Tafat-Profiler: una herramienta para la generación automática de modelos de simulación a partir de perfiles

José Évora Gómez

Máster Oficial SIANI.
Universidad de Las Palmas de G.C.

Máster Oficial SIANI. Universidad de Las Palmas de G.C.

Trabajo fin de Máster

Título: Tafat-Profiler: una herramienta para la generación automática de modelos de simulación a partir de perfiles.
Apellidos y nombre del alumno: Évora Gómez, José
Fecha : 30 de Junio de 2011

Tutor: Hernández Cabrera, José Juan
Tutor: Hernández Tejera, Francisco Mario

Máster Oficial SIANI. Universidad de Las Palmas de G.C.

Agradecimientos

En primer lugar, quiero agradecer a mis dos tutores la inestimable ayuda que me han prestado para llevar a cabo este trabajo de fin de máster, guiándome y aconsejándome para tratar de conseguir los mejores resultados posibles.

En segundo lugar, agradecer a Enrique Kremers su ayuda a la hora de capturar los requisitos de usuario de las herramientas que se presentan en este trabajo, así como, el asesoramiento a la hora de realizar complementos en el ámbito de las redes eléctricas.

En tercer lugar, agradecer a todas las personas relacionadas con el European Institute For Energy Research (EIFER) que me han ayudado a crear los complementos existentes hasta la fecha.

Por último, a todos los familiares, amigos, compañeros y, en especial a mi novia, que se han preocupado y me han animado a realizar el trabajo.

Máster Oficial SIANI. Universidad de Las Palmas de G.C.

Índice general

Capítulo 1

Introducción

El estudio de los sistemas complejos[1] es un área de interés de la comunidad científica. Puede encontrarse mucha literatura acerca de como abordar un estudio de un sistema complejo; sin embargo, no existe un método único que sea adecuado para abordarlos todos.

La complejidad de un sistema se mide en base a la cantidad y naturaleza de los elementos que lo constituyen y sus interrelaciones. A mayor cantidad de elementos la complejidad del sistema será mayor [9]. Los sistemas complejos se pueden estudiar a través de la simulación en ordenadores. Este proceso requiere una etapa de diseño (modelizado) y de implementación (creación de la simulación) [5]. Estos procesos son más complicados cuanto mayor es la complejidad del sistema a estudiar.

Las simulaciones son utilizadas cuando se quiere evaluar modelos teóricos y, eventualmente, efectuar su posterior implementación, cuando ésta es demasiado costosa o cuando no es posible realizar un estudio directamente sobre los sistemas reales [4].

Los objetivos principales para realizar simulaciones sobre sistemas complejos son, entre otros, el descubrimiento del comportamiento del sistema, la postulación de hipótesis que expliquen el comportamiento emergente y el uso de esas hipótesis para predecir el comportamiento futuro del sistema por medio de la observación de los efectos que se producen cuando se somete el sistema a cambios [16]. Por lo tanto, el diseño del modelado y creación de la simulación deben contemplar una flexibilidad suficiente que permita efectuar cambios sobre el sistema de estudio.

Un objetivo importante de la actividad de investigación en sistemas complejos es el de productividad. Ser productivo en el diseño y creación de la simulación permite hacer un estudio del sistema complejo en menor tiempo. En este punto es dónde encajan las visiones del investigador y del ingeniero del software. El investigador tiene el estudio del sistema complejo como problema a resolver, mientras que el ingeniero del software ve en ese problema el requisito principal de un software a desarrollar.

De este modo, surgió el framework Tafat[2] (sección: 2.2). Este framework permite un desarrollo rápido del modelo de simulación así cómo del simulador que lo soporta. Basado en Ingeniería

[1] Los sistemas complejos se componen de varias partes interconectadas cuyos vínculos crean información adicional no visible por el observador. El resultado de estas interacciones entre elementos es la emergencia de nuevas propiedades que no se puede explicar a partir de las propiedades de los elementos aislados

[2] Tafat significa *luz* en lengua aborigen canaria

Dirigida por Modelos (MDE) su sustento principal es un metamodelo. El metamodelo describe los elementos del ámbito que se quieren representar y simular a la hora de estudiar un sistema complejo. El simulador se genera a partir de la descripción de los elementos contenidos en el metamodelo. Por otro lado, el modelo de simulación será la descripción de la escena concreta en términos de instancias de elementos de simulación. Gracias a la separación entre el modelo y el simulador, es posible introducir cambios con mucha mayor facilidad y flexibilidad.

En la realización de este modelo es dónde entra en juego este trabajo. En ocasiones, el modelador tendrá una idea concreta de lo que quiere simular, para lo cual no necesitará ser asistido. Sin embargo, no es habitual disponer de todos los detalles de la escena a simular, sino que se tengan datos estadísticos que la describen. En este caso, el modelo podría ser autogenerado acorde a los parámetros de entrada que el modelador proporciona.

De este modo, la concepción de una aplicación que se encargue de esta tarea cobra sentido. Bajo esta idea, Profiler se postula como una herramienta que es capaz de generar un modelo de simulación compatible con Tafat a partir de datos descriptivos de la escena.

Este trabajo tiene como principal objetivo la creación de la herramienta Profiler que será uno de los componentes del conjunto de herramientas de Tafat. La etapa de diseño de la interfaz se aborda a través de una aplicación web que permite la computación en la nube. La realización de esta página web cobra más importancia a lo largo del desarrollo ya que se le añade el requisito de, aparte de ser la interfaz de Profiler, ser la interfaz del resto de herramientas, convirtiéndose así en la página web oficial de Tafat.

Capítulo 2

Contextualización

En este capitulo se hablará en mayor detalle del estado actual del tema respecto a la simulación de sistemas complejos basada en agentes y se hará un breve resumen de la ingeniería dirigida por modelos. Posteriormente, se introducirá el framework Tafat.

2.1. Estado actual del tema

2.1.1. Sistemas complejos: modelado y simulación

Las simulaciones se utilizan para evaluar, comparar y analizar un modelo de simulación [7]. De este modo se separa el concepto de simulación del modelo, siendo la simulación una herramienta de análisis de sistemas que se retroalimenta del propio sistema y sus cambios. El modelo, sin embargo, es una representación de un sistema o proceso que puede incorporar aspectos lógicos, matemáticos y estructurales. Por otro lado, se deben incluir los conceptos de evento y entidad. Los eventos son hechos que ocurren a lo largo del tiempo y que modifican el estado del modelo. Sin embargo, una entidad es un objeto que se encuentra en el modelo. Adentrando toda esta definición de simulaciones en el estudio de sistemas complejos, se puede decir que las simulaciones son útiles para identificar problemas, cuellos de botellas y fallos de diseño antes de construir o modificar un sistema, además de, estudiar la dinámica del mismo, observando como cambia con respecto al tiempo y como los subsistemas y componentes interactúan [4].

En las simulaciones de sistemas complejos el principal objetivo es la explicación o exploración de procesos naturales a partir de la descripción del sistema [12]. Son muchas las aproximaciones que existen para llevar a cabo la simulación de sistemas complejos. Entre ellas, destaca la aproximación basada en agentes [10]. Por otro lado, la implementación de patrones de diseño de la ingeniería del software para abordar este tipo de simulaciones empieza a considerarse [11], ya que el proceso de desarrollo de la simulación debe ser relativamente fácil y rápido.

La separación conceptual entre los diferentes elementos de la simulación también se considera en el campo de los sistemas complejos. Separar los objetos o agentes de sus propios comportamientos da flexibilidad a la simulación [6]. De este modo se puede tener un mismo objeto que funcione bajo diferentes comportamientos. Por ejemplo, en un proceso de simulación se puede hacer convivir los comportamientos termodinámicos de un edificio con los sociológicos de los usuarios del mismo.

Así, se separa el concepto de los comportamientos, permitiendo con mayor facilidad escalar las funcionalidades que un determinado objeto puede tener o la creación de comportamientos que representen el funcionamiento de varios objetos.

En el estudio de un sistema complejo es importante el nivel de granularidad [8] para equilibrar los resultados a obtener y su calidad con el coste computacional. A mayor detalle más posibilidades hay de analizar con mayor precisión qué elementos influyen y de qué modo. Por ello, es recomendable el estudio de un sistema complejo usando una aproximación de abajo hacia arriba [3] que permita ver el comportamiento de las unidades más pequeñas. En el caso de las redes eléctricas, que es uno de los ámbitos que se trabajan en Tafat, esto permitiría ver la firma que cada electrodoméstico, presente en una vivienda, genera en su funcionamiento, permitiendo saber la influencia de ese aparato en la red, así cómo agrupar en tipos de consumo. Un análisis siguiendo esta arquitectura, de abajo hacia arriba, en el sistema complejo permitiría, además de generar conocimiento, pensar en posibles cambios que se pudieran hacer sobre el sistema para obtener un comportamiento deseado. En el caso de las redes eléctricas, esto sería la aplicación de técnicas de la gestión de la demanda (Demand Side Management - DSM) [14] sobre unidades pequeñas de la escena (electrodomésticos) para generar una curva agregada deseada [1].

2.1.2. Ingeniería dirigida por modelos

El desarrollo de aplicaciones vino de la mano de la aparición de los primeros sistemas de computación. La complejidad en el desarrollo de las aplicaciones radicaba en la necesidad que tenía la máquina de recibir las instrucciones en 0s y 1s. Como consecuencia, por medio de la creación de lenguajes de ensamblador, se creo una capa de abstracción entre el programador y la máquina. De este modo, el programador podía escribir sus aplicaciones en un lenguaje de más alto nivel. La aparición de lenguajes de alto nivel, como por ejemplo C, permitieron facilitar el desarrollo de aplicaciones al establecer una segunda capa de abstracción por medio de la traducción de las instrucciones de alto nivel a código máquina. Sin embargo, la elección del lenguaje de alto nivel vincula el desarrollo de la aplicación a la capacidad de éxito que pueda tener dicho lenguaje [15].

La ingeniería dirigida por modelos (MDE) nace como una tercera capa de abstracción que permite eliminar la dependencia al lenguaje de programación [2]. En este caso, el desarrollo de aplicaciones se hace en base a un modelo el cual expresa como debe ser la aplicación. Este modelo, posteriormente, es procesado por generadores y traductores para crear la aplicación en uno o varios lenguajes. De este modo, la aplicación puede ser modificada cambiando el modelo que la describe [13].

2.2. Tafat

Tafat es un framework con un conjunto de herramientas que han sido creadas para el desarrollo de simulaciones. Este framework ha sido diseñado y desarrollado en el instituto universitario de Sistemas Inteligentes y Aplicaciones Numéricas en la Ingeniería (SIANI). El framework está basado en la Ingeniería Dirigida por Modelos (MDE) para el desarrollo de simuladores, evitando así,

dependencias con lenguajes de programación y facilitando el cambio de ámbitos de simulación. El principal componente de este framework es el metamodelo, que especifica como un modelo puede ser expresado. Este framework ofrece las siguientes ventajas:

- Modo estandarizado de modelar
- Guía para la creación de modelos
- Rendimiento en la ejecución de modelos
- En continuo desarrollo.

Más información acerca de Tafat en la sección 4.1.1.

Capítulo 3

Fundamentos teóricos del trabajo

El fundamento que sustenta el framework Tafat es el metamodelo (sección: 4.1.1). Es el elemento clave de todo el framework ya que define el ámbito[1] en el que las herramientas del framework trabajarán. El uso del metamodelo en el framework es concepto nuclear que surge de la Ingeniería Dirigida por Modelos, y que permite independizar las herramientas del framework del ámbito en el que trabajan, así como del lenguaje de programación en el que las simulaciones se ejecutan. Por medio de traductores y generadores se procesa el metamodelo para obtener los elementos de simulación en un lenguaje concreto. Para el usuario final, el metamodelo se postula como la capa de mayor abstracción que es utilizada para describir los elementos de un determinado ámbito.

La estructura del framework es más compleja si se observa desde el punto de vista del desarrollador del framework (figura: 3.1). El desarrollador observa tres capas: el metametamodelo, el metamodelo y el modelo. La primera de ellas describe, de modo abstracto, los tipos de elementos que puede contener el metamodelo. En la etapa de análisis del framework Tafat, después de estudiar diversos sistemas complejos, se llegó a la conclusión de que éstos podían ser descritos en términos de: entidades, agentes, conexiones y sus comportamientos. Esta conclusión es la que recoge la definición de metametamodelo (figura: 3.2). Esto supone una abstracción que guía el desarrollo del metamodelo clasificando todos los elementos que se le añaden en una de estas categorías.

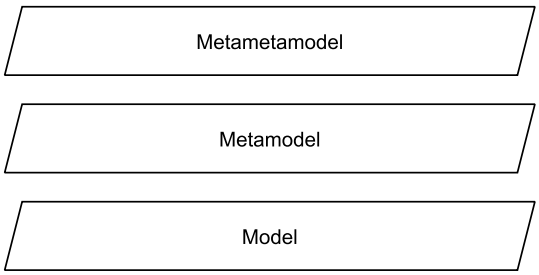

Figura 3.1: Capas que observa un desarrollador

[1]El concepto de ámbito hace referencia al entorno en el que se desarrollan las simulaciones, como por ejemplo: redes eléctricas, tráfico aéreo...

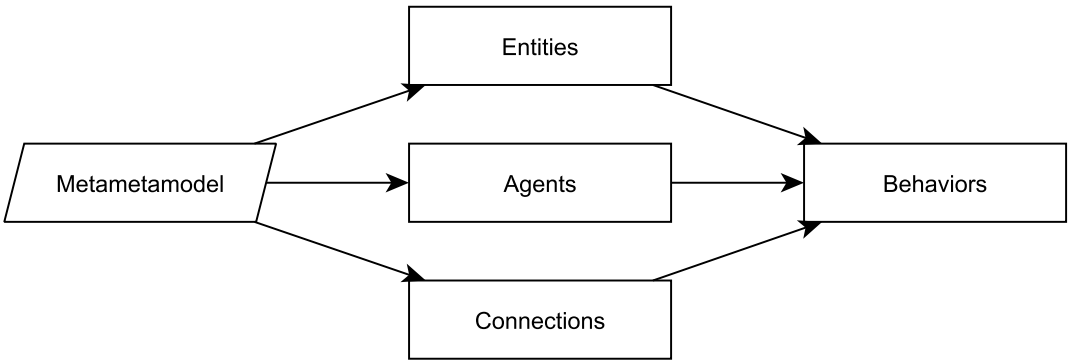

Figura 3.2: Capa del metametamodelo que abstrae los tipos de elementos de simulación que puede contener el metamodelo.

De este modo, el metamodelo contiene los elementos de simulación clasificados en uno de esos tres tipos identificados: entidades, agentes y conexiones. En la descripción de Tafat (sección: 4.1.1) se muestra como se tratan los comportamientos que describen el modo de actuar de estos tres tipos de elementos. El metamodelo está estructurado en capas que responden a las necesidades que un determinado ámbito tiene para ser representado (figura: 3.3). En la figura 3.4 se muestra un pequeño ejemplo de metamodelo de redes eléctricas.

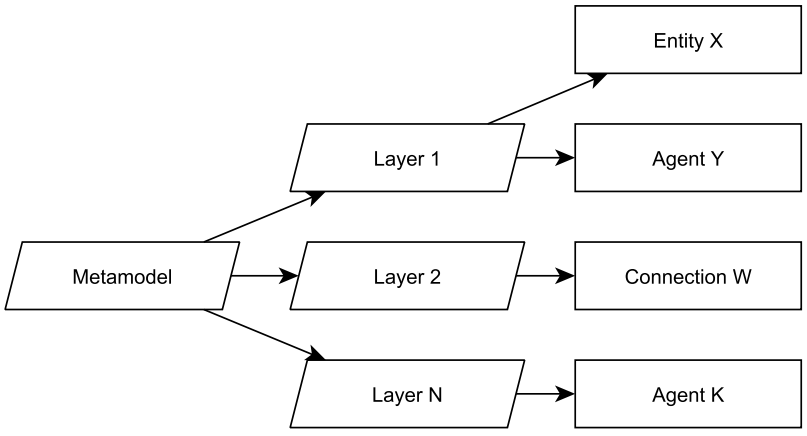

Figura 3.3: El metamodelo es representado en capas acorde a las necesidades del ámbito. Cada capa puede contener entidades, agentes o conexiones.

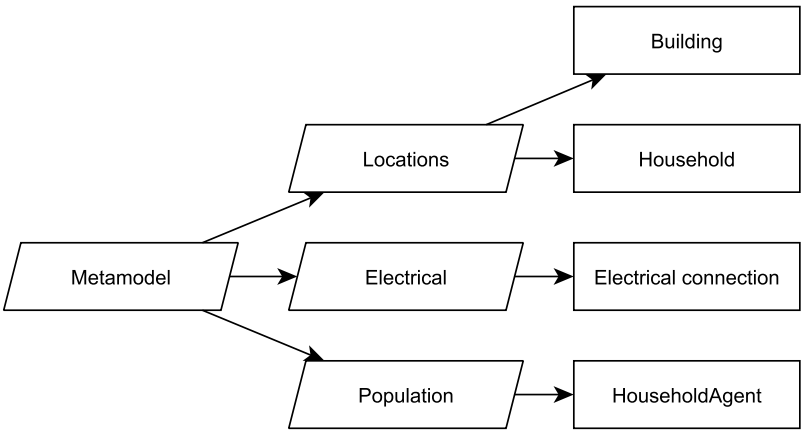

Figura 3.4: Ejemplo de metamodelo

La capa con la que, habitualmente, interactuará el usuario final es la del modelo. En el modelo se representa la escena en forma de instancias concretas de los elementos del metamodelo. Estas instancias describen las características que el elemento a instanciar tiene. Un ejemplo de modelo puede ser observado en la figura 3.5. En este caso, la escena contiene un edificio con dos viviendas y un agente que se relaciona con una de esas viviendas.

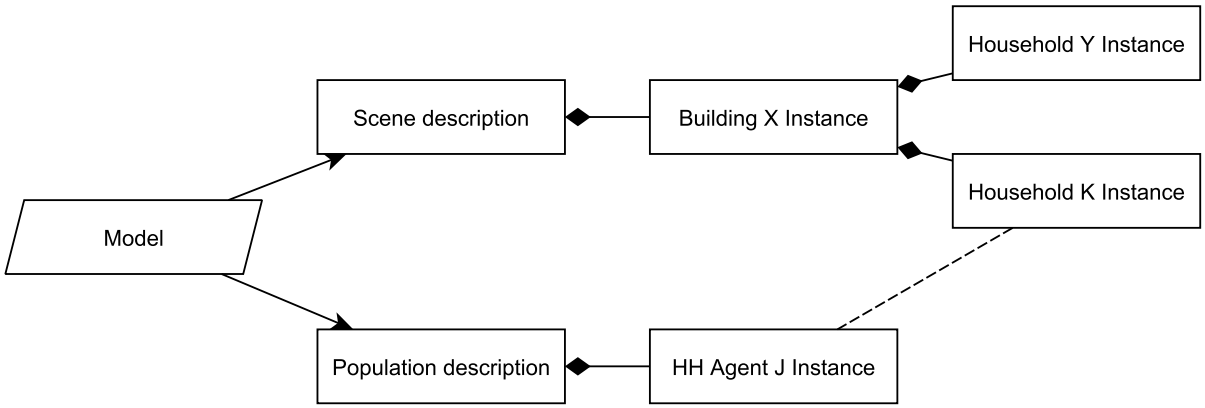

Figura 3.5: Descripción de la escena de simulación en el modelo

Capítulo 4

Recursos, planificación y metodología de desarrollo

En este capítulo se explicará la arquitectura del framework Tafat que es el recurso principal en el que se sustenta Profiler. Posteriormente, se definiran otros recursos necesarios para abordar el desarrollo y explotación de la herramienta. Se mostrará el plan de trabajo trazado y el número de horas empleadas para cada etapa. Por último, se explicará la metodología de desarrollo desglosándola en las fases de las que ésta se ha compuesto.

4.1. Recursos

4.1.1. Tafat

Tras haber introducido lo que es Tafat en el capítulo anterior (2.2), en esta sección se presenta la arquitectura y descripción de los componentes que forman el framework.

Arquitectura

La arquitectura de este framework (figura: 4.1) se ha desarrollado usando la Ingeniería dirigida por modelos (MDE) y considerando diversos patrones de diseño propios de la Ingeniería del Software.

El metamodelo contiene una descripción estructural de todos los elementos que pueden ser usados en el modelo de simulación. El modelo es un subconjunto del metamodelo y se usa para describir un caso de estudio concreto en el cual los elementos son instanciados y conectados entre sí. El metamodelo y los modelos son procesados por herramientas para generar, de modo automático, otras herramientas (MDE).

El elemento central, el metamodelo, es procesado por un traductor para obtener versiones en XSD, HTML y Java. El modelo es una descripción de la escena en términos de instancias de elementos que, previamente, deben estar contenidos en el metamodelo.

Un modelo puede ser escrito usando un editor de XML que permita el uso de archivos XSD. El uso del fichero XSD ayuda a la hora de crear el modelo ya que realiza sugerencias en todo

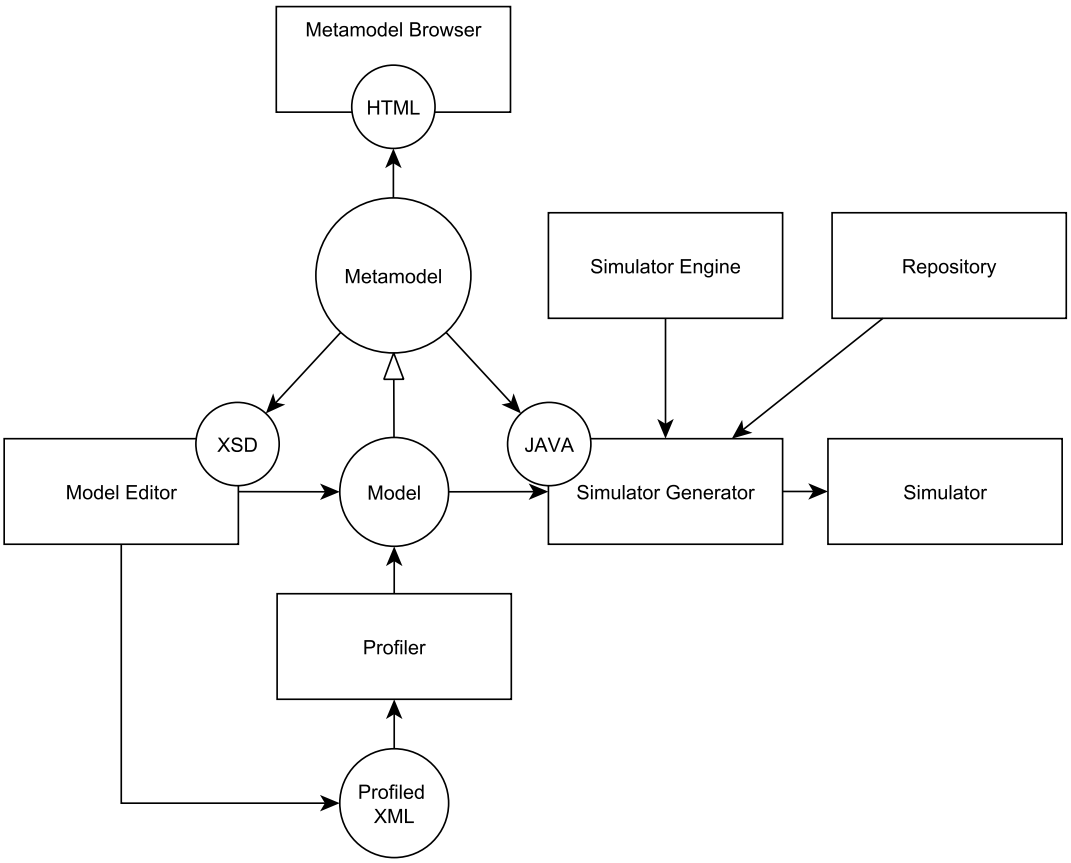

Figura 4.1: Arquitectura de Tafat

momento, indicando elementos a autocompletar, que atributos faltan, listas de posibilidades... En este diagrama se observa dónde encaja Profiler. La generación del modelo se puede abordar de tres modos: usando únicamente el XSD y describiendo la escena manualmente, usando únicamente Profiler para generar el modelo automáticamente o una solución mixta en la que se describan partes concretas con el XSD y otras se generen con Profiler.

Después de tener el modelo escrito, éste puede ser procesado por la aplicación Simulator Generator, que se encargará de recopilar los elementos del metamodelo en su traducción a Java, el motor del simulador y los comportamientos del repositorio. Con todo ello, generará un simulador especifico para este modelo que se le ha proporcionado, evitando tener elementos de simulación que no se usan en la escena descrita.

Una vez el simulador se ha generado de modo automático, éste estará listo para comenzar la simulación. No obstante, el uso de la herramienta Simulator Generator no es necesaria si se utiliza el simulador completo, con todos los componentes.

Roles

Varios roles se han identificado en el framework de Tafat. Los roles identificados son: el desarrollador, el programador, el modelador y el experto del dominio.

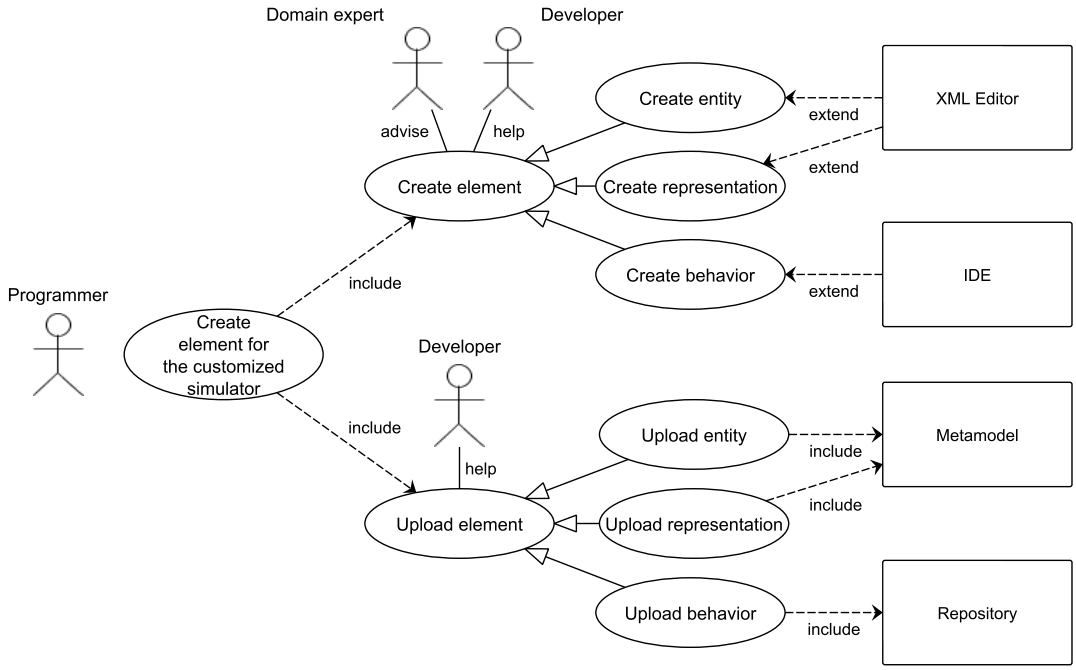

Figura 4.2: Caso de uso del programador

- Desarrollador: Ingeniero del Software que está a cargo del desarrollo del framework Tafat. Este rol es personal contratado para mantener y desarrollar nuevas funcionalidades del framework.

- Programador: persona con conocimientos de programación que es capaz de desarrollar nuevos elementos de simulación (entidades, comportamientos...). El programador puede ser asesorado por el desarrollador del framework para guiar la creación de nuevos componentes de simulación. Por otro lado, también puede ser asesorado por el experto del dominio para identificar los parámetros de las entidades o identificar como debería funcionar el comportamiento.

- Modelador: el principal de los usuarios finales que creará los modelos para simularlos en el framework. Para ello, debe diseñar previamente el escenario que quiere analizar. Una vez diseñado, con ayuda del metamodelo, se comenzará a trascribir ese diseño al modelo por medio de cláusulas XML.

- Experto del dominio: aporta conocimiento técnico al programador o modelador para crear entidades, comportamientos o modelos.

Las figuras 4.2 y 4.3 son casos de usos en los que se observa de qué modo participa cada uno de los roles en el framework Tafat. El primero de ellos muestra un caso de uso que está relacionado con el programador mientras que el segundo lo está con el modelador.

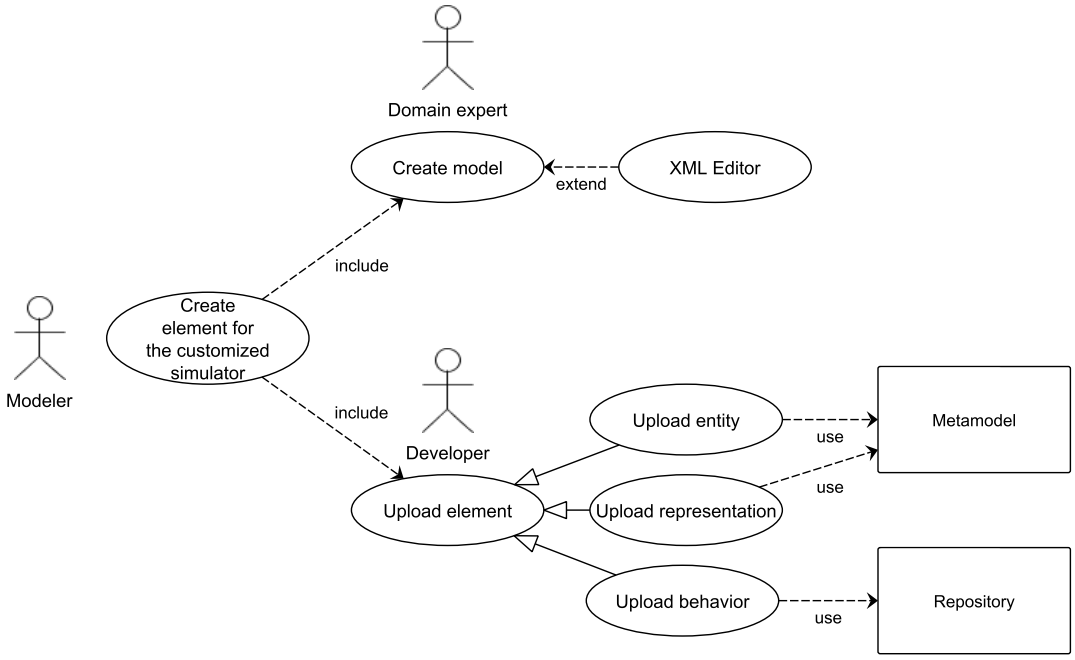

Figura 4.3: Caso de uso del modelador

Metamodelo y modelo

En este apartado se hará especial énfasis en diferenciar los conceptos de metamodelo y modelo, explicando la funcionalidad que cada uno tiene dentro del framework Tafat.

Antes de comenzar hablando del metamodelo, la separación de los diferentes elementos de simulación debe ser explicada y situada en el conjunto de herramientas que el framework contiene. Los tipos de elementos de simulación que el framework contempla son:

- Entidades: objetos que se pueden encontrar en la descripción de la escena. La descripción de este tipo de elementos se encuentra en el metamodelo.
- Agentes: un agente es una entidad computacional capaz de realizar acciones autónomas en un entorno, con la intención de alcanzar metas. El agente es capaz de percibir el entorno y tener una representación del mismo a los efectos de su actividad; es proactivo, es decir, tiene unos objetivos que cumplir y es capaz de planificar su actividad para alcanzar esos objetivos, actuando convenientemente sobre su entorno [17]. En este caso, el agente es un elemento de simulación que interactúa con otros objetos u otros agentes. Este elemento no forma parte de la escena, sino de la población de la simulación. La descripción de este tipo de elementos se encuentra también en el metamodelo.
- Comportamientos: describen el modo de actuar, o una faceta del mismo, de una entidad o agente. Este tipo de elementos se encuentra en el repositorio.
- Conexiones: describen una relación entre una entidad o agente con otra entidad o agente. Este tipo de elementos se encuentra en el metamodelo. El elemento que se escoja será el que

defina que tipo de relación existe entre ellos (eléctrica, social, de comunicación...).

El framework Tafat considera diferentes niveles de granularidad. En simulaciones de abajo hacia arriba lo ideal es representar todos los elementos del sistema complejo. Sin embargo, en ocasiones, existen limitaciones (temporales, computacionales...) que impiden llevar a cabo experimentos con todo nivel de detalle. Por ello, los elementos de simulación del framework permiten ser instanciados bajo diferentes representaciones. A través de la representación se puede modular el nivel de detalle que se desea para cualquier elemento de simulación que se emplee en la escena. Las representaciones que cada elemento de simulación puede tener son dos: completa (full) y fingida (mock). La primera de ellas supone la representación completa del elemento, lo cual permite continuar definiendo los elementos que contiene (por ejemplo: un edificio con todas las viviendas). Sin embargo, la representación fingida no contiene elementos internos y se describe con la cantidad mínima de atributos. Posteriormente, su comportamiento se puede basar en datos históricos obtenidos en otras simulaciones.

Por ejemplo, la simulación de un edificio con diez viviendas, y cada una con sus electrodomésticos, implica la ejecución de comportamientos de, aproximadamente, 150 elementos de simulación (en el ámbito de las redes eléctricas). Si no fuera necesario representar dicho edificio a ese nivel de detalle, éste podría representarse mediante una entidad fingida cuyo comportamiento responde al de una curva de consumo agregada de las diez casas que puede haber sido capturada de simulaciones anteriores o de datos históricos. En este caso, la ejecución de los 150 comportamientos se reduciría a uno: el del edificio.

De este modo, gracias al concepto de representación, es posible mejorar el rendimiento de las simulaciones, permitiendo al modelador definir con mayor o menor detalle los elementos de simulación acorde a su objeto de estudio.

Metamodelo El metamodelo es una descripción de todos los componentes que pueden ser usados en un modelo. Describe las características de los elementos que pueden ser utilizados en las simulaciones. Sin embargo, no describe como las variables varían con respecto al tiempo. Esta tarea la realizan los comportamientos. De este modo, se establece una separación entre la definición y el funcionamiento de un elemento.

Debido a las descripciones que tiene, el metamodelo es el elemento central para transitar de un ámbito de simulación a otro. Por ejemplo, si se trabaja en el campo de las simulaciones de redes eléctricas, se tendrá un determinado metamodelo que describa todos los elementos que puede albergar una simulación de este tipo; sin embargo, el metamodelo será diferente en simulaciones de otro ámbito, como por ejemplo, el tráfico aéreo.

El metamodelo es el núcleo principal del framework ya que, a raíz de él, se generan diversas aplicaciones. El diseño del metamodelo es realmente crítico porque definirá cómo se debe generar un modelo de simulación. Un mal diseño del metamodelo desemboca en una pobre descripción de las escenas de simulación. Las principales consideraciones, desde un punto de vista estructural, que se deben tener al diseñar un metamodelo son:

- Elementos de simulación bien organizados dentro de la estructura, estableciendo correctamente las relaciones entre elementos.

- Coherencia en el diseño de elementos de simulación y en su relación con el resto de elementos (padres e hijos).

El metamodelo se representa por un sistema basado en ficheros y carpetas. Los elementos de simulación se describen a través de archivos (XML), mientras que, los elementos contenidos en las carpetas indican la herencia de éstos con el poseedor de la carpeta. El diseño de elementos que se añaden al metamodelo, y que por tanto, son susceptibles de ser simulados, debe ser asesorado por un experto del dominio. El diseño debe aportar las propiedades que describen a un determinado elemento: características, variables, elementos contextuales, elementos contenidos y su localización dentro de la estructura del metamodelo. Estos parámetros son descritos a continuación:

- Características: información estática del elemento
- Variables: información dinámica del elemento. Esta información varía a lo largo del tiempo de ejecución de la simulación a través de los comportamientos.
- Elementos contextuales: elementos del metamodelo que son necesarios para la ejecución de alguno/s comportamientos asociados a este elemento.
- Elementos contenidos: elementos que puede contener internamente un determinado elemento. Por ejemplo, un edificio puede contener diversas viviendas.
- Localización en el metamodelo: el elemento debe colocarse correctamente en el metamodelo para mantener la coherencia con todos los elementos existentes ya en el metamodelo.

El metamodelo puede ser traducido a XSD, Java y HTML. La traducción a XSD permite al modelador la creación de modelos de modo más sencillo, ya que el XSD hace de guía en el proceso de escritura. Éste se encarga de ofrecer las diferentes alternativas disponibles en un momento determinado de la escritura del modelo, así como, de informar sobre fallos cometidos o ausencia de datos requeridos.

La traducción a Java permite la ejecución del simulador. En el caso de elementos instanciables[1], la traducción genera tres clases: la genérica, la representación completa y la fingida del elemento. Estas clases son utilizadas por el simulador para crear un objeto por instancia encontrada en el modelo que describe la escena de simulación.

Por último, la traducción a HTML permite observar el metamodelo desde un punto de vista gráfico. Esto ayuda al modelador a entender como está estructurado el metamodelo pudiendo observar en cada elemento sus características y sus relaciones con otros elementos.

[1]Elementos que no son abstractos. Haciendo analogía de la orientación a objetos, los elementos abstractos presentan un nivel de abstracción tan elevado que no sirven para instanciar objetos de ellos. Representan los escalones más elevados de algunas jerarquías de elementos y sirven para derivar otros elementos, en los que se van implementando detalles y concreciones, hasta que finalmente presentan un nivel de definición suficiente que permita instanciar objetos concretos. Por ejemplo, animal (abstracto), mamífero (abstracto), perro (no abstracto)

Modelo El modelo contiene instancias de todos los elementos de simulación (entidades, agentes, comportamientos y conexiones) que describen la escena. Expresado en XML cada cláusula indica el deseo de utilizar un elemento en la simulación. En el modelo se establecen las relaciones entre diferentes elementos de simulación. Las relaciones entre entidades y agentes se hace por medio de conexiones. La especificación de qué comportamiento utilizará un determinado elemento de simulación se hace por medio de introducir una cláusula comportamiento (behavior) dentro de la cláusula de la entidad o agente. De este modo se indica que esa entidad o agente va a funcionar bajo el comportamiento referenciado.

El motor del simulador se encargará de comprobar que la relación establecida entre comportamiento y agente o entidad es correcta; además de comprobar las relaciones de contención implícitas en el modelo (una vivienda dentro de un edificio). La figura 4.4 muestra un ejemplo de la escritura de un modelo usando el fichero XSD que se obtiene de traducir el metamodelo. La figura 4.5 muestra una escena que tiene un edificio en el cual hay una vivienda que funciona con un determinado comportamiento (single person). A continuación se muestra el fragmento de código que conlleva la figura 4.5:

```
<simulation>
   <scene>
      <outdoorFull>
         <buildingFull>
            <householdFull>
               <behavior name='HouseholdFullBehavior' release='SinglePerson' />
            </householdFull>
         </buildingFull>
      </outdoorFull>
   </scene>
</simulation>
```

Figura 4.4: Escritura de un modelo de simulación usando el XSD como ayuda

Figura 4.5: Ejemplo de una escena descrita con un modelo de simulación

Repositorio

El repositorio es un almacenamiento digital que contiene los comportamientos de los elementos de simulación. De este modo, los comportamientos están disponibles para ser utilizados en las simulaciones.

El comportamiento simula el modo de actuar de la entidad o agente calculando en cada paso el valor de sus variables a lo largo del tiempo. El concepto de comportamiento es realmente amplio, pues cada elemento puede ser simulado bajo infinitos comportamientos de diferentes índoles.

Por ejemplo, un edificio puede tener un comportamiento que calcule el número de personas que están dentro, modificando en tiempo de ejecución la variable que indica el número de personas que están dentro. Sin embargo, este mismo edificio puede estar actuando, además, bajo otro comportamiento que calcule la temperatura interna del edificio.

De este modo, y gracias a la separación entre descripción y comportamiento, surge el concepto de versión (release). El comportamiento del edificio que cuenta el número de personas podría denominarse versión *social* mientras que el otro podría llamarse *temperatura*. Este concepto tan amplio de comportamiento permite al modelador tener diversas entidades o agentes funcionando bajo diferentes comportamientos a la vez, encargándose cada uno de un aspecto concreto.

Motor del simulador

Está compuesto por diversas clases que permiten la lectura de modelos expresados en XML y crea objetos por cada instancia encontrada en el modelo. Por otro lado, contiene utilidades que pueden ser usadas por otros elementos del motor o comportamientos. La figura 4.6 muestra los paquetes que existen dentro del motor de simulación.

El motor se compone por las clases principales contenidas en el paquete del motor y el resto de paquetes, como el de conversión, funciones interpoladas, cartas de estados, timeouts y herramientas. A continuación, se ofrece una breve descripción de lo que realiza cada paquete del simulador:

- Paquete principal: se encarga de deserializar el XML y generar instancias de los elementos de simulación que describen la escena. Además, asocia los elementos de simulación que son comportamientos con la entidad o agente que le corresponda.

- Paquete de conversión: transforma valores dados en una unidad determinada en el valor equivalente a la unidad que se le pide.

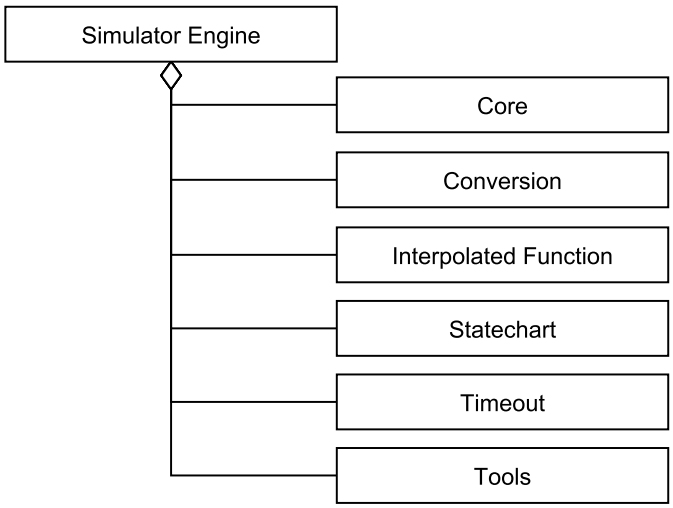

Figura 4.6: Motor del simulador

- Paquete de funciones interpoladas: proporciona una utilidad que, basándose en un conjunto de coordenadas, interpola una función entre ellas considerando ciertos parámetros.
- Paquete de cartas de estado: esta utilidad implementa cartas de estado que pueden ser utilizadas para representar el comportamiento de un elemento de simulación en forma de estados y transiciones.
- Paquete de timeouts: es una utilidad que permite gestionar el tiempo en base a eventos. Su funcionamiento es similar al de las interrupciones.
- Paquete de herramientas: diversas herramientas que pueden ser útiles para las inicializaciones de elementos de simulación, comportamientos... Por ejemplo: obtener valores aleatorios.

Simulator Generator

El objetivo de esta herramienta es la generación de un simulador que contenga todo lo necesario para ejecutar un modelo determinado. A partir del modelo, los elementos de simulación instanciados son recopilados de las clases Java y los comportamientos del respositorio. Posteriormente, el simulador se completa uniendo lo anterior al motor de simulación (figura: 4.7).

Esta herramienta extraerá los elementos (entidades, agentes o conexiones) que el modelo instancia de la traducción del metamodelo a Java. Para ello, el metamodelo debe haber sido procesado previamente por el traductor para generar las clases Java que representan los elementos de simulación.

Por otro lado, los comportamientos se extraen del repositorio cuando éstos son referenciados desde el modelo. Después de este proceso de agrupación, tanto de elementos como de comportamientos, la herramienta debe analizar los comportamientos para obtener otras clases necesarias para su ejecución, si las hubiera. Finalmente, se añade el motor del simulador.

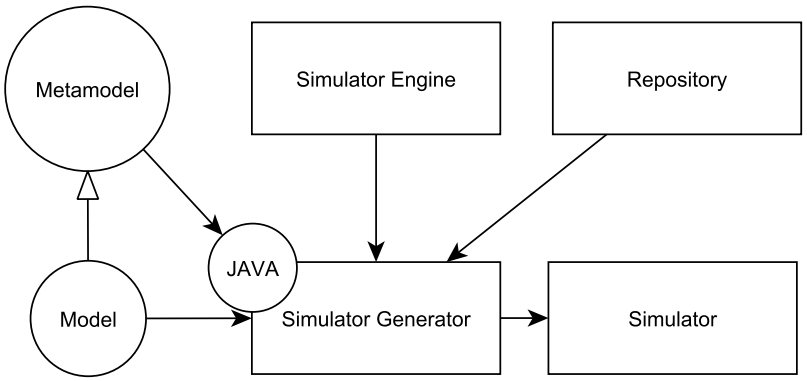

Figura 4.7: Diagrama de relaciones de Simulator Generator.

Simulación

La simulación está dividida en dos partes principales. La primera es la carga de componentes instanciados en el modelo. La segunda es la ejecución de la simulación a través del envío de tics de reloj a todos los comportamientos de los elementos de la simulación.

Carga del modelo En esta primera etapa, todos los elementos de simulación instanciados en el modelo XML son creados e inicializados para comenzar con el segundo proceso: la ejecución de la simulación. Para realizar este proceso, la ruta del modelo a cargar debe ser proporcionada para abrir el fichero y comenzar a analizarlo, generando por cada instancia del modelo un objeto de simulación. Para cada elemento encontrado en el modelo, las acciones que se realizan son:

- Creación del objeto que representa al elemento instanciado.
- Inicialización del elemento.
- Configuración de atributos del elemento a valores por defecto.
- Lectura de atributos que provengan del modelo. De este modo, aquellos valores por defecto serían sustituidos por éstos.

A continuación, se muestra un modelo. El proceso de carga que el simulador realiza con cada instancia puede ser observado en la figura 4.8.

```
<simulation>
   <scene>
      <outdoorFull>
         <buildingFullareaFloor='30'>
            <householdFull/>
         </buildingFull>
      </outdoorFull>
   </scene>
</simulation>
```

El analizador creará todos los elementos recorriendo el modelo de arriba hacia abajo. Por lo tanto, el primer elemento que creará será la escena. Posteriormente, el outdoor que representa las

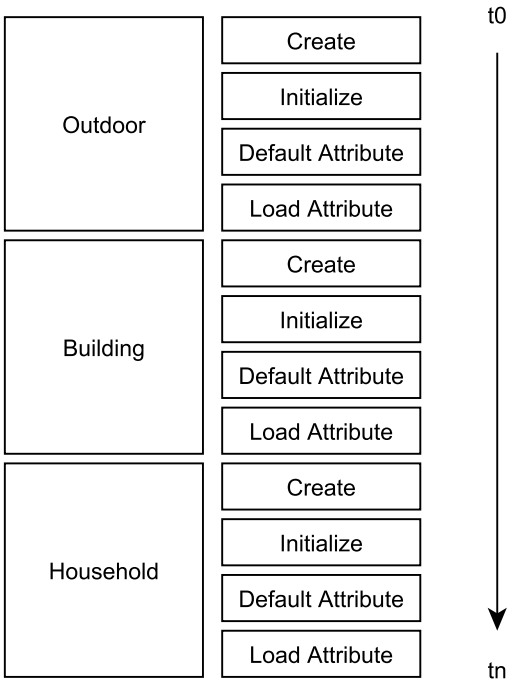

Figura 4.8: Proceso de carga del modelo

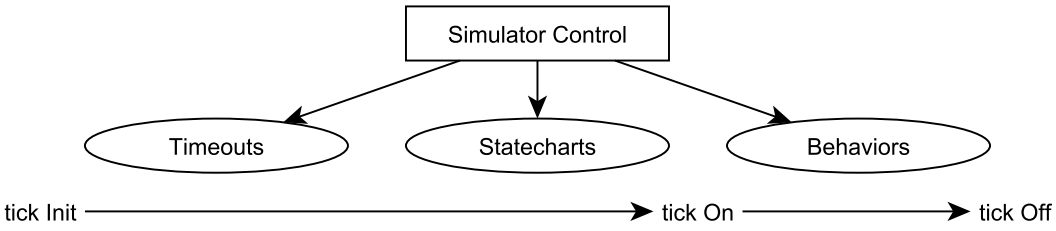

Figura 4.9: Orden de ejecución de elementos en un tic de reloj del simulador

condiciones ambientales en las que una localización está. Finalmente, realizará la carga del edificio para terminar cargando la vivienda.

Ejecución de la simulación En la parte de ejecución de la simulación, el motor del simulador envía tics de reloj a todos los comportamientos registrados en la simulación (tras el análisis del fichero de entrada). En cada segundo de simulación, el simulador se encarga de enviar tics de reloj (en este orden) a los timeouts, cartas de estado y comportamientos (figura: 4.9).

Dentro de estos subconjuntos de elementos que son susceptibles de ser actualizados mediante tics, los elementos que se ejecutarán primero son aquellos que se han creado antes (FIFO). Por lo tanto, los timeouts más antiguos se ejecutarán antes, al igual que las cartas de estado. Los comportamientos seguirán el orden en el que se han creado las instancias, es decir, aquellos

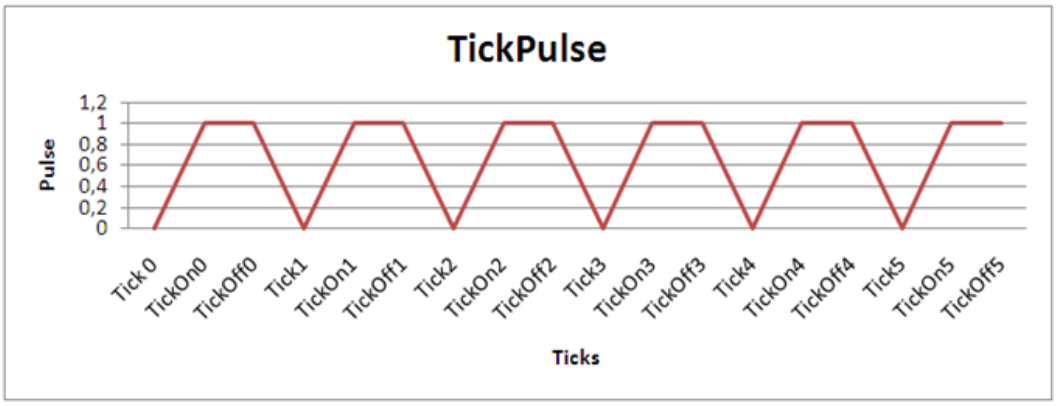

Figura 4.10: Ejecución de un tic de reloj en el simulador respecto a los comportamientos

comportamientos que se encuentren más arriba en el modelo se ejecutarán antes.

Los comportamientos tienen dos momentos de tic de reloj por cada tic. El primero es el de comienzo de tic (tick on) y el segundo es el de finalización del tic (tick off) (figura: 4.10). De este modo, se sugiere que en el comienzo del tic se tome del escenario la información necesaria y en el final del mismo se modifique el valor de las variables de los elementos de simulación.

4.1.2. Recursos utilizados

En esta sección se hablará sobre los recursos, aparte del framework Tafat, que se han utilizado para el desarrollo del trabajo de fin de máster, además de los que serán necesarios para poder utilizar las aplicaciones que en este trabajo se desarrollen.

Recursos utilizados en el desarrollo

Los recursos que se han utilizado para el desarrollo del Profiler se pueden dividir en dos categorías: recursos software y recursos hardware. Debido a que los requerimientos hardware dependen totalmente del software utilizado, se comenzará describiendo el software utilizado. Posteriormente, habiendo aclarado la complejidad del software, se describirá el hardware empleado.

Software Los recursos software utilizados no destacan por ser muy específicos, pues la mayoría de éstos son los que usualmente un desarrollador de software utiliza. Lo más específico que hay son determinadas librerías que se han utilizado.

- *JVM:* Java Virtual Machine. Es un entorno de ejecución de una plataforma independiente que convierte el código Java a lenguaje máquina y lo ejecuta. Una JVM simula un procesador real Java, permitiendo que el código Java sea ejecutado como acciones o llamadas al sistema a cualquier procesador sin importar el sistema operativo.

- *JRE:* Java Runtime Environment. JRE incluye la JVM, librerías y componentes que son necesarios para ejecutar programas escritos en Java. JRE está disponible en diversas plataformas, entre las que se encuentran MAC, Windows y Unix.

- *JDK:* Java Development Kit. Es un kit de desarrollo de software (SDK) para la creación de programas en Java. JDK viene con un JRE completo.

- *Eclipse:* es una comunidad de software libre, cuyos proyectos se centran en la construcción de una plataforma de desarrollo abierta compuesta por frameworks extensibles, herramientas y runtimes para la construcción, desarrollo y gestión del software a lo largo de su ciclo de vida. (Extraído de la web oficial de Eclipse: http://www.eclipse.org).

- *Sistema operativo:* el uso de Java y Eclipse no condiciona la elección del sistema operativo, pues ambos se encuentran disponible en diversas plataformas. Por tanto, se utilizó Windows 7 por ser el sistema operativo que el autor del trabajo tenía instalado previamente. Posteriormente, para la realización de pruebas, se instaló todo lo necesario en un sistema operativo basado en GNU/Linux.

- *Librería Jxl:* librería que permite la lectura de ficheros XLS, facilitando a los programadores de Profiler la creación de modelos basados en datos obtenidos de una tabla de Excel.

- *Librería Matemática de Apache:* es una librería ligera, que contiene componentes matemáticos y estadísticos que tratan de solventar los problemas más comunes que no están contemplados en la librería Math de Java. Extraído de: http://commons.apache.org/math/.

- *Librería POI de Apache:* Apache POI es un proyecto dirigido por Apache Software Fundation que proporciona librerías Java para la lectura y escritura de archivos en formato Microsoft Office como Word, PowerPoint y Excel.

- *Librería Jackaccess:* es una librería Java para la lectura y escritura de bases de datos de Microsoft Access. Esta librería esta dirigida a desarrolladores de aplicaciones en Java.

Software específico en el desarrollo de la aplicación web La aplicación web ha requerido el uso de dos tecnologías: Apache Tomcat y Velocity. Apache Tomcat es un servlet de código abierto desarrollado por Apache Software Foundation (ASF). Tomcat implementa las especificaciones del servlet Java y JavaServer Pages (JSP) y proporciona un entorno de servicios HTTP en Java. Por otro lado, Velocity, de Apache también, es un motor de plantillas que proporciona un lenguaje simple pero potente de plantillas para referenciar objetos definidos en Java. De este modo, se separa aspectos de presentación / visualización de la funcionalidad propia de la aplicación (patrón de diseño modelo-vista-controlador). Con respecto a la pagina web, Tomcat permite gestionar las peticiones web con código en Java y por tanto, conectar la aplicación web con las herramientas de Tafat. Por otro lado, Velocity permite el diseño de la página web por medio de plantillas que son rellenadas dinámicamente en la aplicación Java de la web.

Hardware Una vez visto el software utilizado, se expone a continuación el hardware empleado para el desarrollo del Profiler. Sin embargo, cabe destacar que el software utilizado no requiere el uso de un hardware de alto rendimiento.

- *Equipo informático:* el equipo utilizado fue un Dell 15R con las siguientes características:
  - *CPU:* Intel Core i5-460M.
  - *Ram:* 3.87 GB.
  - *HDD:* 500 GB.

Recursos necesarios en el despliegue

En los casos de usos que se muestran en el apartado de requisitos del software (sección: 4.3.4), podemos observar que existen diversos roles aparte del desarrollador de la aplicación. Uno de ellos es el programador, que se encarga de ampliar las capacidades del Profiler en la generación de modelos. Por otro lado, está el rol del modelador, que es el usuario final que utiliza el Profiler para generar los modelos de modo automático. Atendiendo a cada uno de ellos, los recursos necesarios para la utilización del Profiler variarán.

Programador Los recursos que utilizará el programador en cuanto al software serán exactamente los mismos que se utiliza en el desarrollo de la aplicación, pues su trabajo consiste en ampliar las capacidades del Profiler en la generación de modelos. Para ello, es necesario el uso de herramientas de desarrollo.

- *JVM:* Java Virtual Machine. Es un entorno de ejecución de una plataforma independiente que convierte el código Java a lenguaje máquina y lo ejecuta. Una JVM simula un procesador real Java, permitiendo que el código Java sea ejecutado como acciones o llamadas al sistema a cualquier procesador sin importar el sistema operativo.
- *JRE:* Java Runtime Environment. JRE incluye la JVM, librerías y componentes que son necesarios para ejecutar programas escritos en Java. JRE está disponible en diversas plataformas, entre las que se encuentran MAC, Windows y Unix.
- *JDK:* Java Development Kit. Es un kit de desarrollo de software (SDK) para la creación de programas en Java. JDK viene con un JRE completo.
- *IDE:* Integrated Development Environment. Es una aplicación software que proporciona facilidades a los programadores para el desarrollo de software.
- *Sistema operativo:* la elección del sistema operativo vendrá condicionada por, en primer lugar, Java y, en segundo lugar, por el IDE escogido (en el caso de que se vaya a utilizar uno).

- *Librería Jxl:* librería que permite la lectura de ficheros XLS, facilitando a los programadores de Profiler la creación de modelos basados en datos obtenidos de una tabla de Excel.
- *Librería Matemática de Apache:* es una librería ligera, que contiene componentes matemáticos y estadísticos que tratan de solventar los problemas más comunes que no están contemplados en Java. Extraído de: http://commons.apache.org/math/.
- *Librería POI de Apache:* Apache POI es un proyecto dirigido por Apache Software Fundation que proporciona librerías Java para la lectura y escritura de archivos en formato Microsoft Office como Word, PowerPoint y Excel.
- *Librería Jackaccess:* es una librería Java para la lectura y escritura de bases de datos de Microsoft Access. Esta librería esta dirigida a desarrolladores de aplicaciones en Java.

Hardware Las labores que realiza un programador de Profiler no requieren el uso de un hardware de alto rendimiento. Cualquier equipo informático con prestaciones iguales o superiores a las que se muestran a continuación debería ser suficiente.

- *Equipo informático:*
  - *CPU:* Intel Pentium III 500mhz o equivalente.
  - *Ram:* 384 MB.
  - *HDD:* 125 MB.

Modelador El modelador hará uso de la aplicación en calidad de usuario final de ésta. Por ello, éste no requerirá el uso de una IDE y del JDK. Por otro lado, puede ser recomendable el uso de un editor de XML para la escritura de los ficheros de entrada de Profiler.

Respecto al hardware necesario, cualquier equipo informático capaz de ejecutar aplicaciones Java, debería ser suficiente.

Software El modelador debe tener como software base, para ejecutar el Profiler, un sistema operativo y el JRE.

- *JVM:* Java Virtual Machine. Es un entorno de ejecución de una plataforma independiente que convierte el código Java a lenguaje máquina y lo ejecuta. Una JVM simula un procesador real Java, permitiendo que el código Java sea ejecutado como acciones o llamadas al sistema a cualquier procesador sin importar el sistema operativo.
- *JRE:* Java Runtime Environment. JRE incluye la JVM, librerías y componentes que son necesarios para ejecutar programas escritos en Java. JRE está disponible en diversas plataformas, entre las que se encuentran MAC, Windows y Unix.
- *Sistema operativo:* la elección del sistema operativo vendrá condicionada por Java.

Hardware Los requisitos mínimos son dependientes del sistema operativo que se utilice. Los que se muestran a continuación pertenecen a la ejecución de aplicaciones Java en Windows 7.

- *Equipo informático:*
  - *Ram:* 128 MB.
  - *HDD:* 98 MB.

Información extraída de http://www.java.com/en/download/help/6000011000.xml.

4.2. Plan de trabajo y temporización

En este apartado se verá cual ha sido el plan de trabajo establecido y la temporización para cada una de las etapas. Este trabajo ha sido dividido en diferentes etapas de las que suele constar un proyecto software; con la adición de algunas etapas propias de lo que un trabajo de fin de máster requiere. Las etapas en las que se ha dividido el proyecto son:

- Realización de la propuesta.
- Estudio de las herramientas.
  - Estudio del framework Tafat.
  - Estudio de librerías.
  - Estudio de sistemas de información geográficos.
- Análisis de los requisitos de usuario.
- Análisis de los requisitos del software.
- Diseño arquitectónico del sistema.
- Diseño arquitectónico de las aplicaciones.
- Implementación.
- Diseño y ejecución de pruebas.
- Documentación.

A continuación, para cada una de ellas, se indicará las horas que se han empleado en ellas.

La realización de la propuesta supuso en sí misma unas 20 horas, en el cual se incluyen, aparte de la redacción de la misma, los esfuerzos necesarios para acordar y pactar el trabajo de fin de máster en relación al framework de simulación Tafat.

El estudio de las herramientas ha requerido buena parte del tiempo de este trabajo. El estudio de librerías, del framework Tafat y de sistemas de información geográfico ha supuesto unas 40 horas.

La etapa de análisis también ha sido especialmente larga, debido a que los requisitos no estaban establecidos desde el principio. Por ello, se ha empleado metodología evolutiva en el desarrollo. Los requisitos se iban obteniendo a medida que se tenía contacto con los modeladores que trabajan en el Instituto Europeo para la Investigación en Energía (EIFER). El tiempo dedicado a esta tarea es, aproximadamente, de 50 horas.

La parte de diseño del software ha conllevado el diseño de la arquitectura del sistema, de la arquitectura de las aplicaciones y del formato de los perfiles que se iban a implantar para ser leídos por el Profiler. Esta etapa ha durado 60 horas.

La etapa de implementación se ha realizado 50 horas. En esta etapa se ha implementado el Profiler, algunos complementos y la página web del proyecto Tafat.

El diseño de pruebas y de validación del software con la consecuente ejecución de éstos ha conllevado, aproximadamente, unas 60 horas de la ejecución de este trabajo. En esta etapa se ha llevado a cabo el desarrollo de casos prácticos a medida que se iban añadiendo funcionalidades que ampliaban la creación de modelos a nivel de complementos.

Por último, el número de horas empleado en la documentación es de 40 horas, entres las que se incluye la escritura de esta memoria y la realización de la presentación del trabajo de fin de máster.

Actividad	**Número de horas**
1. Realización de la propuesta	20
2. Estudio de las herramientas	40
3. Requisitos de usuario	25
4. Requisitos del software	25
5. Diseño del formato de los perfiles	15
6. Diseño arquitectónico del sistema	5
7. Diseño arquitectónico de las aplicaciones	40
8. Implementación	50
9. Diseño y ejecución de pruebas	60
10. Documentación	40

Cuadro 4.1: Resumen de las horas empleadas para llevar a cabo el trabajo de fin de máster

4.3. Metodología

4.3.1. Etapa 1: realización de la propuesta

La etapa de la realización de la propuesta fue compleja ya que el framework del que Profiler depende (Tafat) no estaba totalmente desarrollado, haciendo difícil concretar los objetivos a superar.

Por ello, la propuesta se ha ido realizando a medida que se desarrollaba el trabajo en cada una de las iteraciones (metodología evolutiva), pudiendo definir con mayor precisión los objetivos de Profiler.

Inicialmente, se comenzó realizando una propuesta en la que el enfoque era el de la creación automatizada de modelos de simulación en el ámbito de las redes eléctricas, considerando los requisitos obtenidos de los modeladores del European Institute for Energy Research (EIFER).

Posteriormente, y tras sucesivas iteraciones, se aplicó un proceso de desvinculación al ámbito de las redes eléctricas, pudiéndose utilizar Profiler como un generador de modelos para simulaciones de diversa índole, gracias a su arquitectura basada en complementos (sección: 4.3.7). De este modo, los complementos establecerán el ámbito en el que se trabaja: redes eléctricas, tráfico aéreo...

Tras este cambio de rumbo en el trabajo, el enfoque que debía tener la propuesta debía ser modificado para que ligara con esta nueva concepción. Por ello, Profiler se postula como un generador de modelos multi-ámbito que, en este trabajo, se centra en la generación automatizada de modelos de simulación en el ámbito de las redes eléctricas. Por ello, en el apartado de objetivos de la propuesta, se especifica como objetivo principal la creación de un generador de modelos para el framework Tafat, sin vincularlo a un ámbito concreto

Finalmente, la planificación temporal se ha estimado en función del número de horas que se va a dedicar a cada una de las etapas. No obstante, en un desarrollo evolutivo es difícil estimar el tiempo de dedicación de las etapas, ya que éstas se aglomeran en las iteraciones, habiendo iteraciones en las que se dedique más tiempo a unas etapas que a otras.

4.3.2. Etapa 2: estudio de las herramientas

El estudio de las herramientas ha ocupado buena parte del desarrollo de este trabajo. Al inicio de este trabajo, el framework Tafat estaba empezando a diseñarse, experimentando muchos cambio. Por ello, el estudio del framework era más complicado al no tener un diseño estable.

Sin embargo, parte del desarrollo del framework ha sido realizado por el propio autor del trabajo, lo cual hacía que el propio estudio y conocimiento de las herramientas se hiciera a medida que se realizaba el desarrollo.

No obstante, cuando el autor del trabajo se integró en el grupo de desarrollo, el proyecto Tafat ya había comenzado, haciendo necesaria la adaptación a los instrumentos y herramientas que se habían desarrollado hasta ese momento.

Aparte del estudio de Tafat, ha sido necesario estudiar otras herramientas para llevar a cabo el proyecto. Entre éstas se destaca el estudio de librerías y de sistemas de información geográficos.

Estudio de Tafat

Inicialmente, el autor del trabajo no formaba parte del desarrollo del framework y, por ello, el esfuerzo para adquirir los conocimientos necesarios fue mayor. Posteriormente, siendo ya desarrollador del framework, el conocimiento acerca del mismo se iba asimilando a medida que se iba desarrollando éste.

Por otro lado, al ser un framework en desarrollo, la definición de requisitos para Profiler fue más complicada, pues no sé podían fijar correctamente sin tener previamente los de Tafat fijados.

Al igual que los requisitos de Profiler, los requisitos de Tafat eran recabados a través del grupo de geosimulación del Instituto Europeo para la Investigación en Energía (EIFER).

Estudio de la librería Jxl

La inclusión de una librería para leer ficheros de Excel se hizo necesaria tras ver los requisitos que modeladores de EIFER tenían. La idea de Profiler, en sí, es muy genérica: la generación automática de modelos. Los datos de los que se dispone para generar una escena son variantes. Sin embargo, no se suele disponer de datos que permitan alcanzar un nivel de detalle alto. Por ello, Profiler debe adaptarse a los datos de los que sí se disponga, para posteriormente generar el resto basándose, por ejemplo, en estadísticas.

Por ello, un buen modo de proporcionar los datos es a través del uso de ficheros de Excel. De este modo, los generadores de modelos pueden hacer uso de éstos para extraer los datos que se necesiten.

La dotación de esta funcionalidad en Profiler requirió hacer una búsqueda exhaustiva sobre las diferentes alternativas que existen para la lectura de ficheros Excel. Después de estudiar diversas librerías se escogió Jxl por ser una librería ligera y de fácil uso.

La interfaz que presenta esta librería es bastante completa, sin embargo, decidió hacerse un conjunto de funciones que usan esta interfaz, creando una capa superior que permite abstraer al programador de los detalles técnicos de los ficheros Excel. La realización de esta interfaz de alto nivel requirió un estudio intensivo de la librería Jxl.

Estudio de la librería matemática

El uso de una librería matemática es indispensable para la ejecución de Profiler, pues en muchas ocasiones éste debe partir de datos incompletos o estadísticos que definen el escenario a generar. Habitualmente estos datos estadísticos pueden conllevar el uso de distribuciones de cualquier naturaleza.

Por ejemplo, se puede crear un escenario en base a la descripción de una sociedad descompuesta en grupos socio-económicos. Cada grupo, a su vez, tiene una descripción que indica la posibilidad de, siendo de ese grupo, disponer de un determinado electrodoméstico. Estas posibilidades pueden representarse mientras curvas de distribución. Pese a que la librería Math de Java tiene ciertas funciones para el cálculo de herramientas estadísticas, ésta no es suficiente en algunas ocasiones.

Por ello, se realizó una búsqueda de librerías en Java que tuvieran soporte para distribuciones. Aunque existen muchas librerías que implementan distribuciones matemáticas, pocas son las que aseguran funcionar correctamente.

Tras observar unas cuantas, se decidió escoger la librería matemática desarrollada por Apache, ya que es una de las que certifica funcionar correctamente y, además, contiene otras muchas utilidades matemáticas.

Estudio de la librería Apache POI

Esta librería permite la lectura y escritura de ficheros con formatos propios de Microsoft Office (Word, PowerPoint y Excel). El interés de añadir esta librería es, sobretodo, por la interacción

con ficheros de Excel. Jxl ya realiza esta tarea, sin embargo, no tiene compatibilidad con el nuevo formato de Excel (.xlsx). Por ello, se ha incluido también esta librería que ha sido estudiada para la implementación de alguno de los complementos. Además, se ha incluido en el motor de simulación de Tafat para la exportación de datos de simulación.

Estudio de la librería Jackcess

En la etapa de análisis, un modelador indico la necesidad de trabajar con bases de datos de Microsoft Access. En ese momento, se comenzó a realizar una búsqueda para realizar esta tarea en aplicaciones Java. Finalmente, se optó por esta librería, ya que facilita el acceso a los datos. No obstante, siempre existe la posibilidad de hacer una conexión con la base de datos usando el driver JDBC de Java.

Sistemas de información geográficos

Uno de los requisitos de la realización de este trabajo de fin de máster es la adquisición de algunos conocimientos en el uso de sistemas de información geográficos; pues estos conocimientos serán necesarios para crear escenarios a partir de una base de datos procesada con sistemas de información geográfico.

Para ello, se realizó una práctica para la extracción de datos de un conjunto de ficheros de información geográfica utilizando la aplicación ArcGIS. Estos ficheros contenían información sobre la distribución y posición de edificios de una determinada región de Francia

El procesamiento de los datos condujo a la creación de una base de datos que contenía la información más importante de cada uno de los edificios de la región. Esta base de datos serviría como un fichero de datos de entrada de Profiler para la generación del escenario de esa región

4.3.3. Etapa 3: análisis de los requisitos de usuario

La metodología de desarrollo de este proyecto es la evolutiva. Esto implica que no se capturaron todos los requisitos de usuario al mismo tiempo, sino que se han ido capturando a medida que las iteraciones de la metodología evolutiva se sucedían.

En este caso, existen dos tipos de usuarios finales: el programador y el modelador. El usuario final más importante de estos dos es el modelador, ya que es el que hará uso del framework con la finalidad con la cual se concibió: el estudio de sistemas complejos. Sin embargo, la figura del programador es imprescindible para los modeladores, pues es el que se encarga de ampliar las funcionalidades que permiten generar los modelos. Esta ampliación de funcionalidades debe responder a las necesidades con las que el modelador se ha encontrado a la hora de generar el escenario.

Requisitos de usuario del programador

Los requisitos de usuario del programador se han ido descubriendo en cada etapa del ciclo evolutivo, pudiendo refinar cada vez más la arquitectura de la aplicación para que ésta resulte cómoda y fácil de manipular a la hora de ampliar funcionalidades.

En la primera iteración se extrajo el requisito de realizar ampliaciones de funcionalidades de modo incremental.

Posteriormente, en la segunda iteración, se consolidó esa concepción de realizar ampliaciones en un formato de complementos; en el cual cada complemento se encargaría de realizar una función determinada a la hora de generar el modelo. Concretamente, cada complemento generaría las partes del modelo correspondientes a un elemento de simulación determinado. De este modo, cada complemento se relacionaría con los elementos de simulación contemplados en el metamodelo para un ámbito determinado. Teniendo una arquitectura basada en complementos, el siguiente requisito debía versar en la necesidad de procesar el fichero principal de entrada de Profiler de modo transparente a los complementos. Además, debía diseñarse una interfaz homogénea entre los diferentes complementos de Profiler.

En una última iteración, se añadió como requisito de usuario la creación de un conjunto de métodos que faciliten el manejo de los documentos XML, ayudando en la generación de los modelos de simulación.

Requisitos de usuario del modelador

La adquisición de los requisitos del modelador es en lo que se ha centrado la ejecución de esta etapa.

Para ello, fue necesario realizar diversas reuniones con los modeladores del instituto EIFER, los cuales establecieron sus prioridades a la hora de generar escenarios, teniendo en cuenta los datos con los que suelen contar para recrear un lugar.

A raíz de esas reuniones, se estableció como necesidad del modelador la flexibilidad en el desarrollo de modelos basados en diferentes conjuntos de datos y tipos de entidades. Por ejemplo, la creación automatizada de edificios podría hacerse basándose en la distribución de edificios de una base de datos de Stuttgart o de Estrasburgo.

Por otro lado, tras las reuniones, otro requisito que se extrajo es la necesidad de añadir recursividad en la aplicación, haciendo posible llamar de un complemento a otro. Por ejemplo, la generación de un edificio a través del complemento que los genera podría conllevar la ejecución del complemento de viviendas para generar las que se hallan en el interior del edificio.

Por lo tanto, en una primera iteración se capturó la necesidad de tener una herramienta lo suficientemente versátil cómo para permitir leer diferentes tipos de datos.

En la segunda iteración, surgió el requisito de poder crear un tipo de entidades de distintos modos, a través del cambio de la base de datos o simplemente por crearlos usando diferentes métodos. Por ejemplo, podría generarse una ciudad cuyos edificios se agrupan en barrios o, simplemente, generarla como una agrupación de edificios que no se agrupan en barrios.

Finalmente, surgió la necesidad de poder conectar los diferentes módulos de generación automática de elementos de simulación. De este modo, sería posible concatenar la creación de un edificio con la de una vivienda y ésta a su vez con la de un electrodoméstico.

Capacidades generales de Profiler

Tras recopilar todos los requisitos de ambos usuarios finales a lo largo de las diferentes iteraciones el resumen de las capacidades que debe tener la aplicación es:

- Facilidad para la creación de nuevos complementos de Profiler
  - Transparencia en el procesamiento del fichero principal de entrada de Profiler.
  - Interfaz homogénea en los complementos, facilitando la conexión con el núcleo y entre ellos
  - Conjunto de métodos que faciliten el tratamiento de nodos de ficheros XML
- Versatilidad para la generación automatizada de modelos
  - Capacidad para leer diferentes tipos de datos (bases de datos, ficheros...) en los cuales se basarán la generación
  - Generación de diferentes modos de un mismo tipo de elementos de simulación: cambiando el origen de los datos ó usando diferentes modos para generarlos
  - Generación recursiva en la que un complemento pueda utilizar otro complemento

Características de usuario

A lo largo de todo el capítulo se ha distinguido dos tipos de usuarios: el programador y el modelador. Cada uno de ellos tendrá que disponer de conocimientos específicos para hacer uso de la herramienta Profiler.

El programador debe tener conocimientos de programación en Java. Además, debe conocer correctamente la arquitectura del Profiler, de cara a saber cómo debe programar los complementos. Por otro lado también debe tener conocimientos del metamodelo de Tafat, en el ámbito en el que trabaje, para que la generación del modelo sea compatible con el simulador.

El modelador debe conocer los complementos existentes en Profiler para poder saber cual le conviene en cada momento y qué parámetros de entrada puede configurar. Por otro lado, puede ser necesaria la manipulación de los ficheros de datos para alinear los datos disponibles con el formato aceptado por un determinado complemento de Profiler. Además, debe tener conocimientos del framework Tafat.

Ambiente operacional

Dependiendo del usuario final, las características del sistema informático variarán. En líneas generales, el programador requerirá un sistema informático de mayor rendimiento que el modelador quién sólo necesita un sistema capaz de ejecutar aplicaciones en Java. En la sección 4.1.2 se explica con mayor detenimiento.

4.3.4. Etapa 4: análisis de los requisitos del software

En esta etapa se deben identificar cuáles son los requisitos que el software debe cumplir a la hora de ser diseñado y programado. Para ello, hay que apoyarse en los requisitos de usuario que se identificaron. Teniendo en cuenta los requisitos de usuario se procedió a elaborar los del software. En esta etapa, lo importante es saber "traducir" esos requisitos capturados en la anterior etapa a requisitos que el software debe cumplir para implementarlos.

Los requisitos de usuario que se han capturado se pueden dividir en dos tipos: facilidad para crear nuevos complementos y versatilidad para la generación automatizada de modelos. El primero de ellos, está claramente ligado con el rol de programador, mientras que el segundo se relaciona más con el del modelador. Teniendo clara esta distinción, basada en roles, la realización de diagramas de caso de uso para cada rol es de gran ayuda para observar los requisitos que el software debe tener.

Casos de uso

Los casos de uso son una herramienta muy útil para transferir el conocimiento capturado en la etapa de requisitos de usuario a los diseñadores del software. La creación de los casos de uso aclarará las funcionalidades que el software debe tener.

En primer lugar, se comentarán los diferentes roles que se han identificado en el proceso de análisis y el trabajo que realiza cada uno de ellos en el entorno de Profiler. Posteriormente se mostrará los casos de uso de aquellos roles que interactúan directamente con Profiler.

Roles

En el análisis del framework Tafat se han identificado cuatro roles: experto del dominio, modelador, programador y desarrollador de Tafat. Estos mismos roles continúan estando presente en el desarrollo de esta herramienta.

En primer lugar, el experto del dominio será el que asesora el diseño de los datos de entrada para Profiler, indicando que elementos son necesarios considerar y cuales no. Además, en el caso de que sea necesario, puede diseñar, junto con el modelador, la salida que se debe esperar tras ejecutar Profiler. Es decir, se encarga de aquellos aspectos del diseño del modelo y de los datos que tienen que ver con el objeto de estudio y del cual él es experto.

El modelador es el usuario final por excelencia del framework Tafat y, en consecuencia, de esta herramienta. Su misión es preparar el conjunto de datos de entrada que servirán para la generación automática de modelos. Posteriormente deberá crear el fichero de entrada de Profiler y procesarlo para obtener el modelo de simulación compatible con el simulador de Tafat. En este proceso, el modelador puede darse cuenta de la necesidad de modificar o crear un complemento de Profiler para generar el modelo deseado. Para ello, debe contactar con el programador.

El programador puede encargarse de diversas tareas dentro del framework Tafat. Entre ellas está la creación y edición de complementos de Profiler. Una vez el modelador ha declarado su intención y necesidad de modificar Profiler para procesar sus datos, el programador entra en escena. Para ello, debe hacer un estudio previo de los complementos ya disponibles en Profiler. A

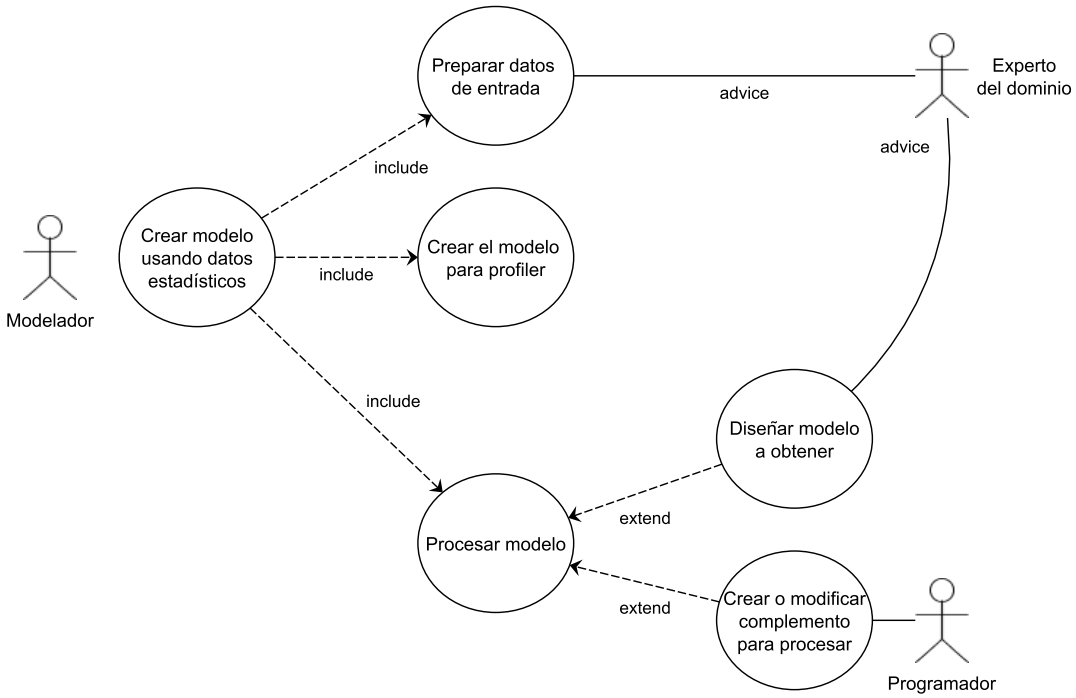

Figura 4.11: Caso de uso del modelador

la hora de implementar cambios o crear nuevos complementos, éste puede ser asesorado por un desarrollador de Tafat.

El último de los roles identificados es el desarrollador de Tafat. Este rol se encarga del desarrollo y mantenimiento del framework, sustentando, entre otras herramientas, Profiler.

Modelador

En primer lugar, se mostrará el caso de uso del modelador, ya que el modelador puede conllevar la creación o modificación de un complemento, momento en el cual es necesario el programador.

El principal objetivo para el modelador es la generación de un modelo sea ejecutable en el framework (figura: 4.11). Para ello, en primer lugar, el modelador debe preparar los datos de entrada que guiarán la creación del modelo. En esta acción puede ser ayudado por un experto del dominio que le indique los datos que son necesarios considerar y cuales no.

Una vez los datos de entrada están preparados para ser procesados, se puede comenzar a crear el fichero que será el principal dato de entrada del Profiler. Este fichero contendrá las directrices de creación del modelo e indicará la o las fuentes de datos que se deben utilizar en el proceso de generación del modelo.

Una vez se ha diseñado el fichero de entrada del Profiler y los datos de entrada, se puede proceder a la generación del modelo basado en ambos parámetros. En ocasiones, y si se ha hecho considerando los complementos ya existentes en Profiler, no será necesario hacer ningún trabajo extra. Sin embargo, por la diversidad de tipos de datos de entrada y de posibilidades de generación de elementos de simulación, es posible que haya que modificar o crear un complemento para el

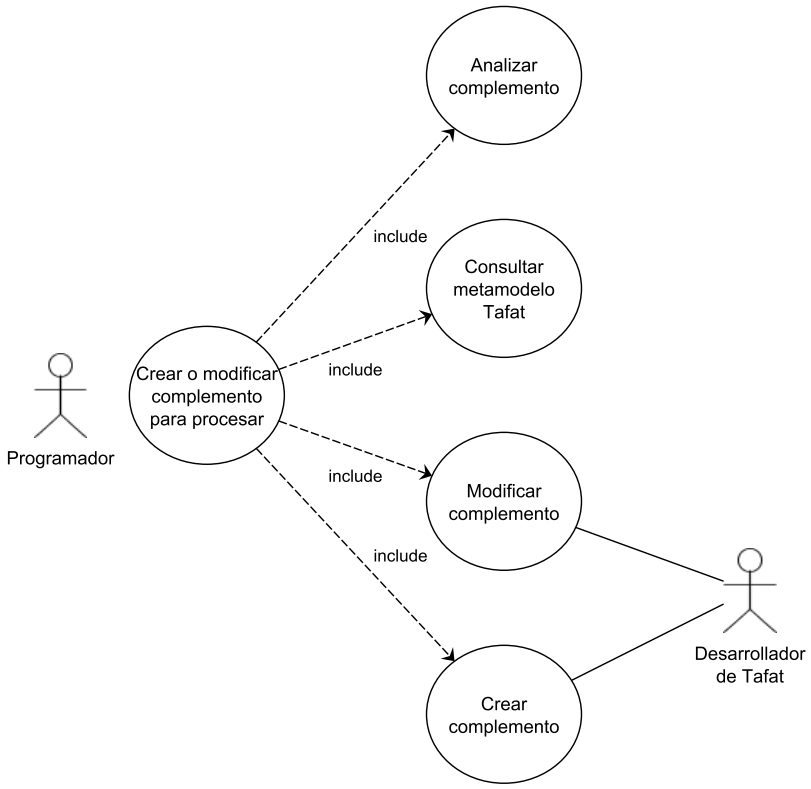

Figura 4.12: Caso de uso del programador

procesado del modelo. En este caso, antes de comenzar a crear o modificar el complemento, se debe diseñar el modelo que se pretende obtener, es decir, que es lo que quiere que el Profiler proporcione. En el diseño del modelo, el modelador puede ser asesorado por el experto del dominio. Una vez diseñado el modelo, se continua con la etapa de creación o modificación de un complemento, momento en el que intervendría el programador.

Una vez se ha creado el complemento, el fichero de entrada de Profiler puede ser procesado, obteniendo como resultado un modelo ejecutable en el simulador de Tafat.

Programador

El programador entra en acción en esta herramienta cuando es requerido para el diseño y creación de un complemento así cómo para la modificación de éstos. Una vez el modelador se pone en contacto con el programador (figura: 4.11), éste debe realizar un análisis de los requisitos del modelador para comprobar si ya existe algún complemento que sea similar o si es necesario crear uno nuevo.

Por lo tanto, en el caso de uso del programador (figura: 4.12), existe una primera etapa de análisis de complementos en la que se decidirá que vía seguir: modificar un complemento ya existente, generando uno nuevo pero alternativo a ese, o crear uno nuevo. El proceso de análisis de complementos puede ser también compaginado con la consulta del metamodelo de Tafat, observando los atributos existentes en cada elemento de simulación.

Una vez terminada esta etapa de análisis, el programador puede comenzar a trabajar en los complementos, ya sea para modificarlos o para crear uno nuevo. Este proceso se realizará en un entorno de programación de Java. El complemento deberá tener un formato predeterminado, acorde con el diseño propuesto por los desarrolladores de Profiler, para poder ser añadido y usado. En este proceso de implementación, el programador puede ser asesorado por un miembro del framework Tafat.

Restricciones

La mayoría de los desarrollos software están sujetos a restricciones del tipo que sean: temporales, económicos, tecnológicos... En este apartado se comentará brevemente las restricciones a las que está sujeto este desarrollo.

Una restricción temporal es impuesta por el propio estudiante de máster que realiza el trabajo. Este trabajo debe finalizarse antes de que termine Julio, ya que es la época más conveniente para presentarlo debido a las circunstancias en las que éste se encuentra. El estudiante está realizando un doctorado que requiere estancias en el extranjero, por eso el trabajo de máster no puede ser presentado en cualquier momento, siendo Julio el mes idóneo para ello.

En cuanto al aspecto tecnológico, esta herramienta forma parte de un framework para el desarrollo de simulaciones. Esto implica que la tecnología empleada para su desarrollo debe ser acorde a la que el framework emplea. Este framework, pese a tener un diseño que permite migrar a otras tecnologías (MDE), ha sido desarrollado en Java. De este modo, el uso de Java como tecnología de desarrollo se impone como restricción.

Requisitos

El principal objetivo de esta herramienta es tener la capacidad, usando diferentes tipos de datos, de generar modelos que sean ejecutables en el simulador de Tafat. Este objetivo se puede conseguir a través de requisitos que el software debe cumplir.

Desde el punto de vista del modelador, esta herramienta debe de ser capaz de generar un modelo, de modo automático, haciendo uso de los datos que el modelador facilita. Para ello, puede requerir el asesoramiento de un experto del dominio o un programador.

Por otro lado, el programador tiene como principal requisito la posibilidad de realizar o modificar complementos. Al ser éste un requisito muy abstracto, debe ser transformado en requisitos software como: claridad conceptual y semántica de lo que es un complemento, interfaz unificada para complementos y posibilidad de interactuar entre complementos.

Hasta ahora sólo se han comentado requisitos de alto nivel desde el punto de vista del usuario. A continuación se explicarán los requisitos a nivel software que son necesarios.

Uno de los más importante es acerca de los datos de entrada. Este requisito es realmente complicado, ya que la variedad de datos que se pueden suministrar son infinitos (formato, tipos de datos...). Es decir, es imposible crear un sistema capaz de aceptar e interpretar correctamente cualquier tipo de datos de entrada. Sin embargo, si es posible crear una herramienta que deje abierta la alternativa de incluir la aceptación de nuevos tipos de datos de entrada. No obstante, debe traer de serie algún soporte de acceso a bases de datos.

Por otro lado, la generación de modelos debe ser flexible, pudiendo cambiar el origen de datos (siempre que tenga un formato determinado) y/o el modo de generar el modelo. Es decir, que para un mismo tipo de generación de modelos, debe poder ser capaz de aceptar cambios en el origen de los datos y también en el modo en el que se genera el modelo. Un ejemplo de ello sería la creación de un complemento para la generación de edificios, con viviendas y dispositivos eléctricos. Este complemento, podría permitir modificaciones en el modo de generación, pudiendo indicar, por ejemplo, que se quiere la generación de todos los dispositivos menos neveras.

Gracias a la modularidad que el framework Tafat tiene implícito, Profiler se puede diseñar teniendo complementos reutilizables que generen elementos de simulación concretos (relacionándose uno a uno). Esto permitiría que, para crear un edificio con viviendas y dispositivos, se pudiera utilizar el complemento de creación de edificios y que éste, a su vez, tuviera la capacidad de utilizar el complemento de creación de viviendas y éste el de dispositivos. De este modo, sería posible la reutilización de complementos.

Aparte de los diferentes datos de entrada, debe haber un fichero principal que indique a Profiler lo que se quiere generar y que datos de entrada se van a utilizar para ello. Este fichero debe ser procesado por Profiler de modo transparente al programador, de modo que éste pueda centrarse en la creación o modificación del complemento. Además, el núcleo de Profiler debe ser capaz de detectar la existencia de un complemento sin que el programador modifique el núcleo. De modo resumido, los requisitos hallados son:

ID	NECESIDAD	PRIORIDAD	FUENTE	DESCRIPCIÓN
01	No negociable	Alta	Aplicación	Flexibilidad en los datos de entrada
02	No negociable	Alta	Aplicación	Soporte de acceso a algún tipo de base de datos
03	No negociable	Alta	Aplicación	Flexibilidad en la generación de modelos
04	No negociable	Alta	Aplicación	Conexión entre complementos
05	No negociable	Alta	Aplicación	Tratamiento del fichero de entrada
06	No negociable	Alta	Aplicación	Detección automática de nuevos complementos

4.3.5. Etapa 5: diseño arquitectónico del sistema

Antes de comenzar a diseñar la aplicación, siempre es importante dedicar un tiempo a pensar cómo va a ser la arquitectura de desarrollo y de explotación del sistema. El diseño de la arquitectura del sistema puede condicionar, en cierto modo, el diseño arquitectónico de las aplicaciones.

En este punto, también debe decidirse que tecnologías se van a utilizar en el desarrollo de la aplicación así cómo de la interfaz que la misma tendrá. En el caso de la aplicación, la tecnología a utilizar será Java, pues es una restricción que se ha introducido desde la etapa de requisitos del software. Sin embargo, en el apartado de desarrollo de la interfaz no se ha mencionado nada. Para este aspecto se ha pensado en una interfaz web cuya justificación puede verse en la siguiente sección. A continuación se mostrará la arquitectura del sistema de desarrollo para la aplicación y

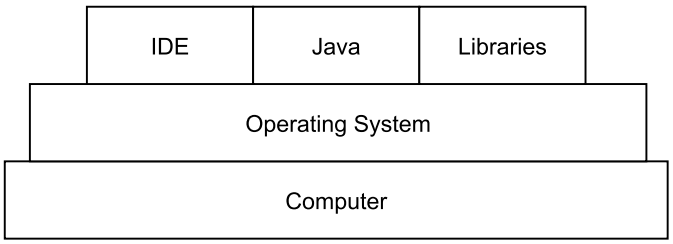

Figura 4.13: Arquitectura del sistema de desarrollo de la aplicación Profiler

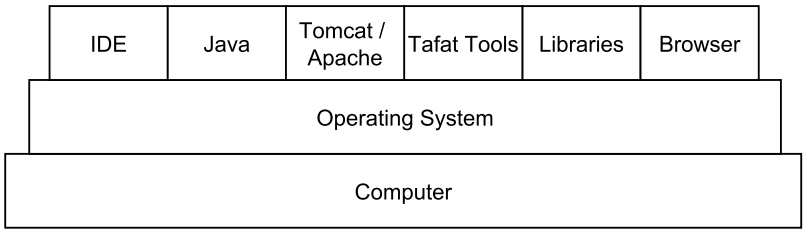

Figura 4.14: Arquitectura del sistema de desarrollo de la interfaz web

su interfaz web así cómo la de explotación acorde con el rol (programador o modelador).

Arquitectura de desarrollo

La aplicación será desarrollada en Java acorde con la restricción que proviene de la etapa de requisitos del software. Para ello, es recomendable utilizar un entorno de desarrollo para Java. En este aspecto, la elección es bastante libre, sin embargo se ha optado por Eclipse (al tener conocimientos previos de su manejo). No existe dependencia con el sistema operativo ya que el desarrollo y ejecución de aplicaciones en Java es posible en cualquier sistema operativo que tenga soporte para Java. Sin embargo, como ya se indicó en el apartado de recursos, se utilizó Windows 7. Todo ello se ejecutará sobre un ordenador que tenga capacidades suficientes para ello (figura: 4.13). Finalmente, cabe destacar la necesidad del uso de librerías para el desarrollo de la aplicación, tal y como se comenta en el apartado de recursos.

La interfaz de la aplicación será en sí otra aplicación web. Esto permitirá la ejecución de la aplicación en la nube. La arquitectura del sistema necesaria para llevar a cabo su desarrollo puede verse en la figura 4.14. Nuevamente, el sistema operativo y el ordenador no son aspectos que deban estar definidos, ya que, en principio, cualquiera de ellos es válido mientras soporte la ejecución de aplicaciones Java. El uso de un entorno de programación (IDE) es, nuevamente, sugerido y para esta ocasión, también se ha vuelto a escoger Eclipse. En este caso, cabe destacar la integración de Tomcat / Apache para el desarrollo de la página web, ya que permite procesar las peticiones web a través de una aplicación Java. De este modo, conectar la herramienta Profiler con un entorno web es mucho más sencillo. Cómo se verá en la siguiente sección, esta interfaz web finalmente será la página web del framework de Tafat por lo que será necesario disponer de las herramientas de Tafat para integrarlas. Finalmente, es necesario el uso de un navegador para comprobar que las respuestas (responses) que la aplicación web genera son correctas.

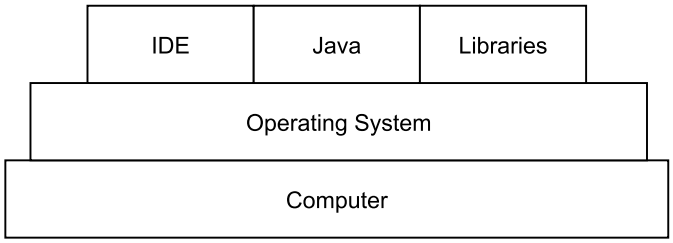

Figura 4.15: Arquitectura del sistema de explotación de la aplicación del programador

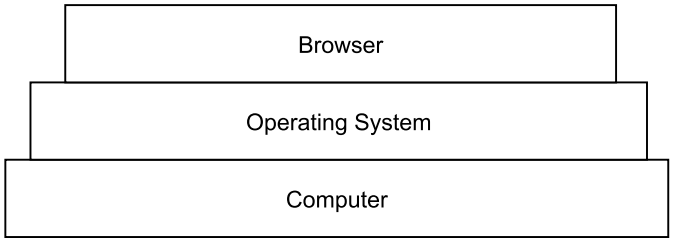

Figura 4.16: Arquitectura del sistema de explotación de la aplicación del modelador

Arquitectura del sistema de explotación

En este apartado se mostrará la arquitectura del sistema en el despliegue acorde a los diferentes roles identificados (sección: 4.3.4). Uno de los roles identificados era el programador. El programador será el que esté al cargo del desarrollo de nuevos complementos de procesamiento de modelos, siguiendo las directrices del modelador y/o experto del dominio. Cómo desarrollador de complementos, necesita la misma arquitectura del sistema que se describió para el proceso de desarrollo (figura: 4.15). En este caso, el programador tendrán también libertad a la hora de elegir el ordenador, sistema operativo o entorno de programación, mientras sean compatibles con Java.

En el caso de explotación del modelador, éste sólo necesitará el navegador para procesar su fichero de entrada de Profiler en la página web de Tafat, apartado Profiler. En este caso, la elección del ordenador, sistema operativo y navegador es también indiferente.

4.3.6. Etapa 6: diseño de la interfaz

La etapa de diseño de la interfaz es importante, y debe ser estudiada cuidadosamente. En el momento en el que se realizó el estudio de la interfaz, no existía prácticamente ninguna interfaz definida para otras herramientas del framework. Sólo existía la página web del metamodelo de Tafat que muestra los diferentes elementos que se pueden poner en una simulación (figura: 4.17).

La realización de esta herramienta en Java permite una fácil utilización de la aplicación en un entorno de cloud computing o de computación en la nube. Gracias a tecnología como JSP (Java Server Pages) es posible realizar páginas web que ejecuten aplicaciones en Java. El uso de aplicaciones en la nube permite mantener actualizada la versión de la página web permitiendo a los usuarios disponer de ella. En cambio, si se usara aplicaciones de escritorio, el usuario tendría que actualizarla cada vez que se publicara una nueva versión.

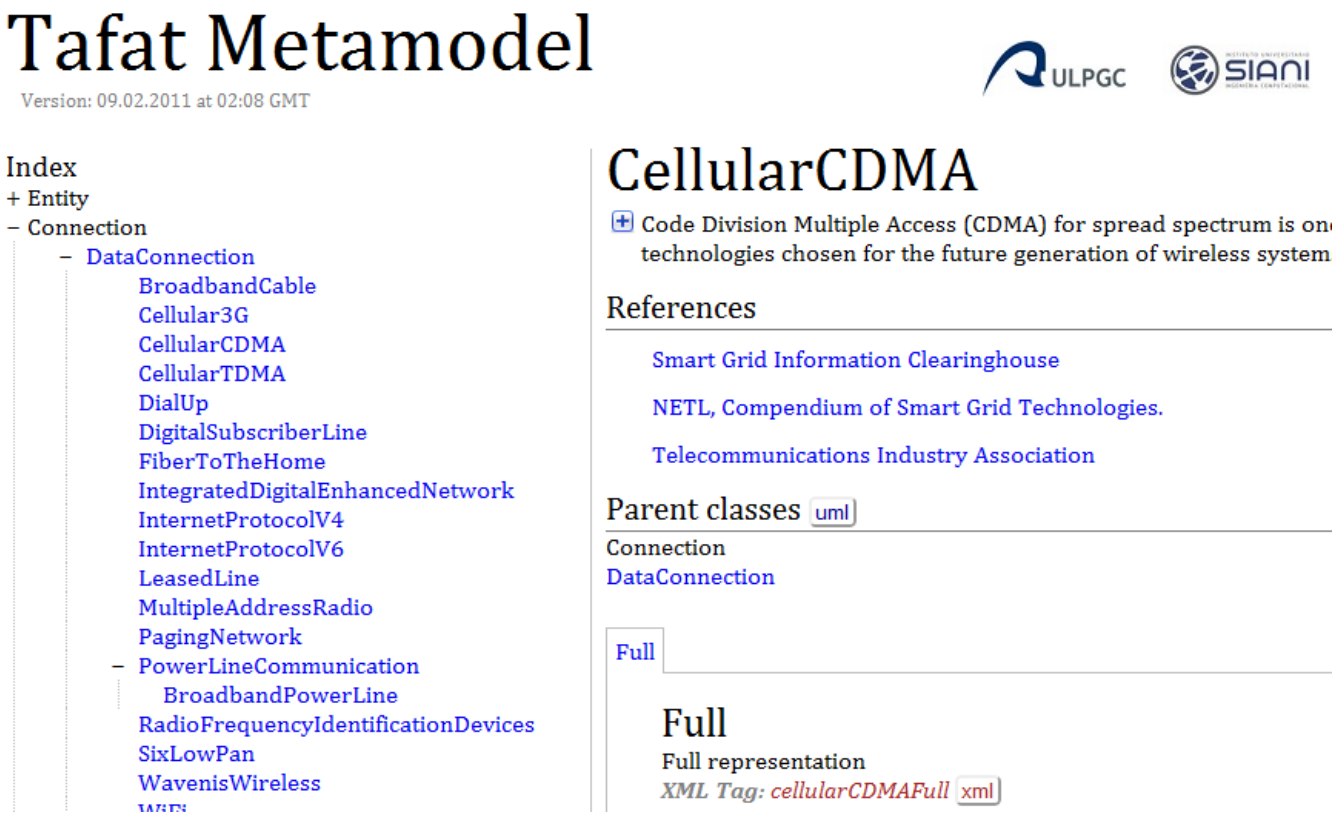

Figura 4.17: Página web generada por el traductor del metamodelo a HTML en el ámbito de las redes eléctricas

Por otro lado, uno de los primeros objetivos que persigue el framework Tafat es la reutilización y compartición de elementos creados. Siguiendo esta filosofía, la computación en la nube es un modelo de prestación de servicios que parece muy adecuado. Por ello, se ha optado por la realización de una página web cómo interfaz de ésta y otras herramientas del framework Tafat. En cierto modo, el diseño de la página web (de la interfaz) conllevará un gran desarrollo en el momento de implementación, pues la página web no sólo abarcará la herramienta Profiler, sino que integrará otras, convirtiéndose así en la página web principal del framework Tafat.

Diseño de la página web

Debido a que la página web va a ser la principal de la plataforma Tafat, ésta debe ser diseñada para permitir la adición de nuevos ámbitos de simulación. Actualmente se están trabajando dos ámbitos: redes eléctricas y control del tráfico aéreo. Sin embargo, en el futuro, el número de ámbitos podría ampliarse. Por ello, la página web debe estar diseñada para la fácil adición de nuevos ámbitos y herramientas que cada ámbito tenga.

Cómo se observa, la página web no será una simple interfaz del Profiler sino de toda la plataforma de Tafat. El diseño de la página web debe ser bastante cómodo para navegar y las interfaces serán dependientes del contenido que se muestre. Por ello, también es conveniente que el diseño se

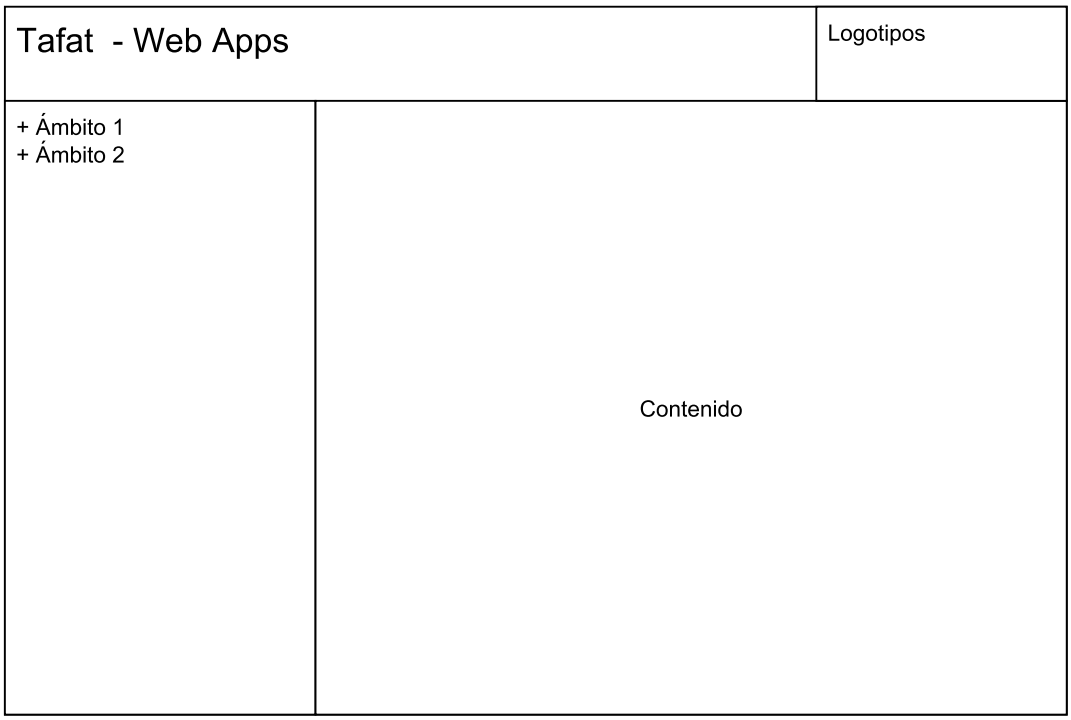

Figura 4.18: Diseño general de la página web. Gestión de ámbitos.

realice permitiendo escalar el sistema, no sólo en el número de ámbitos, sino también en el número de herramientas.

Otro requisito es el control de la accesibilidad a los ámbitos. Algunos de los ámbitos en los que se trabaja deben tener una especial privacidad, de modo que, no sea posible descargar contenidos por parte de cualquier usuario. Para ello, la página web debe permitir configurar contraseñas por ámbitos. Dos prototipos se pueden observar en las figuras 4.18 y 4.19. En la primera se observa cómo visualizarán los ámbitos, mientras que las segunda muestra cómo se podría acceder a las aplicaciones o elementos de cada ámbito.

4.3.7. Etapa 7: diseño arquitectónico de las aplicaciones

Hasta ahora, ya se han extraído los requisitos que debe cumplir el software para satisfacer las necesidades de los usuarios finales. A partir de esta etapa, los esfuerzos se centran en el análisis de esos requisitos para diseñar una arquitectura que los contemple. En esta sección se mostrarán las soluciones arquitectónicas adoptadas, tanto para el Profiler cómo para la página web del framework Tafat.

Diseño arquitectónico del Profiler

Tras el estudio de los requisitos de usuario y, sobre todo, del software se ofrecerá una solución arquitectónica. Teniendo en cuenta el paradigma de desarrollo del framework Tafat, se debe mantener la abstracción entre la funcionalidad de la herramientas y los ámbitos en los que se emplea.

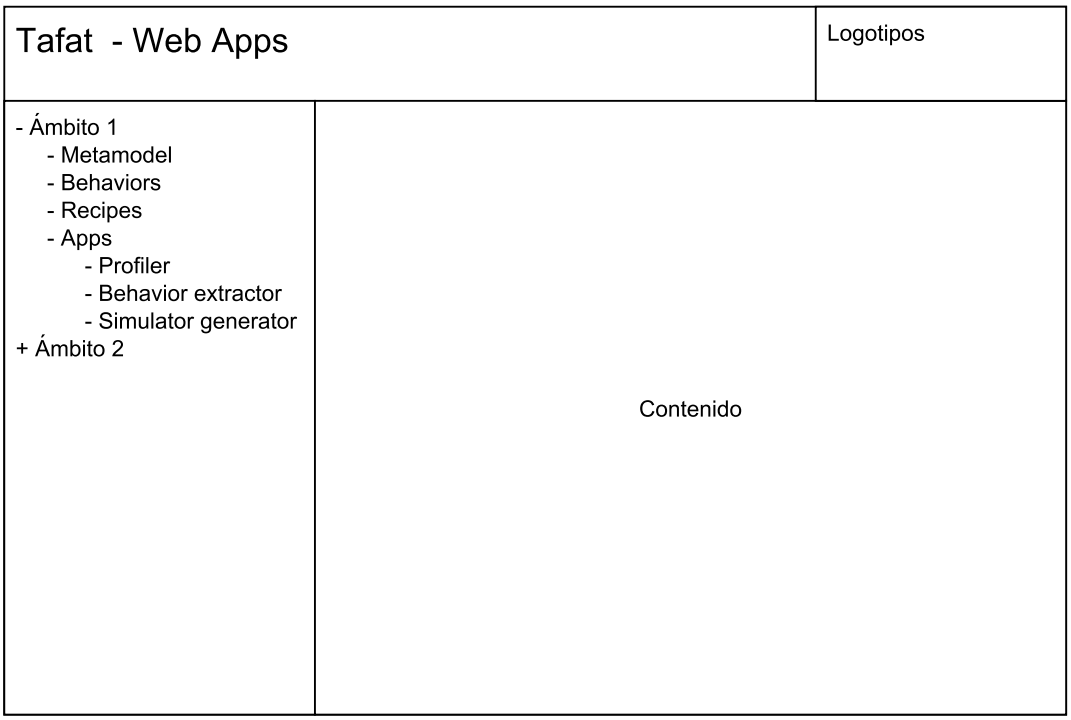

Figura 4.19: Diseño de la web en la gestión de elementos dentro de ámbitos

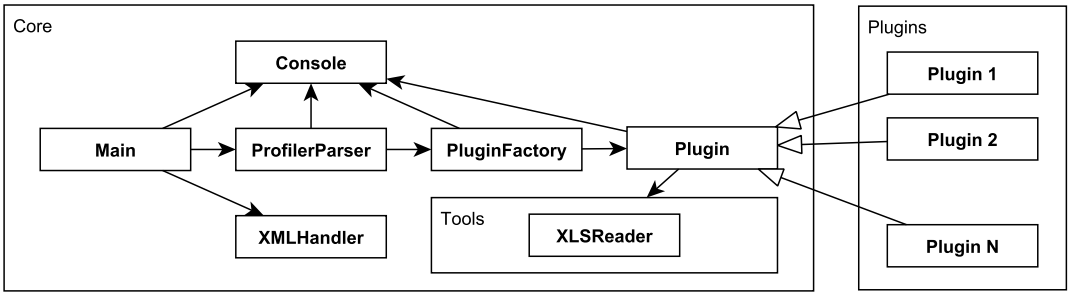

Figura 4.20: Arquitectura del profiler

Por ello, el diseño debe tener en cuenta que esta herramienta debe ser lo suficientemente versátil para poder funcionar en diferentes ámbitos: redes eléctricas, tráfico aéreo...

Atendiendo a esta necesidad de abstracción, el diseño considera una arquitectura desacoplada en un núcleo y complementos (plug-in). El núcleo será la parte que mantiene el nivel de abstracción que se observa en el resto del framework Tafat y que permitirá ser usado en diferentes ámbitos. Por otro lado, los complementos se encargarán de contextualizar la herramienta para que genere modelos acorde con el ámbito en el que se trabaje. Por ejemplo, en el ámbito de redes eléctricas habrá complementos para la creación de electrodomésticos y, en el tráfico aéreo habrá complementos que generen aeropuertos, aviones... Partiendo de estas premisas, la arquitectura que se presenta puede verse en la figura 4.20.

La ejecución del Profiler comienza pidiendo la ruta del archivo que se quiere procesar (sección:

4.3.8). Posteriormente, éste es abierto usando la clase XMLHandler que devolverá el fichero con una estructura de documento XML (Document de Java). El documento XML se enviará al Profiler-Parser para que lo procese buscando las cláusula profiler. Cada vez que se encuentre una cláusula profiler, ProfilerParser enviará el nodo XML que contiene la cláusula a la clase PluginFactory.

Esta clase se encargará de procesarlo y encontrar el complemento que atiende dicha cláusula profiler. Una vez lo encuentre, la clase PluginFactory enviará el nodo a procesar al propio complemento, el cual tiene una interfaz bien definida (sección: 4.3.8). Todos los complementos deben cumplir con esa interfaz para que el PluginFactory pueda invocarlos. Por ello, todos complementos heredan de la clase Plugin. Esta clase proporciona la interfaz que los complementos debe tener, además de, diversas herramientas que facilitan el manejo de los nodos XML, ya que son funciones que deben usar los complementos para generar la porción de modelo que sustituya la cláusula profiler. Por otro lado, la clase Console puede ser utilizada en todo momento para imprimir por pantalla; ya sea desde el propio complemento o desde el núcleo.

El núcleo, además de las clases comentadas, también posee un apartado de herramientas (Tools). Estas herramientas ayudan a los desarrolladores de complementos para Profiler. Entre las diferentes herramientas, destaca la del manejo de ficheros de Excel (XLS) que permite usar los datos de entrada de un fichero Excel a la hora de guiar la creación automatizada de un modelo. Esta funcionalidad se lleva a cabo gracias a la clase XLSReader que hace de interfaz entre la librería Jxl (sección 4.3.2) y los complementos. Esta interfaz se compone de procedimientos que permiten la lectura de elementos de un fichero XLS.

Diseño arquitectónico de la página web

Inicialmente se pensó en la página web como una herramienta para la ejecución de Profiler única y exclusivamente. Sin embargo, surgió como requisito crear la página web general de Tafat. En ella, se ofrecerán servicios relacionados con el manejo de modelos. El aspecto más importante que se ha plasmado en la arquitectura es la escalabilidad, tanto en el número de ambientes, como en el número de aplicaciones que se pueden ejecutar por ambiente.

Este aspecto de escalabilidad se ha contemplado de dos modos diferentes acordes con la naturaleza de cada uno de ellos. La escalabilidad a nivel de ambientes se regula por medio de un fichero XML que define los ambientes y los apartados que cada ambiente ofrece. A su vez, estos apartados se consideran internamente como aplicaciones que se buscarán cuando se envíe una petición al servidor. La escalabilidad en cuanto aplicaciones se ha conseguido al crear una interfaz genérica para todas ellas que permite añadir o quitar aplicaciones que el núcleo puede ejecutar.

Entrando en detalle con los elementos que se observan en la figura 4.20 se puede observar que todas las peticiones entran a través del AppController. Este paradigma de desarrollo web es el que se emplea en la creación de páginas web usando JSP. Se ha escogido esta tecnología para el desarrollo de la página web ya que todas las herramientas que contiene el framework de Tafat están escritas en Java. Por ello, usando JSP, la ejecución de las herramientas es más sencilla.

Las peticiones que recibe el AppController son enviadas al ContentHandler. Esta clase se encarga de comprobar si esa sesión tiene acceso al uso del ambiente que se indica en la petición. Para ello, consulta el acceso en el ControlAccess. Si este le responde afirmativamente, continuará procesando la petición. En caso de que no se tenga acceso, se mostrará el formulario de acceso para

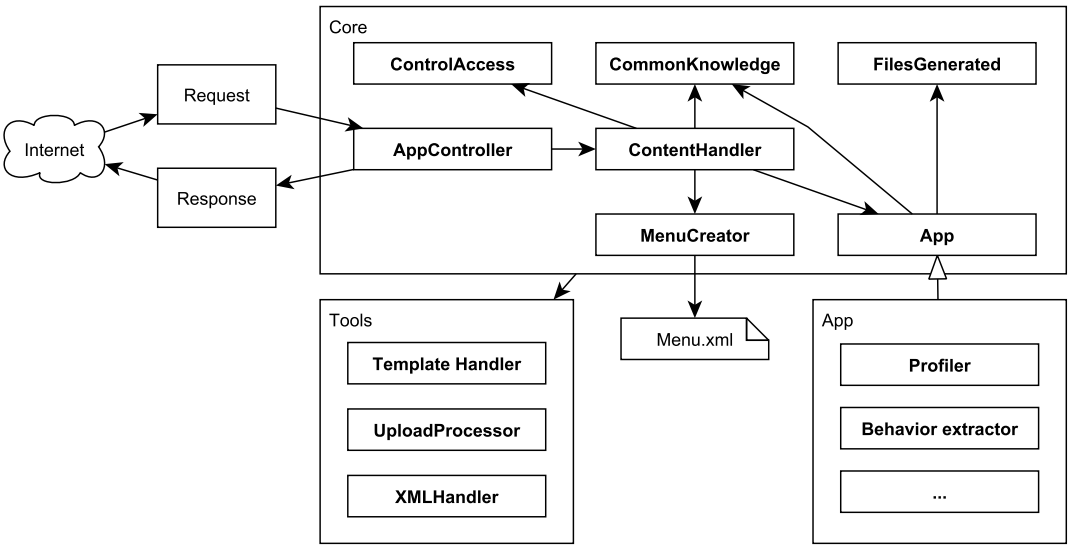

Figura 4.21: Arquitectura de la página web

que se introduzca el nombre de usuario y contraseña.

En caso de que sí se tenga acceso, se derivará la petición a la aplicación correspondiente, de acuerdo con los parámetros que se incluyen dentro de la petición. En última instancia, la aplicación se encargará de devolver el contenido que se quiere mostrar en la respuesta (response), de acuerdo con la petición que le llega. Una vez el ContentHandler recibe la respuesta que proviene de la aplicación, genera el contenido que debe ir en el marco del menú llamando a MenuCreator. Una vez se tiene el contenido que genera la aplicación y el del menú, ContentHandler está listo para devolver ambas cosas como resultado al AppController, quien se encargará de crear una respuesta (response) con esos contenidos.

Las aplicaciones que se observan en el diagrama no son directamente las aplicaciones que se han creado para el framework, si no que son manejadores que se encargan de conectar la aplicación web con la aplicación del framework. Es decir, si llega una petición para generar un modelo a partir de un fichero profiled, esta clase se encargará de llamar a Profiler y gestionar toda la interacción necesaria entre la página web y la aplicación acorde con las peticiones que le llegan.

La función de la clase CommonKnowledge es la de abastecer conocimientos comunes de la página web a todos los elementos del núcleo, aplicaciones y herramientas, cómo por ejemplo, la cantidad de ambientes que hay, punteros a los manejadores de las aplicaciones...

4.3.8. Etapa 8: diseño de otros aspectos

En este apartado se explicará el diseño del fichero de entrada de Profiler y de los complementos (o plug-ins) que se han hecho hasta la fecha de la entrega de este documento.

Diseño del fichero de entrada

El diseño de este fichero se ha visto sujeto al diseño de los modelos ya existentes en Tafat. Los modelos existentes en Tafat se escriben en XML y tienen un formato cómo el siguiente:

```
<simulation>
   <scene>
      <entity parameterX='valueX'>
         <behavior name='name' release='' />
      </entity>
   </scene>
   <connection>
      <powergrid>
         <powerline from='*' to='*' />
      </powergrid>
   </connection>
   <population>
      <agent linkedTo='...' />
   </population>
</simulation >
```

Como Profiler tiene que generar código de este tipo, lo lógico es que el formato que reciba Profiler sea similar, para que sustituya la cláusula profiler por lo generado. De este modo, es posible tener escrito un modelo con partes de elementos concretos y otras partes con cláusulas profiler. Por lo tanto, en medio del modelo XML se pondrán cláusulas profiler, tal y cómo se ve a continuación:

```
<simulation>
   <scene>
      <building area='100'/>
      <profiler-building/>
   </scene>
</simulation >
```

Como se observa, dentro de la definición de escena, la primera instancia es un edificio cuya área sea 100, es decir, un edificio concreto. Sin embargo, al usar la cláusula profiler, se especifica que en ese punto se quiere generar un edificio de modo automático.

Con esta cláusula profiler-building se pretende ejecutar el complemento que se encarga de la generación de edificios. Sin embargo, pueden existir muchos modos de generar un edificio, ya sea por los datos que se consideren o por como se estructuren los elementos internos del edificio. Por ello, se ha añadido la necesidad de escribir la *release*, es decir, versión, que se desea usar para ello.

```
<simulation>
   <scene>
      <building area='100'/>
      <profiler-building release='conViviendas'/>
   </scene>
</simulation >
```

En este segundo ejemplo, se estaría pidiendo que se ejecutara el complemento para la generación de edificios que los genera incluyendo viviendas dentro. Sin embargo, todavía resulta pobre la definición de esta cláusula. Como ya se ha dicho anteriormente, la idea es generar escenarios basados en datos de entrada. Por ello, debiera ser posible pasar como parámetro al complemento la ruta al fichero que se quiere usar como base de datos para la generación.

```
<simulation>
   <scene>
```

```
      <building area='100'/>
      <profiler-building release='conViviendas' baseDeDatos='prueba.xls'/>
   </scene>
</simulation >
```

En este caso, al complemento se le informa de qué base de datos se quiere utilizar para la generación. Cómo se puede observar, tal y como se añadió la ruta de la base de datos, en esta cláusula se puede poner otros atributos que hagan falta. Por ejemplo, podría indicarse el número de edificios que se quiere generar.

```
<simulation>
   <scene>
      <building area='100'/>
      <profiler-building release='conViviendas' baseDeDatos='prueba.xls' cantidad='10'/>
   </scene>
</simulation >
```

En este caso, tras el procesado, la salida a esperar será un modelo que esté preparado para ser ejecutado en el simulador de Tafat.

```
<simulation>
   <scene>
      <building area='100'/>
      <building area='100'>
         <household/>
         ...
      </building>
      ...
   </scene>
</simulation >
```

Proceso de diseño de los complementos

Los complementos han ido variando en su forma desde el principio hasta la versión final. Esto es un proceso normal en los desarrollos evolutivos. En una primera iteración, y con la necesidad de tener versiones operativas, se implementaron de un modo diferente a como finalmente han quedado.

En un primer diseño, los complementos tenían un número de atributos dinámicos que variaban en función de los atributos que se proporcionara en el nodo Profiler del fichero de entrada. Esto hacía difícil la tarea de búsqueda del complemento dentro de Profiler, ya que habitualmente había que retocar el código que se encargaba de buscar el complemento y ejecutarlo (PluginFactory).

Aparte de esta desventaja, los argumentos que debían escribirse en el fichero profiled era más rígido, ya que debían estar todos los que el complemento precisara. Inicialmente se optó por esta solución ya que daba facilidades a los programadores de complementos al no tener que deserializar los atributos del fichero de entrada.

En el último diseño, esto se cambió a una estructura más definida de acceso a los complementos, haciendo que sólo se enviaran dos parámetros: el documento XML a modificar y el que contiene la cláusula profiler. De este modo, se puede extraer todos los atributos que el usuario desee enviar al complemento a través de la propia cláusula profiler. De este modo, los complementos pueden disponer de cuantos atributos deseen, pues se pasa el nodo completo, disfrutando de una interfaz homogénea para todos. Una vez el complemento recibe el nodo, éste debe hacer uso de las funciones de tratamiento de nodos XML para obtener los valores de los atributos.

En el apartado de implementación se dará una breve descripción de cada complemento. Por razones de confidencialidad, no es posible suministrar el código fuente de todos los complementos, ya que algunos contienen información sensible.

4.3.9. Etapa 9: implementación

En este apartado se describirá el proceso de implementación de la aplicación. Cómo se ha indicado a lo largo del documento, este desarrollo se ha realizado utilizando una metodología evolutiva. Por ello, aunque en este apartado se describa todo cómo un proceso continuo de implementación, en realidad se ha realizado iteración a iteración.

Inicialmente, se describirá el apartado de implementación de Profiler, ya que fue el primero que comenzó a desarrollarse. Posteriormente, se realizará un somero repaso sobre las implementaciones de los diferentes complementos que existen en la herramienta hasta la fecha. Por último, se hablará de la implementación de la página web de Tafat.

Profiler

Tras haber descrito los requisitos, tanto de usuario como del software, y haber trazado un diseño para abordarlos, el siguiente paso es la implementación. En este caso, fue necesario integrar diversas tecnologías que se habían aprendido en la etapa de estudio de herramientas.

Desde clases para el manejo de documentos XML hasta librerías para leer ficheros Excel entraron en juego para llevar a cabo la implementación. Por ello, antes de comenzar con la parte principal, se crearon las clases que harían de interfaz interoperativas. Estas clases son XMLHandler y XLSReader. La primera hace uso de los propios métodos que ofrece Java para el acceso a documentos XML. La segunda, XLSReader, hace de interfaz entre la librería expuesta Jxl y los complementos de Profiler. Esta interfaz proporciona un conjunto de métodos de alto nivel para acceder a los datos almacenados en ficheros XLS.

La clase Console fue también añadida. Ésta permite interactuar con la consola de Java, desacoplando el resto de componentes de la aplicación de escribir directamente en consola, permitiendo en un futuro cambiar la salida o entrada estándar de datos. Esta clase ya existía en el simulador de Tafat, con lo cual, gracias a la reutilización de componentes, se pudo utilizar directamente en esta aplicación. En el apartado de implementación de la página web se observa como esta clase es modificada para cambiar la fuente y destino de salida y entrada estándar de datos.

Una vez resuelta la parte de interoperatividad, se comenzó a implementar el cuerpo central (core) de la aplicación: clase Main, ProfilerParser, PluginFactory, Plugin. En cada iteración, estas clases se han ido mejorando para otorgarles mayor funcionalidad y rendimiento a la aplicación, siendo el foco central a mejorar.

Complementos

Los complementos (plug-ins) no se desarrollaron acorde a una metodología evolutiva, sino que se iban desarrollando según necesidad de los modeladores. Muchos de éstos han sido generados con ayuda de personal de EIFER. Debido a las condiciones de confidencialidad de este instituto, sólo se puede ofrecer una explicación de alto nivel acerca de lo que realizan los complementos. Los

complementos que se recogen en este documento son los desarrollados antes de 5 de Junio de 2011. La mayoría de estos complementos se han realizado bajo el asesoramiento de Enrique Kremers, personal de EIFER.

Agent Household En un modelo en el que ya se han generado todas las viviendas, este complemento crea agentes sociológicos acordes a las estructuras de la casas.

En algunas reuniones se ha discutido sobre cómo implementar los agentes. Actualmente, no está claro cómo hacerlo. No se sabe si implementarlos como un agente único que habita la casa y se encarga de manipular los elementos que hay dentro de ella, o si, por el contrario, se debe poner cada unidad que vive en la casa; es decir, en el caso de una familia con un hijo, el agente padre, madre e hijo.

En esta versión de creación automática de agentes se usa la última concepción. Para ello, este complemento hace uso del parámetro que indica el número de habitantes que tiene la casa, y genera tantos agentes como ese número indique.

Buildings Este complemento crea un conjunto de edificios con todos los elementos internos (viviendas, dispositivos eléctricos...) de acuerdo con una base de datos que se le debe proporcionar. Esta base de datos debe contener, en cada fila, información sobre uno de los edificios. De este modo, si se tiene todos los edificios de una región, éstos podrían ser añadidos a una escena de simulación gracias a Profiler. Este hace uso de otros complementos para la generación de viviendas y electrodomésticos.

Connection Permite crear un conjunto de conexiones entre diversas entidades de acuerdo a los parámetros que se le proporcionan. Los parámetros que este complemento reciben son:

- Tipo de conexión: aquí se indicará que elemento de conexión del metamodelo se desea usar para las conexiones.
- Fuente: el tipo de entidades que se quieren conectar como fuente de la conexión. En caso de que la conexión sea de tipo bidireccional, ambos se considerarán fuente y destino.
- Padre de la fuente: en caso de que se quiera, se puede arrojar información sobre cual debe ser el padre de la fuente de la conexión. Esto permite aclarar con mayor exactitud que elementos se quieren establecer como fuentes. Por ejemplo: conexión de todos las neveras que estén en una casa (fuente: nevera, padre: casa); de este modo se evita manejar el resto de neveras que no estén contenidas en una casa.
- Multiplicidad de la fuente: cantidad de elementos que se pueden conectar a esta fuente a través de la conexión especificada en el tipo. La multiplicidad puede ser una o muchas.
- Destino: el tipo de entidades que se quieren conectar como destino de la conexión. En caso de que la conexión sea de tipo bidireccional, ambos se considerarán fuente y destino.
- Padre del destino: en caso de que se quiera, se puede arrojar información sobre cual debe ser el padre del destino de la conexión.

- Multiplicidad del destino: cantidad de elementos que se pueden conectar a este destino a través de la conexión especificada en el tipo. La multiplicidad puede ser una o muchas.
- Política: modo de interconectar las fuentes con destinos. Se ofrecen tres modos: aleatorio, balanceado o el más cercano.

Household PcDirect Este complemento genera una vivienda con todos los electrodomésticos. En este caso, esta versión añade todo éstos directamente dentro de la casa sin realizar divisiones en habitaciones. Este complemento hace uso de otros para la generación de la iluminación y radiadores acordes al tamaño de la casa.

Household Room Este complemento genera una vivienda con todos los electrodomésticos. Esta versión, tras generar la casa, genera todas las habitaciones de ésta e inserta, para cada una, los electrodomésticos acorde a la semántica de la habitación. Este complemento hace uso de otro para la generación de las habitaciones.

Household Statistical En esta versión para la creación de casas se utiliza un fichero de definición de sociedades. Este fichero indica cual es la proporción de cada grupo socioeconómico que se exprese en él con respecto al conjunto global de la sociedad. Este complemento genera tantas casas como se indiquen por parámetro e incluye los electrodomésticos que se asocian a cada grupo socioeconómico. Esta inclusión de electrodomésticos la realiza usando el complemento para la creación de electrodomésticos basados en grupos socioeconómicos. Para la realización de este complemento, los requisitos se capturaron de una persona que trabajaba en asociación con Eifer: Susana Morales.

Lighting I Este complemento genera la iluminación necesaria expresada en potencia instalada que necesita una casa o habitación. Acorde a sus dimensiones y el tipo de iluminación que se desea, este complemento crea una entidad de iluminación con la potencia instalada que precisa la casa. La realización de este complemento se hizo en colaboración con personal asociado a EIFER: Izaskun Beitia.

Lighting II En esta ocasión la creación de la iluminación se basa en muchos de los cálculos del complemento Lighting I, pero la cantidad de iluminación que se necesita se convierte en potencia instalada atendiendo a otros parámetros, como la tecnología de la iluminación a instalar o la eficiencia energética de ésta. Este complemento se hizo en colaboración con personal asociado a EIFER: Jennifer Costa.

LocationsGroup Acorde con la base de datos que se le indica por parámetro, este complemento genera un conjunto de edificios. Es posible parametrizar la penetración de tecnologías como baterías y células fotovoltaicas. Por otro lado, permite generar parcial o completamente el escenario. Un parámetro de entrada se encarga de indicar cual será la cantidad, en porcentaje, de edificios. Este complemento, gracias a sus métodos de parametrización, permite calcular el impacto de la adición de tecnología, baterías y células fotovoltaicas, en la curva de demanda.

PowerConsumer Social Este complemento recoge como parámetro de entrada el tipo de dispositivo a generar. Basado en una descripción de electrodomésticos acorde a cada grupo socioeconómico, éste es capaz de generar cada electrodomésticos con los parámetros adecuados para el grupo socioeconómico que habita la vivienda. Este complemento se asocia con el Household Statistical y permiten la generación de una escena de simulación basada en grupos socioeconómicos. Este complemento se realizo con la ayuda de personal de EIFER: Susana Morales.

Radiator Area En la misma línea que el complemento de iluminación, éste genera un conjunto de radiadores acorde a las dimensiones de la vivienda o habitación. Estos se generan siempre con una potencia instalada predeterminada cuya suma debe superar la potencia instalada calculada para el área de esa vivienda o habitación.

Room Household De acuerdo con el funcionamiento del complemento Household Room, éste genera todas las habitaciones de la casa y los electrodomésticos que deben ir en cada una de ellas. Por otro lado, hace uso de los complementos de generación de iluminación y radiadores para cada una de las habitaciones.

Página web

La realización de esta aplicación también requirió un cierto esfuerzo en el aprendizaje de tecnologías. En el desarrollo de esta página web se han visto envueltas diversas tecnologías cómo JSP y Velocity.

Debido a la inexperiencia en este tipo de desarrollos, se comenzó a realizar unas primeras páginas de prueba que fueran, poco a poco, integrando la funcionalidad acordada en el diseño. Posteriormente, se fueron implementando las interfaces y los manejadores que conectan la aplicación web con las aplicaciones del framework de Tafat.

En la mayoría de las aplicaciones del framework Tafat supuso un cambio en la entrada / salida estándar de datos: la consola. Gracias al desacople que se había realizado en todas ellas, fue sencillo añadirle ese cambio para que las escrituras en consola que se iban realizando a lo largo de la ejecución se pudieran mostrar al usuario a través de la página web. Respecto a la interfaz, se utilizó el modelo observado para enseñar el metamodelo de Tafat en HTML, tal y como se propuso en la etapa de diseño.

4.3.10. Etapa 10: diseño y ejecución de pruebas

Una vez implementada las aplicaciones, se procedió a diseñar las pruebas. Por la naturaleza de las aplicaciones, los objetivos de las pruebas irían en tres direcciones: el núcleo de Profiler, los complementos y, por último, la página web. El diseño de la prueba del núcleo debía ser riguroso para validar los requisitos planteados en las etapas de análisis. Respecto a las pruebas de los complementos, éstas siempre dependerán de los requisitos que se identificaran en la etapa de análisis del desarrollo de dicho complemento. Por último, el diseño de pruebas de la página web irá en la dirección de comprobar el correcto funcionamiento de las aplicaciones y visualización.

ID	NECESIDAD	PRIORIDAD	FUENTE	DESCRIPCIÓN
01	No negociable	Alta	Aplicación	Flexibilidad en los datos de entrada
02	No negociable	Alta	Aplicación	Soporte de acceso a algún tipo de base de datos
03	No negociable	Alta	Aplicación	Flexibilidad en la generación de modelos
04	No negociable	Alta	Aplicación	Conexión entre complementos
05	No negociable	Alta	Aplicación	Tratamiento del fichero de entrada
06	No negociable	Alta	Aplicación	Detección automática de nuevos complementos

Cuadro 4.2: Resumen de los requisitos capturados

Diseño y ejecución de pruebas del núcleo

Muchos de los requisitos recogidos en el apartado de análisis debían ser contemplados en el núcleo de la aplicación; sin embargo, algunos de ellos debían ser atendidos por los complementos que se integran en la aplicación. Los requisitos capturados en el análisis pueden verse resumidos en la figura 4.2.

De ellos, los que deben ser capturados por el núcleo son todos menos el 01 y el 03. Los requisitos con identificadores 01 y 03 deben ser recogidos por los complementos, ya que son los que deben aceptar la flexibilidad necesaria en los datos de entrada y en las formas de generación del modelo. Sin embargo, los otros cuatro requisitos deben ser recogidos por el núcleo de la aplicación. La verificación y validación de los requisitos se realiza mediante pruebas.

El primero de los requisitos, segundo en la tabla, es acerca de la adición de un soporte para la introducción de datos. El soporte por defecto para la entrada de datos que se ha puesto en el núcleo es el de ficheros Excel. A través de estos ficheros se puede aportar los datos necesarios para la creación de la escena. Para ello, se diseñó una interfaz, XLSReader, que permite hacer peticiones al fichero Excel con una lenguaje de más alto nivel. Para realizar la prueba de su funcionamiento, se creó un pequeño complemento que intentara leer ciertos campos de algunas filas de un determinado fichero de Excel. Los resultados que se obtuvieron fueron satisfactorios en cuanto a la exactitud de los datos leídos, así cómo, la facilidad para obtenerlos. Por otro lado, las pruebas de este apartado se han ido sucediendo a lo largo que se iban desarrollando complementos que usaran bases de datos en Excel.

Cabe destacar, que aunque no se creara una interfaz como XLSReader, se ha añadido la librería Jackcess para el acceso a ficheros de Access. Además, Java incluye una API que permite el acceso a bases de datos. Esta API es conocida como JDBC (Java DataBase Connnectivity).

La conexión entre complementos es posible gracias a la posibilidad de, en tiempo de ejecución, crear cláusulas profiler que permitan conectar con otros complementos. Una vez se ha creado, el núcleo de Profiler recorre nuevamente todo el fichero XML para comprobar si falta alguna cláusula profiler por procesar. La prueba diseñada para comprobar el correcto funcionamiento de las conexiones consistió en la creación de dos de los complementos ya mostrados: el de creación

de edificios (Building) y el de creación de viviendas (Household). Entonces, desde el primero se generaron cláusulas profiler para crear viviendas usando el otro complemento. Los resultados obtenidos al hacer estas pruebas también fueron satisfactorios, observando como una creación de arriba hacia abajo era posible a través de este modo recursivo de llamar a los complementos.

Por otro lado, el tratamiento del fichero principal de entrada, el modelo profiled, debía ser transparente de cara a los complementos. De este modo, cuando se detecta una cláusula profiler, ésta es enviada al complemento. Posteriormente, esta cláusula es procesada por el complemento dando lugar a un nodo XML (que puede ser compuesto) que contiene el elemento de simulación generado. Una vez se ha generado, el componente sustituye el nodo profiled por el nodo XML creado. Una vez el complemento devuelve el testigo al núcleo, el documento XML ya contiene el nodo que sustituye a la cláusula profiler. En este caso, la comprobación de este aspecto se ha ido realizando a medida que se hacía las comprobaciones de requisitos anteriores. Las prueba anteriores no habrían podido realizarse con éxito sin que este requisito se cumpliera previamente.

El último requisito que debe soportar el núcleo de Profiler es la detección automática de nuevos complementos. De este modo, no es necesario reescribir el núcleo cada vez que se añada un nuevo complemento. Por medio de Reflection [2] es posible instanciar clases no conocidas (de las que no se ha hecho un import) buscándolas por su nombre. Al saber que todos los complementos están dentro de un mismo paquete, Reflection permite obtener el complemento que atiende la cláusula profiler. Para comprobar el funcionamiento, se creó un nuevo complemento que se utilizó dentro de un fichero XML de entrada de Profiler. Sin tocar el código del núcleo central se proceso el mismo utilizando Profiler. El resultado obtenido también fue satisfactorio, ya que fue capaz de reconocer el nuevo complemento e invocarlo.

Diseño y ejecución de pruebas de los complementos

Debido a la gran amplitud del concepto complemento, no es posible identificar todos los requisitos que pudiera tener uno, ya que éste depende de las necesidades que tenga el modelador o experto del dominio. No obstante, si se han identificado dos de ellos que deben guiar el proceso de desarrollo de un complemento de Profiler. Estos requisitos son el 01 y 03 de la tabla 4.2.

El primero de ellos aconseja que el diseño de un complemento contemple la posibilidad de recibir diferentes formatos de entrada para describir una misma escena. Por ejemplo, los edificios podrían crearse bien en base a sus características como área, volumen y grado de aislamiento térmico, o bien, atendiendo a la antigüedad del edificio y, a partir de ese dato, generar el resto de parámetros. No obstante, el diseño también debe cuidar que no se realice un complemento que abarque demasiados aspectos. En estos casos, habría que considerar si separarlo en dos versiones (releases). Por lo tanto, este requisito no sólo tiene por que encontrarse en un complemento determinado, sino que sería posible encontrarlo en el conjunto de complementos que generan un elemento de simulación concreto pero de diferentes modos y que cada uno admite un formato de datos concreto.

El segundo, el requisito 03, debe ser considerado en el proceso de desarrollo del complemento. Cuando se diseña el complemento para, por ejemplo, la creación de edificios, está claro que el objetivo final es generar un edifico creado de modo automático en el modelo. No obstante, a la

[2] En la Ingeniería Informática, Reflection es el proceso por el cual una aplicación puede observar (hacer una especie de introspección) y modificar su propia estructura en tiempo de ejecución.

hora de diseñar el complemento, se deben considerar aspectos técnicos que deben discutirse con el modelador y/o experto del dominio. Dependiendo del nivel de granularidad, puede interesar mostrar el edificio con todas las viviendas y electrodomésticos que contiene. Sin embargo, en otras ocasiones sólo puede interesar el edificio en sí (en simulaciones a macroescala, por ejemplo). Por ello, la arquitectura del complemento debe contemplar mecanismos sencillos para que, por medio de atributos en la cláusula profiler, se pueda modificar el modo en el que se generan los elementos de simulación para un escenario determinado.

Aunque en este caso no se ha comentado el diseño y ejecución de pruebas concretas de los complementos, si se han dado ideas de cómo podrían diseñarse para que el complemento contemple todos los requisitos previstos al inicio de su desarrollo. El proceso de diseño y ejecución de pruebas dependerá de cada complemento y los requisitos que éste deba cumplir.

Diseño y ejecución de pruebas de la página web

El diseño y ejecución de pruebas de la página web comprendieron en su mayoría la comprobación de las conexiones con las aplicaciones, aspectos de correcta visualización y comprobación de los elementos de seguridad.

Para comenzar con la validación de las conexiones con las aplicaciones, fue necesario comprobar que las propias aplicaciones funcionaban correctamente. Posteriormente, hubo que crear la conexión con la aplicación. Para ello, la página web debía contener la interfaz mientras que la aplicación debía permitir devolver el resultado de la ejecución en un parámetro de entrada. Tras crear la conexión se generó el ejecutable de la aplicación (Jar) y se añadió cómo librería de la aplicación web. Posteriormente, se utilizó la batería de pruebas que se había diseñado previamente para cada una de las aplicaciones, pero ahora usando la interfaz web. De este modo se pudo verificar el correcto funcionamiento de las aplicaciones. Las pruebas que conciernan los aspectos visuales de la página web se han realizado para, principalmente, comprobar la compatibilidad con diversos navegadores web. Por último, se diseñó una batería de pruebas para comprobar el correcto funcionamiento del sistema de seguridad diseñado para restringir el acceso por ámbitos de simulación.

Capítulo 5

Resultados y conclusiones

En este capítulo se mostrará el resultado final de cada una de las aplicaciones creadas: Profiler y la página web. Para mostrar el funcionamiento del Profiler, se hará un pequeño caso de estudio que aclare su funcionamiento y utilidad. Posteriormente, se mostrará el resultado de la página web haciendo especial énfasis en la ejecución de Profiler en la nube.

5.1. Profiler

El Profiler (figura: 5.1) es una de las herramientas de las que se compone el framework Tafat. Este framework, basado en la ingeniería dirigida por modelos, facilita la tarea de creación de un simulador de sistemas complejos. Profiler se basa en la idea de la creación de modelos de simulación basados en datos incompletos y/o estadísticos.

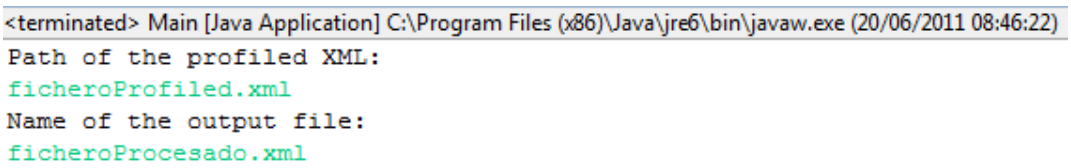

Figura 5.1: Captura de pantalla de Profiler

Esta herramienta posee las siguiente características:

- Arquitectura basada en complementos: esta característica permite al núcleo de la aplicación trasladarse de un ámbito a otro, pasando de, por ejemplo, generar modelos de simulación de redes eléctricas a generarlos para simulaciones de tráfico aéreo. Esto es gracias al desacople que se ha realizado en la aplicación entre el núcleo y los complementos. El núcleo permanece invariante mientras que los complementos delimitan el ámbito sobre el que se trabaja (figura: 5.2). De este modo, puede existir una herramienta Profiler para cada ámbito de simulación.

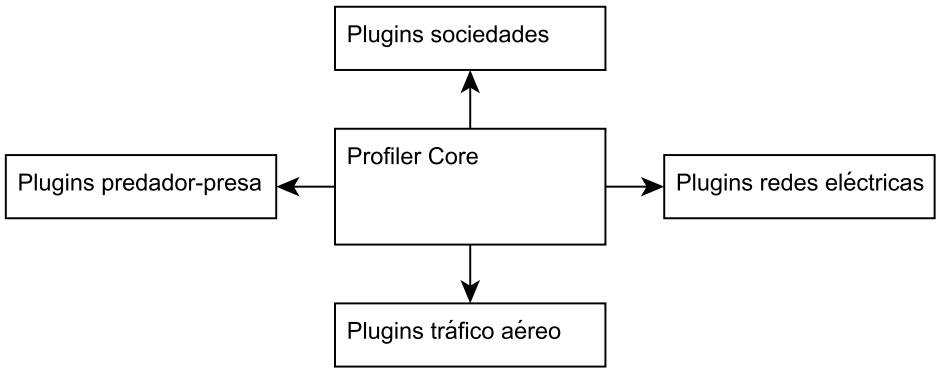

Figura 5.2: Arquitectura de Profiler basada en complementos.

- Soporte para lectura de datos de entrada. La aplicación permite la lectura de ficheros Excel gracias a la interfaz de alto nivel que se ha generado, permitiendo a futuros desarrolladores de complementos acceder a ficheros Excel para extraer datos. Por otro lado, la arquitectura contempla la adición de más formatos de ficheros de entrada de datos gracias a su arquitectura fácilmente escalable.

- Conectividad entre complementos. La aplicación esta diseñada para posibilitar la conectividad entre complementos, permitiendo generar, de modo recursivo, una escena. Esta característica permite modular la generación de un modelo, de tal modo que, cada complemento atienda la generación de un elemento de simulación concreto. Si, a su vez, este elemento de simulación tuviera que contener otros elementos de simulación (por ejemplo, el edificio que contiene viviendas), el complemento podría llamar recursivamente a los complementos que atienden la generación de esos elementos de simulación (figura: 5.3).

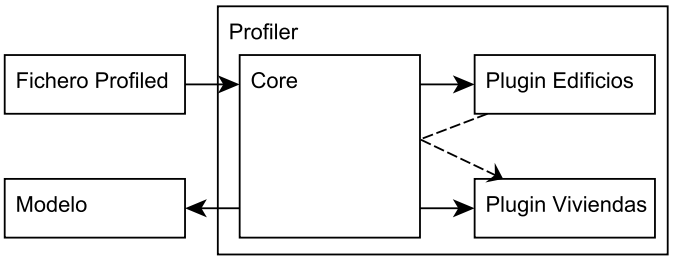

Figura 5.3: Conectividad entre complementos. El complemento de edificios hace uso del complemento de viviendas.

- Transparencia en el procesamiento del fichero de entrada. Profiler está diseñado para atender el procesamiento del modelo profiled, de modo que, para los complementos es transparentes. Cada complemento recibe la cláusula profiler que ha generado la invocación del mismo e, internamente, éste debe generar el código XML que sustituya dicha cláusula (figura: 5.4).

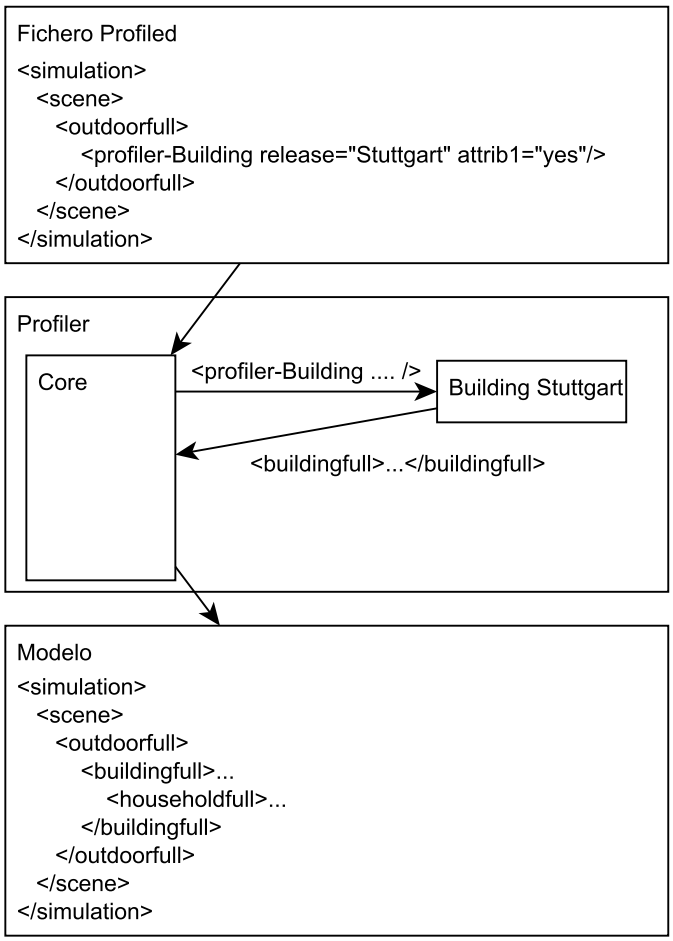

Figura 5.4: Procesamiento del fichero Profiled.

5.1.1. Caso de estudio

El siguiente caso de estudio muestra la generación de una vivienda acorde a la descripción de una sociedad clasificada en grupos socioeconómicos. Esto implica que el punto del vista de este caso de estudio es el del modelador. Para ello, se utilizarán los complementos: Household Statistical y PowerConsumer Statistical. El primero de ellos, basado en una descripción de la sociedad clasificada en grupos socioeconómicos, genera el número de viviendas que se le indica a través de la cláusula profiler. Dentro de cada vivienda, se incluyen cláusulas profiler para la creación de los electrodomésticos. Posteriormente, el complemento PowerConsumer Statistical atenderá esas cláusulas profiler para generar los electrodomésticos acorde al grupo socioeconómico de la vivienda.

Inicialmente, el modelador debe generar el modelo profiled. Para ello, es necesario que el modelador conozca los diferentes complementos que existen y cuáles son sus parámetros de entrada. El Household Statistical tiene los siguientes parámetros de entrada:

- householdCount: el número de casas que se desean generar.

- hDatFile: fichero de datos de definición de la sociedad en grupos socioeconómicos. Este fichero contiene una descripción de los grupos indicando, por cada uno, su porcentaje de participación del grupo respecto a la sociedad, el número de personas que tiene un núcleo familiar de ese grupo y los tipos de dispositivos que ese grupo tiene.
- pDatFile: fichero de datos que complementa la definición de la sociedad. Este fichero muestra características concretas que puede tener un determinado dispositivo en un grupo socioeconómico. Por ejemplo, un grupo socioeconómico de familias de características X poseen, habitualmente, televisiones de características Y.
- behavior: comportamiento social que interactuará sobre los componentes de la casa.

El modelo profiled que se va a emplear para este caso de estudio es el siguiente:

```
<simulation>
   <scene>
      <outdoorfull>
         <buildingfull>
            <profiler-Household release='Statistical' householdCount='1'
            hDatFile='socialgroups.xml' pcDatFile='socialgroups.xml'
            behavior='HouseholdFullBehavior@SocialGroups'/>
         </buildingfull>
      </outdoorfull>
   </scene>
</simulation>
```

En este caso, los ficheros de definición de grupos sociales que se van a usar ya han sido creados previamente, al igual que todos los elementos de simulación que se generarán con Profiler. Si se quisiera utilizar otros ficheros de definición de sociedades habría que observar cuales existen o, en caso de que ninguno satisfaga, generarlos. Por lo tanto, este fichero ya está listo para ser procesado por Profiler. A continuación se muestra la salida que Profiler ha escrito por pantalla:

```
Loading document...
Profiler found, type: household
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: powerconsumer
Profiler found, type: lighting
```

Lo esperable, de acuerdo con la naturaleza del fichero de entrada, era que sólo se encontrara una cláusula profiler: la de Household Statistical. Sin embargo, esto no es así. Esto se debe a que el complemento Household Statistical se encarga de conectar con otro complemento para llevar

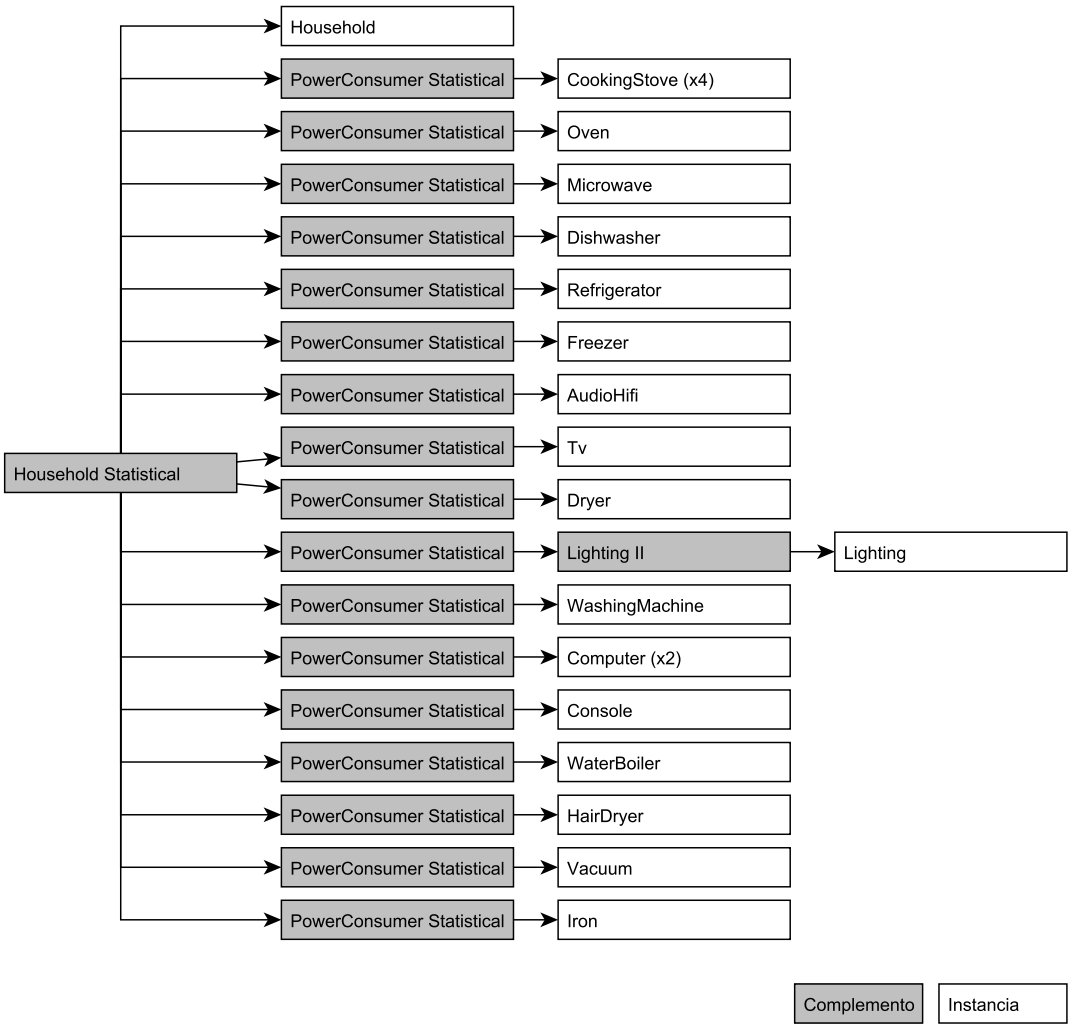

Figura 5.5: Diagrama de ejecución de Profiler. Se muestra las interconexiones entre complementos así cómo que instancia es generada por qué complemento.

a cabo su tarea. La conexión que realiza es con el complemento PowerConsumer Statistical para generar todos los electrodomésticos que debe contener la vivienda. Éste a su vez se encarga de conectar con el de Lighting para generar la iluminación de la vivienda acorde con su área. El diagrama 5.5 muestra la ramificación que se produce cuando Profiler procesa el fichero de entrada.

A continuación, se muestra el modelo XML que es generado tras la ejecución de Profiler. El modelo está preparado para ser ejecutado en el simulador de Tafat a falta de establecer las fechas de comienzo y fin de la simulación.

```
<simulation>
   <scene>
      <outdoorfull>
         <behaviorfull>
            <householdfull areaFloor='109' id='hh0' personCount='4'>
               <behavior filePath='socialgroups.xml' name='HouseholdFullBehavior'
               release='SocialGroups' socialGroup='3a'/>
```

```
<cookingstovefull>
   <behavior name='CookingStoveFullBehavior' release='Pulse'/>
</cookingstovefull>
<cookingstovefull>
   <behavior name='CookingStoveFullBehavior' release='Pulse'/>
</cookingstovefull>
<cookingstovefull>
   <behavior name='CookingStoveFullBehavior' release='Pulse'/>
</cookingstovefull>
<cookingstovefull>
   <behavior name='CookingStoveFullBehavior' release='Pulse'/>
</cookingstovefull>
<ovenfull>
   <behavior name='OvenFullBehavior' release='Pulse'/>
</ovenfull>
<microwavefull>
   <behavior name='MicrowaveFullBehavior' release='Basic'/>
</microwavefull>
<dishwasherfull>
   <behavior name='DishwasherFullBehavior' release='Complex'/>
</dishwasherfull>
<refrigeratorfull>
   <behavior name='RefrigeratorFullBehavior' release='Basic'/>
</refrigeratorfull>
<freezerfull>
   <behavior name='FreezerFullBehavior' release='Basic'/>
</freezerfull>
<audiohififull>
   <behavior name='AudioHifiFullBehavior' release='Basic'/>
</audiohififull>
<tvfull>
   <behavior name='TvFullBehavior' release='Complex'/>
</tvfull>
<dryerfull>
   <behavior name='DryerFullBehavior' release='Basic'/>
</dryerfull>
<lightingFull installedPower='3780.0'>
   <behavior name='LightingFullBehavior' release='InstalledPower'/>
</lightingFull>
<washingmachinefull>
   <behavior name='WashingMachineFullBehavior' release='Complex'/>
</washingmachinefull>
<computerfull>
   <behavior name='ComputerFullBehavior' release='Basic'/>
</computerfull>
<computerfull>
   <behavior name='ComputerFullBehavior' release='Basic'/>
</computerfull>
<electricaltoolfull installedPower='100'>
   <title>console</title>
   <behavior name='ElectricalToolFullBehavior' release='Normal'/>
</electricaltoolfull>
<electricaltoolfull installedPower='2200'>
   <title>waterBoiler</title>
   <behavior name='ElectricalToolFullBehavior' release='Pulse'/>
</electricaltoolfull>
<electricaltoolfull installedPower='1800'>
   <title>hairDryer</title>
   <behavior name='ElectricalToolFullBehavior' release='Normal'/>
</electricaltoolfull>
<electricaltoolfull installedPower='2000'>
   <title>vacuum</title>
   <behavior name='ElectricalToolFullBehavior' release='Normal'/>
</electricaltoolfull>
<electricaltoolfull installedPower='1800'>
```

```
                <title>iron</title>
                <behavior name='ElectricalToolFullBehavior' release='Normal'/>
            </electricaltoolfull>
        </householdfull>
    </buildingfull>
</outdoorfull>
</scene>
</simulation>
```

La generación del modelo se realiza con éxito. Del mismo modo que se ha generado esta vivienda, sería posible generar 100 ó 1.000 viviendas, o una ciudad entera. Gracias a Profiler es posible generar escenarios de miles de objetos con facilidad, lo cual aumenta la productividad del modelador.

5.2. Página web

El trabajo de fin de máster incluía la realización de una página web para dar a conocer el trabajo. Por otro lado, se debía realizar una interfaz para la herramienta Profiler. Uniendo ambos requisitos se decidió crear una página web que contemplara la ejecución de la herramienta en la nube. En futuras evoluciones se decidió ampliar la funcionalidad de la misma para que fuera la página principal (figura: 5.6) del framework Tafat para cualquier ámbito y aplicación relacionada. Esta aplicación web, desarrollada usando JSP y Velocity, tiene las siguientes características.

Figura 5.6: Página web de Tafat

- Arquitectura escalable: el diseño de la aplicación se ha realizado considerando la escalabilidad del sistema. Esto se ve reflejado en la arquitectura de la aplicación web. La escalabilidad se ha conseguido desacoplando del núcleo aquellos elementos susceptibles de ser escalables: número de ámbitos y de aplicaciones por ámbito. El primero se ha solventado por medio de la especificación de un XML que indica los ámbitos existentes y las aplicaciones que tiene cada

uno. Por otro lado, las aplicaciones se han separado del núcleo por medio de una interfaz, de tal modo que éstas pueden ser llamadas fácilmente desde el núcleo (figura: 5.7). En el fichero menu.xml se especifica los diferentes ambientes que hay denominados como 'framework' y dentro del mismo las diferentes opciones disponibles en ese ámbito. El contenido del fichero se muestra a continuación:

```
<frameworks>
   <framework name='Electrical Grid'>
      <option name='Metamodel' ref='http://tafat.sourceforge.net'/>
      <option name='Behaviors'/>
      <option name='Recipes'/>
      <option name='Full Simulator'/>
      <option name='Apps'>
         <option name='Profiler'/>
         <option name='Behavior Extractor'/>
         <option name='Simulator Generator'/>
      </option>
   </framework>
   <framework name='Air Traffic'>
      <option name='Metamodel'/>
      <option name='Apps'/>
         <option name='Profiler'/>
         <option name='Simulator Generator'/>
      </option>
      <option name='Full Simulator'>
   </framework>
</frameworks>
```

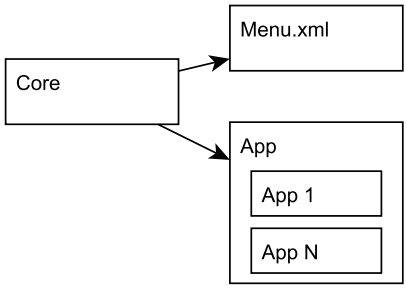

Figura 5.7: Diseño arquitectónico de la página web considerando el requisito de escalabilidad.

- Ejecución de aplicaciones en la nube: la aplicación web tiene soporte para ejecutar herramientas de Tafat. A través de interfaces, dichas herramientas se pueden conectar para ser accesibles desde la página web. Las aplicaciones que se han integrado en la página web son:

  - Profiler: es la herramienta que se ha desarrollado en este trabajo de fin de máster. Esta herramienta se encarga de la generación automática de modelos de simulación compatibles con el framework Tafat basado en datos de entrada.
  - Simulator generator: basado en un modelo de simulación esta herramienta genera un simulador hecho a medida para ese modelo. El simulador generado contiene los elementos necesarios para llevar a cabo la simulación.

- Behavior extractor: permite extraer los comportamientos contenidos en un fichero con extensión alp[1]. Esta herramienta permite reutilizar comportamientos realizados. Los comportamientos son extraídos y almacenados para ser utilizados en futuras simulaciones.

- Conexión con el metamodelo: la visualización del metamodelo se realiza mediante un enlace a una URL. Este enlace permite visualizar la página web que contiene el metamodelo de un ámbito concreto. En el menu.xml se puede especificar este enlace como se pudo ver en el ejemplo mostrado. Esta independencia del metamodelo con la página web general de Tafat, permite gestionar autónomamente los elementos del metamodelo y sus versiones.

- Visualización de elementos no contenidos en el metamodelo. Los elementos no contenidos en el metamodelo como comportamientos (figura: 5.9) o recetas (figura: 5.8) pueden ser visualizados. Éstos deben estar documentados a través de ficheros XML que la aplicación web leerá y mostrará en una lista desplegable.

Figura 5.8: Visualización de recetas. Las recetas son listados de acciones que, habitualmente, representan comportamientos sociales.

[1]Formato del fichero que usa Anylogic. Anylogic es una herramienta de modelado y simulación

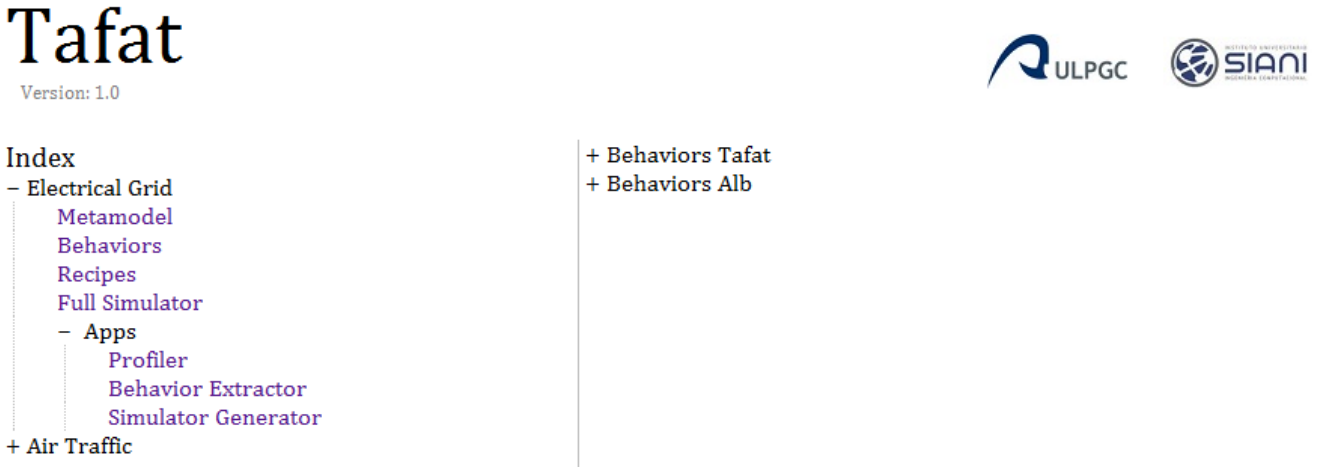

Figura 5.9: Visualización de comportamientos.

- Seguridad por ámbito: cada ámbito puede ser configurado con ciertos parámetros de seguridad y privacidad. Por medio de unas cláusulas que se pueden especificar en el fichero menu.xml, es posible establecer criterios de seguridad de acceso como: el nombre de usuario, la contraseña y el tiempo de inactividad para invalidar sesiones. El nombre de usuario y contraseña es el mismo para todos los usuarios que quieran acceder al ámbito. El formulario de acceso puede verse en la figura 5.10. Si los datos de acceso son erróneos aparecerá el mensaje de la figura 5.11. El siguiente fragmento del fichero menu.xml muestra como se establece la seguridad de acceso:

```
<framework name='Electrical Grid' secure='true'
user='usuario' pass='contraseña' idleTime='120'>
```

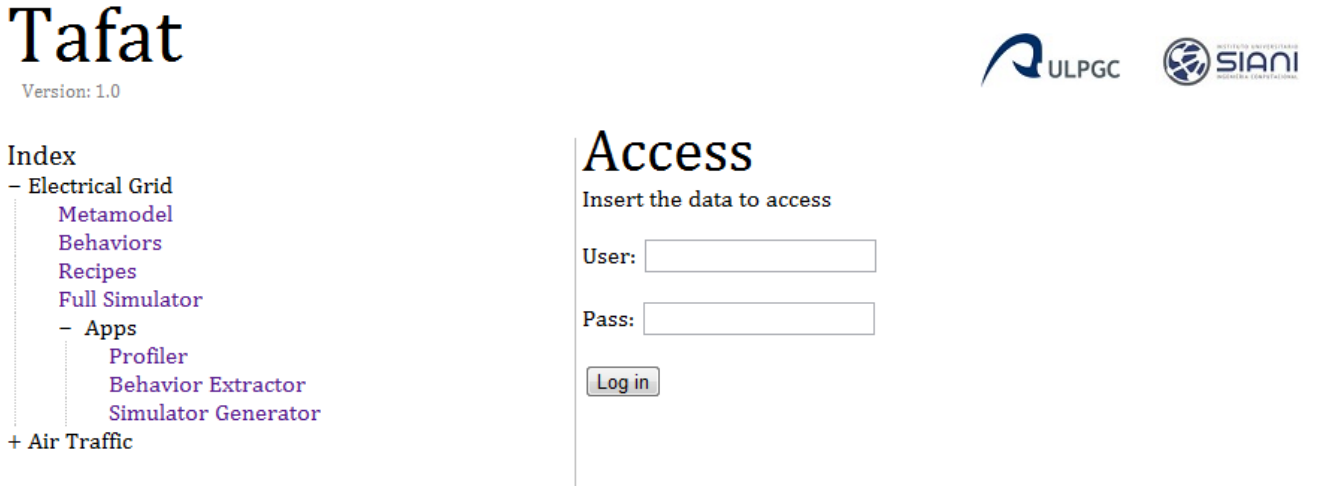

Figura 5.10: Formulario de acceso a un ámbito.

Figura 5.11: Notificación de un fallo en el acceso.

5.2.1. Interfaz de las aplicaciones

Se ha realizado una interfaz para conectar algunas de las herramientas que existen en Tafat con la página web. Estas interfaces permiten la ejecución de dichas herramientas a través de la propia página web. La ejecución de herramientas en la nube mejora la productividad de los desarrolladores de la herramienta así como de los usuarios finales. El desarrollador no se tiene que encargar de informar y/o ayudar a instalar la herramienta en cada actualización. El programador puede difundir los complementos creados con mucha facilidad y el modelador puede acceder al servicio con sólo entrar en la página web. Por otro lado, estas interfaces hacen uso de Velocity para visualizar en HTML los resultados de las interacciones con las aplicaciones.

Profiler

La interfaz del Profiler amplia la funcionalidad que la propia aplicación tiene. Además de permitir el procesamiento de ficheros Profiled, la interfaz web permite subir nuevos complementos para que sean analizados y añadidos al Profiler (figura: 5.12). Por otro lado, también se pueden observar los complementos que están actualmente añadidos a la versión ejecutable de Profiler que está en la página web (figura: 5.13). En caso de que se procese un fichero, el entorno web mostrará la salida que el Profiler ha generado así cómo un enlace de descarga que enlaza con el modelo procesado (figura: 5.14).

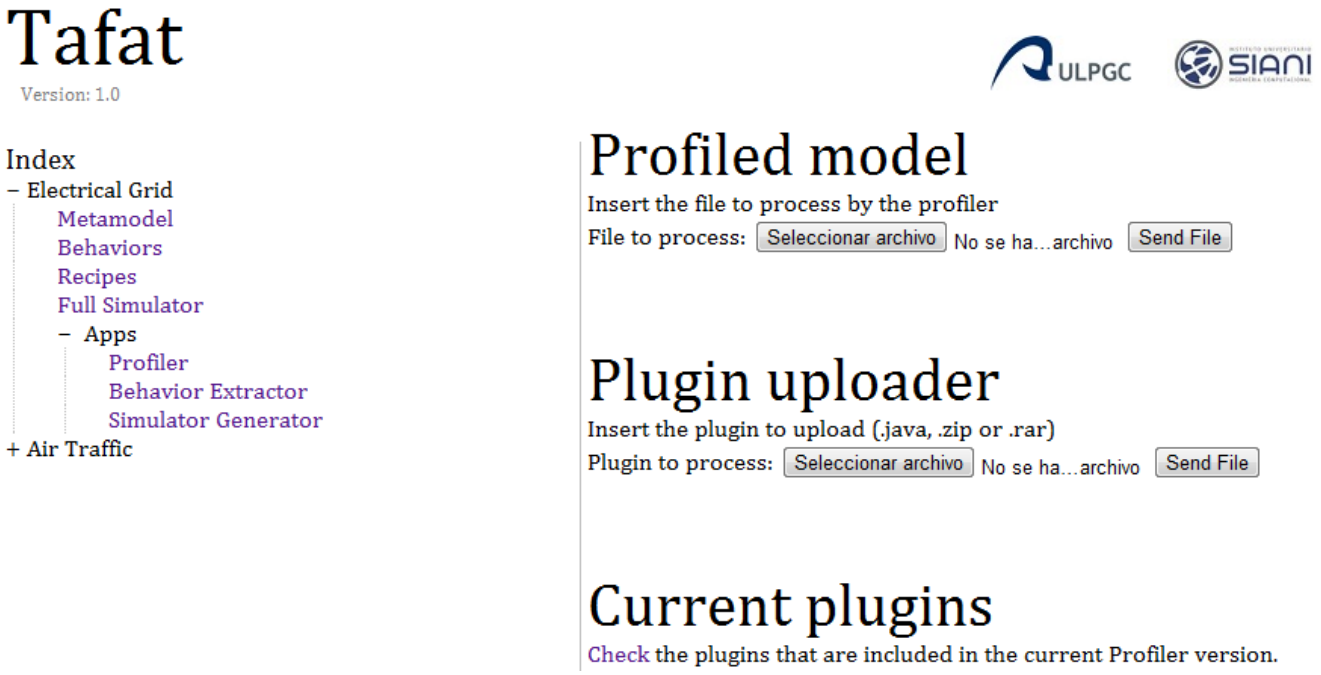

Figura 5.12: Interfaz web de Profiler. En el primer apartado es posible procesar un fichero Profiled. En el segundo es posible subir un complemento desarrollado para que sea añadido. Para visualizar los complementos que hay añadidos actualmente habría que acceder al enlace del tercer apartado.

Figura 5.13: Complementos que están disponibles en la versión actual de Profiler.

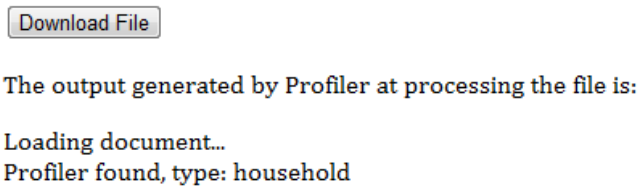

Figura 5.14: Salida generada por Profiler y mostrada en la interfaz web.

Simulator Generator

Esta herramienta se encarga de generar un Simulador acorde con el modelo de simulación que se le proporciona. Ésta analiza el modelo y extrae las dependencias que requiere para incluirlas en el simulador a generar. La interfaz web diseñada para esta aplicación se puede ver en la figura 5.15).

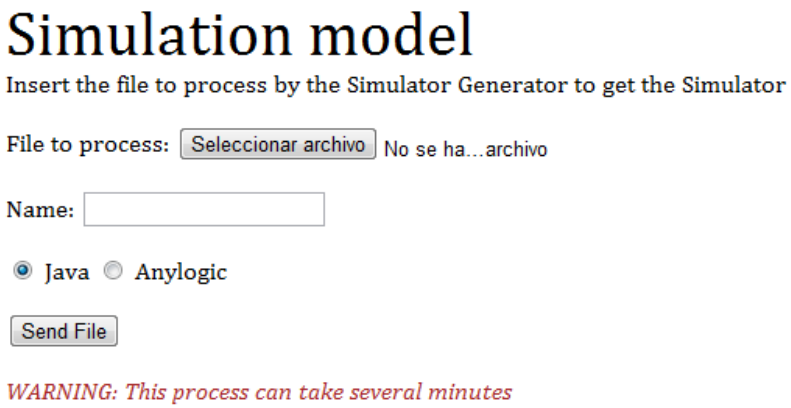

Figura 5.15: Interfaz web para interactuar con el generador de simuladores.

Behavior extractor

Partiendo de un fichero .alp (AnyLogic) se encarga de extraer los comportamientos que éste contenga para almacenarlos en el repositorio. Posteriormente, éstos podrán ser utilizados a través de empacarlos en los simuladores generados por la aplicación explicada anteriormente. Esta aplicación tiene la interfaz web que puede observarse en la figura 5.16. Tras procesar el fichero se mostrará los comportamientos extraídos (figura: 5.17) y se almacenarán para que sean revisados por un administrador.

Figura 5.16: Interfaz web para subir el fichero del que se quiere extraer los comportamientos.

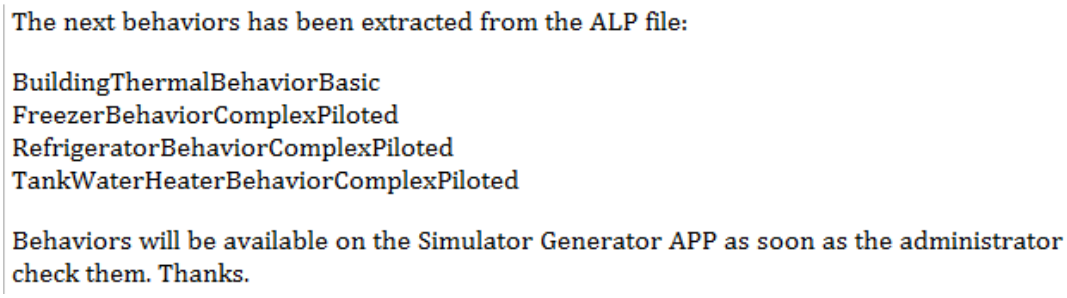

Figura 5.17: Salida generada por Behavior Extractor.

5.3. Conclusiones

El estudio y análisis de los sistemas complejos consiste, en ocasiones, en la simulación de éstos en ordenadores. Debido a la dificultad que supone el modelado del sistema y la creación del simulador el uso de herramientas que faciliten esta tarea es indispensable para ganar en productividad. Para conseguir una mayor efectividad, se debe usar sistemas que proporcionen un marco de trabajo para la realización de simulaciones de sistemas complejos. Éstos resuelven la mayoría de los problemas a los que uno se enfrenta cuando comienza a realizar simulaciones de sistemas complejos.

El framework Tafat es un conjunto de herramientas que permite el desarrollo de simuladores. Gracias a su conjunto de herramientas, Tafat facilita el desarrollo de simulaciones de sistemas complejos aumentando la productividad de los modeladores. La adición de herramientas en este framework ha sido un proceso paulatino que responde a las necesidades que tienen los usuarios finales.

Una de las necesidades es la de facilitar la tarea a la hora de describir el escenario de simulación. Esto se ha abordado por dos vías diferentes pero complementarias: uso de XSD y Profiler. La primera de ellas, ayuda por medio de guiar el proceso de escritura de la escena mientras que la segunda genera, de modo automático, un escenario acorde a unas pautas determinadas. Cuando se realiza un estudio de abajo hacia arriba en el que se utiliza un nivel alto de detalles, Profiler es de gran utilidad para generar, automáticamente, un escenario de grandes dimensiones.

Otra de las necesidades no emana directamente del usuario final, sino de la observación de los analistas del software. El trabajo directo con el usuario final es siempre deseable en el desarrollo de un software, pues permite extraer requisitos que no se han indicado. La mayoría de los usuarios finales con los que los desarrolladores han tenido contacto no proceden del área de la ingeniería

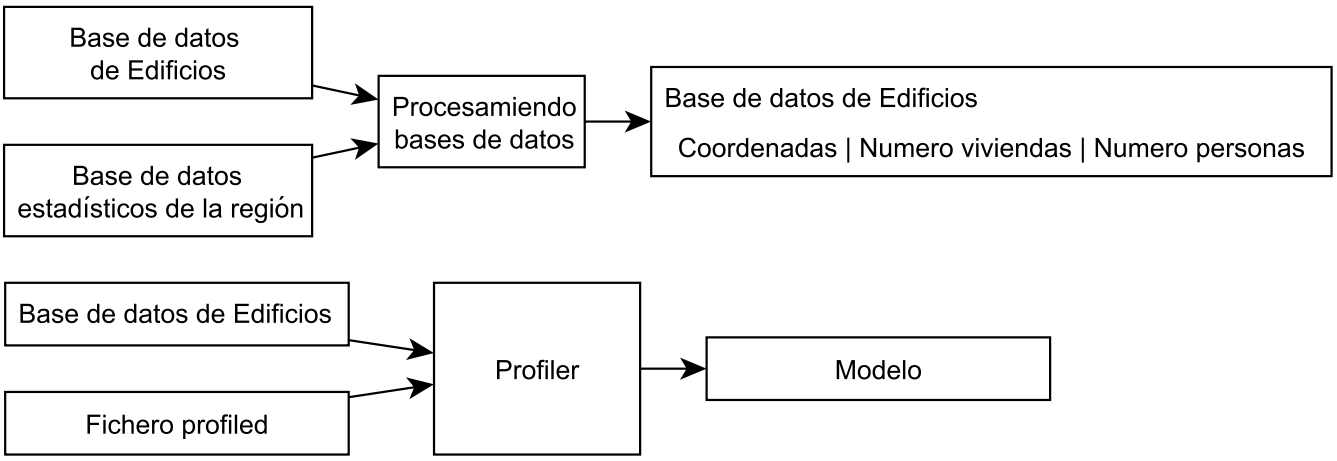

Figura 5.18: Procesamiento de datos para usar Profiler y su generación.

informática. Esto implica que la mayoría de ellos no están familiarizados con entornos no amigables. Debido a que el software debe resultar atractivo, un requisito que se extrae es el de crear interfaces con las que el usuario pueda interactuar, haciendo menos costoso el uso del framework por parte del usuario final. El uso de una interfaz web, no sólo suple esta necesidad, sino que facilita la tarea de mantenimiento del software ya que todos los usuarios tienen acceso a la versión final de cada herramienta del framework, evitando un costoso sistema de actualizaciones en ordenadores locales.

5.3.1. Profiler

El éxito que ha cosechado Profiler proviene, en buena medida, al constituirse éste como la respuesta a un requisito importante formulado por los usuarios finales del framework Tafat. Es habitual que no se disponga de información lo suficientemente completa y detallada para describir un escenario propio del sistema complejo. Lo normal es tener un conjunto de datos que describen una escena basado en sus estadísticas. Por ello, cuando la escena a recrear es grande, se debe recurrir a Profiler.

El contacto día a día con los modeladores de EIFER ha resaltado la necesidad de una herramienta que permita la creación automática de un modelo basado en los datos de entrada proporcionados. Materializando esta necesidad en un caso real se recuerda el caso de un modelador que debía generar un escenario de una región concreta. Disponiendo de una base de datos de edificios, en forma de capa GIS con su geolocalización, necesitaba tener un escenario que contuviera los edificios con viviendas y electrodomésticos. De acuerdo con la información que había, se podía hacer una estimación acerca de cuantas viviendas podría contener cada edificio; pero no los electrodomésticos de cada vivienda. De este modo, se generó una base de datos de edificios con el número de viviendas y personas por edificio. Esta base de datos fue, junto al modelo profiled, el conjunto de datos de entrada para la generación del modelo. Para ello, se creó un complemento que, acorde con la base de datos, generara los edificios y, aprovechando la conectividad entre complementos, llamara a otros complementos para generar las viviendas y electrodomésticos (figura: 5.18).

De este modo, esta herramienta ha sido, en su corto tiempo de vida, de gran utilidad para

varios modeladores de EIFER, pues permite generar escenarios haciendo uso de datos procesados previamente como, por ejemplo, datos que provienen de sistemas de información geográficos.

El desacoplamiento en complementos o extensiones permite a la aplicación trasladarse de un ámbito a otro; pues son los complementos los que vinculan un ámbito a Profiler. Este desacoplamiento, por tanto, ha supuesto una escalabilidad en cuanto al número de ámbitos en el que se puede emplear, debido a su núcleo independiente, y el número de funciones que puede realizar para la generación de modelos, gracias la fácil adición de complementos.

Debido a la arquitectura modular basada en complementos, el desarrollo de los modelos también es modular, pues cada complemento (modulo) de Profiler se encarga de generar un tipo concreto de elementos de simulación. De este modo, se proporciona a los usuarios finales de la aplicación un marco de trabajo sobre el cual apoyarse a la hora de diseñar el modo de generar el modelo deseado. El usuario final, insertado ya en la dinámica de los complementos existentes, realizará un diseño modular basado en conexiones en el cual cada complemento se encargará de una parte concreta de la generación del modelo.

En la misma línea en la que el framework Tafat ha sido desarrollado, el Profiler se postula como una herramienta que afianza y mejora un importante criterio con el cual el framework fue desarrollado: productividad. El framework Tafat, en un determinado ambiente, se desarrolla progresivamente a través de la adición de elementos de simulación por parte de los programadores y modeladores. Estos últimos aprovechan estos elementos de simulación ya creados para describir escenas, pudiéndose preocupar sólo en lo concerniente a la tarea del modelado. En esta tarea es donde Profiler puede ser de gran utilidad para generar modelos de modo automático.

Profiler ha sido reconocido como una herramienta muy útil por jefes de proyectos, modeladores y alumnos de proyectos de fin de carrera. Junto con el framework Tafat, su grado de aceptación es alto y su presencia es cada vez mayor dentro del ámbito de redes eléctricas en EIFER. El uso del framework Tafat y sus herramientas comienza a ser estudiado en otros ámbitos, como por ejemplo, tráfico aéreo, predicción, turismo... Por otro lado, el framework Tafat empieza a hacerse eco en la comunidad científica a través del primer artículo que lo menciona (anexo: A).

5.3.2. Página web

La página web realizada en este trabajo de fin de máster se postula como la web principal del proyecto Tafat. En ella, se permite la visualización de los elementos de cada ámbito y la ejecución de herramientas del framework en la nube.

Este framework es usado y conocido fuera del instituto por lo que es de vital importancia que tenga presencia en internet, al menos, informativa. La adición de funcionalidades visuales y de procesamiento de datos incrementa el valor que ésta tiene.

La adición de técnicas de computación en la nube permiten un fácil acceso, propagación y mantenimiento de las aplicaciones. Éstas pueden ser accedidas desde cualquier punto de acceso que disponga de internet y navegador web. La propagación de nuevas versiones es mucho más fácil al no tener que instalarla localmente en cada ordenador de cada modelador que use la herramienta. Las actualizaciones de las herramientas pueden ir en dos líneas: mejoras del núcleo o adición de elementos pertenecientes al ámbito en el que se trabaja. El mantenimiento de la aplicación es más sencillo al poder disponer de la retroalimentación de información que los usuarios generan al hacer

uso de las aplicaciones, descubriendo fallos con mayor facilidad y rapidez.

La primera versión de la página web permite al usuario realizar diversas funciones: ver el metamodelo, ver comportamientos, ver recetas, ejecutar Profiler, subir un complemento de Profiler, ver los complementos de Profiler, extraer comportamientos de un fichero alp, generar el simulador tanto en alp como en java a partir de un modelo y descargar el simulador.

5.3.3. Tecnologías empleadas

La realización de este trabajo de fin de máster ha supuesto el estudio, aprendizaje y uso de un conjunto de tecnologías. En el desarrollo se han empleado las siguientes tencologías: Java, JSP, Velocity, HTML, CSS, Javascript, AJAX, además de, diversas librerías.

El uso de Java para el desarrollo de las aplicaciones se debe a que el framework Tafat se está desarrollando en este mismo lenguaje. Las aplicaciones resultantes de este trabajo se unen a este framework agrandando así el conjunto de herramientas que este posee. El uso de la tecnología Java en el desarrollo de aplicaciones permite, en primer lugar, la independencia al sistema operativo en el que se ejecute la aplicación. En segundo lugar, al ser uno de los lenguajes de programación que más se utiliza, es fácil encontrar librerías que ayuden a la hora de desarrollar aplicaciones.

En el desarrollo de Profiler ha sido necesario hacer uso de librerías. En general, éstas amplian las capacidades de Profiler para el acceso a bases de datos. La librería con la que más se ha trabajado es Jxl. Haciendo uso de ella se ha generado una interfaz que permite el acceso a ficheros Excel en un lenguaje de alto nivel (accediendo por nombre de columna y número de fila). El acceso a ficheros Excel de nueva generación (.xlsx) se ha resuelto añadiendo la librería POI de Apache. Futuras mejoras podrían adaptar la interfaz de alto nivel, que se apoya en Jxl, para que utilice POI. Del mismo modo se podría facilitar el acceso a ficheros de Microsoft Access haciendo que la interfaz se apoye en Jackcess, librería agregada al proyecto. Debido al componente estadístico que los complementos de Profiler suelen llevar, ha sido necesaria la adición de una librería matemática: Apache Math.

El uso de estas librerías ha implicado el estudio y aprendizaje de las mismas, tanto para el desarrollo de la interfaz mencionada anteriormente así como para el desarrollo de complementos que hagan uso de éstas. La búsqueda y estudio de estas librerías es, por así decirlo, reutilizable, pues estas librerías se pueden integrar en las simulaciones de Tafat para otorgarles funcionalidades que faciliten el acceso a bases de datos o realizar cálculos matemáticos.

El uso de Java como tecnología para el desarrollo de las herramientas del framework ha sido determinante en la elección de la tecnología utilizada para el desarrollo de la página web. De este modo, Apache Tomcat se postuló como la tecnología adecuada para el desarrollo de la página web ya que permite realizar una aplicación web en lenguaje Java. Esto facilitó las conexiones entre la página web y las herramientas del framework, permitiendo así la ejecución de éstas en la nube. El aspecto visual se ha resuelto por medio del uso de tecnologías como HTML, CSS, JavaScript, AJAX y Velocity. El uso de Velocity ha permitido desacoplar aspectos de visualización de la página web del procesamiento interno de la misma. De este modo, los elementos que procesan las peticiones sólo se tienen que encargar de cargar la plantilla y de rellenarla con los contenidos que sean necesarios. El uso de Apache Tomcat, Velocity, CSS, JavaScript y AJAX ha requerido una etapa de estudio y aprendizaje, pues eran tecnologías desconocidas para el autor.

5.3.4. Internacional

Este trabajo de fin de máster se ha desarrollado en un ambiente internacional en el que diversas personas de diferentes nacionalidades se han visto involucradas. Por ello, un aspecto a cuidar ha sido el de mantener una interfaz, codificación y documentación en Inglés. La internacionalización de un proyecto software es enriquecedor al permitir el acceso de personas de todo el mundo al mismo. El primer paso para conseguirlo es la generación de las aplicaciones y su documentación en un lenguaje que sea hablado en muchos sitios, como lo es el Inglés. En este caso concreto, además de por esta razón, el tener contacto con personas extranjeras ha convertido ese objetivo en una obligación.

5.3.5. Artículos de investigación

La creación de este conjunto de herramientas en el ámbito del framework Tafat es, sin duda, algo innovador. El punto de vista que proporciona Tafat para llevar a cabo la simulación de sistemas complejos es novedoso ya que no se había visto antes la aplicación de técnicas de la ingeniería dirigida por modelos en este campo.

Este framework, que usa una concepción diferente para llevar a cabo simulaciones de sistemas complejos, puede ser de alto interés para la comunidad científica. Por ello, se ha comenzado a redactar diversos artículos de investigación en los que Tafat se comienza a dar a conocer. Algunos de ellos están todavía escribiéndose mientras que otros ya han sido aceptados en congresos (anexo: A).

Capítulo 6

Trabajo futuro

El trabajo desarrollado es susceptible de tener continuidad en el futuro. No es un trabajo cerrado: con principio y fin. Debido a la vinculación de este trabajo con el proyecto Tafat los trabajos futuros están ligados con el propio futuro del proyecto.

Estos trabajos pueden provenir de tres fuentes: adición de requisitos a las herramientas creadas en este trabajo, colateralidad en la adición de requisitos en el proyecto Tafat y la adición de mejoras.

6.1. Profiler

Esta herramienta tiene un nivel de madurez bastante alto, con lo que no hay especial margen para la adición de mejoras. El trabajo futuro en este caso puede ir por la adición de requisitos por parte de los usuarios finales (modeladores) que hagan uso de él o a través de efectos colaterales en la adición de requisitos en otras partes del proyecto Tafat.

Pese a que el margen de mejoras sea pequeño, si hay aspectos de Profiler que pueden ser ampliados. Actualmente, sólo existe una interfaz que permite la lectura de ficheros de Excel (.xls). En un futuro debiera añadirse soporte para otros tipos de bases de datos, adaptando la interfaz creada, que es de alto nivel, para que funcione con el resto de bases de datos.

Aumentar las funcionalidades del conjunto de herramientas (tools) que tiene Profiler internamente podría ser de gran utilidad. Por un lado, se podría realizar mejoras que permitan añadir más abstracción entre lo que se quiere generar y el tratamiento de los nodos XML. Por otro lado, se podría facilitar la escritura de bases de datos por si se deseara, desde un complemento, generar una salida (por ejemplo, todas las localizaciones geográficas de los elementos generados para poderlos representar en un mapa).

La realización de complementos para la generación de código XML que sea compatible con el framework Tafat se hace artesanalmente. Esto es, que los dispositivos y las características que se expresan en los modelos generados por Profiler no están sincronizados con el metamodelo o el repositorio. Es el propio programador el que debe hacerse cargo de comprobar que lo que escribe en el XML es correcto. Una mejora podría tratar de suplir esta carencia que tiene la aplicación actualmente, permitiendo a los programadores referenciar elementos del metamodelo o

del repositorio con la seguridad de que éstos y sus atributos existen.

A medida que el uso de Profiler es mayor es lógico que aparezcan nuevos requisitos que actualmente no contempla. La infinidad de usos que puede tener esta herramienta convierte el desarrollo de la misma en un proceso en el que no se puede establecer un final en el cual, a partir de él, se puede decir que la herramienta (el núcleo) no necesite ninguna modificación. Por otro lado, requisitos que modifiquen el proyecto Tafat pueden afectar al núcleo de la misma en cuanto al tratamiento de los ficheros de entrada y el modo de generar los modelos.

6.2. Página web

La página web, como ya se indicó en conclusiones, desde el punto de vista de funcionalidad está bastante avanzado. Sin embargo, se puede trabajar más en el modo en el que se le presentan a los usuarios las opciones de la página web.

Actualmente, el usuario debe conocer las herramientas del framework Tafat para gestionar sus modelos. La interfaz puede ser mejorada para que esta gestión sea más transparente y de alto nivel. Para ello, se propone la adición de un sistema de usuarios que almacene los modelos que cada usuario ha ido creando, subiendo o procesando. En caso de subir un modelo profiled, la aplicación web lo procesaría y situaría en el entorno de trabajo del usuario el modelo ya procesado. De este mismo modo, la aplicación de extracción de comportamientos puede unirse con la de subida de comportamientos, lo que para un caso se envía un fichero .alp y en el otro una clase .java. De este modo, el usuario no tiene porque conocer que existe una aplicación Behavior Extractor que sirve para procesar ficheros .alp.

Por otro lado, una mejora funcional muy importante podría venir en la ejecución de simulaciones en la nube. Teniendo los modelos en el perfil de la página web, podría permitirse realizar simulaciones en el servidor con ese modelo y depositar los ficheros de resultados en el perfil del usuario tras finalizar la simulación. Esta mejora funcional, empleada en un servidor con potencia suficiente, podría evitar la necesidad de ejecutar Simulator Generator, pues siempre se ejecutaría en el servidor con el simulador completo. No obstante, esta mejora es más dependiente del hardware del que se disponga que del software que haya que programar para permitirlo.

Bibliografía

[1] Braithwait, S.: Behavior modification. Power and Energy Magazine, IEEE 8, 36–45 (2010), internal-pdf://Behavior Modification-1869937665/Behavior Modification.pdf, 3

[2] Butler, M., Petre, L., Sere, K., Kent, S.: Model driven engineering. In: Integrated Formal Methods, Lecture Notes in Computer Science, vol. 2335, pp. 286–298. Springer Berlin / Heidelberg (2002)

[3] Capasso, A., Grattieri, W., Lamedica, R., Prudenzi, A.: A bottom-up approach to residential load modeling. Power Systems, IEEE Transactions on 9(2), 957–964 (1994)

[4] Carson, J.: Introduction to modeling and simulation. In: Simulation Conference, 2003. Proceedings of the 2003 Winter. vol. 1, pp. 7–13 Vol.1 (2003)

[5] Gaudou, B., Ho, T.V., Marilleau, N.: Introduce collaboration in methodologies of modeling and simulation of complex systems. In: Intelligent Networking and Collaborative Systems, 2009. INCOS '09. International Conference on. pp. 1–8 (2009)

[6] Iba, T., al, e.: From Conceptual Models to Simulation Models: Model Driven Development of Agent-Based Simulations

[7] Ingalls, R.: Introduction to simulation. In: Simulation Conference, 2008. WSC 2008. Winter. pp. 17–26 (2008)

[8] Khlif, M., Shawky, M.: Co-modelling and simulation with multilevel of granularity for real time electronic systems supervision. In: Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on. pp. 348–353 (2008)

[9] Law, A., McComas, M.: How to build valid and credible simulation models. In: Simulation Conference, 2001. Proceedings of the Winter. vol. 1, pp. 22–29 vol.1 (2001)

[10] Macal, C., North, M.: Agent-based modeling and simulation. In: Winter Simulation Conference (WSC), Proceedings of the 2009. pp. 86–98 (2009)

[11] Polack, F.A.C., Andrews, P.S., Ghetiu, T., Read, M., Stepney, S., Timmis, J., Sampson, A.T.: Reflections on the simulation of complex systems for science. pp. 276–285, crossref =

[12] Polack, F.A.C., Hoverd, T., Sampson, A.T., Stepney, S., Timmis, J.: Complex systems models: Engineering simulations. pp. 482–489

[13] Poole, J.D.: Model-Driven architecture: Vision, standards and emerging technologies. In: In In ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models (2001)

[14] Saffre, F., Gedge, R.: Demand-Side management for the smart grid. In: Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP. pp. 300–303 (2010)

[15] Schmidt, D.C.: Guest editor's introduction: Model-Driven engineering. Computer 39, 25–31 (Feb 2006)

[16] Vinerbi, L., Bondavalli, A., Lollini, P.: Emergence: A new source of failures in complex systems. In: Dependability (DEPEND), 2010 Third International Conference on. pp. 133–138 (2010)

[17] Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons (Jun 2002)

Enlaces de las librerías usadas

Las páginas web de las librerías contienen manuales de uso de éstas que han sido necesarios para la realización de este trabajo. A continuación, se muestra las páginas web de las librerías utilizadas:

- *Librería Jxl*: http://jexcelapi.sourceforge.net/ (visto por última vez: 30/06/2011)
- *Librería Apache POI*: https://poi.apache.org/ (visto por última vez: 30/06/2011)
- *Librería Apache Tomcat*: https://tomcat.apache.org/ (visto por última vez: 30/06/2011)
- *Librería Apache Velocity*: https://velocity.apache.org/ (visto por última vez: 30/06/2011)
- *Librería Apache Math*: https://commons.apache.org/math/ (visto por última vez: 30/06/2011)
- *Librería Jackcess*: http://jackcess.sourceforge.net/ (visto por última vez: 30/06/2011)

Apéndice A

Artículo de investigación

En este apartado del anexo se incluye uno de los artículos de investigación que ya ha sido escrito y aceptado. Este artículo será presentado en un congreso en Paris. Los datos del artículo son los siguientes:

- Título: Agent-Based Modelling of Electrical Load at Household Level
- Autores: Jose Evora, Enrique Kremers, Susana Morales, Mario Hernandez, Jose Juan Hernandez and Pablo Viejo
- Lugar: CoSMoS Workshop 2011, European Conference on Artificial Life (ECAL11) in Paris, France
- Fecha: 8th August 2011.

Se está realizando, por otro lado, la escritura de un artículo que será, probablemente, presentado en un congreso que tendrá lugar en Viena. El abstract del mismo ya ha sido aceptado en dicho congreso. Los datos del artículo, a día de hoy, son:

- Título: Towards an interdisciplinary approach for the simulation of future smart grid architectures from a complex systems science point of view.
- Autores: Pablo Viejo, Enrique Kremers, Mario Hernández Tejera, Jose Juan Hernandez, José Évora, Eric Daudé, Jose Mª Gonzalez de Durana.
- Lugar: Vienna
- Fecha: September 12-16, 2011 (Sept 13, 2011): Main Conference (and Satellite).

A continuación, se añade como anexo el artículo de investigación presentado en el congreso que tendrá lugar en Paris. En este artículo se hace referencia al framework Tafat así como a sus herramientas: entre ellas Profiler.

Agent-Based Modelling of Electrical Load at Household Level

Jose Evora[1], Enrique Kremers[2], Susana Morales Cueva[2], Mario Hernandez[1], Jose Juan Hernandez[1], and Pablo Viejo[2]

[1] SIANI, University of Las Palmas de Gran Canaria, Las Palmas, Spain, jose.evora@siani.es
[2] EIFER, European Institute for Energy Research (KIT & EDF), Karlsruhe, Germany enrique.kremers@eifer.org

Abstract. Regarding electrical systems as complex systems offers new approaches for analysing, modelling and simulating those systems. Using software engineering techniques like Model Driven Engineering, a disaggregated model for household electricity demand is created. The Tafat framework for simulating complex energy systems is presented, including the concepts of the metamodel, models and behaviours. A first case study simulating the load curve of 1000 households composed of five different social groups is discussed and compared with an aggregated curve. The model is able to represent the load curve of a sample of households using a bottom-up approach.

Keywords: agent based model, electrical system, domestic load curve simulation, residential demand, demand side management, appliances, model driven engineering, simulation framework, demand size management, complex system

1 Introduction

During recent years, an increasing interest in knowing more about electrical demand at low levels of the power grid can be observed. This is mainly due to the process of change the electrical system are undergoing, motivated by causes such as the introduction of renewable energy sources (RES) as well as distributed generation (DG). These changes require a better knowledge of the load curves at a distributed level, which involves different factors such as electrical equipment, user behaviour, environmental conditions, etc. that have a potential impact on the consumption. Aggregated data at substation level is no longer accurate enough to describe the consumption processes at the level of the distribution grid.

Modelling the consumption locally can help us to better understand the composition of aggregated load curves (also represented by load profiles) and analyse how local measures can have an effect on the global curve. Current aggregated models do not provide the possibility to model local measures on the grid, as for example punctual actions taken by individual consumers, like switching on

a cooking stove or an oven. They can only be included by modelling their aggregated effect and integrating it at aggregated level, due to their resolution. When considering the proposed changes in the electricity system, for planning and control activities at distribution level, these curves are not suitable, as they only fit for a large number of consumers because they describe the average use of energy over time [12, 4].

Some approaches already exist found which considered these facts and implement a demand modelling at lower scale. In [10] a simplified demand model for domestic lightning is presented that provides estimating the aggregated demand or even the distributed loading for groups of households. In [4] a multi-use bottom-up domestic consumption modelling tool was developed and verified. [14] gathered and analysed high resolution domestic electrical data and found that logging at intervals of few minutes is necessary to capture the fine detail of load patterns for evaluating on-site generation. In [11], a high resolution stochastic approach, using heterogeneous Markov chains is chosen to model domestic electricity demand.

2 Electrical systems

The electric power system is composed of different electrical components that allow for the production, transmission and consumption of electric power. Production or generation of electric power is the process of converting energy in other forms (chemical, mechanical, nuclear, etc.) into electrical energy. For the transmission, different kinds of electrical networks are used. Generally, they are classified by their nominal voltage at each level. Long range transmissions over hundreds of kilometres are performed at high voltages of several hundreds of kilovolts (kV). These networks are called transmission systems. Transformer stations (substations), can reduce the voltage at a given point in order to feed electricity to the so called distribution system. The distribution system carries the electricity to the final consumers. These networks operate at medium voltage levels, usually between 1-50 kV. In a final stage, distribution transformers can convert from medium voltage into low voltage (less than 1 kV) which is the typical voltage level find at residential or tertiary customers. Some specific customers (such as industries) may have direct connections at medium voltage level, too.

In a *classical* energy system, generation is injected at high or medium voltage level and consumed in the distribution system. Following the trend of introduction of renewable energy sources and distributed generation, injections of energy at almost all levels of the system are possible. Also, and in order to better match the fluctuating production introduced to the system, Demand Side Management (DSM) mechanisms, which allow to manage the consumption are being developed. The electricity system can be seen as a complex system, being composed of a large number of interacting entities. The reproduction of the behaviour of the system is therefore not possible by modelling only individual objects or the system in a monolithic way. In this paper, we use a disaggregated method to model a part of the electricity system. A detailed model of low-voltage demand

is constructed for simulating the load curve of individual households. This includes the loads of each household, representing the final end consumers of this electrical system. Simulating a large number of entities, the individual consumptions can be aggregated and a representation of a load curve at e.g. substation level is created.

2.1 Complexity and intelligence in electrical systems

Classically, the electrical system management has been done based on the aggregated consumption data at a global level. The nature of this management is has been centralised since the system was considered as an indivisible unit. When new devices that apply demand side management strategies are introduced in real electrical systems, new management conceptions must be considered. So, the management of the future electrical system must overcome the restrictions that are introduced now and start analysing the electrical system as a distributed system.

This conception of the electrical system shows analogies with other living complex systems, such as ants or bees colonies, in which there are several agents that are taking decisions and acting locally. Actions that each agent are executing locally are aggregated, and these actions can lead to emergent phenomena.

From our perspective, in the electrical systems, people living in the household are agents [13]. They can be considered as intelligent [3] since they are self interested units and exhibit an adaptive behaviour to their environment. These agents are able to take decisions and coordinate their actions with other agents. The main difference with the study of living complex systems is that in the electrical system, we are not interested in the study of the emergent behaviour starting from the local actions, but the modification of local behaviours to get the desired emergent behaviour [9]. However, in a second point of view, a complex system simulation model can serve to observe unwanted emergent phenomena and study these effects on the system. An approach to analyse complex systems from this point of view, at the very bottom, can be seen in [6].

2.2 Analysing electrical systems

From an analytical point of view, in electrical systems, we can classify objects according to their behaviour in three categories: first entities that can be described in term of their physics such as a building that exhibits a thermodynamical behaviour, second entities that can be described with a mechanistic model such as a washing machine and third agents that can be described with a intentional model such as people living in a household or smart meters.

This behaviour separation match perfectly with the differentiation proposed in [2]. In this work, the author proposes the following three behaviour categories from a observer point of view:

- **Physics Stance:** at the level of physics and chemistry. It is concerned with things such as mass, energy, temperature, velocity, and chemical composition.

- **Design Stance:** at the level of biology and engineering. It is concerned with things such as the function of a living system or the design of a system.
- **Intentional Stance:** at the level of software and minds. It is concerned with things such as belief, thinking and intention.

3 Simulation of electrical systems

Like other complex systems, electrical system simulations involve the behaviour execution of the many elements that exists in a real grid.

The process starts by modelling the electrical system using Tafat. Tafat is a framework for building simulation through Model Driven Engineering which is currently under development by SIANI and EIFER. It is currently implemented in JAVA, but designed to be language-independent. Tafat uses an object oriented and agent-based approach. In this framework, each element of the system is represented including a behaviour that explains how it changes over time. Then, a simulation is run by executing all the behaviours concurrently. During the simulation, behaviours can modify the attributes of these entities. One entity can have different kinds of behaviours, that are stored in a repository. The separation of structural and behavioural elements (entities, defined in the metamodel; and behaviours, stored in a repository) should be underlined here.

However, since the electrical system is locally and massively affected by the human interventions, it is also required to model this behaviour along with other entities that belong to the electrical system architecture. Representing the human behaviour means that it is required to define what people do during a time period, which electrical devices they have and how they use them.

Until now, the human behaviour has not been analysed and electrical companies only have historic aggregated data of the consumption at global level. From now, it is quite important to have data at the household level to be able to model households and simulate the behaviour of people inside. So, the main challenge in modelling electrical systems is to describe all the electrical appliances in the households and the behaviour profiles whose actions generate electrical consumption.

3.1 Software engineering approach

It is needed to approach the software engineering of the simulator with a methodology that supports the modelling of a large number of entities. The goal is to make the simulator development and maintenance easier.

The approach that has been taken is based on a software engineering paradigm called Model Driven Engineering (MDE) in which systems are developed at an abstraction level close to the problem domain. In this approach, models are the artifacts that drive the development process. From an evolutionary point of view, development methods have been trying to get more abstractions in order to reduce the gap between the programming semantic and domain semantic.

MDE is the result of the combination of technologies such as Domain Specific modelling Languages (DSML) and engines that analyses models for synthetising software artifacts. This paradigm allows to improve productivity and flexibility at developing a simulator, since the application is developed and modified just by changing the model [8].

3.2 Metamodel and models

To model the electrical system, a metamodel for this domain has been developed. The metamodel is a formal representation that supports the description of an electrical system with a semantic that is closer to the domain expert. The metamodel can be understood as a representation language for modelling electrical systems.

This metamodel comprises the classes of entities that have been identified in the real world. So, the metamodel defines what kind of elements can be represented in the model and provides a standardised way of representing elements. Classes are represented in a model by means of context, features, variables and behaviour (Figure 1). Context defines where elements can be placed in the scene. Features are defined as static data of an element; while variables are defined as dynamic data. The behaviour is the logic that updates the dynamic data.

In the metamodel, we distinguish different types of model elements: entities, static objects that belong to the scenario (e.g. refrigerator); agents, intentional objects that interact with entities and communicate with other agents (e.g. person); connections that define relations between entities or agents (Figure 2).

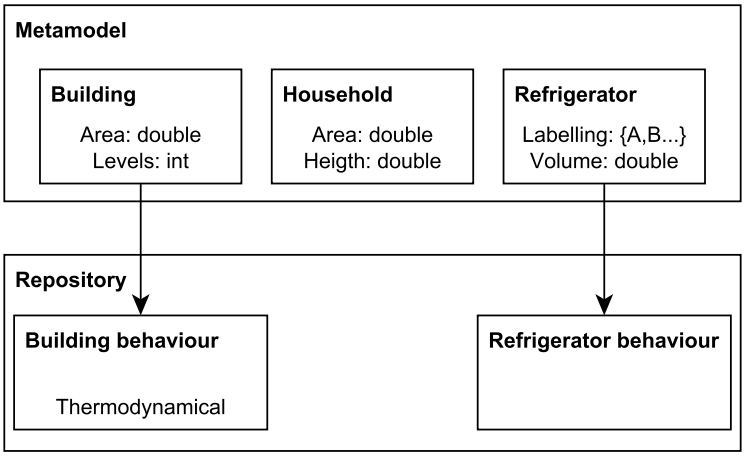

Fig. 1. Structure of the metamodel where the element definitions are exposed.

A model is a representation of a specific electrical system using the metamodel classes. Elements that are going to be simulated are instantiated from

the metamodel in the model. Both, models and the metamodel, are represented using XML.

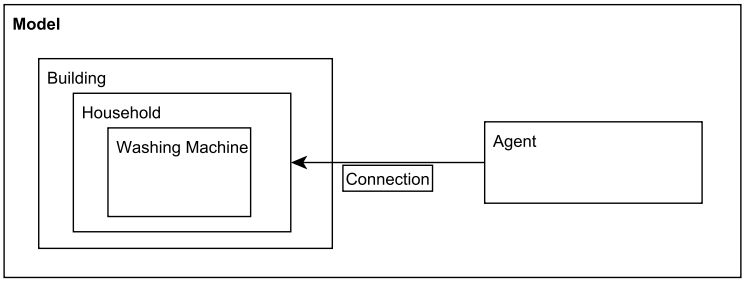

Fig. 2. Structure of the model shown by an example including behaviours

3.3 Programming behaviours

To simulate the electrical system, all the metamodel classes have to include behaviours that must be programmed.

A separation between the different types of behaviours must be done according to the discussion exposed in the section 2.2. In this section, an example of each kind of behaviour will be exposed.

Environmental behaviour Environmental behaviours represent the change over time of some environmental variables. Environmental variables are normally common to a group of entities or devices and describe the surroundings or "outdoor". These can be for example solar radiation models, which represent the insolation and can be used for calculation of the thermal gains of a building, or further for energy production (photovoltaics, solarthermal use, etc.). These behaviours do not directly change the attributes of a device or agents, but rather allow some interactions in an indirect way (e.g. through heat exchanges, etc.).

Device behaviour Most of the electrical devices used in a household are major appliances such as washing machines, refrigerators, etc. There are also some other, smaller appliances, such as CD players, TVs, HiFi Audio equipment, etc. Usually, the major appliances cause a larger part of the electrical consumption. In order to recreate the individual load curves, EIFER (European Institute for Energy Research, a common research institute by KIT and EDF) has developed individual models for the behaviours of electrical appliances, which were integrated into Tafat. Simplified technical models are used, which take into consideration different technical parameters of a specific appliance. So, for example, the load curve a TV will be characterised by the size and technology (CRT, LCD, Plasma, etc.). Major appliances also are modelled using the EU Energy Label

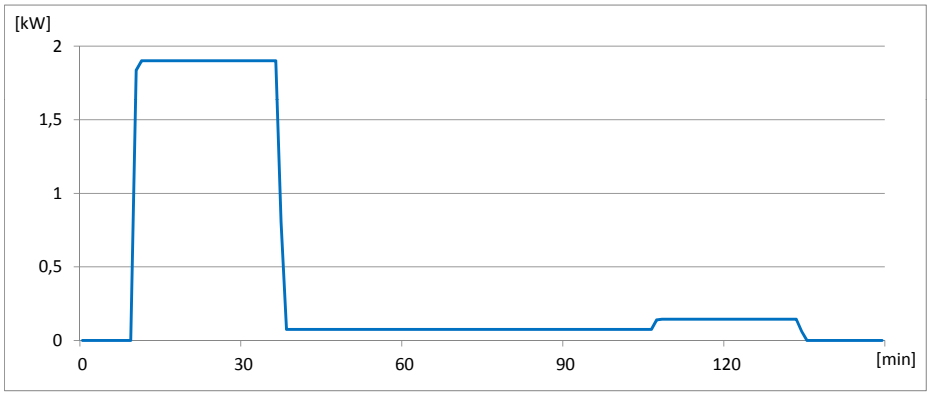

Fig. 3. Simulated load curve of a washing machine

as an input parameter, which is an indicator for the energy consumption of a device and is compulsory for appliance sold in the EU. Different releases for the behaviours of the electrical devices were created. Using this modular approach, a behaviour of a single device can be *exchanged* in a simple way. The different releases include simplified technical models with varying degrees of accuracy, thus allowing for an optimisation of execution time vs. accuracy of the model. In Figure 3 an example of the load curve generated by the behaviour of a washing machine can be seen. This load curve is created by a simplified technical model of this appliance.

Social behaviour A flexible architecture is proposed to carry out a social behaviour. As described in section 2.2, intentional stances are the most complex behaviours. For this reason, the architecture must be flexible to allow a range of behaviours from simple behaviour based on a list of tasks to a complex behaviour implemented as a neural network.

The mission maker is the intention launcher, the decision maker is in charge of choosing a recipe to accomplish the mission launched and the action maker is the executor of the recipe. The recipe is a list of actions that executes to accomplish a mission. With this architecture, a simple behaviour can be developed by creating a big recipe in which all the tasks are described and having a mission and decision maker very simple. Otherwise, in a complex social behaviour, the task can be launched by the mission maker according to several parameters of its own agent or the environment, having a hard process to choose a recipe in the decision maker, but easier recipes that only describe how to arrange a task as, for example eat.

3.4 Simulation

The main problem for developing good models that represents accurately a place (town) as the lack of data. Often, it is quite difficult to gather the needed data.

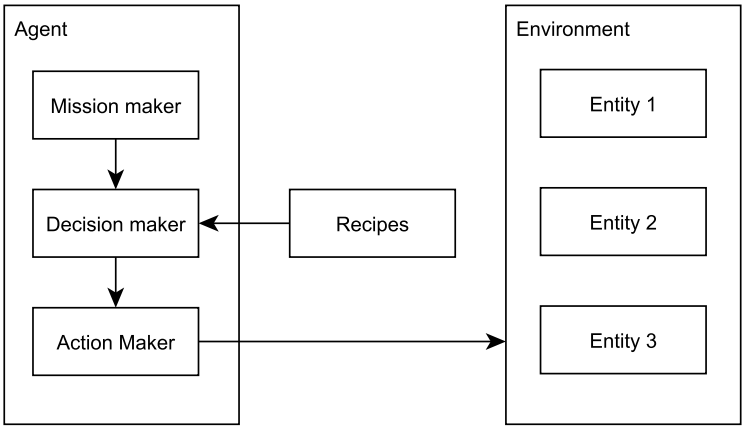

Fig. 4. Agent architecture composed by a mission maker (intentions), decision maker (recipe selector) and action maker (recipe executor)

Ideally, models should be built using real data, since the simulation will help to understand what happens in the electrical system. However, since data is not available at all, a model approximation is done. A tool called profiler has been developed (Figure 5), which helps to carry out this task. Using a high-level description of a place (for example, amount of buildings and population), Profiler automatically generates an electrical system model that can be simulated directly. The profiler is part of Tafat, a MDE platform that supports the development of simulations.

Electrical models can be created to represent the load accurately but light-weight enough for use in large scale simulations, and handle demand side management mechanisms through the use of an agent-based approach.

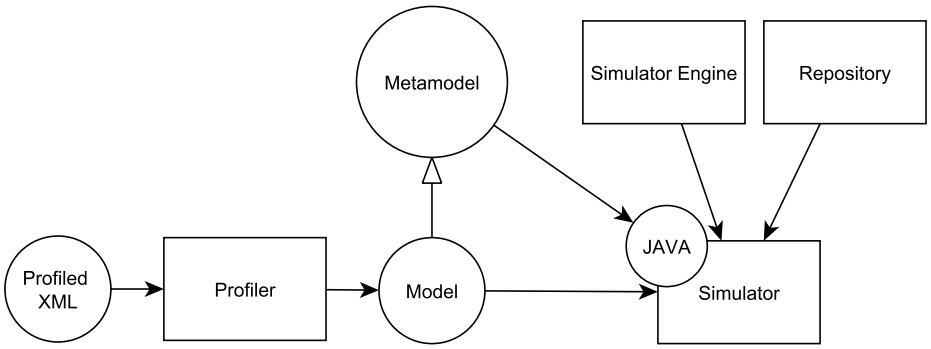

Fig. 5. Tafat architecture

4 Case Study

4.1 Modeled scene

The case study proposed is an analysis of the electrical load of households according to social group characteristics. The following five different social groups have been taken into account to develop the case study:

1. junior single,
2. senior single,
3. junior couple,
4. senior couple and
5. family with children.

These groups were taken as a part which represents about 70% of the population of Germany, being the most numerous groups identified by the German Socio-Economic Panel Study (SOEP). The constitution of this sample is shown in Figure 6.

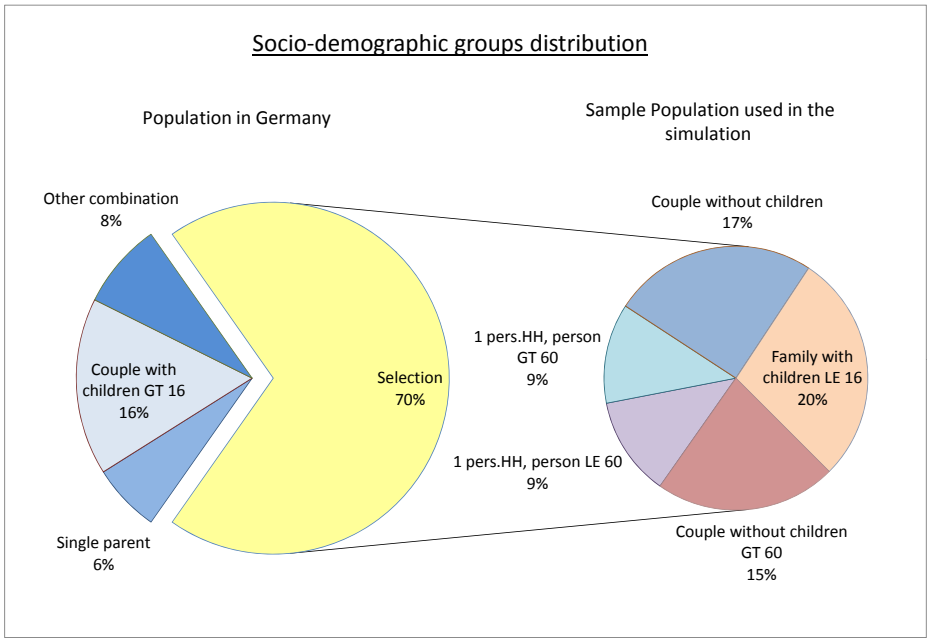

Fig. 6. Socio-demographic groups used in the case study. Source: SOEP.

A survey[3] has provided the information about the timetables and appliance usage of a household according to the social group. Based on this information the

[3] The survey consisted in local interviews with around 20 persons in Karlsruhe, Germany in order to obtain data like usage times and durations of specific socio-

agent behaviour which is related to the household is implemented. The electrical usage behaviour data employed for this study was obtained through a local survey. Thus the gathered data from a small sampling was used as input parameter in the Tafat model. Hence, 20 different social behaviours (i.e. 20 different model recipes) are been used to simulated the five socio-demographic groups. Each of these recipes includes some randomness through the definition of intervals at which devices are switched on or off. The exact time instant of time is obtained as a uniformly distributed random variable over this interval.

To arrange this study, an important utility of Tafat to build automatically a scene has been used. This utility uses statistical data from the SOEP and the survey to create a model scene in which the social groups are distributed in the households. Those households contain the electrical devices, and their amount, according to the social group.

4.2 Simulation and results

In the simulation, the device load curves within a household are generated. The curves were aggregated using an individual behaviour for each household taking into account some randomness (variation of the duration and use time in a defined range of the different electrical devices).

The simulation results show the load curve of a day of 1000 households with around 12 appliances each, composing thus an amount of approximately 12000 simulation elements. This type of simulation can provide the relevance of a determined power consumer in the household consumption as well as the influence of a specific type of consumer on the global load. The number of 1000 entities was chosen, as it was determined that larger amounts did not change substantially the results and only increased the execution time of the simulation. This is probably due to the use of a limited number of recipes (taken from the number of surveyed persons). In this case, the execution time was around 20 minutes on a standard desktop PC for the simulation of a period of 24 hours.

The 1000 household sample is composed of a distribution of the different social groups according to real statistical data, in order to obtain a sample of households that is as close as possible to reality. In Figure 7, the simulated load curve for one day can be seen. The simulation is run in a high time resolution, being the time step one minute. This allows observing effects which are neglected in simulations at lower resolutions, provided by 15 minute or hourly models. Some sharp peaks can be observed, which are caused by the use of high power consuming devices in the household. A general trend to use more power during the day is clearly visible. During the night, the base load (devices that are constantly running, such as refrigerators and other permanent loads) cause a consumption that is only around a third part of the daily peak load. The configuration of the simulation can be seen in Figure 8.

---

demographic groups. It has to be noted that the survey is not representative but rather a sample of the user behaviours of those groups.

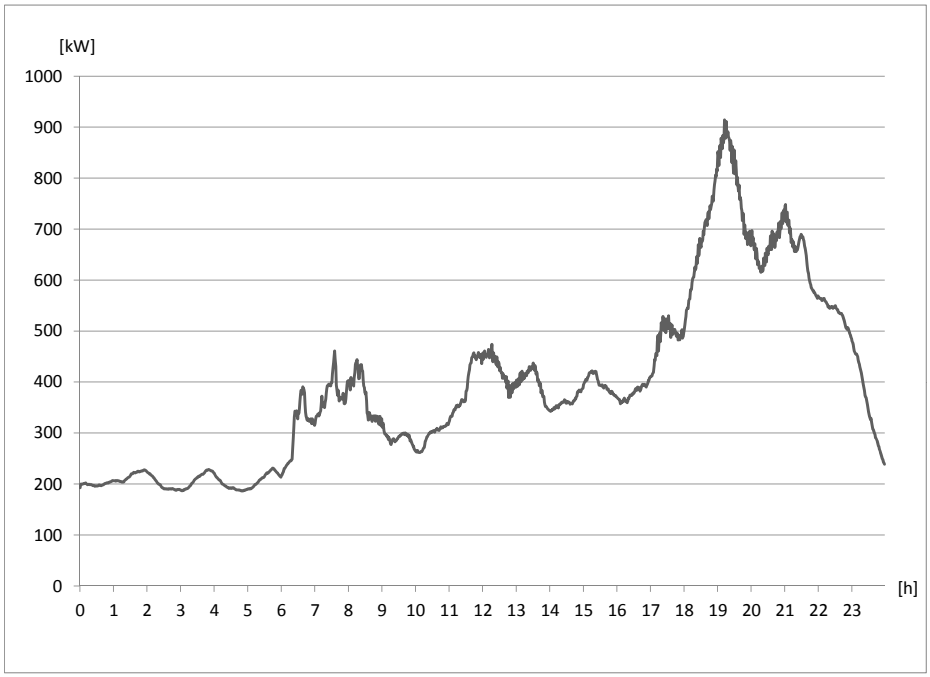

Fig. 7. Simulated load curve of 1000 households for one day

In Figure 9 the curve is compared to a standard load profile of Germany for a winter weekday (according to [1] and made public by [7]) for household demand. The profile is provided in a normalised form in order to be weighted with a given number of energy. In this case, and for comparison purposes, the profile was weighted with the same amount of energy as the simulation curve, i.e. that the daily consumed energy [kWh] is the same in both cases. The load profile is a smooth curve, representing an aggregated load behaviour at high levels of the electricity system for large number of consumers. The are the result of a statistical analysis based on representative samples from different consumer groups [5].

The simulated curve represents the total power consumed by a sample of 1000 households, modelled individually and with an autonomous behaviour for each of them. The curve has been adapted by averaging periods of 15 minutes in order to match the same time granularity as given by the standard load profile. The curve is more peak shaped, which is probably due to the relatively small amount of households (in comparison to the statistical samples taken to obtain a profile, which are representative) and the reduced number of behaviours (in total, only 20 different behaviours have been used). Furthermore, only 70% of the household population is modelled, neglecting other social groups which may change the curve.

Entity	Attributes	Components	Behavior
Building		Household: n=1	-
Household	sg = montecarlo() Households are classified into one of the 5 social groups (sg) using montecarlo method area = $f_{area}(sg)$ Area is variable depending on the social group	PowerConsumers CookingStove n=4 Microwave n=1 Oven: n =1 Dishwasher: n = 1 Refrigerator: n = 1 AudioHifi: n = 1 TV: n = f_{tv}(sg) Ligthing: n = 1 Washing Machine: n = 1 Computer: n=$f_{computer}$(sg) Waterboiler: n = 1 Hairdryer: n = 1 Vacuum: n = 1 Iron: n = 1 Console: n = $f_{console}$(sg) PowerBus	Sociological Behavior Each social group has 4 different recipes. A recipe is randomly selected. The recipe defines how the power consumers **mode** is changing
PowerConsumer	mode power		DeviceBehavior Power is calculated depending on the device mode and on its technical characteristics
PowerBus	power		DeviceBehavior power is calculated by aggregating the consumption of Household PowerConsumers

Fig. 8. Configuration of the entities used in the simulation.

Even though the selected samples in the survey are not representative for all Germany nor society as a whole (only five social groups were used), the general trend of both curves are similar. Three peaks can be observed, which are closely synchronised in time and correspond to morning, noon and evening peaks. These peaks are correlated with a large and concurrent usage of high power devices, such as cooking plates, ovens, microwaves, etc. due to alimentation habits, as well as lightning use in the evening hours. The morning and noon peaks are lower in the simulation than in the profile, whereas the evening peak is higher. The timely synchronisation of the ramps of the peaks matches quite well; this indicates that the activities (having breakfast, lunch, returning home, etc.) were modelled according to the average German user behaviour. Even though, some differences can be observed at the evening drop (21-23h), as well as a small second peak that cannot be found in the profile.

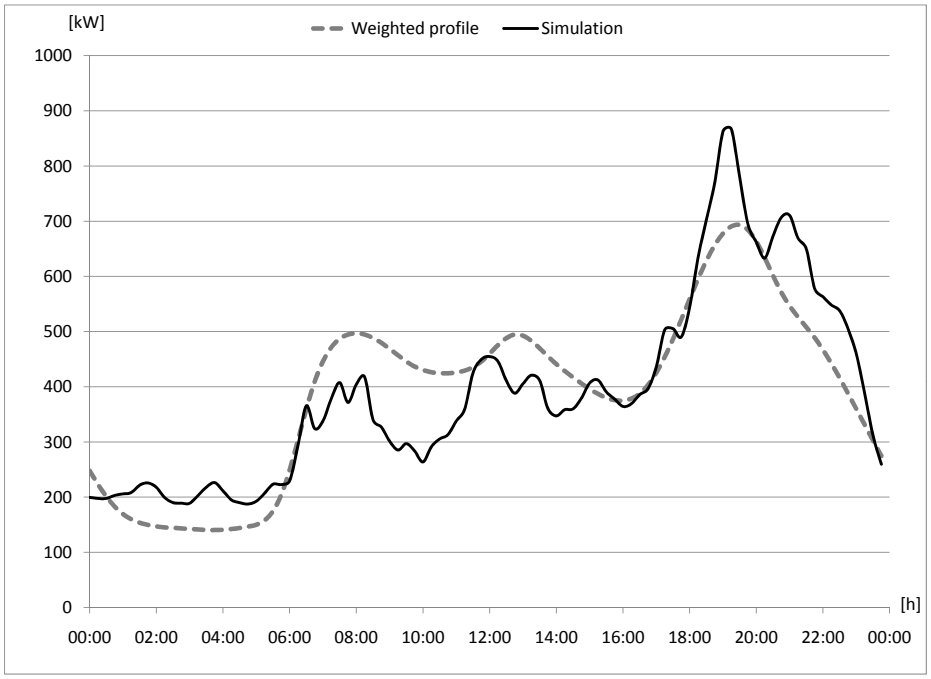

Fig. 9. Simulated load curve vs. standardised residential load profile

4.3 Discussion

Even using a relatively small sample of households and reduced number of behaviours, a curve that represents the major characteristics (peaks and troughs, as well as their timely synchronisation) is generated. Concerning the differences in the height of the peaks, the model could be reviewed in order to check the individual power curves of each electrical device. This seems to be quite hard as almost no data is available for such a validation (at a representative sample). However, the differences could also be related to the use of only 70% of the socio-demographic population share. Further, behaviour itself is another factor to consider, as a largely simplified and almost static model was used. Some specific characteristics of the model create peaks, which could be explained by a rather homogeneous behaviour of the groups. For example the second evening peak (around 21:30h) is possibly caused by some activities (watching TV, other evening activities) which start and end at similar times, because of the relatively small sample. Using data from a more representative survey or a more stochastic-based social behaviour, this could be avoided, though.

5 Conclusions and outlook

In this paper, a vision of the electrical system as a complex system has been introduced. This is necessary as the future grid behaviour becomes more distributed. Furthermore, the simulation of the electrical system at a household level as a complex system has been addressed.

The case study demonstrates that a bottom-up simulation of the residential consumption using an agent-based approach has been successful since the result curves show similarities to aggregated load curves. A comparison with a national aggregated profile shows similarities in the main characteristics of the curve. Moreover, due to the high resolution of the model, a large number of parameters (individual appliances types and models, socio-technical behaviours, etc.) is available for variation.

From now, the simulation that has been developed will allow us to experiment and study new algorithm and strategies for electrical system management. New scenarios, new problems and new challenges will arise in the near future with the introduction of renewables and a distributed production in the electrical system. Simulation is particularly necessary to design a new management approach.

The simulation will allow us with further work to study the integration of demand side management strategies. Strategies, such as adaptive or reactive technologies, incentives or campaigns, can be addressed for studying their impact at load curve level.

However, social behaviours need to be validated and improved. This validation is necessary for studying the emergent behaviours and for identifying the local actions of agents which provokes the desired emergent behaviour. Moreover, social behaviours must be improved in order to introduce a higher degree of heterogeneity to the models. Due to the high resolution of the household model, individual actions such as changing specific parameters on a device can be performed.

Furthermore, the model developed could be expanded in order to simulate not only the demand side, by including distributed generation or other injections to the grid (like storage), which could interact with the existing elements. This is already contemplated within Tafat, and would allow a disaggregated analysis of offer-demand balance and the possibility to estimate the impact of those measures at a system level.

References

[1] Bundesverband der Energie- und Wasserwirtschaft: BDEW - Standardlastprofile (SLP) (2011)
[2] Dennett, D.: The Intentional Stance. M.I.T. Press (1987)
[3] Monti, A., Ponci, F., Benigni, A., Liu, J.: Distributed intelligence for smart grid control. In: Nonsinusoidal Currents and Compensation (ISNCC), 2010 International School on. pp. 46–58 (2010)
[4] Paatero, J.V.: Computational Studies on Variable Distributed Energy Systems. Phd thesis, Helsinki University of Technology (2009)

[5] Palensky, P., Kupzog, F., Zaidi, A.A., Kai, Z.: Modeling domestic housing loads for demand response. In: Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE. pp. 2742–2747 (2008)
[6] Polack, F.A.C., Hoverd, T., Sampson, A.T., Stepney, S., Timmis, J.: Complex systems models: Engineering simulations. In: ALife XI, Winchester, UK, August 2008. pp. 482–489. MIT Press (2008)
[7] RWE Rhein-Ruhr Verteilnetz: Lastprofile (2011), http://www.rwe-rhein-ruhr-verteilnetz.com/web/cms/de/201616/rwe-rhein-ruhr-verteilnetz/netzzugang-strom/netzzugang-netznutzung/lastprofile/
[8] Schmidt, D.C.: Guest editor's introduction: Model-Driven engineering. Computer 39, 25–31 (Feb 2006), http://doi.ieeecomputersociety.org/10.1109/MC.2006.58
[9] Stepney, S., Polack, F.A.C., Turner, H.R.: Engineering emergence. In: Engineering of Complex Computer Systems, 2006. ICECCS 2006. 11th IEEE International Conference on. p. 9 pp. (2006)
[10] Stokes, M., Rylatt, M., Lomas, K.: A simple model of domestic lighting demand. Energy and Buildings 36(2), 103–116 (2004)
[11] Widén, J., Wäckelgard, E.: A high-resolution stochastic model of domestic activity patterns and electricity demand. Applied Energy 87(6), 1880–1892 (2009)
[12] Willis, H.L., Scott, W.G.: Distributed power generation: planning and evaluation. Marcel Dekker, New York (2000)
[13] Wooldridge, M.J.: An Introduction to Multiagent Systems. John Wiley, Chichester, West Sussex, United Kingdom (2002)
[14] Wright, A., Firth, S.: The nature of domestic electricity-loads and effects of time averaging on statistics and on-site generation calculations. Applied Energy 84(4), 389–403 (2007)