendinger, B. Brandherm, and S. Ullrich. A simulation framework for sensor-based systems in second life. *Presence: Teleoperators and Virtual Environments*, 18(6):468–477, Dec. 2009.

- [115] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.
- [116] L. Rabiner and B.-H. Juang. Fundamentals of speech recognition. 1993.
- [117] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing*, 9(2):177–200, 2013.
- [118] J. Reimer. Why next-gen games have next-gen prices. Disponible en <http://arstechnica.com/gaming/2006/12/8479/>, 2006. (último acceso: Diciembre, 2015).
- [119] P. Remagnino and G. L. Foresti. Ambient intelligence: A new multidisciplinary paradigm. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(1):1–6, 2005.
- [120] V. Reynolds, V. Cahill, and A. Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of the First International Conference on Integrated Internet Ad Hoc and Sensor Networks*, InterSense '06, New York, NY, USA, 2006. ACM.
- [121] P. F. Riley and G. F. Riley. Next generation modeling iii-agents: Spades—a distributed agent simulation environment with software-in-the-loop execution. In *Proceedings of the 35th conference on Winter simulation: driving innovation*, pages 817–825. Winter Simulation Conference, 2003.
- [122] L. Roalter, A. Moller, S. Diewald, and M. Kranz. Developing intelligent environments: A development tool chain for creation, testing and simulation of smart and intelligent environments. In *Intelligent Environments (IE), 2011 7th International Conference on*, pages 214–221, July 2011.
- [123] J. Ruiz. *Parkinson—un as en la manga*. Biblioteca Parkinson. Asociación Parkinson Madrid, 2008.
- [124] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.
- [125] D. Sanchez, M. Tentori, and J. Favela. Activity recognition for the smart hospital. *Intelligent Systems, IEEE*, 23(2):50–57, March 2008.
- [126] K. Sanmugalingam and G. Coulouris. A generic location event simulator. In G. Borriello and L. E. Holmquist, editors, *UbiComp 2002: Ubiquitous Computing*, number 2498 in Lecture Notes in Computer Science, pages 308–315. Springer Berlin Heidelberg, Jan. 2002.

- [127] P. Sernani, A. Claudi, P. Calvaresi, D. Accattoli, R. Tofani, and A. F. Dragoni. Using 3d simulators for the ambient assisted living. In *Proceedings of the 3rd International Workshop on Artificial Intelligence and Assistive Medicine*, volume 1213, pages 16–20, 2014.
- [128] E. Serrano and J. Botia. Validating ambient intelligence based ubiquitous computing systems by means of artificial societies. *Information Sciences*, 222(0):3 – 24, 2013. Including Special Section on New Trends in Ambient Intelligence and Bio-inspired Systems.
- [129] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, and N. I. Badler. Parameterizing behavior trees. In *Motion in Games*, pages 144–155. Springer, 2011.
- [130] A. Shoulson, N. Marshak, M. Kapadia, and N. I. Badler. Adapt: the agent development and prototyping testbed. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 9–18. ACM, 2013.
- [131] J. Silva, O. Ribeiro, J. Fernandes, J. Campos, and M. Harrison. The apex framework: Prototyping of ubiquitous environments based on petri nets. In R. Bernhaupt, P. Forbrig, J. Gulliksen, and M. Lárusdóttir, editors, *Human-Centred Software Engineering*, volume 6409 of *Lecture Notes in Computer Science*, pages 6–21. Springer Berlin Heidelberg, 2010.
- [132] C. Stocker, L. Sun, P. Huang, W. Qin, J. M. Allbeck, and N. I. Badler. Smart events and primed agents. In *Intelligent virtual agents*, pages 15–27. Springer, 2010.
- [133] L. Tang, Z. Yu, X. Zhou, H. Wang, and C. Becker. Supporting rapid design and evaluation of pervasive applications: challenges and solutions. *Personal and ubiquitous computing*, 15(3):253–269, 2011.
- [134] L. Tang, X. Zhou, C. Becker, Z. Yu, and G. Schiele. Situation-based design: A rapid approach for pervasive application development. In *Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 128–135, Sept 2012.
- [135] D. Trenholme and S. P. Smith. Computer game engines for developing first-person virtual environments. *Virtual reality*, 12(3):181–187, 2008.
- [136] J. D. Ullman. *Elements of ML Programming, ML97 Edition*. Prentice Hall, 2 edition, 1998.
- [137] A. Van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000.
- [138] T. Van Nguyen, J. G. Kim, and D. Choi. ISS: The interactive smart home simulator. In *11th International Conference on Advanced Communication Technology, 2009. ICACT 2009*, volume 03, pages 1828–1833, Feb. 2009.

- [139] J. van Oijen and F. Dignum. Agent communication for believable human-like interactions between virtual characters. In F. Dignum, C. Brom, K. Hindriks, M. Beer, and D. Richards, editors, *Cognitive Agents for Virtual Environments*, volume 7764 of *Lecture Notes in Computer Science*, pages 37–54. Springer Berlin Heidelberg, 2013.
- [140] R. G. Vega, R. H. Bretones, and B. J. del Olmo. Desarrollo de un sistema de detección de caídas basado en acelerómetros. Trabajo de Fin de Grado en Ingeniería Informática y en Ingeniería del Software (Universidad Complutense, Facultad de Informática, curso 2015/2016), 2016.
- [141] C. Velasquez, C. Soares, R. Morla, R. S. Moreira, J. M. Torres, and P. Sobral. A 3d simulation framework for safe ambient-assisted home care. In *The Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UIC/ATC)*, pages 61–66, Nov. 2011.
- [142] M. Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.
- [143] D. Zhang, H. Huang, C.-F. Lai, X. Liang, Q. Zou, and M. Guo. Survey on context-awareness in ubiquitous media. *Multimedia tools and applications*, 67(1):179–211, 2013.

Lista de acrónimos

AAL	<i>Ambient Assisted Living</i> , Vida cotidiana asistida por el entorno
AMI	<i>Ambient Intelligence</i> , Inteligencia Ambiental
API	<i>Application Programming Interface</i> , Interfaz de programación de aplicaciones
BIM	<i>Building Information Modeling</i> , Modelado de la información de edificios
CAD	<i>Computer-Aided Design</i> , Diseño asistido por ordenador
DSM	<i>Domain-Specific Modeling</i> , Modelado Específico del Dominio
GUI	<i>Graphical User Interface</i> , Interfaz gráfica de usuario
HAB	<i>Hierarchical Automaton of Behaviours</i> , Comportamientos de autómatas jerárquicos
HAI	<i>Hierarchical Automaton of Interactions</i> , Interacciones de autómatas jerárquicos
IFC	<i>Industry Foundation Classes</i> , Clases de Fundación de la Industria
IVA	<i>Intelligent Virtual Agent</i> , Agente inteligente virtual
JME3	<i>jMonkeyEngine3</i> , Motor de juegos en Java
MDD	<i>Model-Driven Development</i> , Desarrollo Dirigido por Modelos
PHAT	<i>Physical Human Activity Tester</i> , Prueba de la actividad física humana
SDK	<i>Software Development Kit</i> , Kit de Desarrollo de Software
SociAALML..	<i>Social Ambient Assisted Living Modeling Language</i> , Lenguaje de modelado para la sociedad de la vida asistida de las personas