Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO DE FIN DE MÁSTER

Comunicación entre un simulador y un optimizador para la mejora del proceso de esterilización de una empresa de conservas

Alumno: **Luis Tormo Rico**

Tutor 1: Gloria Gutiérrez Rodríguez
Tutor 2: Carlos Gómez Palacín

Máster en Investigación en Ingeniería de Procesos y Sistemas Industriales

Convocatoria de defensa: **Septiembre 2020**

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO DE FIN DE MÁSTER

Comunicación entre un simulador y un optimizador para la mejora del proceso de esterilización de una empresa de conservas

Alumno: **Luis Tormo Rico**

Tutor 1: Gloria Gutiérrez Rodríguez
Tutor 2: Carlos Gómez Palacín

Máster en Investigación en Ingeniería de Procesos y Sistemas Industriales

Convocatoria de defensa: **Septiembre 2020**

Resumen

Castellano

En este proyecto se pretende desarrollar una interfaz directa entre un simulador de eventos discretos y un *software* de optimización con el fin de poder establecer una conexión entre ambos. Como demostración de la validez y utilidad de este trabajo, se va a aplicar en un caso real como es el estudio de la mejora de un proceso de esterilizado a través de la optimización de la planificación o *scheduling*.

Primero se deberá estudiar y entender el funcionamiento del proceso industrial para posteriormente, crear un modelo matemático a través de programación lineal mixta-entera que defina su comportamiento teniendo en cuenta las restricciones que tienen este tipo de sistemas. Dicho modelo permitirá obtener la planificación a corto plazo de la planta de esterilizado. Este se codificará usando el lenguaje de alto nivel, Julia. Que además de ofrecer una sintaxis basada en Optimization Programming Language, que facilita la traducción de un modelo de programación mixta-entera, dispone de interfaces con los principales optimizadores disponibles, tanto comerciales como gratuitos (Gurobi, CPLEX, Cbc...).

Por otro lado, se desarrollará una simulación que imite el comportamiento real del proceso mediante un *software* de simulación de eventos discretos, Simio. Mostrando cómo funciona el programa y las capacidades industriales que permite.

El objetivo final será el de unir de manera efectiva el simulador y el optimizador de modo que cada vez que el sistema simulado tenga una variación u ocurra un cierto evento, el estado actual se envíe al optimizador. El modelo matemático devolverá una planificación óptima adecuada al nuevo estado de la planta. Es decir, se producirá una actualización iterativa del sistema. Esto aportará una base virtual sobre la que probar distintas estrategias de planificación sin afectar a la producción de la planta real.

Palabras clave: Industria alimentaria; Scheduling; Optimización; Planificación de producción; Julia; Simio.

Inglés

This project aims to develop a direct interface between a discrete event simulator and optimization software in order to establish a connection between the two. As a demonstration of the validity and usefulness of this work, it will be applied in a real case such as the study of the improvement of a sterilization process through the optimization of planning or scheduling.

First, it is necessary to study and understand the operation of the industrial process to later create a mathematical model through mixed-integer linear programming that defines its behavior taking into account the restrictions that these types of systems have. This model will allow obtaining the short-term planning of the sterilization plant. This will be coded using the high-level language, Julia. That in addition to offering a syntax based on Optimization Programming Language, which facilitates the translation of a mixed-integer programming model, it has interfaces with the main optimizers available, both commercial and free (Gurobi, CPLEX, Cbc ...).

On the other hand, a simulation will be developed that mimics the real behavior of the process using a discrete event simulation software, Simio. Showing how the program works and the industrial capabilities it allows. The ultimate goal will be to effectively link the simulator and optimizer together so that every time the simulated system has a variance or a certain event occurs, the current state is sent to the optimizer. The mathematical model will return an optimal planning appropriate to the new state of the plant. That is, there will be an iterative update of the system. This will

provide a virtual base on which to test different planning strategies without affecting actual plant production.

Keywords: Food industry; Scheduling; Optimization; Production planning; Julia; Simio.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

1.1 Objetivos

El objetivo principal del trabajo pasa por establecer la comunicación entre un simulador y un *software* de optimización con el fin de poder tener una conexión efectiva entre ambos. Esto ayudará de manera considerable a hacer estudios, análisis o incluso implementar dentro de sistemas industriales reales multitud de tecnologías que requieren gran capacidad de cálculo.

Con el fin de aplicar este resultado a un caso real, se va a utilizar esta unión para la ayuda de la mejora de un proceso de esterilizado de una empresa real de conservas. Implementando esta herramienta en la prueba y estudio de aplicación de modelos matemáticos de Programación Lineal Mixta Entera (MILP) para la optimización de tareas o *scheduling* en cortos periodos de tiempo. En el trabajo se pretende:

- Conocer y entender profundamente el funcionamiento del *software* de simulación de eventos discretos, Simio.
- Desenvolverse con el lenguaje de programación dinámica de alto nivel, Julia.
- Modelar de manera eficiente un problema de planificación de tareas y comprobar el resultado de su optimización.
- Desarrollar una simulación de manera exacta a la del proceso industrial real con Simio.

- Estudiar y conseguir hacer efectiva la comunicación entre Simio y Julia.

1.2 Ámbito de aplicación

En las fábricas modernas es común encontrarse una pirámide jerárquica que defina el control de la empresa como la de la figura 1.1. Este esquema clásico destaca por estar formado por diversos niveles. Los superiores son los encargados de hacer planificaciones a largo plazo controlando siempre el nivel de inventario, los beneficios y gastos que se van a tener y en definitiva, disponer de una visión global del conjunto empresarial, llamados ERP (Enterprise Resource Planning). Los niveles inferiores son los dedicados íntegramente a producción y control del proceso, los más bajos están formados por el sistema productivo en sí, pero el más importante en este caso son los niveles intermedios o sistemas MES (Manufacturing Execution System). Estos toman decisiones y planifican en intervalos cortos de tiempo como minutos, turnos, días...(Yang y Takakuwa 2017). El trabajo se enmarca en la mejora de este tipo de niveles.

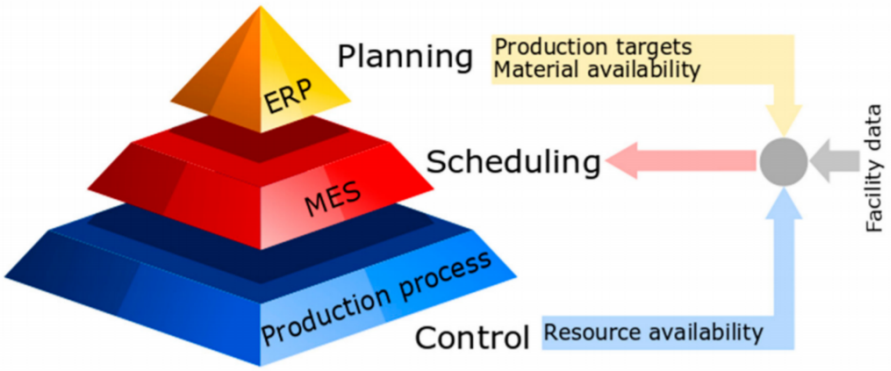

Figura 1.1: Flujo de información entre diferentes niveles de planificación. Fuente: G. P. Georgiadis, Elekidis y M. C. Georgiadis 2019.

Los sistemas MES, encargados del correcto funcionamiento de los niveles más bajos de la pirámide y en contacto siempre con los intereses globales que dictan los ERPs son los más importantes. Las empresas saben que para ellos es imprescindible tener una producción ajustada, donde se deben minimizar todos aquellos tiempos que de un modo u otro no sean rentables para el proceso. Todo ello requiere de una buena planificación, conocer siempre cuál es la mejor combinación de recursos que cumplan un determinado requisito, como puede ser sacar el máximo rendimiento a la planta o tener

un gran ahorro energético. Este dependerá de las determinadas situaciones cambiantes que pueden ocurrir en una empresa.

Dependiendo de la situación, se tendrán que tomar unas decisiones. Estas pueden ser dadas por la experiencia de los propios responsables o a través métodos de planificación. Tradicionalmente los organismos encargados de planificar este tipo de tareas lo hacen en vista de periodos de tiempo medios como turnos o días, teniendo en cuenta determinadas situaciones para asegurar el objetivo de producción establecido. Pero, ¿y si se hiciera una planificación de cortos periodos de tiempo (minutos) que tenga en cuenta la situación cambiante que puede haber en el proceso?

Ante toda mínima propuesta o intención de cambio una empresa debe conocer cuáles van a ser las repercusiones y cómo va a responder el sistema. Por ello una buena solución es optar por el uso de simuladores. La forma de relacionar y probar todas las técnicas de optimización de *scheduling* de cortos horizontes de tiempo puede ser de manera previa, utilizando algún *software* que introduzca este tipo de metodología dentro del mismo. Además de ello dotar al simulador de cierta inteligencia y aumentar su potencia de cálculo.

1.3 Antecedentes

Desde el momento en que la tecnología e internet aparecen en el mundo industrial junto con un mercado globalizado, las empresas deben encontrar la manera de sacar el máximo rendimiento a todos los sistemas de producción, disminuir gastos y obtener el máximo beneficio. A partir de ese momento desde el mundo de la investigación se empieza a tratar métodos de optimización para este tipo de industrias.

Empresas del sector químico, farmacéutico, alimentario, metalúrgico, gas y petróleo o transportes tienen la problemática de tener procesos donde cobra especial importancia el tiempo, las tareas a ejecutar y los recursos disponibles. El *scheduling* o planificación de producción nace como uno de los instrumentos para la optimización de procesos.

Desde el mundo industrial muchas de las empresas han visto este tipo de técnicas demasiado complejas y los planificadores se han centrado más en usar simuladores o manuales de decisión, obteniendo de esta manera solu-

ciones no del todo óptimas (G. P. Georgiadis, Elekidis y M. C. Georgiadis 2019). Debido a la difícil aplicación hace unos años de estos sistemas, los *software* de simulación cobraron especial importancia a la hora de tomar determinadas decisiones en los sistemas MES. Estos siempre han servido para poder probar técnicas antes de llevarlas a planta y poder hacerse una idea de cómo se comportaría un sistema real ante un determinado cambio.

A medida que pasa el tiempo, este tipo de metodología se va haciendo cada vez más necesaria, la digitalización y la llegada de la industria 4.0 hace que el mundo empresarial comience a colaborar con el mundo de la investigación. El objetivo pasa por poder optimizar sus plantas al máximo para que las decisiones que se tomen a pie de máquina se hagan en función de un determinado interés de la empresa y no solo en base a la observación o experiencia del empleado.

En muchos de estos procesos el avance pasa por establecer este tipo de optimización y mejora para la toma de decisiones aprovechando las nuevas tecnologías conociendo el estado en el que se encuentra una determinada planta industrial en todo momento. De esta manera poder obtener decisiones en base a lo que esté ocurriendo. Muchas tendencias de investigación se han centrado en esto, como la llevada a cabo por el grupo Control y Supervisión de Procesos (CSP) de la Universidad de Valladolid a través de proyectos públicos europeos como CoPro o estatales como Inco4in.

1.4 Planteamiento del trabajo

El trabajo pasa por poder implementar en una factoría real el concepto de *scheduling*, hacer todo el modelo matemático que defina el comportamiento del sistema y obtener la planificación de la producción de un periodo corto de tiempo. Por otro lado, crear en un *software* de simulación la planta de fabricación para mostrar cómo funciona y cómo se pueden tomar decisiones. El objetivo final es poder unir estos dos sistemas que hasta ahora han ido separados, es decir, poder experimentar dentro de un simulador cómo funcionaría la planta si las órdenes fueran dadas por un optimizador a partir de un modelo matemático.

Por ello el trabajo se estructura principalmente en 3 partes, la elaboración de un modelo matemático para la optimización del *scheduling*, la elaboración de un modelo de simulación y la unión de ambos.

1.4.1 Scheduling de horizontes cortos de tiempo

Hoy en día el mundo se mueve muy rápido y la industria debe de adaptarse a ello y reinventarse día a día con su respectiva competencia. Las empresas deben sacar un determinado número de productos en un intervalo de tiempo. Los niveles intermedios de la pirámide o departamentos de producción tendrán un papel principal a la hora de cumplir con los objetivos de los ERP. La planificación y el buen secuenciamiento de las tareas de cada uno de ellos serán vitales para el correcto funcionamiento de la empresa.

Observando más cerca la industria, sobre todo la relacionada con la química, farmacéutica o alimentaria es muy probable que en algunas etapas de una producción continua aparezcan procesos por lotes o procesos de tipo *batch*. Este tipo de producción requiere de un importante componente de planificación, pero normalmente es vista en horizontes medianamente largos de tiempo como días. Esto tiene un inconveniente y es que no tiene en cuenta los posibles cambios o problemas que puedan derivarse de la línea de producción y por lo tanto puede no ser del todo efectiva.

El objetivo del *scheduling* en este trabajo es mediante la programación lineal-mixta entera, obtener un modelo matemático que defina el comportamiento del sistema y implantarlo en un lenguaje de programación como es Julia. De esta manera hacer una optimización que proporcione el mejor secuenciamiento de tareas que deberá tener la planta de esterilizado en un horizonte corto de tiempo, dadas unas determinadas condiciones. El resultado se mostrará y se analizará utilizando diagramas de Gantt.

1.4.2 Simulación

Para una planta de producción es vital poder experimentar y conocer el comportamiento en un tiempo futuro. Esto se consigue con un lugar en el que se agrupen todas las variables, materias, recursos o equipos del sistema. Para ello, existen las simulaciones que son capaces de mostrar cómo será la evolución de un determinado proceso a lo largo del tiempo. Los modelos de simulación deben emular con la mayor precisión posible el proceso que se

quiera estudiar, cosa que permitirá a los encargados mejorar y adaptarse a las situaciones futuras que vayan a ocurrir.

La simulación en este trabajo consistirá en emular el comportamiento de un proceso de esterilización real de una planta de producción de conservas, mediante el *software* de simulación Simio.

1.4.3 Unión de la simulación y la optimización

En vista de poder dotar a un simulador como Simio de una gran capacidad de cálculo, se persigue el objetivo de poder integrar un lenguaje de programación cómo Julia con altas capacidades de cálculo matemático dentro del mismo.

Esto se elaborará utilizando el caso de mejora del *scheduling* de la planta de esterilizado. La idea es que se pueda implementar el modelo matemático desarrollado dentro de la simulación. Para ello se va a intentar incluir de alguna manera la planificación de tareas dentro de Simio, cuando este lo considere necesario.

Esto provocará principalmente dos cosas.

- Todos los métodos de optimización de *scheduling* en plazos cortos se podrán probar previamente antes de llevarlos a cabo en la planta real. Lo que ayudará a esta metodología a integrarse más fácilmente, pudiéndose probar si utilizando este tipo de técnicas de decisión se optimiza el sistema a largo plazo.

- La segunda es que se mejora el *software* de simulación, Simio. Gracias a la capacidad de cómputo de un lenguaje de programación como Julia. Su unión dará la posibilidad de utilizar gran cantidad de algoritmos matemáticos y métodos de cálculo.

2 Caso de estudio

En este capítulo se resume toda la información necesaria para conocer en profundidad el sistema industrial elegido. Para ello se hará una descripción de la planta con los elementos que la componen, su nomenclatura y una explicación del proceso en sí. Después se expondrán las partes más importantes del proceso de esterilizado y todo lo que se debe tener en cuenta para poder mejorar este tipo de plantas.

2.1 Planta

El trabajo se centra en una planta de producción de conservas, concretamente en la parte de esterilizado.

En la fábrica de conservas, los alimentos se procesan y son introducidos en latas con sus respectivos aceites y otros elementos. Estos pasan a ser envasados y sellados. Durante este tiempo se puede decir que el procedimiento de producción ha sido continuo, hasta que se llega a la parte de esterilizado. Una vez envasado se tienen que eliminar todos los microorganismos que habiten en los alimentos sean o no patógenos. En este punto el proceso se convierte en semicontinuo, los operarios pasan a ser los encargados de transportar las latas provenientes de la linea de envasado a los correspondientes esterilizadores (G. P. Georgiadis, Ziogou y col. 2018).

2.1.1 Elementos del sistema

Existen 4 tipos de elementos que se tienen que conocer:

- Las latas: Elemento más básico y a la vez principal del sistema.
- Los carros: Recipiente metálico en el que se introducen las latas de un mismo tipo.
- Los lotes o *slots*: Agrupación de carros listos para ser introducidos en los esterilizadores.
- Esterilizadores o autoclaves industriales: Son recipientes metálicos donde se introducen los lotes para eliminar cualquier microorganismo existente en las latas.

2.1.2 Procedimiento

Por tanto, se tiene unas latas que van llegando de la línea de envasado y son introducidas en carros metálicos. El vaciado y el llenado de estos se produce de manera automática pero el desplazamiento es manual. Los operarios van llenando los autoclaves con ellos hasta que se llega a su capacidad o si se deja de producir esa unidad en concreto. Si el esterilizador está lleno, los carros van formando una cola delante del mismo. El grupo de carros que se introduce en el autoclave son los llamados lotes o *slots*. Una vez dentro, el operario debe seleccionar el programa de temperatura correspondiente dependiendo del tipo de latas que haya introducido.

Cuando el programa termina son los mismos empleados los encargados de sacar los lotes de los esterilizadores para que las latas sigan su proceso de empaquetado y almacenamiento en la empresa.

2.2 Partes importantes del proceso de esterilización

Para conocer de lleno el proceso en el que se centra el trabajo se debe hablar de algunas partes importantes. La primera es entender el funcionamiento que tienen los recipientes encargados de la esterilización. La segunda es saber qué es el perfil térmico, de qué depende y cómo de importante es para una empresa alimentaria como esta. La última es conocer de primera mano,

cuáles son las peculiaridades y problemas que acarrean este tipo de plantas industriales y que por tanto se tienen que tener en cuenta en todo momento.

2.2.1 Autoclaves industriales

El proceso de esterilizado ocurre en unos autoclaves industriales como el de la figura 2.1. Este consiste en un recipiente metálico de gran tamaño donde se introducen los carros.

Su funcionamiento se basa en rociar las latas con agua sobrecalentada. Para ello se tienen 2 corrientes principales de entrada, una de vapor y otra de agua caliente. A los productos se les rocía mediante un circuito cerrado de agua que se va calentando a través de un intercambiador de calor. Para conseguir la temperatura adecuada para el perfil térmico requerido se utiliza un lazo de control que varía la entrada de agua caliente y vapor. Al circuito cerrado de agua se le aplica aire comprimido mediante otro lazo de control para aumentar o disminuir la presión en el fluido.

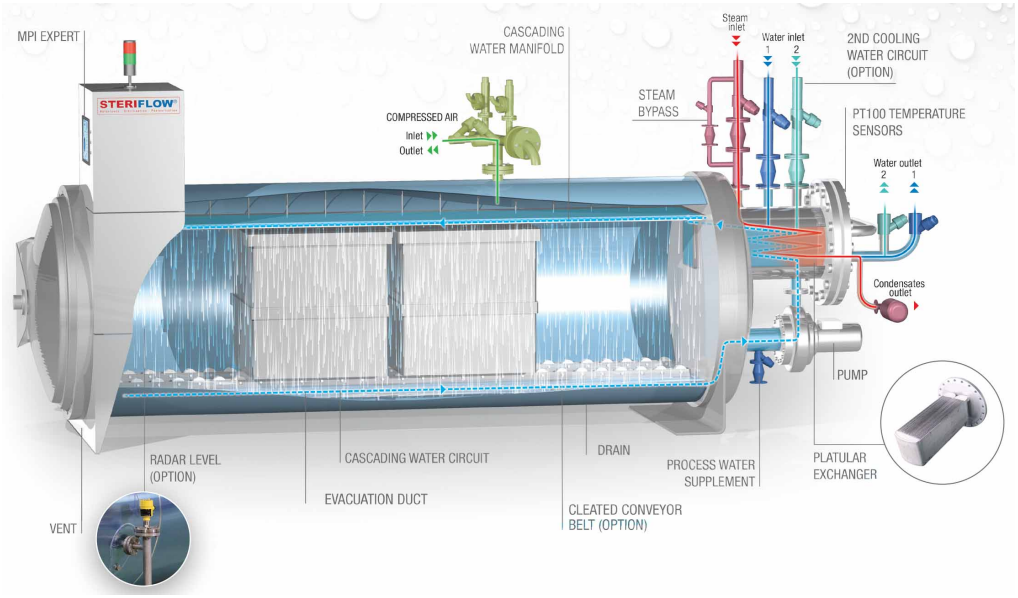

Figura 2.1: Ejemplo de autoclave industrial de la empresa Steriflow. Fuente: Steriflow 2016

2.2.2 Perfil térmico

El objetivo de una esterilización es asegurar que se reduzca la población de microorganismos y por tanto existan unos niveles de seguridad en los productos que van a consumir los clientes. Para ello se utiliza un tratamiento térmico.

Estos tratamientos necesitan un estudio previo, dando un perfil térmico diferente dependiendo del tipo de comida, tamaño o geometría del envase. Este tipo de perfiles están formados por tres fases:

- Calentamiento, con el que se aplica una subida muy rápida de temperatura. La población bacteriana empieza a disminuir.
- Fase de mantenimiento de la temperatura en la que se asegura la letalidad.
- Enfriamiento rápido del sistema que deja la letalidad en un valor estable.

Por tanto, la temperatura y el tiempo de aplicación constituyen un factor muy importante a la hora de garantizar la seguridad de los alimentos. En la figura 2.2 se pueden ver como son los distintos perfiles pueden afectar a la letalidad de los microorganismos.

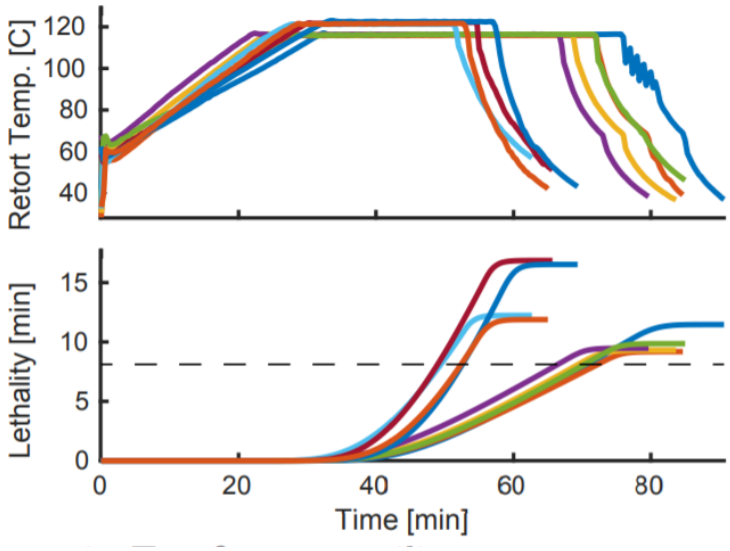

Figura 2.2: Ejemplo de diferentes perfiles térmicos en la esterilización del producto de una empresa real de conservas y su consecuente letalidad. Fuente: Vilas y Alonso 2018.

2.2.3 Restricciones

Las peculiaridades del sistema vienen a través de las restricciones del mismo. Estas están relacionadas sobre todo con que el producto sea un alimento. Este tipo de industrias deben cumplir altos estándares de seguridad para que el estado del producto de consumo que llegue a los clientes sea totalmente adecuado.

Tradicionalmente estos sistemas se basan bastante en el conocimiento de los propios operarios de la empresa. La metodología ha dependido mucho de las condiciones que se detallan a continuación y los encargados de las máquinas solían tomar decisiones, muchas veces, bastante conservadoras para que no hubiera problemas. Esto causaba una falta de aprovechamiento de recursos y tiempo dentro de la planta industrial. Por ello, ante toda mejora, automatización o optimización se debe tener muy en cuenta las peculiaridades que puedan tener este tipo de sistemas (Vilas y Alonso 2018).

- Los carros no pueden superar en espera un determinado tiempo.

  Es muy importante que las latas se esterilicen antes de cierto tiempo dependiendo del tipo de producto, el estado y el momento en el que se haya introducido al envase. Por ello es esencial eliminar los microorganismos antes de que el producto pueda ponerse malo. Si este tiempo se sobrepasa se debe analizar si el alimento de las latas de ese carro en concreto sigue manteniendo unas propiedades adecuadas.

- El perfil térmico se debe adaptar a una curva precalculada.

  Dependiendo del producto y del envase, cada tipo de lata tiene una curva precalculada y estudiada previamente a la que el perfil térmico del autoclave se tiene que ceñir.

- Las restricciones que dependen de las capacidades del sistema.

  En este caso son aquellas que únicamente tienen que ver con las capacidades del sistema. Un ejemplo claro es la cantidad de vapor que necesitan los autoclaves para cumplir con la temperatura requerida en el perfil térmico de los productos. Esto se ve claramente en la figura 2.3. Puede darse el caso que, si dos autoclaves empiezan a calentar a la vez, no haya suficiente vapor o energía para abastecer ambos y por tanto la curva de calentamiento se vea afectada.

Entre estas restricciones, las que tiene que ver con el perfil térmico dependen del sistema de control del propio autoclave. Para esto se usan lazos, que hacen que la temperatura de consigna se ajuste a la estudiada previamente para este tipo de lata y producto. Una solución para la mejora de este tipo de máquinas podría ser la establecida por Vilas y Alonso 2018 en la que se utiliza un modelo para garantizar la seguridad junto a una optimización en

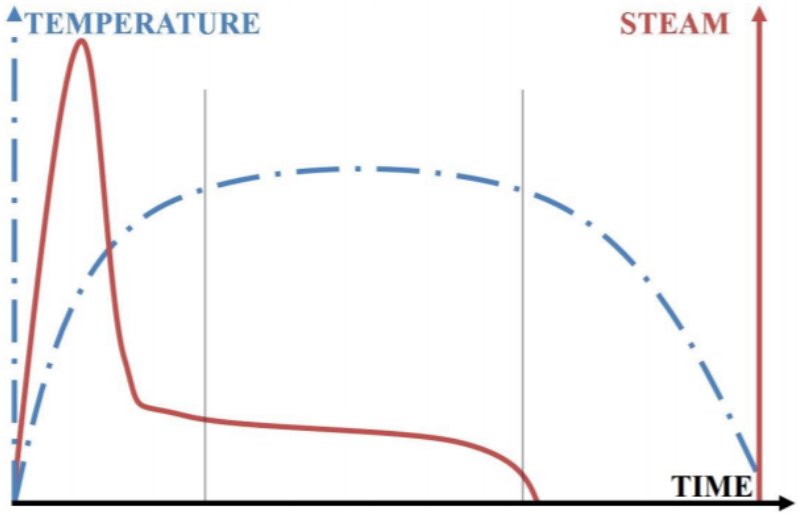

Figura 2.3: Ejemplo del perfil de temperatura y vapor de un autoclave en función del tiempo. Fuente: Palacín y de Prada 2019

tiempo real, que cambie la temperatura de consigna al controlador cuando sea necesario.

Sobre todo, en el sistema, se tienen los problemas que puedan derivarse de un tipo de proceso semi-continuo como este, en el que la asignación de recursos a las máquinas correspondientes se debe hacer de la mejor manera posible evitando problemas de producción que puedan ocasionarse como cuellos de botella. Por ello, las otras dos restricciones se pueden solucionar a través de un *scheduling* óptimo que prevenga a través de sus órdenes, el hecho de que ocurran este tipo de inconvenientes.

3 Elaboración del modelo matemático y optimización del scheduling

En este capítulo se describe la optimización del proceso industrial. Para ello se elige una metodología en tendencia actualmente, que es la planificación o *scheduling* en horizontes cortos de tiempo de procesos *batch* y procesos semicontinuos como este. A partir del análisis hecho en el capítulo 2 de la planta de esterilizado, se hacen dos modelos matemáticos con formulación MILP. El primero del sistema en sí y el segundo en el caso que se tenga en cuenta una restricción de un recurso compartido como puede ser el vapor de suministro a los autoclaves. Para elaborar la optimización se resuelven los modelos implementándolos en Julia. En la parte final se expresarán los resultados obtenidos.

3.1 Scheduling

3.1.1 Definición

El *Scheduling* es el proceso de toma de decisiones que juega un gran papel en muchas industrias como la papelera, metalúrgica, gas y petróleo, químicas, alimentarias, transportes, servicios... Sobre todo, en plantas industriales donde las tareas requieren ser procesadas en unos recursos asignados (Harjunkoski y col. 2014).

En un proceso industrial como uno químico es necesario que la producción esté correctamente planificada para asegurar que en un determinado momento se va a disponer del material, maquinaria o empleados necesarios.

La planificación de producción se hace para tener cuatro cosas claras de las tareas o actividades que se van a llevar a cabo:

- Tareas a ejecutar.
- Asignación a recursos.
- Su secuenciamiento.
- Cuando se producen.

El *scheduling* será diferente dependiendo del objetivo final para el que se haga. Por ejemplo, obtener la máxima producción, tener un ahorro energético, reducir los tiempos muertos... Tradicionalmente es una técnica que se hacía incluso en lápiz y papel, pero dado el incremento del volumen de producción, los diversos tipos de productos que se producen y el aumento de la fabricación flexible utilizando lotes cortos hace que sea difícil hacer un *scheduling* óptimo sin una buena optimización.

3.1.2 Encaje en la industria

El estándar de ISA-S95 ofrece un prototipo de modelo jerárquico tradicional para la organización del sistema de fabricación de una empresa como el que se muestra en la figura 3.1. Este es un esquema más detallado que el mostrado anteriormente en la figura 1.1 donde se establece que existen 5 niveles. El superior es el 4, llamado también ERP (Enterprise Resource Planning) que se encarga de planificar la producción básica de la planta, ver el material necesario y tener en cuenta las entregas y envíos desde una perspectiva global de la empresa. Trabaja a plazos relativamente largos, haciendo sus predicciones para asegurar el máximo beneficio al conjunto.

El 3 son los sistemas MES (Manufacturing Execution System) que trabajan a partir de las planificaciones y la asignación de recursos, materia prima... del nivel superior. Controla el trabajo, las características del producto y optimiza el proceso de producción. Se mide en tiempos más cortos que el nivel 4 como días, turnos, horas, minutos e incluso segundos. Se comunica con el 4 pero también con los de abajo.

Los niveles inferiores incluyen el 2, 1 y 0. En orden de arriba a abajo estaría el nivel responsable de monitorizar, supervisar y controlar el proceso a través

de sistemas SCADA/HMI que interactúan directamente con los responsables humanos. El 1, es el encargado de medir y manipular la parte de abajo, a través de PLCs. El nivel más inferior de todos sería el 0, que es donde se encuentra el proceso en sí, este envía la información a los superiores a través de sensores y actúa sobre el mismo con los actuadores.

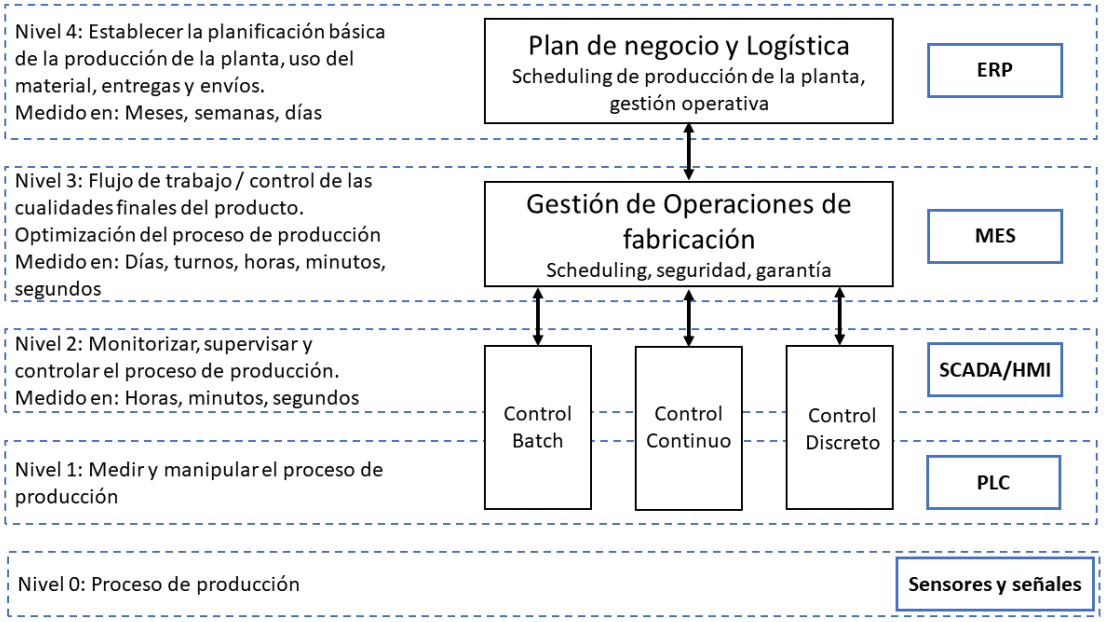

Figura 3.1: Modelo jerárquico asociado a las operaciones de fabricación y sistemas de control y otros sistemas de negocio. Fuente: ISA 2010.

Por tanto, principalmente el *scheduling* es considerada una herramienta de los sistemas MES, es decir, el nivel 3 del modelo anterior.

Los nuevos sistemas de producción flexible y la industria 4.0 hacen que la optimización del sistema sea cada vez más importante y con ello, el *scheduling* puede ser crucial para el aprovechamiento total de las plantas industriales.

3.1.3 Problemas de scheduling

Los problemas de *scheduling* pueden ser muy diversos, dándose en muchos tipos de empresas e instalaciones industriales. En Harjunkoski y col. 2014 exponen que un problema de este tipo debe tener en cuenta 4 cosas.

- El entorno de mercado:

  El mercado delimitará mucho la manera de producción de la empresa. Será importante conocer el volumen y la variabilidad de la demanda ya

que esto afectará a la frecuencia y regularidad con la que se fabrica el producto.

- La interacción con otras funciones de planificación:

  En una cadena de suministro existen muchas otras funciones que poco o nada tienen que ver con el *scheduling*. Estas interaccionarán directamente entre ellas y se deberán tener en cuenta.

- Las instalaciones de producción:

  Son las que tienen que ver con la instalación en sí y la forma de producir. Se dividen en tres clases.

  - El tipo de proceso: *Batch* o continuo.
  - El entorno de producción: Que pueden ser entornos secuenciales donde un proceso *batch* no se mezcle con otro o entornos de red en el que diversos procesos se juntarán con otros para dar otro *batch*.
  - El tipo de operación: Si es una operación de producción (conversión de entradas en salidas) o una transferencia de materia.

- Las características específicas del proceso:

  Los problemas relacionados íntegramente con el proceso en sí, sus restricciones y características.

3.2 Scheduling a corto plazo de procesos batch

Como se ha visto, una de las principales cosas que hay que tener en cuenta es el tipo de proceso que se tiene delante. Distinguir entre uno *batch* o continuo. Los dos se diferencian en la naturaleza de las restricciones de capacidad y el tiempo de procesamiento y la manera en que eso afecta al nivel de inventario. En el primero la capacidad es un límite de la cantidad procesada (kg) y el tiempo de procesamiento es fijo. En el continuo las capacidades se refieren a tasas de procesamiento (kg/h) y el tiempo no es fijo.

En el caso del trabajo se tiene un proceso *batch* en el que los lotes se deben procesar en los esterilizadores.

El mayor problema no es identificar el tipo de *scheduling* en sí, si no la manera de optimizarlo. Esto tendrá una gran repercusión en el coste computacional. El trabajo hecho por Méndez y col. 2006 propone 4 principales aspectos que hay que tener en cuenta a la hora de crear el modelo de optimización.

- Las decisiones de optimización:
  - Dimensionamiento de los lotes.
  - Dónde colocar los lotes.
  - El secuenciamiento.
  - La sincronización.
- Los elementos de modelado: Basados en materiales (STN o RTN) o basados en lotes o *batch*.
- La representación del tiempo (discreta o continua):
  - Los basados en la cuadrícula del tiempo: Representación discreta, cuadrícula común continua o cuadrícula continua específica para cada unidad.
  - Los basados en precedencia: inmediata o general.
- La Función objetivo que se vaya a utilizar: *Makespan* (tiempo desde el inicio al final), tardanza, coste...

En la figura 3.2 se puede ver un resumen de los diferentes tipos de representación temporal que existen.

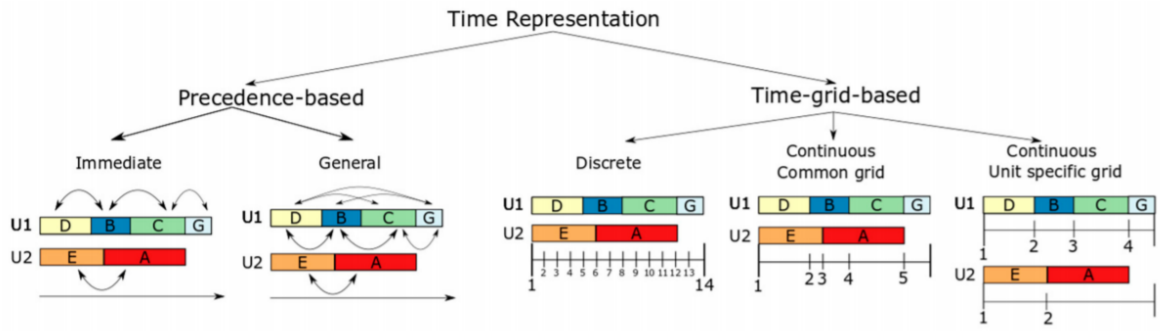

Figura 3.2: Clases de representación por eventos. Fuente: G. P. Georgiadis, Elekidis y M. C. Georgiadis 2019

En la figura 3.3 se ve un resumen detallado del tipo de modelo de optimización. Para este trabajo se han usado las técnicas de precedencia, en concreto la precedencia general. Esta contiene variables binarias que definen que un lote i se ha procesado antes o después que uno i' pero no necesariamente de manera inmediata, también variables que indican a que recurso se ha asignado cada uno de los lotes.

Characteristic	Discrete time models	Continuous time Models					
Event representation	Global time intervals	Global time points	Unit-specific time events	Time slots*	Unit-specific immediate precedence*	Immediate precedence*	General precedence*
Main decisions	Lot-sizing, allocation, sequencing, timing				Allocation, sequencing, timing		
Key discrete variables	W_{ijt} defines if task I starts in unit j at the beginning of time interval t.	Ws_{in} / Wf_{in} define if task i starts/ends at time point n. $W_{inn'}$ defines if task i starts at time point n and ends at time point n'.	Ws_{in} /W_{in} / Wf_{in} define if task i starts/is active/ends at event point n.	W_{ijk} define if unit j starts task i at the beginning of time slot k.	$X_{ii'j}$ defines if batch i is processed right before of batch i' in unit j. XF_{ij} defines if batch i starts the processing sequence of unit j.	$X_{ii'}$ defines if batch i is processed right before of batch i'. XF_{ij} / W_{ij} defines if batch i starts/is assigned to unit j.	$X'_{ii'}$ define if batch i is processed before or after of batch i'. W_{ij} defines if batch i is assigned to unit j
Type of process	General network				Sequential		
Material balances	Network flow equations (STN or RTN)	Network flow equations (STN or RTN)	Network flow equations (STN)		Batch-oriented		
Critical modeling issues	Time interval duration, scheduling period (data dependent)	Number of time points (iteratively estimated)	Number of time events (iteratively estimated)	Number of time slots (estimated)	Number of batch tasks sharing units (lot-sizing) and units	Number of batch tasks sharing units (lot-sizing)	Number of batch tasks sharing resources (lot-sizing)
Critical problem features	Variable processing times, sequence-dependent changeovers	Intermediate due dates and raw-material supplies	Intermediate due dates and raw-material supplies	Resource limitations	Inventory, resource limitations	Inventory, resource limitations	Inventory

Figura 3.3: Características generales de los modelos de optimización. Fuente: Méndez y col. 2006

3.3 Modelado matemático del sistema

Para conseguir hacer el modelo de la planta se debe tener muy en cuenta como es en sí, tanto el proceso como el sistema en general. A partir de todo lo explicado en el capítulo 2 se tiene información suficiente para poder crearlo. Mediante la programación lineal mixta-entera se modela el sistema (Gómez, de Prada y Pitarch 2018) y se intenta buscar la solución de la manera más rápida posible utilizando *solvers* que puedan resolver este tipo de problemas.

El sistema se hace utilizando predicados lógicos al igual que G. Palacín, Riquelme y de Prada 2019 traduciéndolos a un modelo matemático para facilitar su implementación en un lenguaje de programación. Para ello se

utiliza la técnica del Big_M explicada y extraída de Winston y Goldberg 2004.

El primer paso para elaborar el modelo es definir los diferentes conjuntos que van a aparecer.

Conjuntos

- Carros: I
- Grupos de carros a esterilizar (*Slots*): J
- Autoclaves: U

En este apartado se desarrollan dos modelos matemáticos diferentes, aunque el segundo es en parte evolución del primero. Ambos tendrán en cuenta la restricción de la subsección 2.2.3 de que hay un tiempo máximo de espera antes de entrar al autoclave. Pero el segundo se centrará en intentar solucionar el problema que se puede ocasionar cuando dos esterilizadores intenten calentarse al mismo tiempo, donde es probable que la falta de vapor de suministro provoque no llegar a la temperatura de consigna en el tiempo planificado.

Ante una tasa de llegada de carros desde la actualidad hasta un momento futuro, se utiliza también un horizonte robusto en el cual se asegura que los carros que llegan entre el instante actual y un determinado momento sean asignados a un slot. El resto de carros desde el valor del horizonte hasta el final podrán ser incluidos si el optimizador lo considera oportuno (G. Palacín, Riquelme y de Prada 2019). El objetivo es tener un *scheduling* de un corto periodo de tiempo que vaya ejecutándose al pasar unos determinados minutos y por ello, no es necesario que los carros llegados después estén asignados. De manera que, la optimización siempre tenga una buena solución y pueda tener en cuenta si existe algún problema en la planta como un fallo de uno de los esterilizadores, una tasa de llegada de carros diferentes...

3.3.1 Modelo de precedencia general

Para elaborar el modelo se utiliza la precedencia general. El primer paso después de definir los conjuntos es el de definir las variables del problema.

Variables

- X(bin): Variable de tipo binaria que relaciona un carro con un slot. Siendo uno cuando el carro i está asignado al slot j.

$$X_{i\in I, j\in J} \tag{3.1}$$

- Z(bin): Variable binaria que relaciona un slot con un determinado autoclave. Esta vale uno cuando el slot j está asignado al autoclave u.

$$Z_{j\in J, u\in U} \tag{3.2}$$

- Y(bin): Relaciona la precedencia entre slots. Es binaria y vale uno si el slot j va delante del slot j'.

$$Y_{j\in J, j'\in J} \tag{3.3}$$

- $te_{j\in J}$: Tiempo con el que inicia cada slot j el proceso de esterilización.
- $tf_{j\in J}$: Momento en el que un determinado slot j termina el proceso.
- $ts_{i\in I}$: Instante de tiempo en el que se sella un carro i y entra en espera para ser asignado a un slot y posteriormente a un autoclave.
- MK: Makespan, tiempo total que pasa desde el inicio del proceso hasta el final.

Dentro del modelo estarán también los parámetros. Estos tendrán un valor asignado y dependerá del proceso y el sistema que se tenga.

Parámetros

- max_c: Máximo número de carros en un slot. Depende de la capacidad de cada autoclave.
- τ: Tiempo de espera máxima para cada carro, que será diferente dependiendo del alimento o el tipo de lata.
- t_{he}: Duración estándar de la fase de calentamiento, es decir, tiempo que pasa desde que se inicia el esterilizado hasta que se llega a la temperatura de mantenimiento. Tendrá que ver con el perfil térmico que se requiera.
- t_{mc}: Duración de la fase de mantenimiento y enfriamiento, después de que termine el calentamiento. Dependerá al igual que el anterior del perfil térmico.
- h_r: Valor del horizonte robusto, que obliga a que los carros i llegados hasta ese momento sean asignados a slots j.

Un ejemplo del resultado de un diagrama de Gantt con las variables que se han explicado sería como el de la figura 3.4.

Mediante límites y predicados lógicos traducidos a lenguaje matemático se establecen las restricciones del sistema. Estas son las que explican e incluyen su comportamiento, decirle de alguna manera al optimizador lo que debe ocurrir para que este organice las tareas de la mejor manera posible.

Restricciones

- Carros: Un carro i solo puede aparecer en un slot j. Es decir, no puede ser asignado a 2 grupos de carros, es indivisible y únicamente puede estar en un slot.

$$\sum_{j \in J} X_{i,j} \leq 1 \quad \forall i \in I \tag{3.4}$$

- Slots: Los slots j tienen que tener como mínimo un carro y un máximo de max_c. Por ello, para cada j, la suma de todos los carros asignados

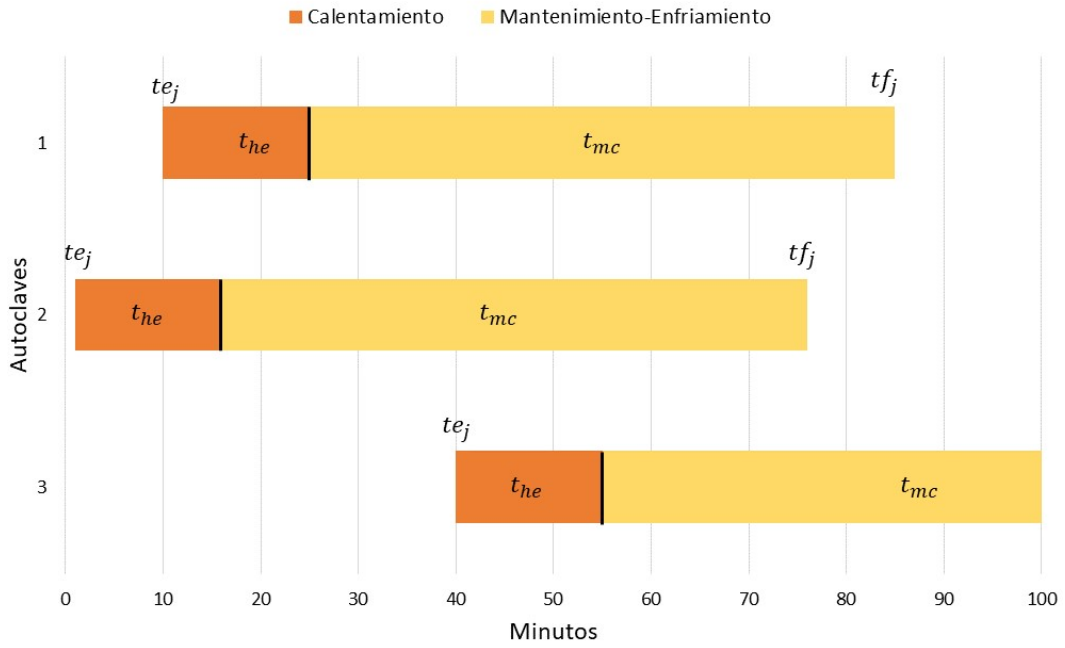

Figura 3.4: Ejemplo de diagrama de Gantt de 3 autoclaves.

debe ser mayor que uno para que se introduzca en el autoclave y menor que la capacidad máxima del esterilizador en este caso.

$$\sum_{i \in I} X_{i,j} \leq max_c \quad \forall j \in J \tag{3.5}$$

$$\sum_{i \in I} X_{i,j} \geq 1 \quad \forall j \in J \tag{3.6}$$

- Autoclave: Un slot j solo puede ir asignado a un esterilizador u. Al igual que en la restricción de los carros es inconcebible que una unidad pueda ser distribuida en dos autoclaves.

$$\sum_{u \in U} Z_{j,u} = 1 \quad \forall j \in J \tag{3.7}$$

- Esterilización: Para el proceso de esterilizar se deben definir algunos predicados. El primero es que el tiempo de inicio de esterilización te_j de un slot j que contenga un determinado carro i, ha de ser mayor que el momento después de que ese carro haya sido sellado y esté a la espera de ser asignado (ecuación 3.8). Pero ese carro que haya entrado nunca

ha de sobrepasar el tiempo máximo de espera τ, por lo que te_j debe ser menor que la suma de ts_i y τ (ecuación 3.9). Por otro lado, se tiene que indicar si un carro que acaba de entrar y no está asignado a ningún slot será porque lo ha hecho después de sobrepasar el horizonte robusto (ecuación 3.10). Por último, de indicará al optimizador que el tiempo final de un slot j (tf_j) será la suma del tiempo de inicio de esterilizado del mismo (te_j) más el calentamiento (t_{he}) y la fase de mantenimiento-enfriamiento (t_{mc}) (ecuación 3.11).

$$te_j \geq ts_i - Big_M(1 - X_{i,j}) \quad \forall i \in I, \forall j \in J \tag{3.8}$$

$$te_j \leq ts_i + \tau + Big_M(1 - X_{i,j}) \quad \forall i \in I, \forall j \in J \tag{3.9}$$

$$ts_i \geq h_r - Big_M(\sum_{j \in J} X_{i,j}) \quad \forall i \in I \tag{3.10}$$

$$tf_j = te_j + t_{he} + t_{mc} \quad \forall j \in J \tag{3.11}$$

- Precedencia: Con ello se define cómo interactúan los grupos de carros entre ellos después de que se cree el primero. En caso de que un slot j preceda a j' y se les haya asignado el mismo autoclave, j' deberá esperar a empezar su esterilización hasta que termine el anterior (ecuación 3.12). Y viceversa, si es j' el que precede a j tiene que esperar (ecuación 3.13).

$$\begin{gathered} te_{j'} \geq tf_j - Big_M(1 - Y_{j,j'}) - Big_M(2 - (Z_{j,u} + Z_{j',u})) \\ \forall j, j' \in J, \forall u \in U \end{gathered} \tag{3.12}$$

$$\begin{gathered} te_j \geq tf_{j'} - Big_M(Y_{j,j'}) - Big_M(2 - (Z_{j,u} + Z_{j',u})) \\ \forall j, j' \in J, \forall u \in U \end{gathered} \tag{3.13}$$

- Makespan: Se define el tiempo total del sistema desde el inicio como una variable MK que tiene que ser mayor o igual que el valor del tiempo en el que termina el último slot de ser esterilizado.

$$MK \geq tf_j \quad \forall j \in J \tag{3.14}$$

Una vez planteado el problema de *scheduling* solamente queda definir la función objetivo. Muchas veces es normal que estas funciones tengan en cuenta el beneficio que pueda producir al conjunto de la empresa esta parte de la planta, de manera que se maximice la ganancia económica. Al no poder ver el conjunto de la empresa ni de disponer estos datos se va a plantear una optimización multiobjetivo que maximice la salida de carros (ecuación 3.15) y que tarde el menor tiempo posible reduciendo el *makespan* (ecuación 3.16). Cosa que en condiciones normales debe reducir gastos y aumentar el beneficio. Antes de la minimización del *makespan* se le deberá indicar que el valor máximo de carros debe aumentar o mantenerse para que una optimización no influya con la otra.

Función objetivo

$$maximizar \sum_{i \in I, j \in J} X_{i,j} \tag{3.15}$$

$$minimizar \quad MK \tag{3.16}$$

3.3.2 Modelado del sistema con restricción de vapor

Una de las grandes desventajas que pueden tener estos sistemas es la red de vapor que deben tener. Esta puede ser común, para cada uno o para unos determinados grupos con el consecuente coste que eso puede tener. El problema puede darse como el ejemplo de la figura 3.5 donde las etapas de calentamiento del 2 y del 1 son al mismo tiempo. El hecho de que no se tenga vapor suficiente para calentar dos autoclaves a la vez es algo a tener bastante en cuenta. Por ello, se propone una solución modificando el modelo de precedencia general que ya se tiene añadiendo las variables, los parámetros y las restricciones necesarias.

La estrategia es poder modificar el *scheduling* de manera que tenga en cuenta que si dos o más calentamientos de diferentes esterilizadores coinciden el autoclave tardará más tiempo a llegar a la temperatura establecida por

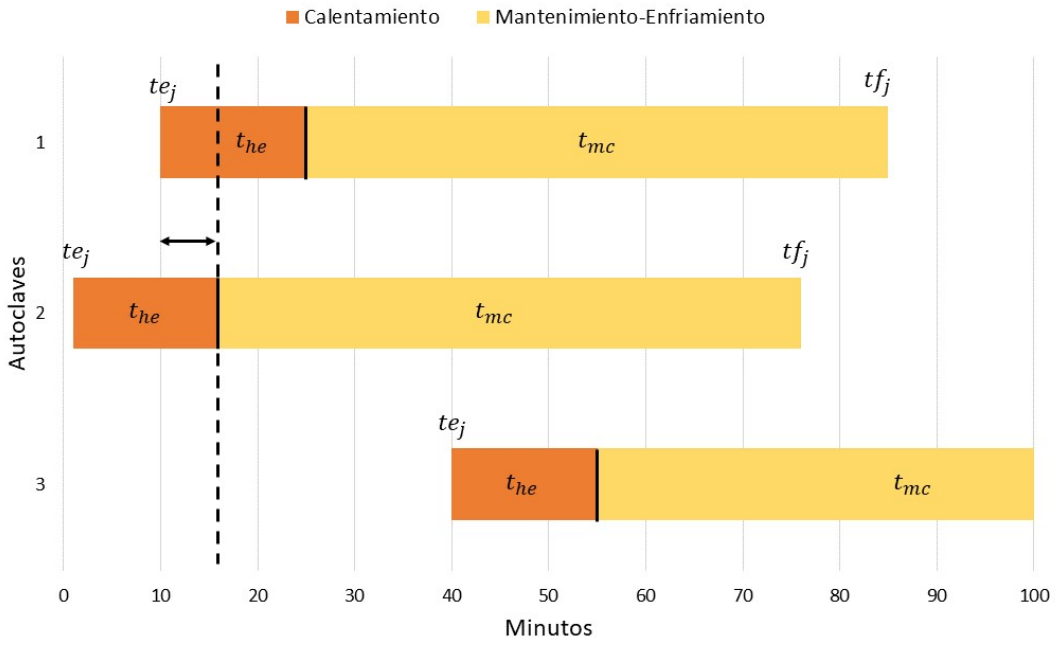

Figura 3.5: Ejemplo de diagrama de Gantt de 3 autoclaves con coincidencia en la fase de calentamiento.

el perfil de temperatura. La solución pasa por incrementar el tiempo de calentamiento para el slot que coincida con su predecesor en esta fase. De manera que si el slot 2 empieza a ser esterilizado y poco tiempo después el slot 3 empieza también, a ambos se les debe sumar un cierto tiempo que asegure que aunque se disponga de menos cantidad de vapor, será suficiente para llegar a la temperatura correcta.

Al sistema anterior se le añaden únicamente dos variables.

Variables

- W(bin): Variable binaria que determina si la fase de calentamiento coincide para dos slots diferentes. Si el slot j y el j' coinciden en su fase de calentamiento temporalmente su valor será uno.

$$W_{j \in J, j' \in J} \tag{3.17}$$

- $th_{j \in J}$: Duración de la fase de calentamiento. En el modelo de antes figuraba solamente con t_{he} pero ahora su valor puede ser diferente en función del grupo de carros j.

El único parámetro diferente es el que se añade del aumento de la fase de calentamiento.

Parámetros

- t_{hp}: Duración añadida a la fase de calentamiento.

Las restricciones son las mismas, pero modificando la definición del tiempo final de esterilizado para cada slot y añadiendo las nuevas de la fase de calentamiento.

Restricciones

- Esterilización: Ahora tf_j pasa a ser la suma del inicio de esterilizado para cada grupo más la suma del tiempo de calentamiento para cada j (th_j) y el tiempo de mantenimiento-enfriamiento.

$$tf_j = te_j + (th_j + t_{mc}) \quad \forall j \in J \tag{3.18}$$

- Restricciones de fase de calentamiento: Con ellas se pretende establecer el comportamiento en el caso que coincidan las fases.

  Una de las cosas que cambian es la definición de la fase de calentamiento, esta será la suma del tiempo de calentamiento estándar establecido t_{he} más el incremento t_{hp} que irá multiplicado por la suma del número de slots que coincidan con j.

  En primer lugar, se tiene las condiciones donde j precede a j'. Si sus fases de calentamiento coinciden se cumplirá que el calentamiento de j va a terminar después de que la esterilización de j' haya empezado (ecuación 3.20). En caso de que no coincidan significará que j' empieza cuando la fase de calentamiento de j ha terminado (ecuación 3.21).

  En segundo lugar, las restricciones donde es j' el que va antes de j serán parecidas a las otras. Si coinciden es j el que empieza esterilizar antes de termine el calentamiento j' (ecuación 3.22), si no que quiere decir que el inicio de la esterilización de j es posterior (ecuación 3.23).

Por último la ecuación 3.24 y la ecuación 3.25 expresan que por un lado, la matriz binaria W tiene el mismo valor si es $W_{j,j'}$ o $W_{j',j}$ y que un slot no puede coincidir en su calentamiento con él mismo.

$$th_j = t_{he} + t_{hp}(\sum_{j' \in J} W_{j,j'}) \quad \forall j \in J \tag{3.19}$$

$$te_j + th_j \geq te_{j'} - Big_M(1 - W_{j,j'}) - Big_M(1 - Y_{j,j'}) \quad \forall j, j' \in J \tag{3.20}$$

$$te_j + th_j \leq te_{j'} + Big_M(W_{j,j'}) + Big_M(1 - Y_{j,j'}) \quad \forall j, j' \in J \tag{3.21}$$

$$te_{j'} + th_{j'} \geq te_j - Big_M(1 - W_{j,j'}) - Big_M(Y_{j,j'}) \quad \forall j, j' \in J \tag{3.22}$$

$$te_{j'} + th_{j'} \leq te_j + Big_M(W_{j,j'}) + Big_M(Y_{j,j'}) \quad \forall j, j' \in J \tag{3.23}$$

$$W_{j,j'} = W_{j',j} \quad \forall j, j' \in J \tag{3.24}$$

$$W_{j,j} = 0 \quad \forall j \in J \tag{3.25}$$

Un ejemplo de lo que va a ocurrir con esto es la figura 3.6 donde se muestra el diagrama de Gantt con la nueva restricción. Para solucionar el problema de que las fases de calentamiento del 1 y del 2 sean coincidentes se le suma un cierto intervalo de tiempo que irá multiplicado por el número de autoclaves que empiecen la esterilización casi al mismo tiempo. En este caso el 2 coincide con el 1 y viceversa, por tanto $t_{hp}(1)$.

3.4 Software utilizado

Para poder hacer el cálculo matemático se necesita un *software* que contenga un entorno de computación numérica como Python, R, Matlab, Mathematica... El requisito es que sea capaz de ejecutar las operaciones matemáticas y mostrar los resultados.

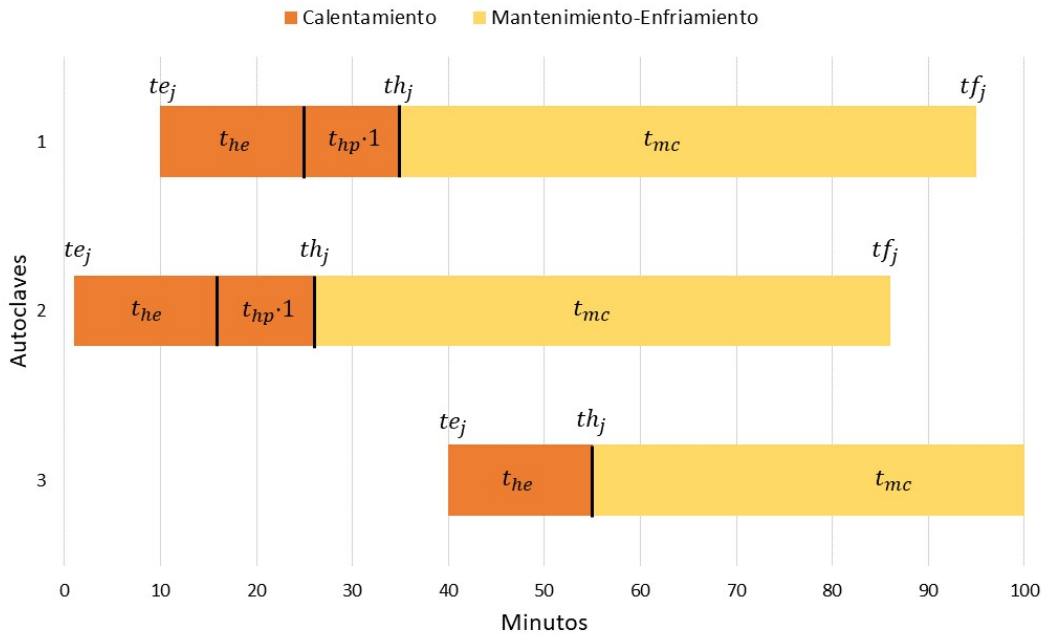

Figura 3.6: Ejemplo de diagrama de Gantt de 3 autoclaves con coincidencia en la fase de calentamiento y solución propuesta.

Los citados utilizan un lenguaje simple de alto nivel sin tener que mencionar y definir el tipo de variable (int, float, double...), se les conoce como lenguajes de tipado dinámico. Lo contrario sería Fortran o C que son de tipado estático. Mientras que los dinámicos ganan en productividad pierden en rendimiento. Los estáticos eran los más utilizados por los investigadores, pero llega un momento en que empieza a ocurrir lo contrario, que perder tanto en productividad no les es conveniente (Bezanson y col. 2017).

Combinando los dos tipos de lenguajes nace Julia language. Una de las cosas más importantes de Julia es que tiene una gran capacidad de computación mediante un sofisticado compilador que le hace reducir el tiempo de cómputo a niveles parecidos a los de los lenguajes estáticos. Aparte, es de código abierto lo que le hace tener una gran cantidad de librerías y foros donde apoyarse y compartir proyectos.

Para hacer cálculos y operaciones complejas, al entorno de programación de Julia se le tienen que añadir paquetes o librerías, que son diseñadas por usuarios o empresas. Estas se deben instalar previamente en el entorno Julia y usarse con un *using ...* al principio del archivo. Para el caso del trabajo se han utilizado las que se citan a continuación.

3.4.1 Librerías utilizadas

JuMP

JuMP es un lenguaje de modelado de dominio específico para la optimización matemática integrado en Julia. Admite varios solucionadores (*solvers*) comerciales y de código abierto para una variedad de clases de problemas, incluida la programación lineal, la programación de enteros mixtos, la programación cónica de segundo orden, la programación semidefinida y la programación no lineal.

Gadfly

Sistema de representación y visualización para gráficos escrito con lenguaje Julia. En este caso usado para mostrar los resultados finales.

DataFrames

Al igual que el anterior escrito con julia, es un sistema de representación y visualización, pero ahora de los datos en formato tablas.

JSON

Un fichero JSON es simplemente un fichero de texto sencillo destinado al intercambio de datos. Esta librería permite crear estructuras de texto, manejarlas y escribir la información en este tipo de archivos.

Dates

Librería de Julia utilizada para manejar fácilmente formatos de fecha, así como para obtener la hora actual, cambiar de formato de tiempo...

3.4.2 Solvers de optimización

A la librería de JuMP se le debe integrar un solver que se encargará de hacer la optimización matemática. En este trabajo se van a usar dos.

CPLEX

Es un solver fabricado y comercializado por IBM de manera gratuita para estudiantes y personal docente, pero de pago para uso comercial. Ofrece bibliotecas C, C ++, Java, .NET y Python que resuelven la programación lineal (LP) y problemas relacionados. Específicamente, resuelve problemas de optimización con restricciones lineales o cuadráticas donde el objetivo a optimizar se puede expresar como una función lineal o una función cuadrática convexa.

La velocidad de cálculo es muy grande por lo que se ha usado principalmente para obtener los resultados del *scheduling* en un tiempo corto.

Cbc

El otro solver utilizado para resolver problemas matemáticos de programación mixta entera es Cbc (Coin-or branch and cut). Como su nombre indica, trabaja a través de un algoritmo llamado *Ramificación y Acotación* que consiste en ir buscando soluciones por diferentes caminos como si fueran las ramas de un árbol y encontrar el valor óptimo a la función objetivo, en caso de no serlo este corta directamente el camino o la rama. A diferencia de CPLEX es totalmente gratuito al ser de código abierto.

Debido a errores que no dependían del trabajo y la facilidad de cálculo requerida se ha utilizado Cbc para la demostración final de la integración de Julia en Simio.

3.5 Resultados

Los archivos utilizados se encuentran en formato Julia (.jl) en el apéndice A para que se puedan usar de manera simple. Todos ellos se han simulado con un ordenador MSI con procesador Intel Core I5-10210U con 16GB de memoria RAM.

Para la obtención de los resultados primero se ha traducido todo el modelo matemático a lenguaje Julia y se le han dado valores a los parámetros. Estos no representan lo que pueda pasar en la empresa realmente ya que sus datos de producción no se conocen, únicamente se expone una metodología de cálculo del *scheduling* de producción de la planta de esterilizado.

Para ambos casos se ha definido una llegada de 100 carros en 200 minutos, distribuidos de manera aleatoria entre 0 y 200. Su valor será el mismo para ambas simulaciones. En estos resultados, por lo tanto, no se ha tenido en cuenta el hecho de que puedan haber carros en el sistema, si no que el número inicial es 0 y a partir de ahí van llegando. En caso de querer hacer optimizaciones de *scheduling* de manera iterativa se deberían conocer los carros que no hayan sido esterilizados y cuánto tiempo llevan dentro del sistema cada vez que se ejecute el archivo.

3.5.1 Sistema con precedencia general

Parámetro	**Valor**	**Parámetro**	**Valor**
Big_M	500	max_c	9
τ (min)	30	t_{he} (min)	15
t_{mc} (min)	75	h_r	100
Número Carros (I)	100	Número Slots (J)	7
Número Autoclaves (U)	5		

Tabla 3.1: Valor de los parámetros usados para la optimización

Los valores para el modelo de precedencia general que se han utilizado son los mostrados en la tabla 3.1. Con un tiempo máximo de espera τ exigente que haga que los carros no puedan quedarse mucho tiempo parados. Mientras que para las fases de esterilizado el calentamiento durará 15 minutos y el mantenimiento-enfriamiento 75 minutos. El número de carros será 100, con 7 slots posibles (9 carros como máximo) y 5 esterilizadores disponibles. Por último, el Big_M tendrá un valor suficientemente alto para que cumpla su función y un horizonte robusto de 100 minutos. Todos los carros que entren después de 100 minutos no se van a incluir en el *scheduling* a no ser que el *solver* lo considere necesario.

Maximizando el número de carros esterilizados

Los resultados cuando se maximiza el número de carros esterilizados es el que se muestra en la tabla 3.2 y gráficamente en la figura 3.7. En la tabla destaca el número de slot al que se hace referencia, cuando empieza a ser esterilizado, cuando termina y el autoclave al que ha ido a parar. En la gráfica se ve de manera visual a través de un diagrama de Gantt cómo se ordenan para sacar el máximo número de carros posibles dada la alta llegada de estos a la planta. Siendo la barra negra, el final del calentamiento de cada uno de los autoclaves. Destaca que las parejas de esterilizadores 5-1, 4-3 y 2-1 coinciden en sus fases de calentamiento.

Slot	Tiempo inicio (min)	Tiempo final (min)	Autoclave asignado
1	27.3917	117.392	2
2	182.304	272.304	2
3	63.0606	153.061	1
4	55.0447	145.045	5
5	102.149	192.149	3
6	95.3834	185.383	4
7	192.228	282.228	1

Tabla 3.2: Resultados de la asignación de los diferentes slots a los autoclaves con el momento de inicio y final maximizando el número de carros esterilizados.

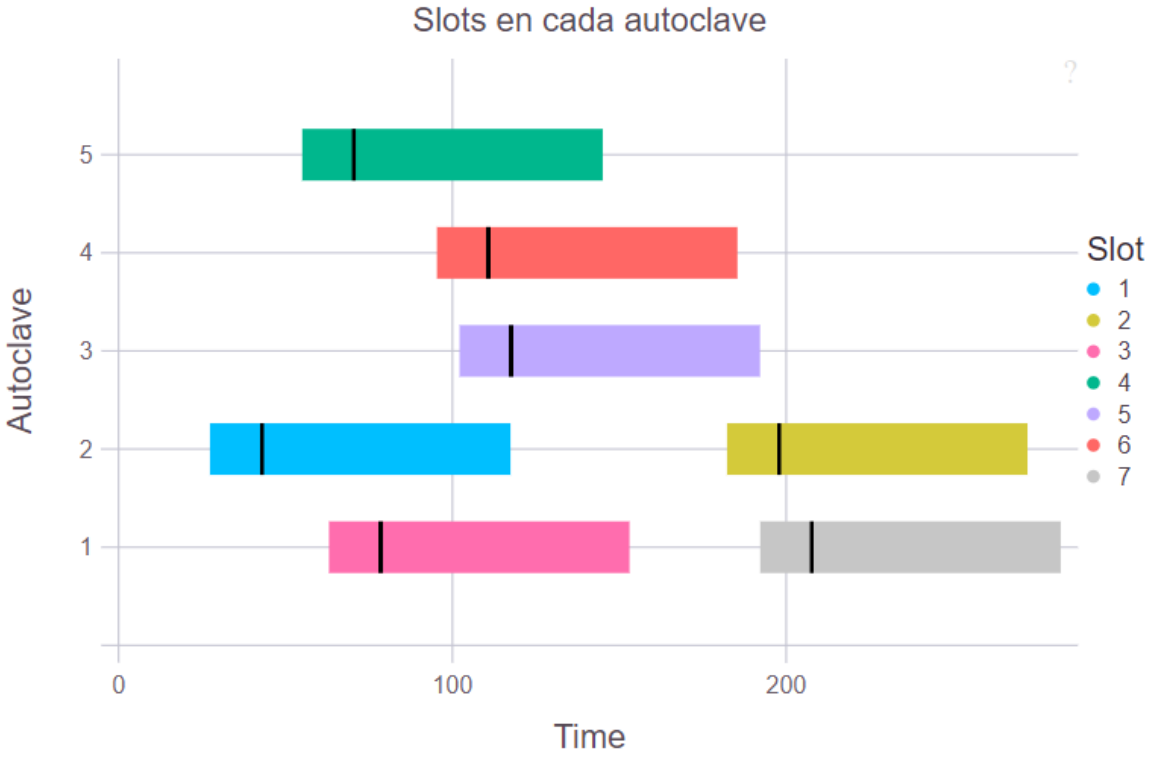

Figura 3.7: Resultados de la asignación de los diferentes slots a los autoclaves en un diagrama de Gantt maximizando el número de carros esterilizados.

Minimizando el makespan

El siguiente paso para una optimización multi-objetivo es el de indicarle al optimizador que tiene que obtener una nueva solución que reduzca el tiempo total de proceso, pero manteniendo el número de carros que ha optimizado anteriormente para que pueda mejorar la solución.

Los resultados son los que se muestran en la tabla 3.3 y en la figura 3.8. En este caso destaca sobre todo que el tiempo total del proceso ha pasado de ser de esta manera de 272.304 a 231.256 minutos. La optimización ha conseguido reordenar la esterilización de los slots para reducir el *makespan*.

Slot	Tiempo inicio (min)	Tiempo final (min)	Autoclave asignado
1	141.256	231.256	1
2	124.621	214.621	2
3	109.817	199.817	3
4	92.1134	182.113	5
5	74.6182	164.618	4
6	44.6895	134.69	1
7	29.2301	119.23	2

Tabla 3.3: Resultados de la asignación de los diferentes slots a los autoclaves con el momento de inicio y final minimizando el *makespan*.

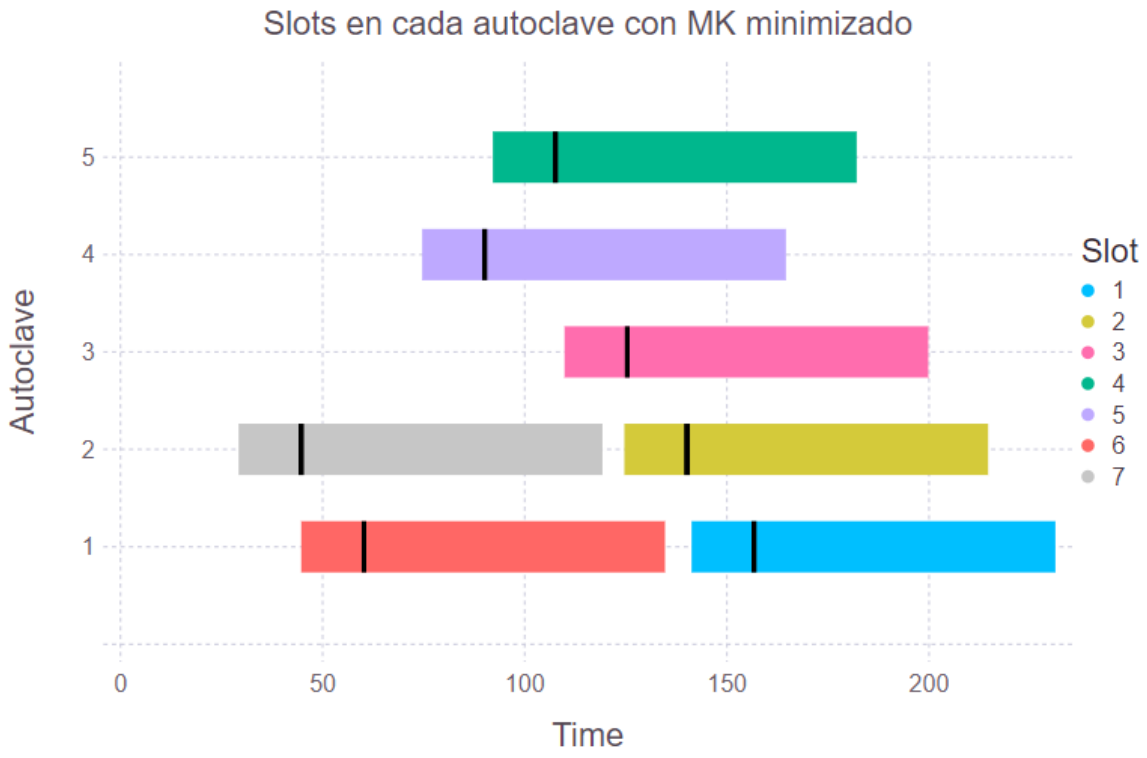

Figura 3.8: Resultados de la asignación de los diferentes slots a los autoclaves en un diagrama de Gantt minimizando el *makespan*.

Finalmente se muestra también cómo ha quedado repartido la unidad más pequeña del modelo, los carros. En la tabla 3.4 se ve el número de carro, cuando ha pasado de la zona de envasado, cuanto tiempo ha estado en espera, que slot lo contiene y el autoclave al que han ido, junto con la información anterior de cuando empieza y termina de esterilizarse. Como caso particular los carros que entran fuera del horizonte robusto no se adjudican a ningún slot a no ser que la solución lo haya considerado oportuno.

La gran utilidad de este tipo de optimizaciones es utilizarlas de manera iterativa cada cortos periodos de tiempo para tener en cuenta los posibles cambios que haya habido en el sistema. El horizonte robusto juega un papel importante al indicar que todos los carros por debajo de su valor deben ser adjudicados a un slot y los que no, es el *solver* el que lo debe de decidir. Si la idea es que se ejecute una optimización cada 15, 30, 45 minutos por ejemplo, estos carros no tendrán importancia ya que el sistema no tendrá las mismas condiciones pasados unos minutos.

En la tabla de resultados se puede ver cómo algunos de los carros tienen una espera mayor a 30, simplemente es por el hecho de que no han sido asignados y por lo tanto en vista a 200 minutos en adelante van a ser dejados en el sistema.

3.5.2 Sistema con restricción de vapor

La optimización en este caso se ha hecho con los valores de la tabla 3.5. La única diferencia es la aparición de t_{hp}, la cual suma 15 minutos de calentamiento a cada esterilizador en caso de que exista coincidencia de las dos fases.

Maximizando el número de carros esterilizados

Los resultados al maximizar la salida de carros del sistema son los que se ven en la tabla 3.6 y la figura 3.9. Destaca cómo la restricción ha funcionado ya que ningún esterilizador coincide con otro en su calentamiento.

Carros	Entrada	Espera	Slot	Autoclave	Inicio	Final
1	162.228	62.2276	0	0	0.0	0.0
2	59.5551	15.0632	5	4	74.6182	164.618
3	73.4625	18.6509	4	5	92.1134	182.113
4	110.051	14.5703	2	2	124.621	214.621
5	186.199	86.1986	0	0	0.0	0.0
6	35.1838	9.50567	6	1	44.6895	134.69
7	178.503	78.5028	0	0	0.0	0.0
8	80.9827	28.8338	3	3	109.817	199.817
9	184.754	84.7537	0	0	0.0	0.0
10	50.4719	24.1463	5	4	74.6182	164.618
...	...	...	...	...	...	...
90	71.5776	20.5358	4	5	92.1134	182.113
91	149.036	49.0363	0	0	0.0	0.0
92	86.9872	22.8293	3	3	109.817	199.817
93	66.0584	8.55979	5	4	74.6182	164.618
94	173.373	73.3727	0	0	0.0	0.0
95	172.789	72.7893	0	0	0.0	0.0
96	92.1134	-2.84217e-14	4	5	92.1134	182.113
97	181.111	81.1108	0	0	0.0	0.0
98	172.406	72.4057	0	0	0.0	0.0
99	9.72954	19.5006	7	2	29.2301	119.23
100	45.3842	29.234	5	4	74.6182	164.618

Tabla 3.4: Resultados de la asignación de los carros con toda la información para el sistema.

Parámetro	Valor	Parámetro	Valor
Big_M	500	max_c	9
τ (min)	30	t_{he} (min)	15
t_{mc} (min)	75	t_{hp}	15
h_r	100	Número Carros (I)	100
Número Slots (J)	7	Número de autoclaves (U)	5

Tabla 3.5: Valor de los parámetros usados para la optimización con restricción en el calentamiento.

Slot	Inicio (min)	Calentamiento (min)	Final (min)	Autoclave
1	200.0	215.0	290.0	1
2	89.068	104.068	179.068	3
3	50.0849	65.0849	140.085	2
4	104.618	119.618	194.618	5
5	71.4796	86.4796	161.48	4
6	27.3917	42.3917	117.392	1
7	179.099	194.099	269.099	2

Tabla 3.6: Resultados de la asignación de los diferentes slots a los autoclaves con el momento de inicio y final maximizando el número de carros esterilizados con la restricción en el calentamiento.

Minimizando el makespan

Cuando se reduce el tiempo total del proceso para todo el sistema este pasa de valer 290 minutos a 231.256. El ahorro de tiempo es considerable, hacien-

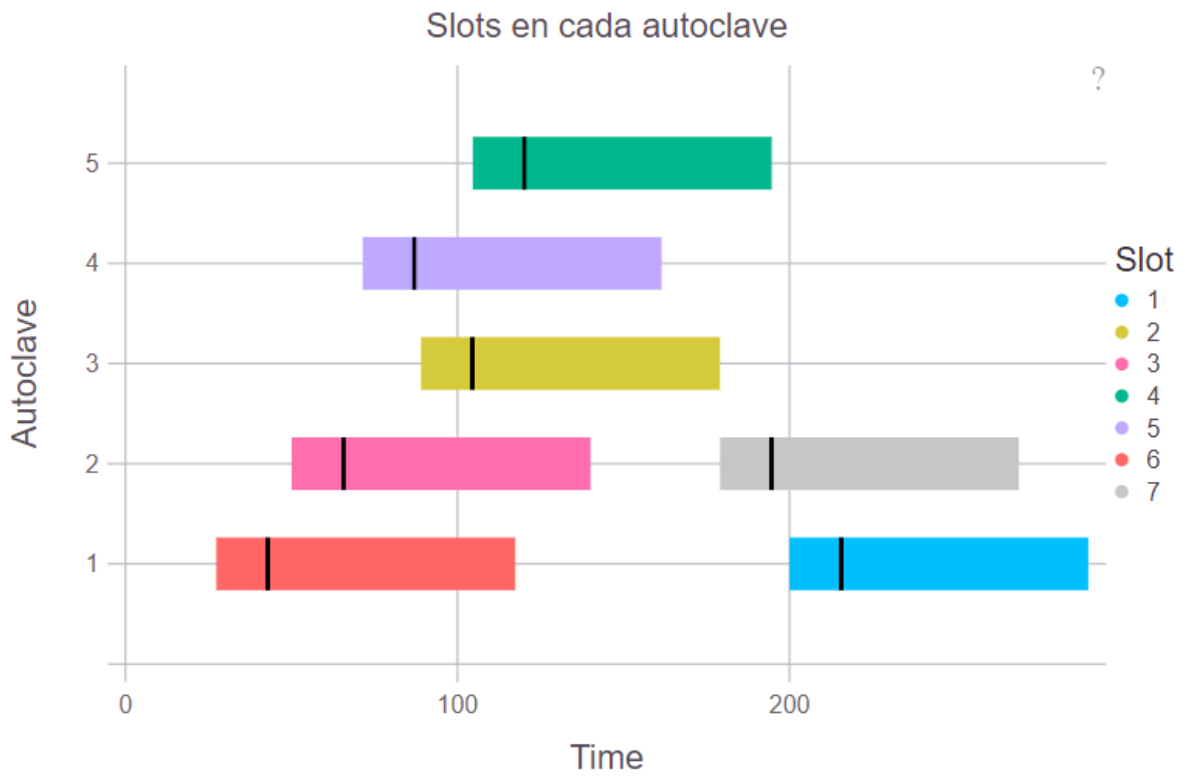

Figura 3.9: Resultados de la asignación de los diferentes slots a los autoclaves en un diagrama de Gantt maximizando el número de carros con la restricción en el calentamiento.

do que a la vez se extraigan el máximo número de carros. Los resultados se muestran para este caso en la tabla 3.7 y la figura 3.10. Igual que en el caso anterior la optimización ha decido que, para cumplir con las dos funciones no le es rentable tener una penalización de 15 minutos más de calentamiento y lo que hace es ajustar todos los esterilizadores para que no ocurra. Por ejemplo, el 3 empieza a calentar a 126.256 min, el 2 15 minutos justo después y el 1 otros 15 minutos.

Slot	Inicio (min)	Calentamiento (min)	Final (min)	Autoclave
1	141.256	156.256	231.256	1
2	126.256	141.256	216.256	2
3	111.256	126.256	201.256	4
4	94.6212	109.621	184.621	5
5	65.1838	80.1838	155.184	3
6	50.0849	65.0849	140.085	1
7	27.3917	42.3917	117.392	2

Tabla 3.7: Resultados de la asignación de los diferentes slots a los autoclaves con el momento de inicio y final minimizando el *makespan* con la restricción en el calentamiento.

Al igual que en caso anterior también se extrae una tabla para conocer que va a pasar con los carros como se puede ver en la tabla 3.8. En ella se tie-

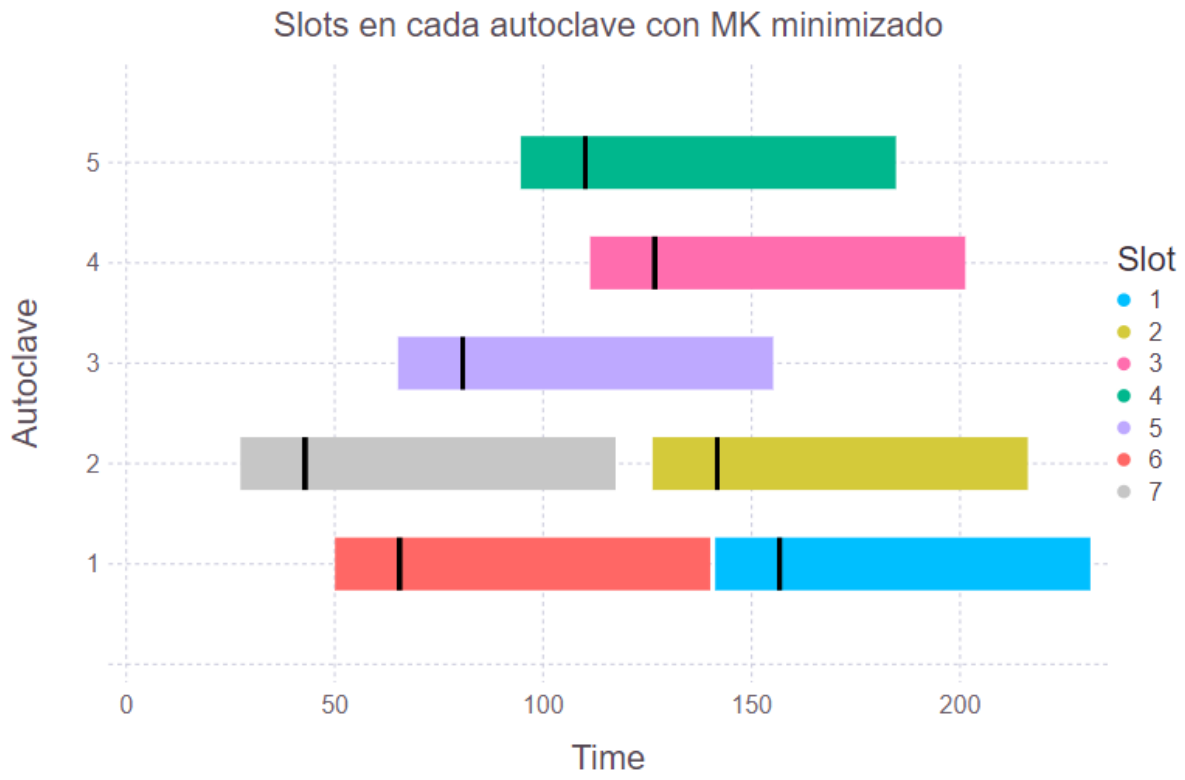

Figura 3.10: Resultados de la asignación de los diferentes slots a los autoclaves en un diagrama de Gantt minimizando el *makespan* con la restricción en el calentamiento.

ne toda la información necesaria sobre los 100 carros que van a entrar al sistema, cuáles van a ser asignados al entrar antes del valor del horizonte robusto y cuáles no. Como novedad también se detalla la duración del calentamiento que va a tener cada carro, para que en el caso de utilizarse se sepa rápidamente cual tiene un calentamiento mayor de 30.

Carro	**Entrada**	**Espera**	**Slot**	**Auto.**	**Inicio**	**Dur.cal.**	**Fin cal.**	**Final**
1	162.228	62.2276	0	0	0.0	0.0	0.0	0.0
2	59.5551	5.62877	5	3	65.1838	15.0	80.1838	155.184
3	73.4625	21.1587	4	5	94.6212	15.0	109.621	184.621
4	110.051	16.2047	2	2	126.256	15.0	141.256	216.256
5	186.199	86.1986	0	0	0.0	0.0	0.0	0.0
6	35.1838	30.0	5	3	65.1838	15.0	80.1838	155.184
7	178.503	78.5028	0	0	0.0	0.0	0.0	0.0
8	80.9827	13.6385	4	5	94.6212	15.0	109.621	184.621
9	184.754	84.7537	0	0	0.0	0.0	0.0	0.0
10	50.4719	14.712	5	3	65.1838	15.0	80.1838	155.184
...	...	...	...	...	...	...	...	...
90	71.5776	23.0436	4	5	94.6212	15.0	109.621	184.621
91	149.036	49.0363	0	0	0.0	0.0	0.0	0.0
92	86.9872	24.2684	3	4	111.256	15.0	126.256	201.256
93	66.0584	28.5628	4	5	94.6212	15.0	109.621	184.621
94	173.373	73.3727	0	0	0.0	0.0	0.0	0.0
95	172.789	72.7893	0	0	0.0	0.0	0.0	0.0
96	92.1134	19.1422	3	4	111.256	15.0	126.256	201.256
97	181.111	81.1108	0	0	0.0	0.0	0.0	0.0
98	172.406	72.4057	0	0	0.0	0.0	0.0	0.0
99	9.72954	17.6622	7	2	27.3917	15.0	42.3917	117.392
100	45.3842	19.7997	5	3	65.1838	15.0	80.1838	155.184

Tabla 3.8: Resultados de la asignación de los carros con toda la información para el sistema con restricción en el calentamiento.

4 Simulación de la planta industrial

Con este capítulo se pretende explicar la utilidad de las simulaciones en el mundo industrial, conocer cómo funciona el simulador elegido para el trabajo y qué tiene de particular. El objetivo final será el de explicar el proceso para construir una simulación que se adapte a la planta real de esterilizado y ver cómo a partir de ella, se pueden tomar decisiones.

4.1 ¿Qué es la simulación?

La simulación consiste en imitar dentro de lo posible un sistema físico junto con su comportamiento. Este es uno de los instrumentos más importantes que pueden proporcionar las nuevas tecnologías a las empresas.

Se define como una posible herramienta para poder disminuir el riesgo que puede conllevar cierta acción como un cambio en una linea de producción o ver cómo de rentable puede ser hacer una determinada inversión. En un entorno industrial las decisiones que se toman pueden ser cruciales o tener una gran repercusión económica por lo que es importante disponer de un buen modelo de simulación que lo tenga en cuenta.

Los sectores industriales que pueden hacer uso de este tipo de tecnologías son muy diversos, ya sea en procesos productivos, logística o incluso empresas del sector servicios.

Dentro de los modelos de simulación se pueden dividir al igual que los procesos de manera simple en dos tipos. Los continuos donde los estados y las variables que delimitan un sistema cambian continuamente en el tiempo y los discretos donde las propiedades únicamente cambian en un determinado

instante. Un ejemplo de los procesos continuos son la simulación de gran parte de sistemas de la industria química como el comportamiento de un reactor a lo largo del tiempo. En cambio, un ejemplo de modelo discreto sería un sistema de fabricación como el del trabajo donde los eventos se producen y terminan en unos determinados instantes de tiempo.

Los *softwares* de simulación son muy variados y existen en multitud de industrias y sectores. La elección de estos dependerá principalmente del tipo de sistema que se quiera simular y de la complejidad del mismo.

4.2 Simio

Simio se define a sí mismo como un entorno de modelado de simulación basado en objetos inteligentes. (**sim**ulation **i**ntelligent **o**bjects). Para explicar cómo funciona Simio internamente se puede utilizar las actas de los congresos donde ha participado la empresa como Prochaska y Thiesing 2016.

En el trabajo se va a utilizar la versión disponible en los ordenadores de la universidad, *Simio University Design Edition*.

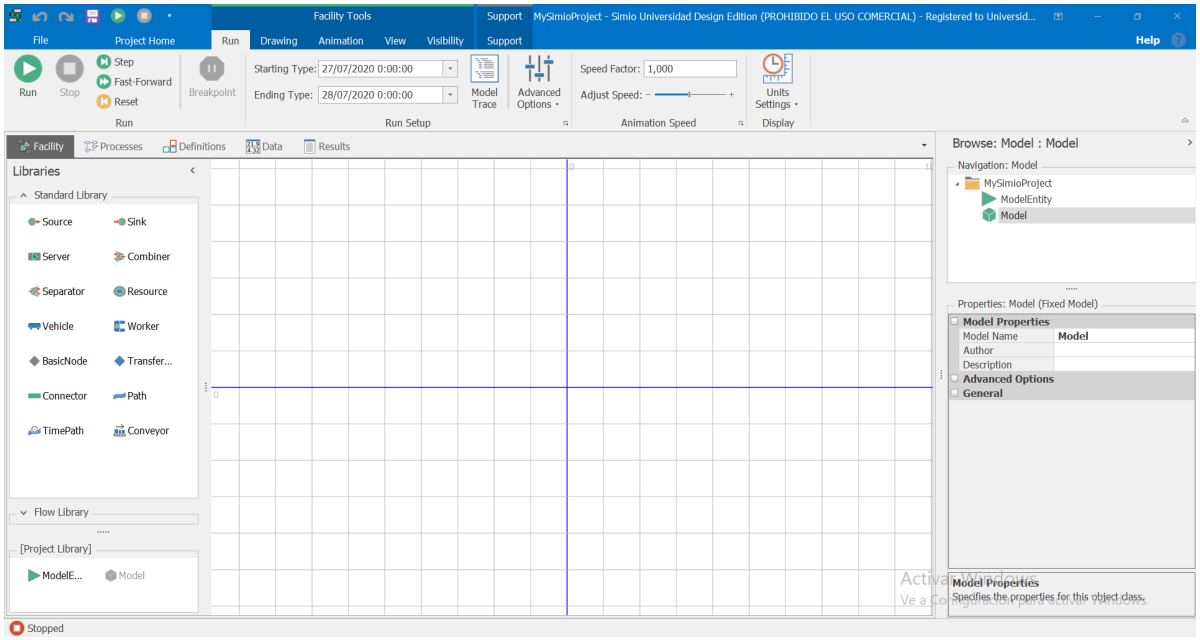

Figura 4.1: Vista de la interfaz de Simio.

4.2.1 Utilidad

Simio tiene mucha utilidad al igual que los *sotfwares* de simulación de eventos discretos y manufactura en general. Según la información del mismo es indicado para las siguientes situaciones.

- Sistemas con gran complejidad o que necesitan hacer determinadas pruebas que puedan tener una importante repercusión económica.
- Sistemas donde sea determinante predecir los distintos escenarios y la variabilidad que pueda ocurrir.
- Sistemas en los que no se tenga muchos datos y se pueda simular en base a su comportamiento.
- En caso de que la presentación sea un factor importante. Contiene vista 3D, en lineas generales se puede considerar un programa muy estético que puede ayudar a plasmar y comunicar ideas.

4.2.2 Scheduling en Simio

Simio tiene su particular manera de analizar las técnicas de *scheduling*. Desde la empresa se hace mucho hincapié en que la planificación y este concepto son diferentes pero complementarios.

Para ellos la planificación es programar en el tiempo el trabajo a realizar y que este tenga las materias primas suficientes, es decir, es un trabajo del nivel superior de la pirámide, concretamente del ERP. Sin embargo, el *scheduling* es el siguiente paso después de la planificación, el trabajo de desglosar las tareas, organizarlas y asignarles unos recursos. Por ello, este concepto entraría más dentro de los sistemas MES.

Tal y como explica Pegden y Thiesing 2015, el *scheduling* se puede abordar como un problema de optimización con un modelo matemático formado por unas ecuaciones que son resueltas por un *solver*. Estos tienen el problema de que suelen ser bastante complejos y por lo tanto se necesita de alguien con conocimientos suficientes, tanto de este tipo de técnica como del proceso que tiene delante. En ellos lo que se quiere es el resultado.

Por otro lado, la simulación sirve para poder ver cómo se va a comportar ante un determinado cambio a lo largo del tiempo. En la simulación el objetivo

está en plasmar la lógica o el comportamiento del sistema que se quiere representar adecuadamente. Para Simio el *scheduling* en una simulación se resume en 2 decisiones críticas.

- La asignación de recursos. Cuando una tarea se va a producir en un servidor se tiene que definir correctamente para que no se produzca en otros sitios.
- La selección del trabajo. Los recursos deben tener capacidad para decidir qué trabajos se tienen que realizar primero.

Simio se puede utilizar para poder simular un sistema en casi todas sus versiones. Con ellas se puede aplicar una determinada lógica para plasmar el sistema que se quiera y experimentar su respuesta ante un cambio. El *scheduling* en sí se obtiene como resultado de su comportamiento a lo largo de la simulación y la opción de poder hacer planificaciones previas que sean las que delimiten su comportamiento están reservadas para otras versiones como *Enterprise Edition* diferentes a la utilizada en el trabajo.

4.2.3 La definición de objeto en Simio

Se puede decir que los objetos son la base del funcionamiento de Simio. Estos son guardados en librerías y pueden ser fácilmente compartidos. Un objeto puede ser una máquina, un avión, un robot o cualquier cosa que se requiera representar en el sistema. Además, son fácilmente personalizables ya que se pueden representar con cualquier forma en 3D.

La filosofía de Simio se basa en que no hay diferencia entre un objeto y un modelo. Un sistema de fabricación con diversos robots y una cinta forma el modelo de una célula de trabajo, pero esta pasa a ser un objeto (objeto compuesto) que puede estar dentro de otros modelos. La actividad de construir un objeto es idéntica a la de hacer un modelo.

Los lenguajes de programación orientados a objetos (OOP) como C++, C# o Java se basan en unos principios claros comunes entre ellos. El *software* se construye como una colección de objetos cooperantes que se instancian a partir de clases (*class*). Aunque Simio, está programado en C#, es más un entorno gráfico de modelado con los principios de un lenguaje de programación orientado a objetos. Por tanto, la capacidad de poder añadir nuevos

objetos al *software* es más de alguien con conocimientos de modelado que de programación.

Para construirlos se pueden utilizar diferentes técnicas.

- Obtener el objeto requerido a través de objetos compuestos, como el ejemplo de la cinta y los robots.
- Construir su proceso lógico que alteran su estado en relación a eventos. Tal y como se haría en otro tipo de simuladores (Arena o GPSS).
- Se utilizan los ya creados y se modifican hasta tener el comportamiento que se quiere.

Simio dispone de 6 tipos de objetos básicos con los que empezar a trabajar y que se pueden modificar gracias a las propiedades y estados que se pueden definir en su interior, estas se pueden ver en la figura 4.2. En la figura 4.3 se pueden ver parte de algunos objetos que se tienen en el panel de librerías de Simio y que se detallan a continuación. Las definiciones se han obtenido con la ayuda de Vieira y col. 2014.

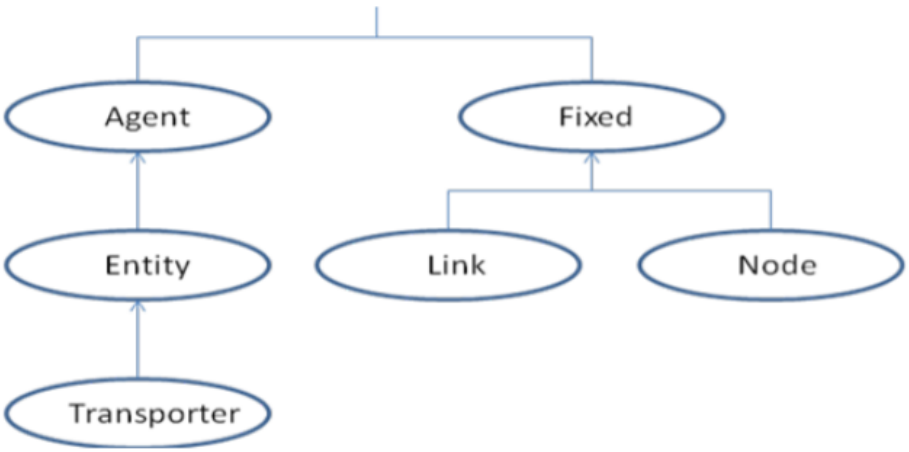

Figura 4.2: Clases de objetos básicos en Simio. Fuente: Prochaska y Thiesing 2016.

Fixed object (Objetos Fijos):

Son aquellos que se colocan en una localización fija, se representan con ellos equipos estacionarios. Clase formada por 6 objetos básicos.

- Source (Fuente): Objeto encargado de crear las entidades. Formado por un nodo de salida, la cola del búfer de salida y el objeto en sí. Se le debe

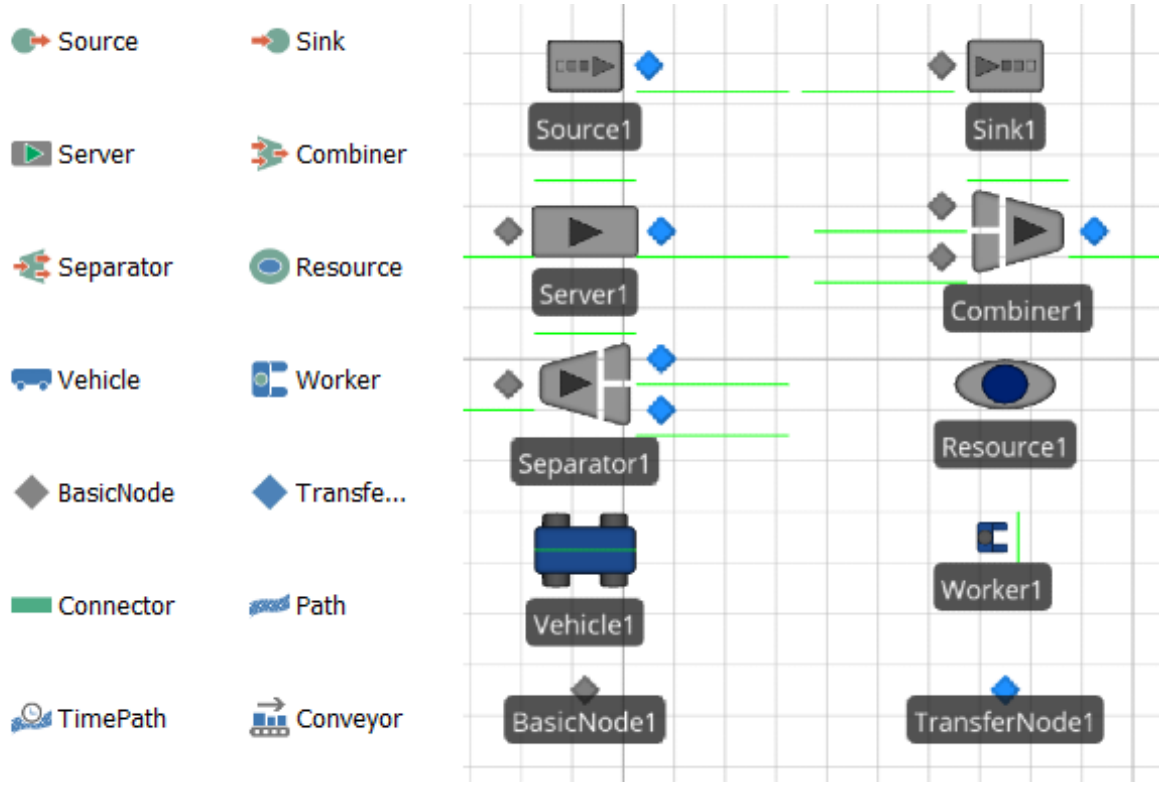

Figura 4.3: Objetos de Simio y representación básica en el mapa

especificar el tipo que se va a crear y toda la información relativa (cuándo crea la primera, frecuencia, número de entidades simultáneas...).

- Sink (Sumidero): Elimina las entidades del sistema. Formado por un nodo de entrada, la cola del búfer de entrada y el objeto.

- Server (Servidor): A diferencia de los anteriores este tiene 2 nodos, uno de salida y otro de entrada, dos colas de búfer y el objeto. Su función sería la de ser una unidad de procesamiento, a la que se le puede indicar información del proceso y de cómo manejar la entidad de entrada (tiempo, capacidad...).

- Combiner (Combinador): Tiene la función de agrupar varias entidades en un lote o *batch* y después de procesarlas asocia los miembros a una única entidad. Formada por 3 nodos, 2 de entrada y uno de salida, 3 colas y el objeto.

- Separator (Separador): Su función es la de separar las diversas entidades que se hayan agrupado en una o incluso es capaz de hacer copias de una misma entidad. Se puede utilizar para deshacer la agrupación del *combiner*.

- Resource (Recurso): Se trata de un objeto genérico con capacidad. A diferencia de todos los otros objetos, las entidades no pasan a través de él.

Agents (Agentes):

Clase de objetos que se mueven por el espacio tridimensional del *softwa-re*. Usados para hacer modelos basados en agentes, utilizados para simular interacciones entre objetos inteligentes. Por ejemplo, los usuarios de un ae-ropuerto.

Entity (Entidades):

Subclase de agentes que se mueven de un objeto a otro a través de una red de enlaces y nodos. Como puede ser un trabajador que lleva las piezas de una máquina a otra, un robot móvil...

Link and Node (Enlace y Nodo):

Encargados de construir la red por la que se pueda mover las entidades. Existen 2 tipos de nodos y 2 tipos de enlaces.

- BasicNode y TransferNode: Mientras que el *BasicNode* es una simple intersección entre diferentes caminos el *TransferNode* es más complejo capaz de hacer cambios en el destino de las entidades.

- Connector y Path: El primero es un camino simple, sirve para definir rutas de duración 0 y se les puede asignar un peso. Mientras que el segundo es bastante más complejo ya que se le pueden asignar velocida-des, pesos o decidir que entidad tiene prioridad a la hora de salir. Existe una versión a medio camino entre ambas llamada *TravelPath* en la que aparte de las funcionalidades del *Connector* se le puede especificar el tiempo de ruta.

Transporter (Transportador):

Aquellos objetos que tienen capacidad para recoger, transportar y dejar una o varias entidades a la vez. Hay de 2 tipos.

- Vehicle (Vehículo): Utilizado para modelar dispositivos que siguen una ruta. Un operario que transporta una determinada unidad, un taxi, un camión, un robot AGV. Se puede definir con las características de un vehículo en la realidad, asignarlo a determinados puntos, darle una prioridad a una acción...
- Worker (Trabajador): Es un recurso móvil que se puede utilizar para transportar entidades entre nodos. La diferencia con el vehículo es que este trabaja siempre bajo demanda y el otro lo hace siguiendo siempre una misma ruta.

4.2.4 Los procesos en Simio

Tal y como se ha explicado, se parte de la base de que los objetos básicos son el inicio para la construcción de objetos de otros niveles. La manera de convertirlos en inteligentes es mediante el apartado *processes* (procesos). Estos definen de manera lógica como va a responder el objeto ante un determinado evento.

Un proceso está constituido a través del secuenciamiento de los *steps* o pasos. Su funcionamiento es sencillo, un determinado evento hace activar un *token* que es el que recorre toda la secuencia lógica.

De esta manera en caso de tener un profundo conocimiento del modelo que se quiere representar se podrá traducir a una secuencia lógica e imitar su comportamiento en la realidad de manera exacta.

La cantidad de *steps* que proporciona Simio es bastante variada, desde retrasos en el tiempo, esperas hasta que ocurra cierto evento, asignar nuevos valores, hasta incluso terminar la simulación.

Por ejemplo, en el caso de tener un robot al que le lleguen 2 tipos de piezas, este deberá decidir qué hacer con ellas. En caso de ser de tipo 1, que aumente su contador y sea procesada de una determinada manera, si es de tipo 2 simplemente se cuenta y se deja pasar. Para hacer esto en Simio se pueden

utilizar objetos básicos como una fuente que cree las piezas (entidades), un servidor (robot)... Cuando una entidad entre en el servidor o en un nodo, se lanza un proceso como el de la figura 4.4. El *token* empieza a recorrerlo, se decide qué entidad es la que ha provocado el evento mediante un paso *Decide*, si el tipo es 1 se aumenta el número del contador al valor anterior +1 con *Assign* y se ejecuta otro proceso a través de *Execute*. Si es de tipo 2 solamente se contabiliza.

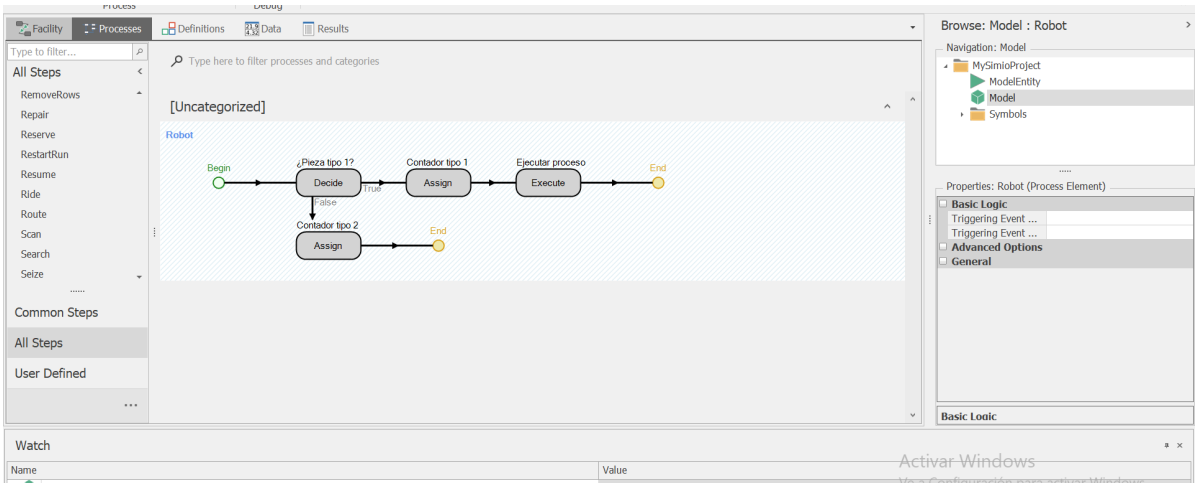

Figura 4.4: Ejemplo de un proceso lógico en Simio

4.3 Construcción de la simulación

El primer paso para poder construir la simulación es tener un conocimiento adecuado de cómo funciona para luego poder plasmar todo su comportamiento en el *software*.

4.3.1 Descripción del comportamiento del sistema

A través de todo lo explicado en el capítulo 2 se tiene un determinado comportamiento que hay que representar.

Este consiste en una llegada continua de carros que se tienen que agrupar para poder introducirse dentro de los esterilizadores.

Para desarrollar la simulación de una manera un poco más simple que en la realidad se va a utilizar una determinada tasa de llegada de carros. La idea es que se forme un grupo cuando se alcance un número exacto y este salga a uno de los esterilizadores. En caso contrario, se quiere que se queden esperando

hasta que formen un grupo suficientemente grande. Las características del sistema que se va a plasmar son las siguientes.

- 2 tipos de carros: Tipo 1 y Tipo 2.
  - Tipo 1: Llegará un carro de tipo 1 cada 13.33 minutos de manera que se forme un slot cada 40. Su tiempo de procesamiento será de 65 minutos.
  - Tipo 2: Los carros de tipo 2 tendrán la misma frecuencia formándose un slot cada 40 pero con 20 min de diferencia. Su procesamiento tendrá un valor de 75 minutos.
- El slot estará formado por 3 carros. Cuando se llegue a esa cantidad se transportarán a uno de los servidores que esté libre.

4.3.2 Creación de la simulación

Simio tiene varios apartados principales para trabajar. El primero es *Facility (Instalaciones)* donde está la representación gráfica de los objetos y con ello sus propiedades básicas. Como se ha comentado existe también el apartado *Processes (Procesos)* desde donde se elabora el comportamiento lógico del sistema. Pero además, hay otros llamados *Definitions (Definiciones)*, *Data (Datos)* y *Results (Resultados)*. Las dos últimas sirven para construir tablas, manejar datos y ver resultados. Por lo tanto a la hora de elaborar un modelo de simulación en sí interesa saber cómo configurar las 3 primeras.

Facility

En la pestaña *Facility* se va a construir el sistema de manera sencilla y representarlo gráficamente. Para ello se van a utilizar los siguientes objetos básicos de Simio, quedándose el sistema de la manera que se ve en la figura 4.5.

- Entidades: Son los carros que se van a mover por el sistema. Para este caso en la simulación se han utilizado 2 tipos (E1 y E2) que representarían dos grupos de latas diferentes.

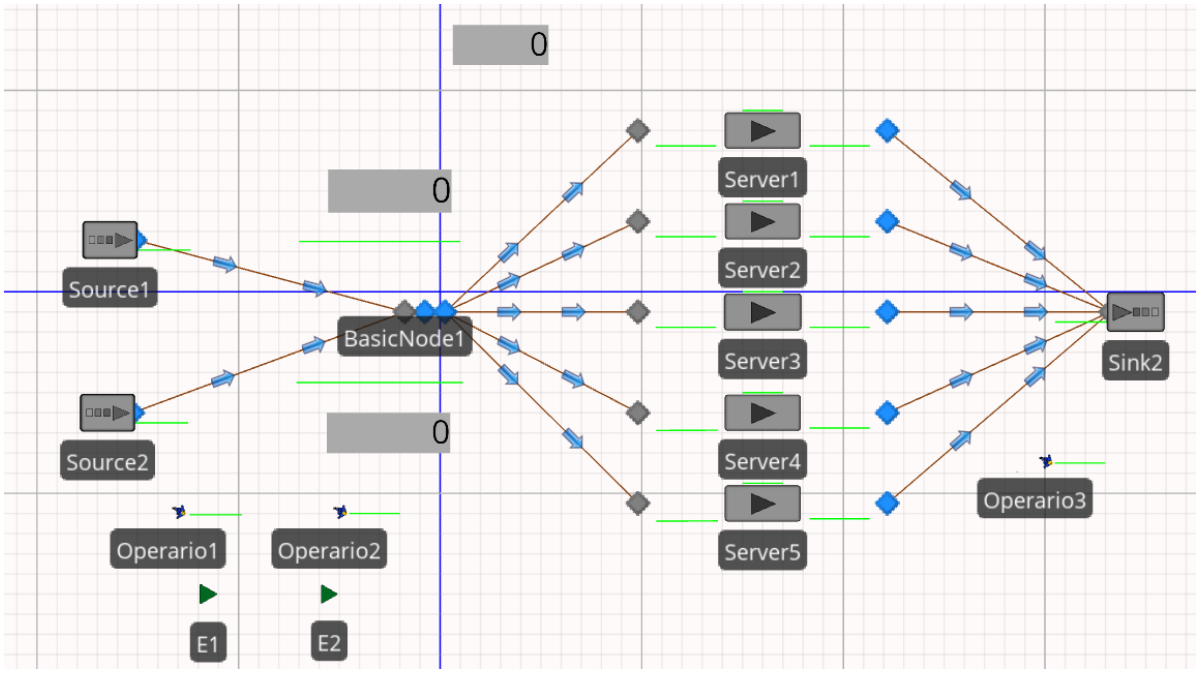

Figura 4.5: Vista de general de los objetos utilizados en la construcción de la simulación.

- Fuentes: Concretamente 2. Crearán las entidades a la frecuencia de tiempo que se le indique. Simulan la llegada de carros que tendría la planta en la realidad, cada una de las fuentes introducirá un tipo de entidad.

- Nodos: Aparte de los propios nodos de los objetos, se utilizan 3, estos servirán para construir una zona de espera donde agrupar los carros en slots antes de introducirlos en los esterilizadores. El primero es un nodo básico (intersección básica) donde entran las entidades provenientes de la fuente. El segundo es un nodo de transferencia o *TransferNode* en el que se le añaden los procesos que definen el comportamiento del sistema. El último será de transferencia también, comunica directamente la zona de espera con los servidores.

- Servidores: Al ser unidades de procesamiento hacen la función de esterilizadores, con un tiempo asignado dependiendo de si el slot es E1 o E2. También se les define una capacidad máxima que es el número de carros límite y que la capacidad de la cola de entrada es 0, ya que la zona de espera se encuentra en los 3 nodos y no a la entrada del autoclave.

- Sumidero: Destruirá las entidades cuando salen del sistema.

- Vehículos: Existen 3 y tendrán la función de hacer de operarios para llevar los carros de las fuentes a la zona de espera, a los esterilizadores y

al sumidero. *Operario1* y *Operario2* están encargados de toda la zona previa a los autoclaves, cuando una entidad requiera ser desplazada entre fuente - zona de espera o zona de espera - esterilizadores acudirá uno de los dos dependiendo la disponibilidad o el que más cerca esté. El tercero traslada los grupos de carros cuando salen de los autoclaves a la zona de almacén (Sumidero).

Definitions

En el apartado de definiciones se pueden crear elementos, propiedades, variables estado, eventos, funciones, listas... Entre ellas destacan los estados, eventos, funciones y listas. Los primeros son variables de la simulación que pueden cambiar de valor con el tiempo.

Para poder plasmar el comportamiento del sistema en procesos va a ser necesarios crear 5 variables estado que se muestran en la figura 4.6.

- *var_e*1: Contendrá el valor del número de carros que están esperando del tipo 1.
- *var_e*2: Valor del número de carros que están esperando del tipo 2.
- *carros*: Variable que indica el número de carros que conforma un slot, en este caso será 3.
- *Tipo*: Indica de qué tipo es la entidad, si 1 o 2.
- *time_server*: Indica el tiempo de procesamiento que va a tener la entidad.

Será importante también crear un evento.

- *mandar_senal*: Evento que indica a la simulación que el grupo de carros está listo para salir.

Aparte de estos, se va a utilizar una lista para manejar distintos vehículos y funciones para hacer cualquier cálculo que se requiera.

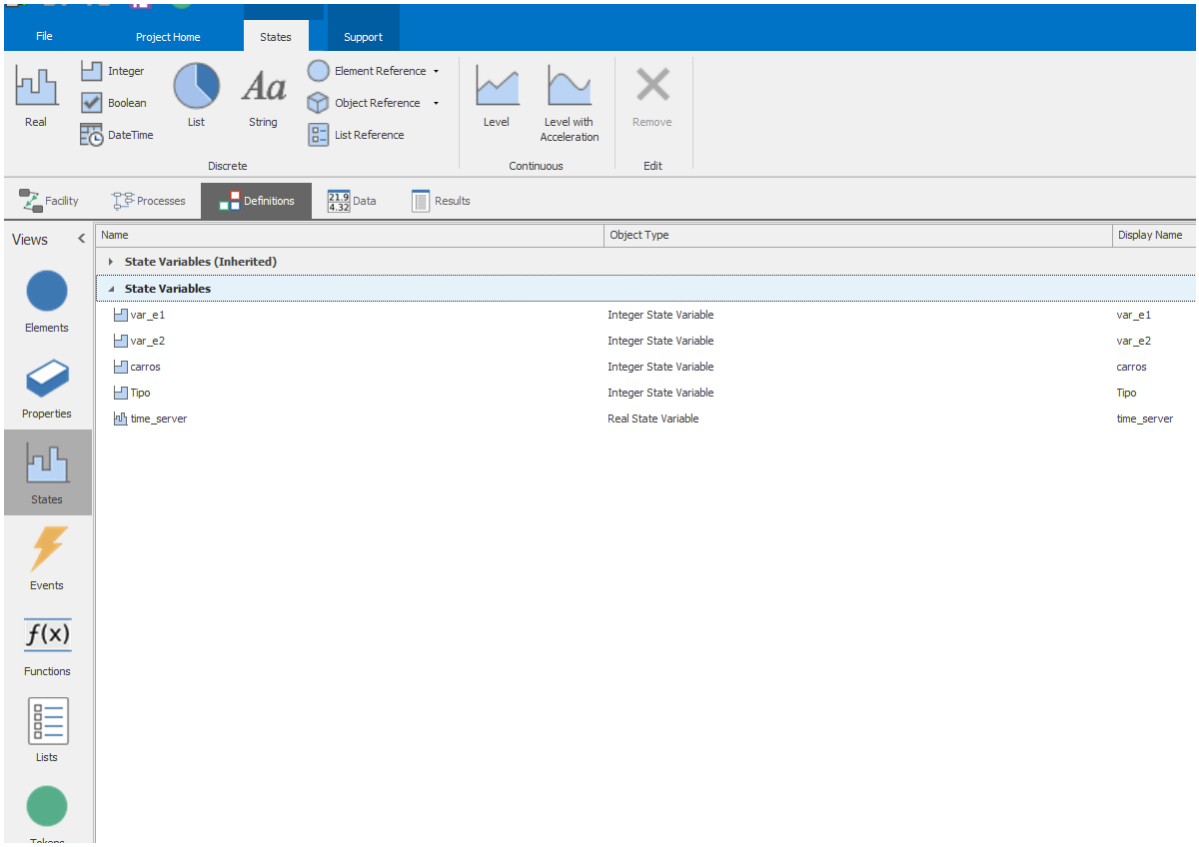

Figura 4.6: Vista de las variables estado utilizadas en la simulación.

Processes

Una vez definidos los elementos del sistema, queda proporcionar a la planta unas reglas que delimiten su comportamiento a lo que es el proceso real. En este sentido cada vez que un carro entre en el sistema debe ser llevado a la zona de espera, en caso de que el número de carros de ese tipo sea igual al definido en la variable *carros* se llevan a los esterilizadores, en caso contrario se dejan en cola, esto se muestra de manera esquematizada en la figura 4.7.

Para traducir esto a un lenguaje lógico dentro de Simio se utilizan tres procesos que se pueden ver en la figura 4.8. El primero de ellos (*TransferNode1_Entered*) será el que se active cuando una entidad entre al segundo nodo de la zona de espera. Cuando este termine y se vaya a producir la salida de alguna entidad, se lanzará *TransferNode1_Exited*. El tercero *Duracion_servidor* se ejecutará cuando un grupo de carros haya entrado en el servidor y estén a punto de ser introducidos en un autoclave, de esta manera dependiendo del tipo se le asignará a ese servidor un tiempo de procesamiento determinado.

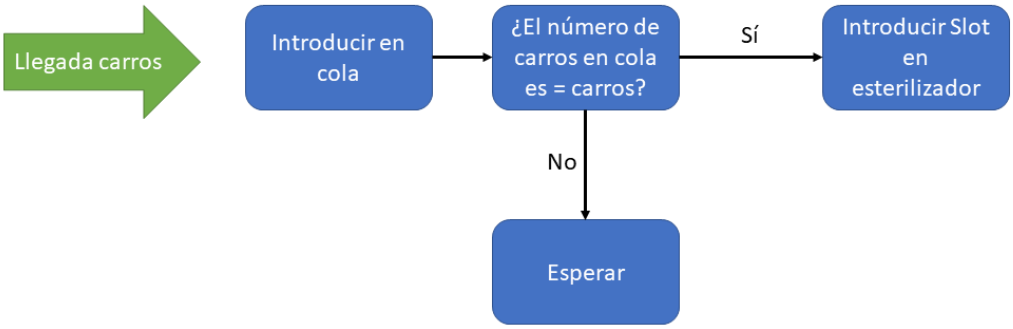

Figura 4.7: Explicación del proceso a implementar en Simio.

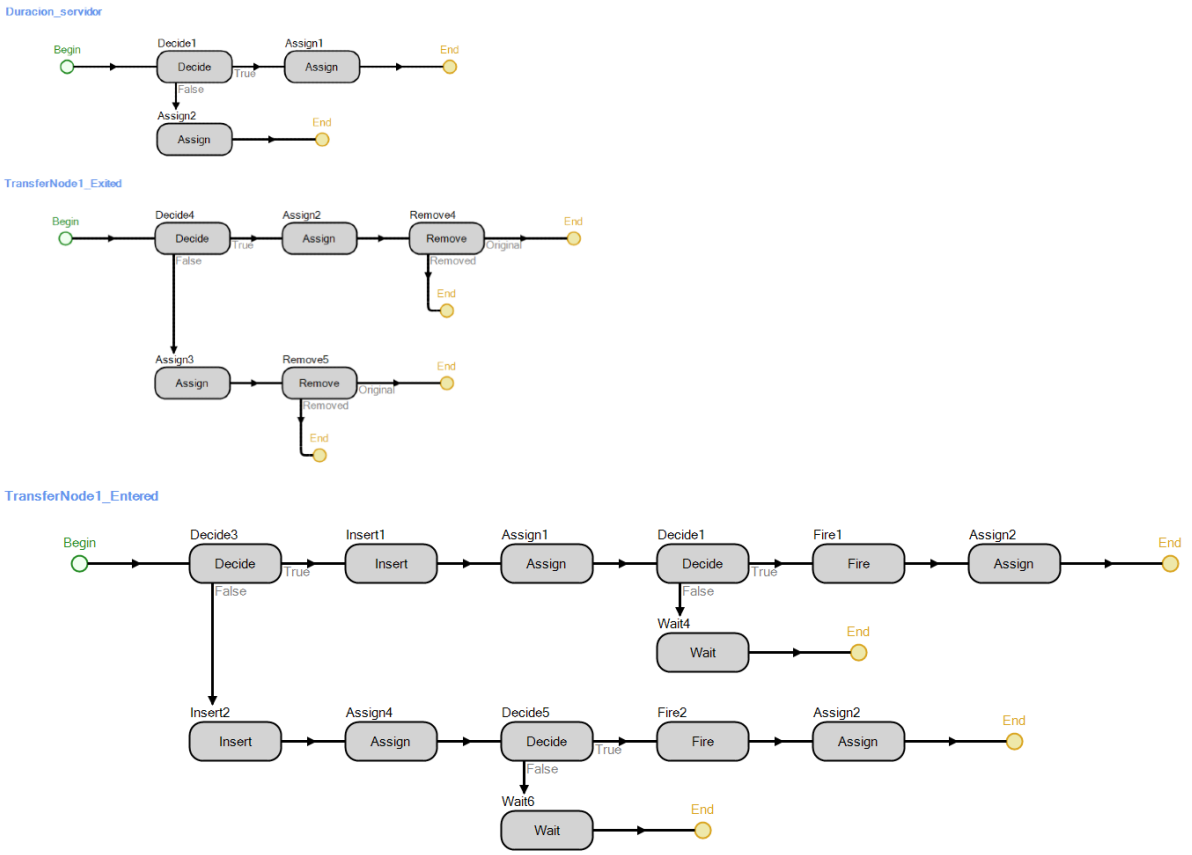

Figura 4.8: Vista de general de los procesos utilizados en la definición del comportamiento de la simulación.

- *TransferNode1_Entered*: Cuando una entidad active el evento comienza a circular el *token*. El primer paso será ver qué tipo ha producido el evento mediante un *Decide* que tendrá una condición lógica del tipo: $Entity.EntityType == E1$. Si es del tipo 1 seguirá el camino de arriba (*True*) y si es del 2 el de abajo (*False*), ambos pasarán por un *Insert* que introduce la entidad en la cola correspondiente. Para llevar la cuenta

de cuántas hay en la cola se utiliza como contadores las variables de estado var_e1 o var_e2, el bloque *Assign* suma una unidad y con el *Decide* posterior se comprueba si el número que hay en la cola es igual a la variable *carros* con la condición $var\_e1 + 1 < carros$. En caso de ser falsa no se hace nada, el sistema espera y se introduce en una cola. Si es efectiva el *token* lanza un evento (*mandar_senal*). Si este evento se produce, se indica al sistema que el grupo de carros está listo para salir.

- *TransferNode1_Exited*: Una vez terminado el anterior proceso, empieza el siguiente. Al igual que se aumentaba el valor de los contadores ahora se deberá quitar. Se examina qué entidad es con *Decide*, se le resta el valor al contador determinado mediante *Assign* y se elimina de la cola utilizando *Remove*.

- *Duracion_servidor*: Cuando se hayan producido los anteriores procesos, previamente a ser procesado el grupo de carros se debe asignar el tiempo correspondiente al esterilizador. Un trabajo que en el caso normal haría un operario al ver qué tipo de latas contiene. Aquí se tiene que describir con un proceso lógico. De esta manera antes de ser procesado en el servidor se aplica un *Decide* para ver si se corresponde con las del tipo 1, si es así, se le asigna a la variable estado *time_server* el valor correspondiente para ese tipo. En caso de ser negativo será el valor de las latas del tipo 2.

Una vez construida la simulación, Simio permite utilizar modelos en 3D en formato *.skp* y está conectado con la plataforma 3D Warehouse. Esto proporciona una mejora evidente al hacer mucho más visual el sistema y por tanto ayuda a su presentación. En la figura 4.9 se ve cómo quedaría desde arriba la planta, con la zona de envasado de la que provienen los carros, la zona de espera, los esterilizadores y la zona de descarga de los Slots, donde se almacenan para su posterior empaquetamiento y distribución. En la figura 4.10 se ve la planta desde diferentes vistas.

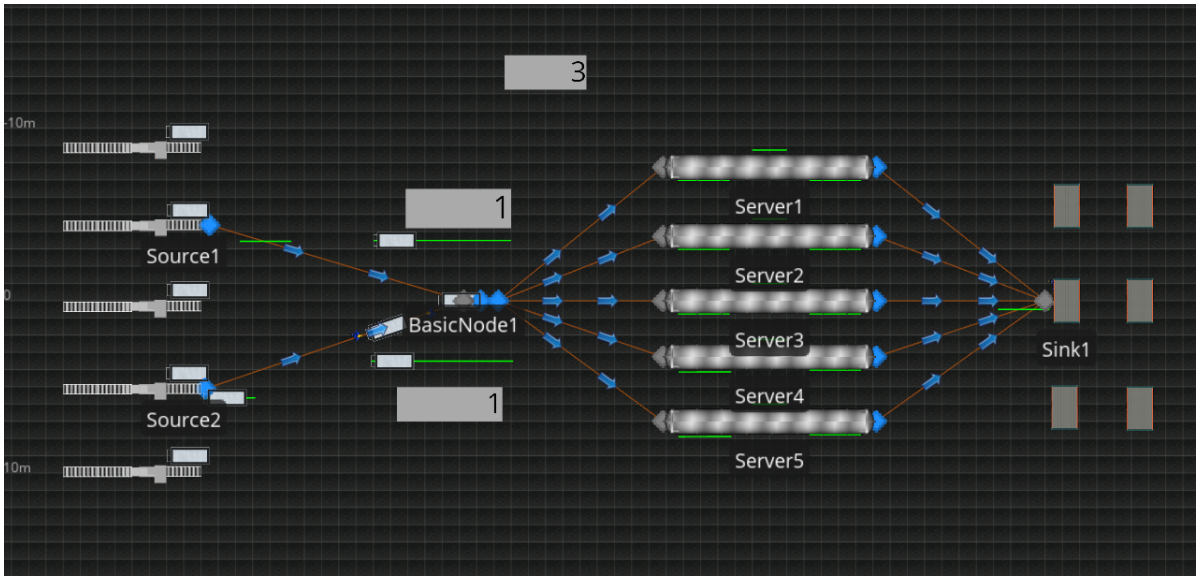

Figura 4.9: Vista de general de los objetos utilizados en la construcción de la simulación decorada.

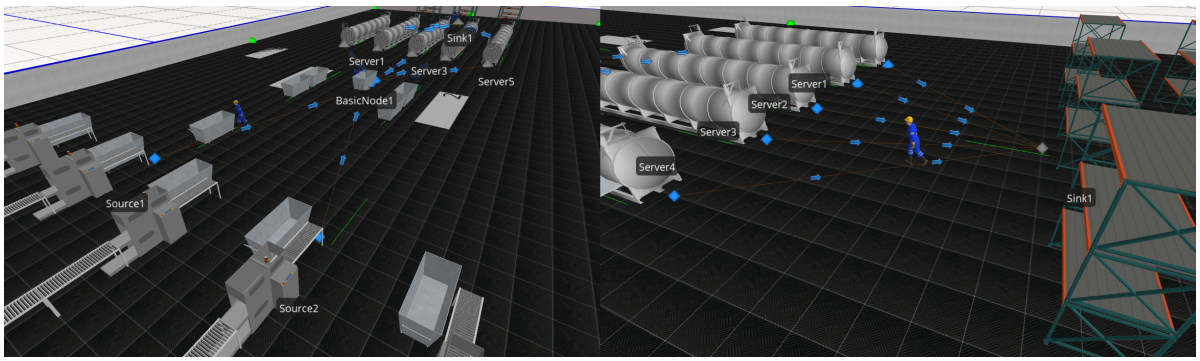

Figura 4.10: Diferentes puntos de visión de la simulación en 3D de la planta de esterilizado.

5 Integración del optimizador en el simulador

Este capítulo pretende explicar cómo se ha integrado el simulador con el *software* de optimización, cuál ha sido la estrategia y la explicación de los archivos realizados.

5.1 Esquema de comunicación propuesto

La idea es poder comunicar Simio con Julia de la manera más efectiva posible y con ello mostrar un ejemplo de cómo se introduciría una optimización de *scheduling* dentro del mismo.

Para ello, se va a utilizar una evolución de la simulación en Simio descrita en la sección 4.3. Para cumplir el objetivo de tener la mejor planificación de tareas posible, Julia será el que le proporcione las órdenes a Simio cada vez que este lo requiera. Por lo tanto, Simio será meramente una plataforma de simulación, pero con la capacidad de tener una inteligencia proporcionada desde un elemento exterior.

Para conseguir esto se propone un esquema como el de la figura 5.1. Ya se ha visto que en la construcción anterior existen dos tipos de entidades, es decir dos tipos de carros diferentes. El propósito será que cada vez que un slot esté preparado para salir de la zona de espera, Simio mande una señal a Julia indicándole el tipo de lata que va a salir y el tiempo de simulación para que este ejecute la optimización. De esta manera la decisión de qué hacer con el grupo de carros dejará de ser del simulador a través de prioridades,

listas o simplemente al que esté libre y pasará a ser una decisión mucho más compleja y meditada en vista a los objetivos globales.

El resultado dará un *scheduling* y el primer esterilizador que se ponga en funcionamiento con el primer lote será el elegido. Julia le transmitirá el número de esterilizador al que tiene que ir.

De esta manera se utilizará un modelo matemático como el desarrollado en la subsección 3.3.1. La gran diferencia estará en los resultados, ya que en este caso sí se va a utilizar un proceso iterativo. Cada vez que ocurra cierto evento, ya sea que esté un slot listo para salir, que haya pasado unos determinados minutos... Julia deberá conocer de alguna manera el estado de la planta, ya sea porque Simio se lo comunique o porque tenga una especie de memoria.

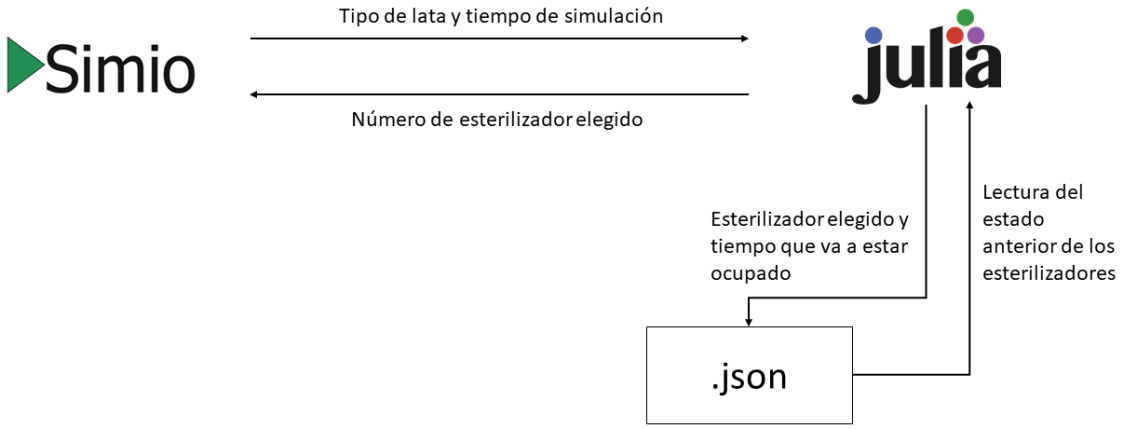

Figura 5.1: Esquema de la conexión propuesta entre Simio y Julia.

Llegado este punto, se tienen dos nuevos problemas.

- ¿Cómo hacer la conexión Simio-Julia?.

  La solución pasará por descubrir e investigar cómo funciona Simio internamente y si permite ser personalizado de alguna manera. También requiere conocer cómo integrar un lenguaje de programación como es Julia dentro del mismo.

- ¿Cómo sabe Julia qué esterilizadores están ocupados en el momento Simio le mande la señal?.

  Para ello se utilizará un fichero *.json* como base de datos para dotar a Julia de memoria y que recuerde de alguna manera qué esterilizadores van a estar ocupados y hasta cuándo.

5.2 Conexión Simio-Julia

5.2.1 Simio

La solución para conseguir integrar algo nuevo dentro de Simio es utilizar una API (Application Programming Interface). Desde Simio explican cómo hacerlo detalladamente en Houck 2018. Mediante una Interfaz de Programación de Aplicaciones se establecen un conjunto de métodos claramente definidos para establecer comunicaciones entre diferentes componentes de *software* y se dan los medios para no tener que programar desde cero, proporcionando de esta manera una base clara de donde partir.

Lo más interesante para este caso es que permite a Simio en concreto extenderse a más soluciones complejas y comunicarse con otros *softwares*. Entre los ejemplos que se detallan están los siguientes.

- Importar y exportar datos de Simio.
- Crear interfaces con lógica existente.
- Cambiar el comportamiento de los objetos de Simio.
- Establecer la conexión con otros sistemas, paquetes o lógica heredada.
- Construir y modificar archivos a partir de datos externos.

Con esto parece que se forman los elementos perfectos para poder conseguir el objetivo. Pero desde Simio avisan que se necesita conocer de antemano una serie de recursos. Entre ellos, tener un conocimiento general de programación y de conceptos modernos como objetos y interfaces, tener competencias con *.NET* y estar familiarizado con Visual Studio y C#. Siendo *.NET* y Visual Studio dos de los conceptos más importantes de este capítulo.

Por un lado, Visual Studio sería el llamado entorno de desarrollo integrado (IDE) donde se incluyen distintos lenguajes de programación. Visual Studio pertenece a la compañía Microsoft y dentro del mismo se puede utilizar el *framework* o marco de trabajo .NET que consiste en una serie de estructuras y tecnología determinadas que facilita bastante la programación. Para utilizar .NET se aceptan lenguajes como C#, Visual Basic, C++, F#, Python... Siendo los dos primeros los más utilizados y son a los que se refiere Simio en su introducción a la API.

Para utilizar su API proponen utilizar dos tipos de mecanismos, una *.dll* o un *.exe* dependiendo de lo que se requiera.

- En caso de querer hacer una extensión en Simio o cualquier modificación se debe utilizar una biblioteca de enlaces dinámicos o *.dll*. Estas son ejecutables, contienen código y funciones que pueden ser usadas a la vez por varios programas, promueven la modularización de código, la reutilización, uso eficiente de memoria y reducción de espacio en disco. Es por ello que hay aplicaciones que se construyen a base de módulos o paquetes de este tipo.

- Si lo que se quiere es hacer es la llamada "simulación sin cabeza", se debe hacer uso de un ejecutable, es decir, un *.exe*. Este tipo de simulaciones funcionan sin una interfaz gráfica definida. Los ejecutables incluyen código y comandos listos para localizar y ejecutar archivos de una determinada aplicación.

Al instalar Simio en el ordenador proporciona una extensión instalable para Visual Studio (*.vsix*) llamada *Simio Visual Studio Templates*. Al instalarla proporciona 3 tipos de plantillas con las que empezar a trabajar.

- Simio UserSelectionRule: Plantilla para un proyecto con el objetivo de crear reglas de selección definidas por el usuario dentro de las propiedades de los objetos.

- Simio User-defined Element with Step: Sirve para personalizar un Elemento de Simio asociado a un *Step*.

- Simio UserAddIn: Con él se crea un complemento para los experimentos llamados *Add-in*. Estos permiten crear múltiples escenarios, analizar la variabilidad y mostrar la mejor solución.

Uno de los estudios previos más parecidos al objetivo del trabajo, la unión de Simio con otro *software* fue el de Dehghanimohammadabadi y Keyser 2017. En él se explica cómo ejecutar un archivo de Matlab desde Simio y viceversa a partir de la creación de un *Step* y todas las posibilidades que esto conlleva. Viendo esto se optó seguir la ruta de estos investigadores, solo quedaría conocer cómo crear un *Step*, configurarlo para que sea adecuado para su uso en la planta de esterilizado y que sea capaz de ejecutar Julia al mismo tiempo. Por ello, se elige como punto de partida la plantilla *User-defined Element with Step*.

User-defined Element with Step

Como su nombre indica, la plantilla está hecha para poder crear o modificar un elemento y un paso. Los primeros representan cosas en un proceso que cambian de estado con el tiempo. Los elementos se agregan a una lista y luego se hace referencia a uno o más pasos del proceso (Simio 2019b). Por ejemplo, uno de los *steps* más utilizados es *Batch* que sirve para agrupar entidades en una sola, es decir si llegan latas A y B y se requiere formar un lote, con este paso se agrupan las dos entidades que llegan (A y B) para formar una única (lote). El elemento *BatchLogic* representa este momento del proceso.

El *Step* sería la formación de un determinado paso. Como ya se ha explicado, estos se activan mediante un *token* y sirven para definir el comportamiento lógico del proceso en Simio. No tienen estado en sí, pero son capaces de cambiar la variable estado ya sea de un elemento, un *token*, una entidad o un objeto.

Por lo tanto, en el trabajo se opta por construir un nuevo *Step* que sea capaz de cambiar los valores de una variable estado dentro de un proceso lógico. De esta manera se podrá definir un comportamiento establecido a uno de los objetos de la simulación. Esta plantilla está formada por dos archivos en lenguaje C#.

- *UserElement.cs*: Sirve para crear un nuevo elemento. Esta plantilla crea directamente un elemento asociado a un *step*, aunque después únicamente se vaya a utilizar el segundo.

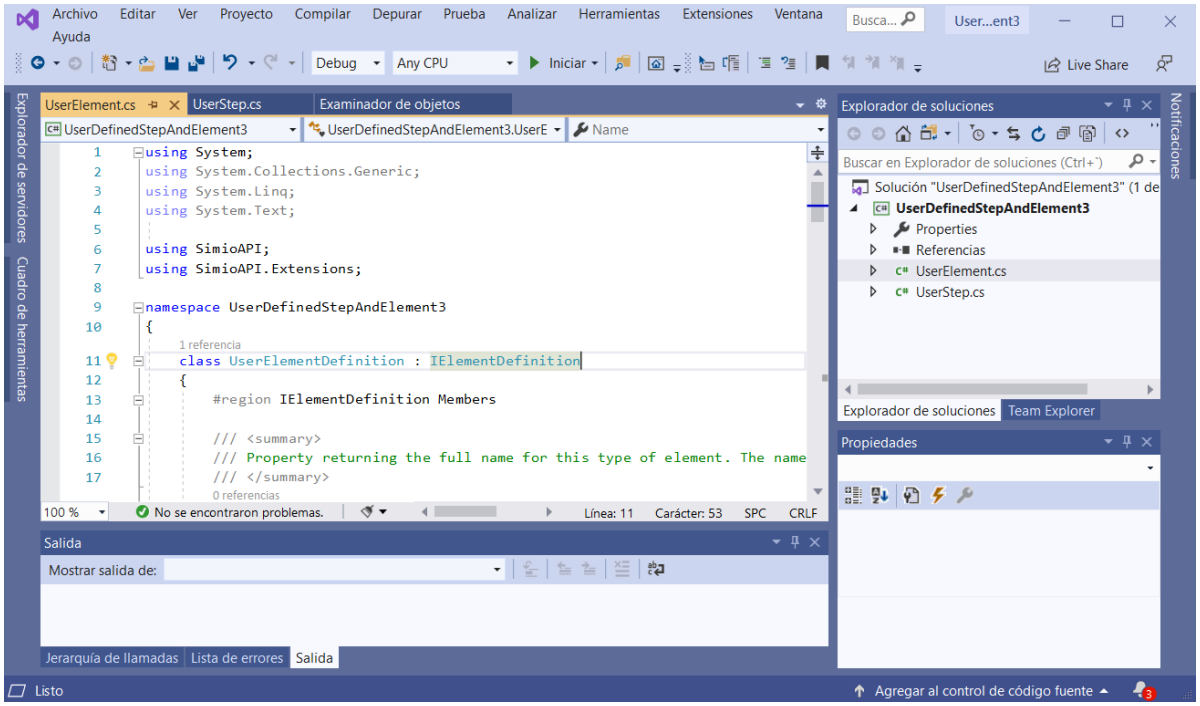

Figura 5.2: Vista de la plantilla del Visual Studio *User-defined Element with Step*.

- *UserStep.cs*: Con él se crea un nuevo paso para los procesos y permite multitud de funcionalidades como introducir o exportar datos, cambiar el valor de variables estado...

Sabiendo esto, el único archivo que haría falta modificar en este caso solamente sería el *UserStep.cs*. Simio detalla cómo hacer esto mediante la figura 5.3.

El procedimiento consistirá en construir una *.dll* mediante Visual Studio y la plantilla elegida. Una vez hecho, se debe llevar a una carpeta interna de Simio llamada Users Extensions. Cuando el programa arranque la cargará y creará el nuevo paso en Simio para poder utilizarse en los procesos lógicos.

Programación orientada a objetos (C#)

Antes de proceder a explicar el archivo sería conveniente conocer unos conceptos previos sobre C# y la programación orientada a objetos. Este tipo de programación se basa puramente en la construcción de objetos a los que le entran datos de entrada, se manipulan y proporcionan datos de salida.

Uno de los términos más importantes para entender los conceptos de este tipo de programación es el de clase, los cuales describen el tipo de los ob-

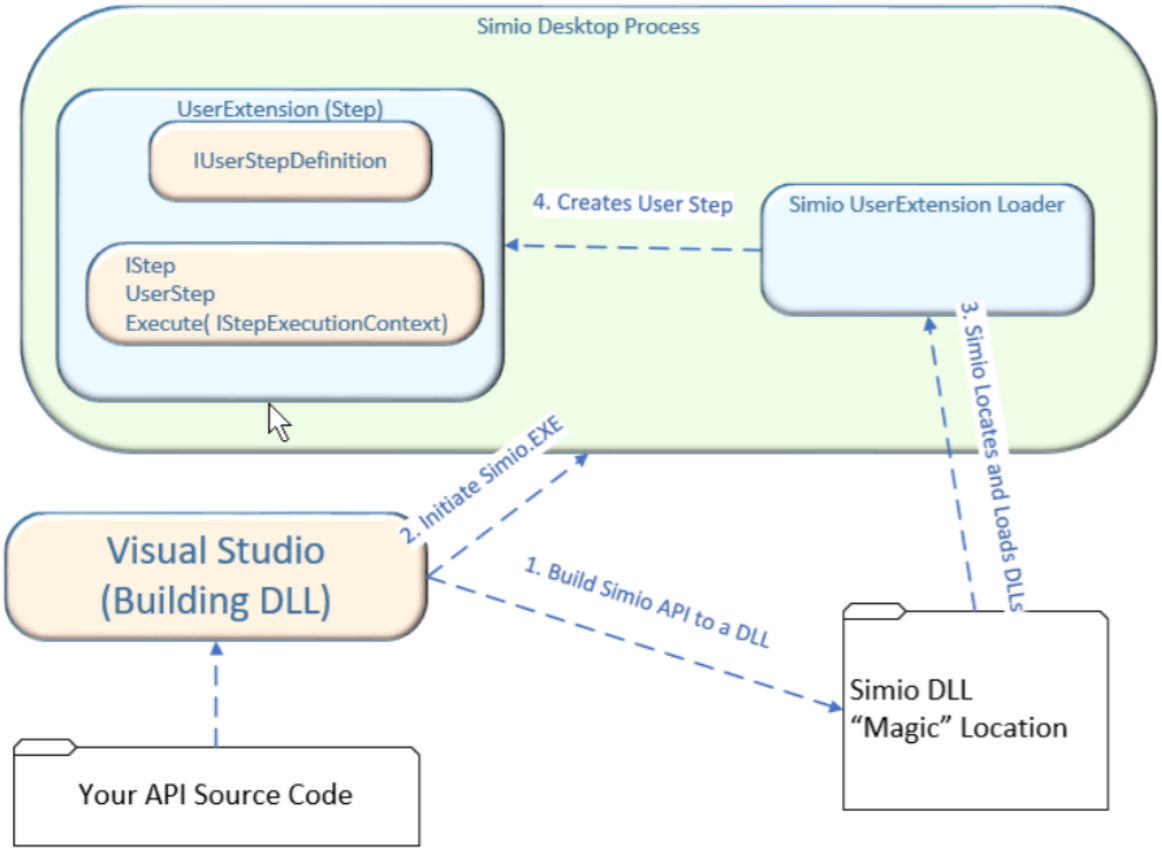

Figura 5.3: Esquema de Simio que explica los pasos a seguir para la construcción de un *Step*. Fuente: Simio 2019b.

jetos. Mientras que los objetos describen las instancias de las clases. Si una clase fuera un plano, el objeto sería un edificio construido a partir de ese plano (Microsoft 2018). Una clase contiene distintos miembros, los 3 más importantes son los siguientes.

- Propiedades: Describen los datos de las clases.
- Métodos: Definen el comportamiento de una clase.
- Eventos: Proporcionan comunicación entre diferentes objetos y clases.

Entre estos miembros es importante quedarse en concreto con la idea de las propiedades y los métodos.

UserStep.cs

La plantilla utilizada es la que viene en la versión de Simio 11.197.19514.0. Para hacer uso de ella lo primero que se debe hacer es referenciar dos bibliotecas dinámicas (*.dll*) de Simio como son *SimioAPI.dll* y *SimioAPI.Extensions.dll*. Con ellas se consigue hacer referencia a los objetos dentro de Simio. Para poder tener una guía de cómo funciona estas referencias que conforman la API, Simio dispone de la *Simio API Reference Guide* (Simio 2019a) que va instalada dentro del mismo programa. Esta guía es la que indica cómo cambiar o simplemente acceder no solo a un objeto, si no una propiedad, estado o cualquier tipo de valor que se requiera. La plantilla viene formada por 2 clases principales con un determinado nombre pero cada una de ellas hace referencia a una *interface* es decir, establece una conexión con el programa.

- UserStepDefinition: Establece la *interface* con *IStepDefinition* que permite describir un tipo de paso o *Step* gracias a un objeto.

  Esta clase debe estar formada por 5 propiedades y 2 métodos tal y como está en la figura 5.4.

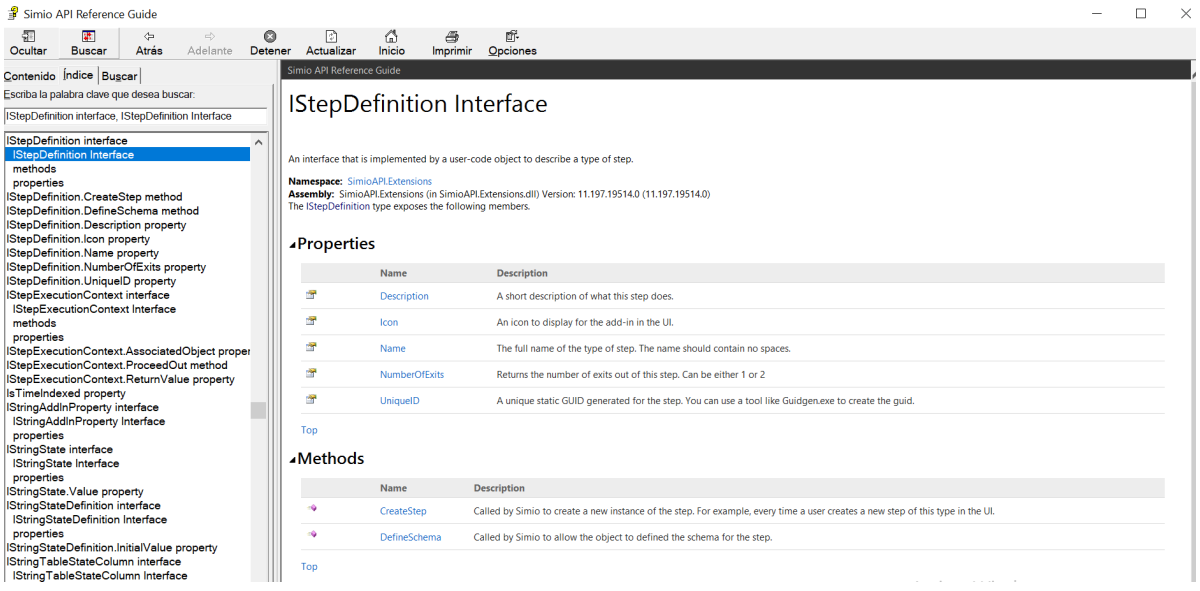

Figura 5.4: Vista de *IStepDefinition* en la *Simio API Reference Guide*.

  - Propiedades:

    - Descripción: Pequeño texto que se va a mostrar en Simio con la explicación de lo que hace el paso.

- Icono: Posibilidad de añadirle un icono. En la propia plantilla se devuelve como nula.
- Nombre: Nombre del paso en Simio.
- Número de salidas: Devolver el número de salidas que puede estar entre 1 y 2.
- *UniqueID*: Se le añade un GUID (Identificador Único Universal) para el paso. Sirve para encontrar desde el *software* de manera precisa una determinada información.

- Métodos:

  - *CreateStep*: Es llamado por Simio para crear un nuevo *Step*.
  - *DefineSchema*: Llamado por Simio para definir el esquema para el paso. En él se pueden expresar las propiedades que va a tener el *Step*, junto con el nombre que se va a mostrar, información adicional...

- UserStep: Establece la conexión con *IStep* que permite dar funcionalidad al paso. Formado únicamente por 1 método.

  - Métodos:

    - *Execute*: Ejecuta la lógica del *Step*.

5.2.2 Unión con Julia

Una vez vista la estrategia para poder crear pasos en Simio, faltaría por ver cómo introducir a Julia dentro de la plantilla en C# de Simio.

En su manual *Embedding Julia* (Julia 2020) se explica cómo incrustar Julia en Visual Studio. Proporciona de esta manera una API, al igual que Simio, para conseguir cumplir el objetivo. Con esto, se dan las claves para poder integrar el lenguaje Julia dentro archivos de C/C++, C#, Python...

Para ello se debe hacer referencias a una librería dinámica llamada *libjulia.dll* que se encuentra en la carpeta *bin* de Julia. Esta tiene como entrada diversas funciones y cada una de ellas se encargará de una determinada acción, llamando si es necesario a otras librerías dinámicas del programa. En el trabajo se utilizan 3.

- jl_init__threading: Introduce en Julia la ruta de la carpeta, que normalmente es *"...\\Julia-1.4.2\\bin"*. Sirve como función de inicialización.

- jl_eval_string: Funciona de dos maneras.

  - Como entrada: Necesita una cadena de caracteres y evalúa el comando en lenguaje Julia. Por ejemplo, para indicarle $a = 3$ sería de la manera $jl\_eval\_string(\text{“}a = 3\text{”})$. Al igual que se le puede introducir una variable también es capaz de ejecutar un archivo *.jl* mediante el comando "*include("programa.jl")*".

  - Como salida: No sólo es necesario ejecutar expresiones, muchas veces se tiene que devolver el valor al programa anfitrión. De esta manera también se puede utilizar para hacer referencia a un objeto de Julia. Es decir si se tiene la expresión $res = jl\_eval\_string(\text{“}a\text{”})$. La variable *res* será un puntero que hará referencia al sitio de la memoria donde se haya guardado el resultado del comando anterior. Ese sitio de la memoria alojará, al valor 3, ya que $a = 3$.

- jl_atexit_hook: Julia recomienda su uso encarecidamente. Con esta función notifica a Julia que el programa está a punto de acabar. por lo tanto, le dará tiempo para limpiar las soluciones de escritura pendientes y ejecutar los finalizadores.

El trabajo restante será el de elaborar una librería dinámica mediante la plantilla de Simio, introducir Julia con estas funciones y crear una estrategia para implementar la optimización de la planta de esterilizado en la simulación.

5.3 Diseño de archivos finales

Como ya se ha comentado se va a trabajar con una evolución de la sección 4.3 y con la estrategia definida en la sección 5.1. Es decir, se tiene un sistema con 5 esterilizadores y la intención es poder ejecutar Julia cuando haya un slot listo, indicándole el tipo de entidad y que este proporcione el autoclave de destino, teniendo en cuenta Julia qué esterilizadores están ocupados a través de su base de datos *.json*.

Ahora que ya se conocen las capacidades que proporciona Simio con las plantillas, se va a definir una estrategia final para construir un *Step* que se pueda poner en un proceso para que sea efectiva la comunicación. Este nuevo esquema es el de la figura 5.5.

Por el lado de Simio, la idea es crear unas nuevas variables estado que pueda obtener la librería dinámica para hacer sus propios cálculos y enviárselas a Julia. Y viceversa, que esta pueda enviar nuevos valores a Simio. Para ello se va a hacer uso de 7 variables de estado. La primera de ellas ya se creó en la sección 4.3 que es la que indica el tipo de entidad. Todas las otras son nuevas y se deberán crear en el apartado de *Definitions*. La segunda será el tiempo de simulación, es decir, una variable que indica los minutos o segundos que han pasado desde el inicio. Para poder indicarle a Simio qué esterilizador debe activarse, se ha optado por la solución de cambiar el peso de los *Paths*, los caminos que llevan a los autoclaves. De esta manera si el peso es 0, ese camino está desactivado y buscará el que tenga un valor de 1, al haber 5 esterilizadores en el modelo anterior, se crean 5 variables una para cada uno de los *Path*. Además, se intentará enviar una cadena de texto para indicar la ruta del archivo de Julia y así aumentar la comodidad a la hora de hacer pruebas con diferentes ficheros.

Por la parte de Julia, la librería dinámica deberá indicarle únicamente el tipo y el tiempo de simulación por el que se ha iniciado el evento. Se ejecutará el archivo y comprobará en su base de datos *.json* que no haya ningún esterilizador ocupado. Hará los cálculos y el solver dará una optimización de *scheduling* con las restricciones establecidas, el autoclave asignado al primer slot será el elegido. En la base de datos se guarda primero el autoclave que se va a poner en marcha y el tiempo que va a estar ocupado. El dato es enviado por parte de Julia y el código de la *.dll* hará el tratamiento correspondiente.

5.3.1 Creación del Step con la plantilla de Simio

El resultado completo se muestra en la sección B.2 pero se va a intentar resumir de manera esquemática su funcionamiento sin entrar en detalles de programación.

El primer paso será cargar las librerías externas que se van a usar y que vienen puestas en la plantilla. Como ya se ha explicado existen dos clases principales.

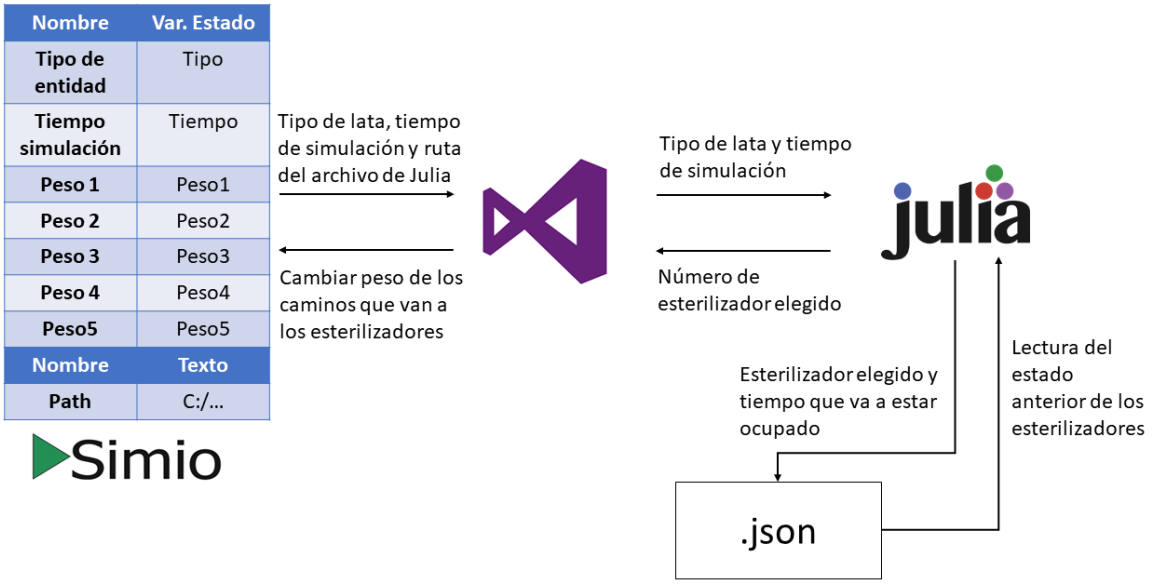

Figura 5.5: Esquema de la conexión propuesta entre Simio y Julia detallado.

UserStepDefinition

Se define el nombre del paso ("Simio_Julia"), la descripción ("Step que indica..."), la imagen, (null) el guid y el número de salidas (1).

Para los métodos en *DefineSchema* se definen 8 propiedades que va a tener el *step* y se añaden con su nombre y una pequeña descripción ("Tipo de entidad", "Peso del path 1"...). Estas serán propiedades o casillas de las cuales, una será para el tipo de entidad, el tiempo, 5 para el peso de cada uno de los caminos y otra con la ruta al archivo de Julia.

Con *CreateStep* se crea el paso en Simio con esas mismas propiedades.

UserStep

La primera acción es introducir las librerías dinámicas para el uso del lenguaje Julia. Una de ellas es *kernel32.dll* para poder hacer uso de la función de entrada *SetDllDirectory*, utilizada para indicar la ruta donde buscar las demás librerías de Julia. La otra es la *libjulia.dll* con las funciones *jl_init__threading*, *jl_eval_string*, *jl_atexit_hook*.

Una vez dentro del método *Execute* se hace todo el proceso operativo.

Se obtiene el valor de las variables estado de ese mismo instante y se le asigna un nombre (*state1*, *state2*...). Aparte, se lee también la cadena de texto que se haya escrito en la casilla de *Path* con el nombre *varPath*.

En este momento es cuando empieza el tratamiento de los datos con Julia. Se le indica el directorio con la variable *juliaDir*, *SetDllDirectory(juliaDir)* y *jl_init__threading(juliaDir)*. Se le pasa el tipo de entidad con el que se ha iniciado el evento *jl_eval_string("tipo_1 = state1")* y el archivo de optimización que se indica a través de Simio mediante la ruta, por ejemplo *jl_eval_string("include("C:/programa.jl")")*. Julia hace los cálculos y devuelve el valor del autoclave que se ha seleccionado mediante la función: *jl_eval_string("autoclave_sel")*. Se cierra Julia en segundo plano a través de *jl_atexit_hook(0)*.

En este punto es donde dependiendo el valor que tenga *autoclave_sel* se deja todos los pesos a 0 menos el seleccionado que se marca como 1. De esta manera se transmite el valor a Simio, se cambia automáticamente el peso del *Path* dejando como único posible el que haya indicado Julia. El *Step* se queda de la manera que indica la figura 5.6

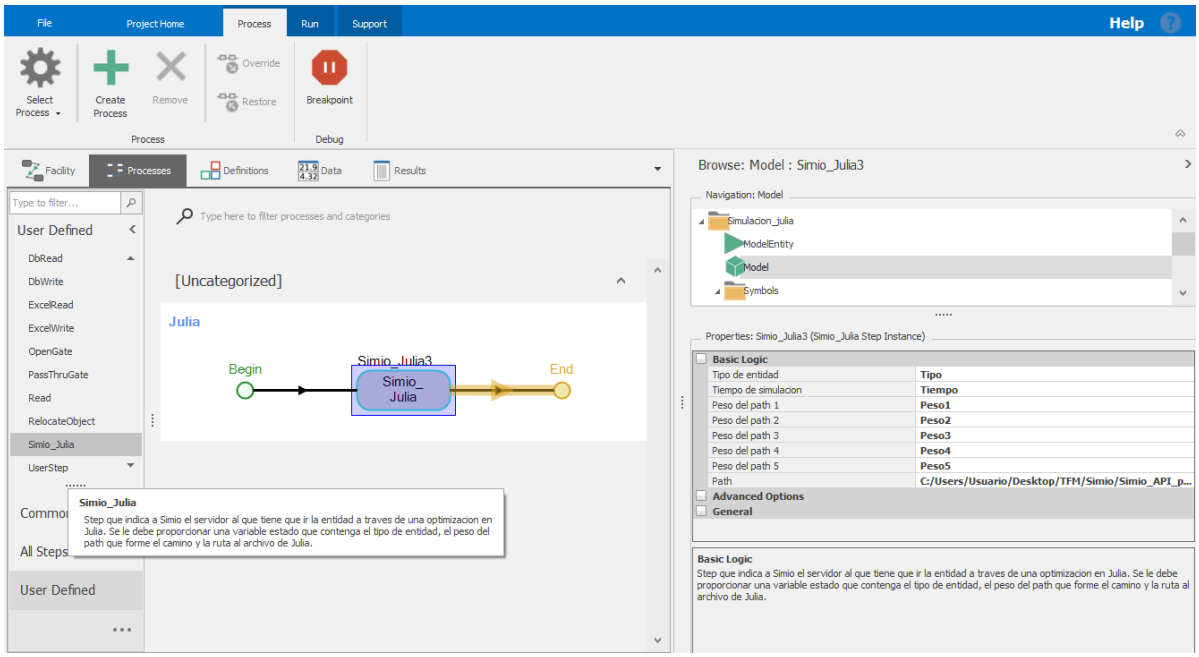

Figura 5.6: Vista del *Step* creado en Simio

5.3.2 Modificación del archivo de Simio

Para poder establecer el *Step* que comunica Simio con Julia es necesario hacer unas modificaciones sobre el modelo de simulación que se ha había hecho anteriormente en la sección 4.3.

Los cambios serán sobre todo del apartado procesos y de las variables estado. Por una parte, se necesitan crear las siguientes variables, aparte de las ya hechas.

- *Tiempo*: Variable que contendrá el valor del tiempo que haya pasado desde que se inicia la simulación.
- *Peso...*: Las 5 variables que cambiarán el valor del peso *(weight)* de los diferentes *Path* o caminos a los esterilizadores. Inicialmente valdrán 0, pero al llamar a Julia, el código de la librería dinámica los cambiará dejando uno de ellos en 1 que será el elegido para que pase el grupo de carros.

La hora de inicio de Julia y de Simio debe ser la misma para que al enviarle los segundos de simulación Julia sepa sobre qué hora hace referencia.

Respecto al apartado de los procesos se necesitan hacer dos cambios. Estos se ven claramente en la figura 5.7.

- El primer paso será añadir 2 bloques *Execute* que sirven para ejecutar otro proceso. Cuando el *token* recorra *transferNode1_Entered*, se decida qué tipo de entidad es y si el número de carros que hay ya es el indicado para formar un grupo, se ejecutará inmediatamente otro proceso llamado *Julia*.
- El segundo paso es crear el proceso *Julia*, que estará formado por un *Assign* y el *Step* creado *Simio_Julia*. La función del primero será cambiar el valor de la variable de estado *Tiempo* con la fórmula $DateTime.Second(TimeNow) + (60 * DateTime.Minute(TimeNow) + (3600 * DateTime.Hour(TimeNow))$. De esta manera el valor de esa variable pasa a ser el valor del tiempo de simulación que lleva Simio en segundos. Una vez listas tanto *Tipo* como *Tiempo* se ejecuta el *Step* diseñado.

Cuando se lance *Simio_ Julia* se introducirán el valor de las dos variables de entrada y la ruta del archivo de Julia, para que haga una optimización de *scheduling*. Simio esperará a que termine y para que la simulación pueda continuar.

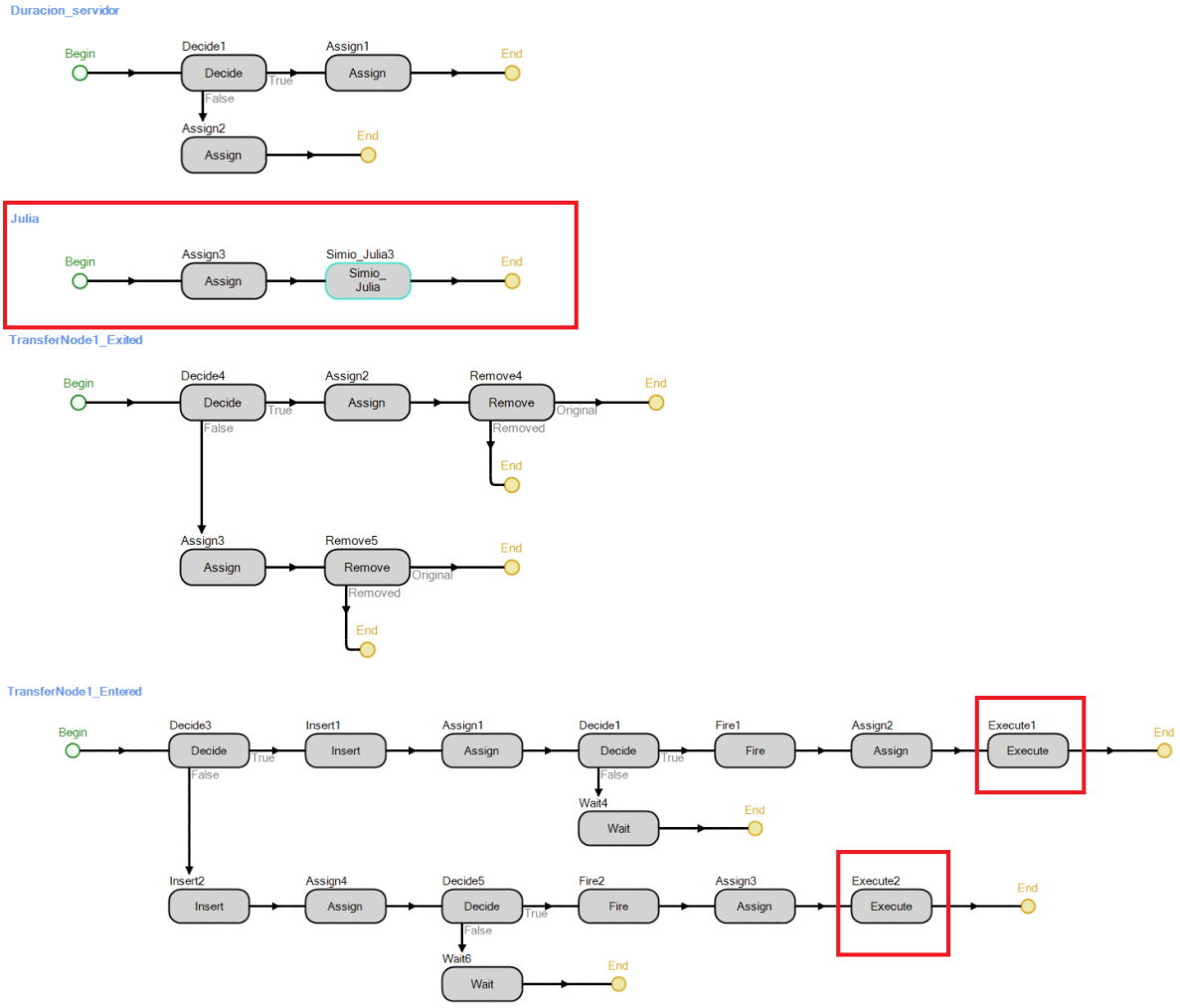

Figura 5.7: Vista los procesos elaborados para la simulación en Simio con las modificaciones.

5.3.3 Creación del archivo de optimización con Julia

La idea para el archivo de optimización es que haga lo mismo que en el capítulo 3. Es decir, aunque Simio llame a Julia para saber a qué esterilizador tiene que llevar el grupo de carros, esta decisión debe de poder perseguir un objetivo. Cada vez que se ejecute el archivo en Julia hará un *scheduling* de cara al futuro conociendo los datos del sistema. A través de las restricciones y función/es objetivo se podrá perseguir un determinado fin. Si se quiere reducir el tiempo total y la solución dada por el *solver* es llevarlo a un deter-

minado esterilizador debe ser porque con esas condiciones y esos autoclaves disponibles esa es la mejor opción en ese momento para ese objetivo.

Como ya se sabe, al archivo de optimización le entran dos datos como es el tipo de entidad y el tiempo. Pero a parte se necesita conocer de antemano algunos datos que tienen que ver con el sistema y que ya se han utilizado en la simulación de Simio descrita en la sección 4.3.

- La tasa de llegada de slots: Al haber carros de dos tipos, va a llegar un slot cada 20 minutos de un tipo. Por ejemplo si llega uno del 1, a los 20 minutos lo hará otro del 2.
- Duración de esterilización: Diferente dependiendo del tipo. La fase de calentamiento será más o menos la misma pero habrá una discrepancia importante en la de mantenimiento-enfriamiento.

El diseño del archivo de Julia será también una evolución del modelo matemático de precedencia general utilizado en la subsección 3.3.1 pero de manera más simple y con algunas variaciones. Se elimina el horizonte robusto, ya que la idea ahora es llamar a Julia cada vez que esté listo un slot y lo indique Simio.

Conjuntos

Dado que la optimización se hace cuando un slot está preparado para salir, no es necesario disponer del conjunto de los carros como sí lo era anteriormente. En este caso se tendrá que introducir una nueva variable que indique el tipo de carro que ha producido el evento en Simio.

- Grupos de carros a esterilizar (Slots): J
- Tipo de carro o entidad: tipo
- Autoclaves: U

Variables

En las variables se elimina cualquiera que esté relacionada con los carros. Destaca la introducción de la binaria T que relaciona los slots con el tipo. Por otro lado, ts_j hará referencia al momento cuando esté listo un slot para ser esterilizado, se obtiene a partir del cálculo de la tasa de llegada de slots. La variable t_t_j, será del mismo tamaño que ts_j e indicará el tipo de slot que hace referencia j. La última diferencia es que t_mc_t ahora depende del tipo que sea por lo tanto es función de t y contendrá el valor de la duración de la fase de mantenimiento-enfriamiento. La variable que relacione el slot con el tiempo que debe durar esta fase será t_tipo_j.

- Z(bin): Variable binaria que relaciona un slot con un determinado autoclave.

$$Z_{j\in J,u\in U} \tag{5.1}$$

- T(bin): Variable de tipo binaria que relaciona un slot con un tipo de carro. Esta valdrá uno cuando el slot j está asignado al tipo de carro o entidad t.

$$T_{j\in J,t\in tipo} \tag{5.2}$$

- Y(bin): Relaciona la precedencia entre slots.

$$Y_{j\in J,j'\in J} \tag{5.3}$$

- $te_{j\in J}$: Tiempo incio de esterilización para un slot j.
- $tf_{j\in J}$: Momento en el que un determinado slot j termina el proceso.
- $ts_{j\in J}$: Instante de tiempo en el que un slot j está listo para entrar a un autoclave.
- $t\_t_{j\in J}$: Variable que acompaña a ts_j y que indica de qué tipo es el slot que se va a procesar.
- MK: Makespan, tiempo total de inicio al final.

- $t\_tipo_{j \in J}$: Tiempo de fase de mantenimiento-enfriamiento que va a tener un slot j dependiendo del tipo t que sea.
- $t\_mc_{t \in tipo}$: Duración de fase de mantenimiento-enfriamiento dependiendo del tipo t que sea.
- $ocupados_{u \in U}$: Vector que contiene la información del tiempo que le queda a cada esterilizador. Valores que salen a partir de la lectura del *.json*.

Parámetros

Los parámetros se quedan en 2, se mantiene τ pero siendo ahora el tiempo de espera máximo de un slot a ser asignado a un autoclave y t_{he} como lo que dura la fase de calentamiento.

- τ: Tiempo de espera máxima para cada slot, que será diferente dependiendo del alimento o el tipo de lata.
- t_{he}: Duración estándar de la fase de calentamiento.

Restricciones

En cuanto a las restricciones se hace lo mismo que en las variables y conjuntos, eliminar las que tengan que ver con los carros.

- Autoclave: Cada slot j solo puede estar asignado a un esterilizador.

$$\sum_{u \in U} Z_{j,u} = 1 \quad \forall j \in J \tag{5.4}$$

- Tipo de entidad: Se añaden las que tienen que ver con el tipo de lata/-carro/slot. En este caso únicamente podrá haber un slot j asignado a un tipo. También se tendrá que siempre que un slot esté asignado a un tipo y la variable $T_{j,t}$ sea 1 será el mismo valor que t_t_j para ese determinado j. La última a tener en cuenta será que si un slot está asignado a un tipo t, la variable t_tipo_j tendrá el mismo valor que t_mc_t.

$$\sum_{t \in tipo} T_{j,t} = 1 \quad \forall j \in J \tag{5.5}$$

$$\sum_{t \in tipo} t \cdot T_{j,t} = t\_t_j \quad \forall j \in J \tag{5.6}$$

$$t\_tipo_j = \sum_{t \in tipo} (t\_mc_t \cdot T_{j,t}) \quad \forall j \in J \tag{5.7}$$

- Esterilización: En este caso la duración final tf_j será la suma del tiempo de inicio de esterilización, la fase de calentamiento y la fase de mantenimiento-enfriamiento para ese slot j. También se ha de tener en cuenta que el inicio de la entrada de un autoclave de j nunca va ser menor que ts_j, pero es esencial que el inicio no se produzca después de el tiempo máximo de espera τ.

$$tf_j = te_j + t_{he} + t\_tipo_j \quad \forall j \in J \tag{5.8}$$

$$te_j \geq ts_j \quad \forall j \in J \tag{5.9}$$

$$te_j \leq ts_j + \tau \quad \forall j \in J \tag{5.10}$$

- Precedencia:

$$te_{j'} \geq tf_j - Big_M(1 - Y_{j,j'}) - Big_M(2 - (Z_{j,u} + Z_{j',u})) \quad \forall j, j' \in J, \forall u \in U \tag{5.11}$$

$$te_j \geq tf_{j'} - Big_M(Y_{j,j'}) - Big_M(2 - (Z_{j,u} + Z_{j',u})) \quad \forall j, j' \in J, \forall u \in U \tag{5.12}$$

- Makespan:

$$MK \geq tf_j \quad \forall j \in J \tag{5.13}$$

- Sistema: Al leer el estado de los esterilizadores en la base de datos puede que haya alguno que esté ocupado. Por lo tanto se debe incluir una

restricción que indique que en caso de asignar un slot j a un determinado autoclave u, deberá esperar a que termine. Siendo *ocupados* un vector que contiene la información del tiempo restante que le queda a cada esterilizador.

$$te_j \geq ocupados_u(Z_{j,u}) \quad \forall j \in J, \forall u \in U \tag{5.14}$$

Función objetivo

Para la función objetivo se utiliza en este caso la minimización del *makespan* igual que se podría utilizar cualquier otra dependiendo del objetivo que se tenga.

$$minimizar \quad MK \tag{5.15}$$

El resultado es el que se puede ver en la sección B.3. Los valores utilizados en el fichero de Julia han sido los mostrados en la tabla 5.1.

	Slot tipo 1	**Slot tipo 2**
Tasa de llegada	40 min (dif. de 20 min con el 2)	40 min (dif. de 20 min con el 1)
T. procesamiento (tmc)	50 min	60 min
t_{he}	15 min	
τ	15 min	
Slots	8	
Autoclaves	5	

Tabla 5.1: Valores utilizados en el fichero de optimización.

5.3.4 Base de datos .json de Julia

Cuando se produzca una optimización de este tipo es necesario que Julia de alguna manera sepa qué esterilizadores van a estar ocupados y durante cuánto tiempo. Para ello, se utiliza un archivo de texto *.json* como base de datos.

La idea es que inicialmente todos los esterilizadores estén a 0 en el fichero de texto como se ve en la ecuación 5.16. Cuando Simio llame a Julia este deberá leer la información del estado de los autoclaves. Mediante el paquete *Dates* se puede utilizar el formato temporal.

$$\{“Esterilizador\_oc'' : [0, 0, 0, 0, 0]\} \tag{5.16}$$

Se produce la optimización y se elige el primer esterilizador que se va a poner en marcha para enviárselo a Simio. Antes de eso Julia, deberá guardar en el *.json* que autoclave ha sido elegido y la hora a la que va a terminar la esterilización. Como en la ecuación 5.17 en caso de ser elegido el 5.

$$\{”Esterilizador\_oc” : [0, 0, 0, 0, ”2020 - 08 - 15T012 : 00 : 00”]\} \tag{5.17}$$

Cuando se produzca una nueva llamada a Julia es decir, está listo un nuevo slot, deberá leer la información del fichero y ver a partir de qué hora va a estar libre cada esterilizador. En caso de que al restarle la hora actual haya pasado el tiempo establecido de esterilizado, se deja en 0, si no, se le deja el valor actual. Si no es 0, quiere decir que sigue ocupado por lo que se le deberá añadir una restricción a la optimización para que tenga en cuenta que autoclaves están disponibles. Los minutos que les queden a los esterilizadores diferentes a 0 se introducen en la variable *ocupados* para proceder con los cálculos. Se optimiza y se guarda la información modificada con el esterilizador elegido.

$$\begin{aligned}\{”Esterilizador\_oc” : [”2020 - 08 - 15T012 : 30 : 00” \\ , 0, 0, 0, ”2020 - 08 - 15T012 : 00 : 00”]\}\end{aligned} \tag{5.18}$$

Por ejemplo, en el caso de que no vuelva a haber otro autoclave listo hasta las 12:25:00. Al ejecutar Julia, este verá que el 5 vuelve a estar disponible ya que se ha pasado la hora guardada y por lo tanto, se deja a 0. Construirá el vector *ocupados* con los valores $[5, 0, 0, 0, 0]$. Incluirá por tanto, una restricción de que no se le puede asignar ningún slot al esterilizador 1 hasta pasar los 5 minutos. Se ejecuta y se guarda la nueva solución, por ejemplo en el autoclave 3.

$$\begin{aligned}\{”Esterilizador\_oc” : [”2020 - 08 - 15T012 : 30 : 00” \\ , 0, ”2020 - 08 - 15T013 : 05 : 00”, 0, 0]\}\end{aligned} \tag{5.19}$$

De este modo se dota a Julia de una memoria con la que poder establecer la comunicación de manera efectiva.

6 Resultados

En este capítulo se muestra finalmente cómo es el resultado de la interacción entre Simio y Julia. Cómo a partir de la simulación se llama al optimizador para que decida qué respuesta es la mejor. En él también se analizan los resultados de la optimización de Julia cada vez que Simio lo requiere y cómo queda finalmente el *scheduling* a lo largo del tiempo.

6.1 Preparación y primera llamada a Julia

Cada vez que se vaya a iniciar una simulación se requiere hacer ciertas acciones antes.

- La primera de ellas es asegurar que la base de datos *.json* tiene todos los valores a 0.
- La segunda es comprobar que el inicio de la simulación en Simio lleva la misma fecha que el archivo de Julia *.jl* que se vaya a utilizar, para los resultados que se muestran la simulación empieza el 09/09/2020 a las 00:00:00.

Una vez comprobado esto se puede iniciar la simulación, los carros se irán produciendo en las fuentes (*sources)* con la tasa establecida y los operarios los irán llevando a la zona de espera. Cuando se llegue al número de carros que conforman un lote se producirá la llamada a Julia indicándole los segundos de simulación que lleva y el tipo de carro que ha producido el evento (1 o 2). La simulación en ese momento se detendrá como la figura 6.1 y esperará una respuesta.

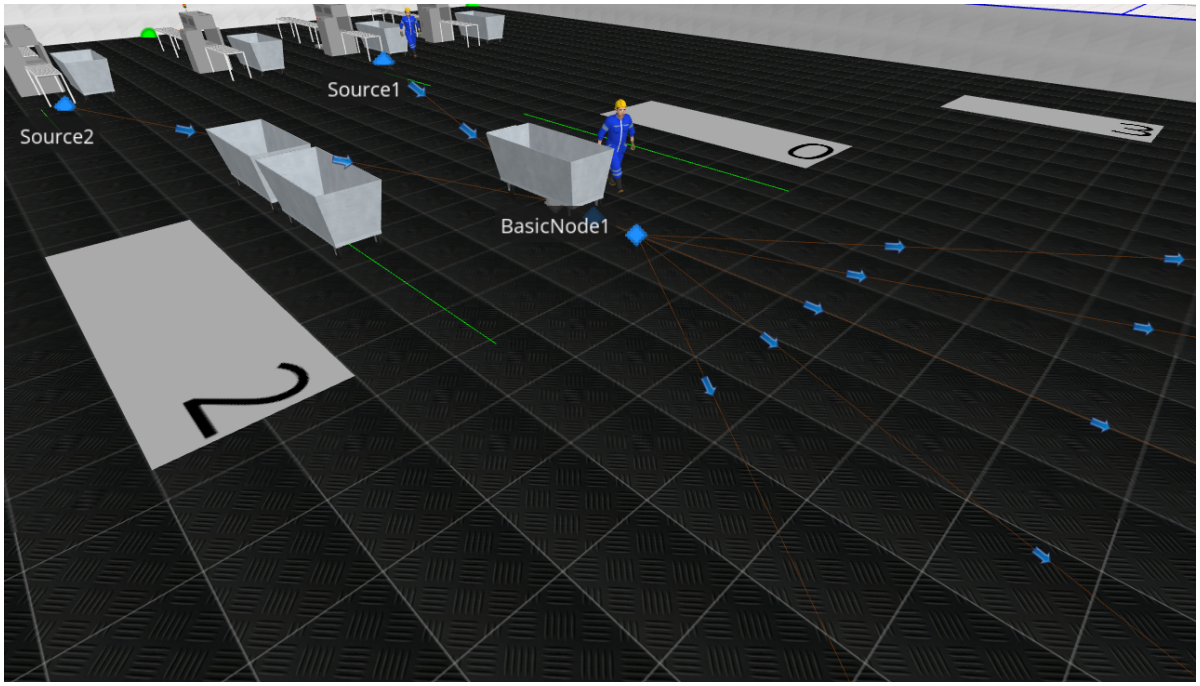

Figura 6.1: Vista 3D de la simulación final en Simio.

Es en ese momento cuando Julia comenzará a ejecutar el fichero que se haya indicado en el *Step* de Simio. El primer paso será el de leer el tipo de entidad y cuál es tiempo actual de simulación. Después lee la base de datos para comprobar qué esterilizadores pueden estar ocupados y hasta qué hora. El contenido en ese momento del archivo *.json* el siguiente.

$$\{"Esterilizador\_oc" : [0, 0, 0, 0, 0]\} \tag{6.1}$$

Al no haber ningún valor diferente al 0, empezará a optimizar con las variables, parámetros y restricciones descritas. Un ejemplo de un resultado en la primera iteración es el de la tabla 6.1. El *solver* habrá optado por la mejor solución en vista a la llegada de los siguientes 8 slots. La asignación del autoclave del primer lote será la solución. Mediante el código escrito en la *.dll* se cambiarán los valores de los pesos de los caminos que llevan a los esterilizadores. Dejando como única opción en este caso el camino número 5. En la figura 6.2 se muestra cómo queda el *scheduling* producido por el solucionador.

Cuando haya terminado la optimización, el fichero modificará la base de datos con la hora a la que va a finalizar la esterilización de ese lote, en la posición 5. La manera de poder conocer el valor de la nueva fecha será

Slot	Tipo	T. inicio (min)	Duración (min)	T. final (min)	Auto. asign.
1	2	0.0	75	75.0	5
2	1	20.0	65	85.0	1
3	2	40.0	75	115.0	2
4	1	75.0	65	140.0	5
5	2	80.0	75	155.0	3
6	1	100.0	65	165.0	4
7	2	120.0	75	195.0	2
8	1	140.0	65	205.0	1

Tabla 6.1: Resultados de la primera optimización en Julia para proporcionar la respuesta a Simio.

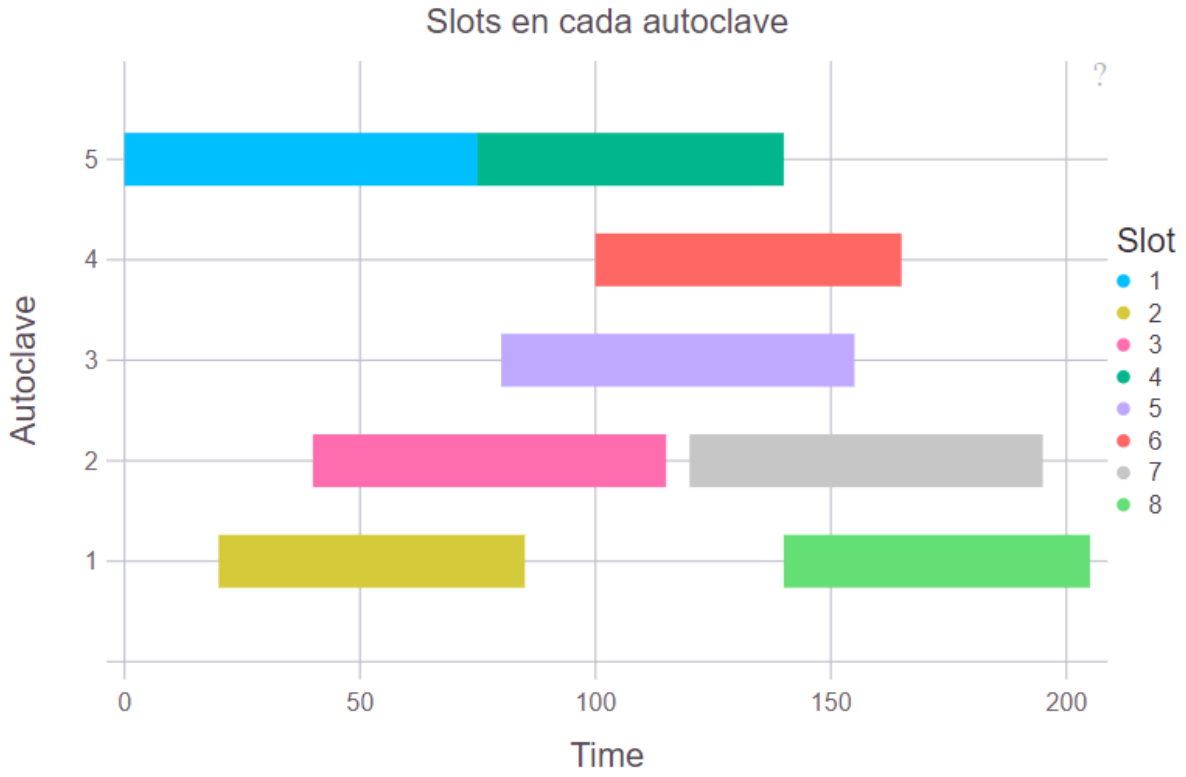

Figura 6.2: Primer Scheduling lanzado en Julia para proporcionar la respuesta a Simio.

sumándole a la inicial (las 00:00:00 del 09/09/2020) el tiempo de simulación que lleve (40 minutos) y la duración de esterilización de ese autoclave (si es de tipo 2, 75 minutos). Lo que indicará que el esterilizador 5 estará libre a las 01:55:00 del día 09/09/2020.

$$\{"Esterilizador\_oc" : [0, 0, 0, 0, "2020 - 09 - 09T01 : 55 : 00"]\} \quad (6.2)$$

Cuando Julia haya terminado y los valores de los pesos de los caminos se vayan a modificar en Simio, se lanzará un mensaje en pantalla como el de la figura 6.3 en el que se muestra el esterilizador que ha sido elegido.

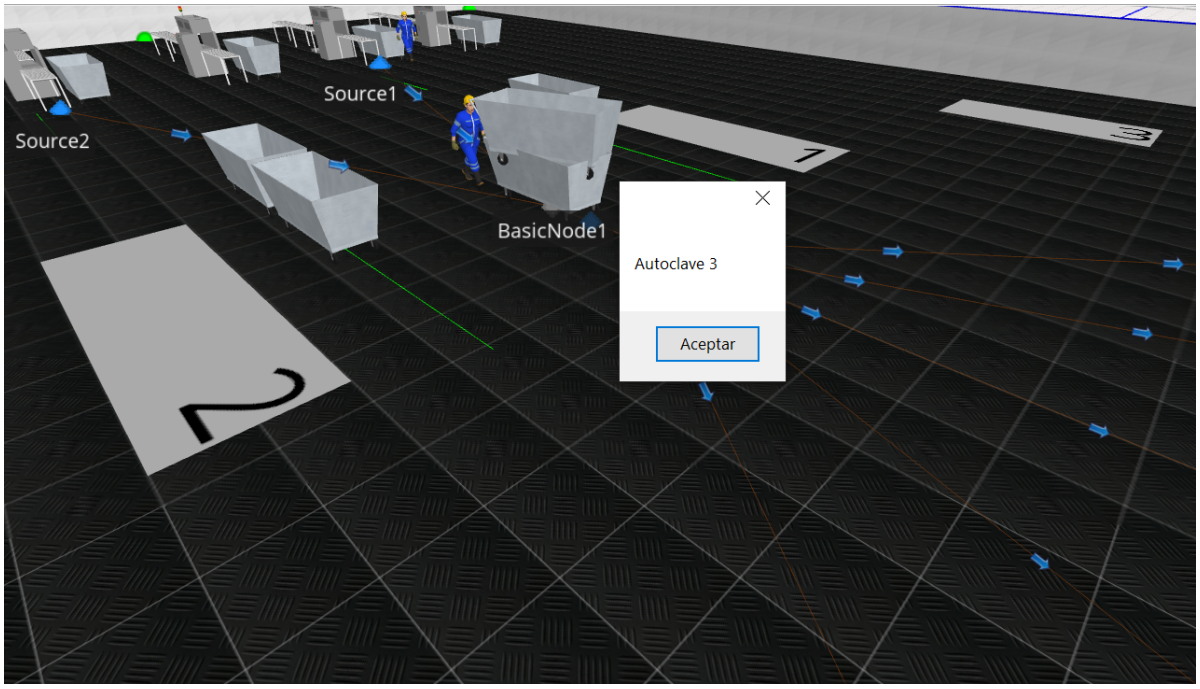

Figura 6.3: Vista 3D de la simulación final en Simio una vez recibida la respuesta de Julia.

Es en ese momento cuando dependiendo del tipo de entidad se cambia el tiempo de esterilizado del autoclave seleccionado.

Cuando vuelva a haber otro grupo de carros listo para entrar en un autoclave se volverá a llamar a Julia. Por la tasa de llegada de carros que se ha utilizado el siguiente será del tipo 1. El procedimiento será el mismo que en la anterior iteración. Se envía el tiempo de simulación y el tipo y por otro lado, se lee qué esterilizador está ocupado y hasta cuando mediante la base de datos. Por los resultados anteriores se conoce que el autoclave 5 no estará disponible hasta la 1 : 55 : 00.

6.2 Segunda y posteriores llamadas

Los resultados son los de la tabla 6.2 y la figura 6.4. En este caso el slot 1, será irá al autoclave 2 con 65 minutos de duración. Destaca el gran retraso que sufre el 5 para ser asignado debido a la restricción que sufre de que no puede ser ocupado hasta que termine el lote anterior.

Slot	Tipo	T. inicio (min)	Duración (min)	T. final (min)	Auto. asign.
1	1	0.0	65	65.0	2
2	2	20.0	75	95.0	4
3	1	40.0	65	105.0	1
4	2	60.0	75	135.0	3
5	1	80.0	65	145.0	2
6	2	100.0	75	175.0	4
7	1	120.0	65	185.0	5
8	2	140.0	75	215.0	1

Tabla 6.2: Resultados de la segunda optimización en Julia para proporcionar la respuesta a Simio.

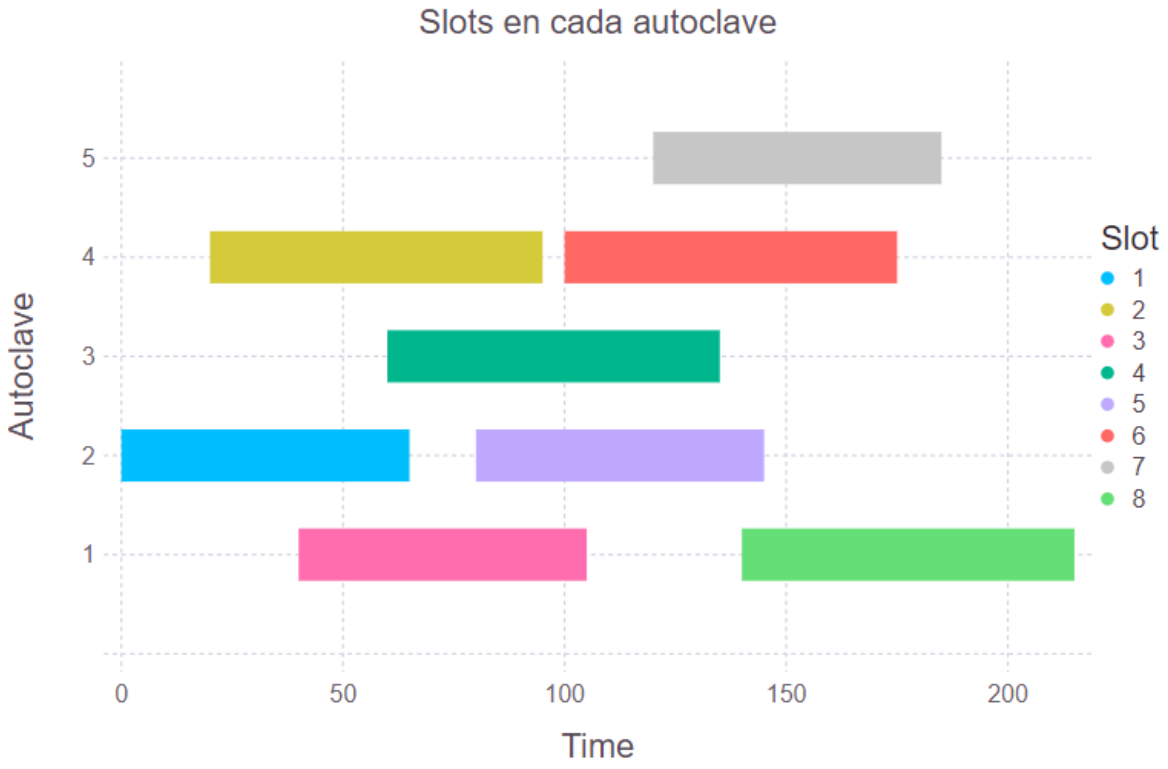

Figura 6.4: Segundo Scheduling lanzado en Julia para proporcionar la respuesta a Simio.

Una vez hecha la solución se guarda en la base de datos la información sobre la hora a la que va a estar disponible el esterilizador 2, si el evento se ha iniciado a las 01 : 00 : 00 (60 minutos), al ser de tipo 1 terminará a las 02 : 05 : 00. La hora del autoclave 5 se mantiene al seguir aun en esos momentos ocupado.

$$\{"Esterilizador\_oc" : [0, "2020-09-09T02:05:00", 0, 0, "2020-09-09T01:55:00"]\} \tag{6.3}$$

De esta manera se irán produciendo iteraciones cada vez que un grupo de carros esté listo para ser esterilizado.

Un ejemplo de lo que ocurre cuando la simulación se deja durante un tiempo es lo que se ve en la figura 6.5 de manera gráfica y en la tabla 6.3 para los 8 primeros lotes. Cada vez que el sistema se ponga en marcha se irán acumulando los carros a medida que se fabriquen hasta un determinado momento, en este caso a los 40 minutos y después estará listo un slot cada 20. Simio y Julia colaboran de la manera descrita anteriormente. En este caso el primer lote es de tipo 1 y se asigna al 5, el segundo de tipo 2 al 3... Para este último caso al permitirse una espera del primer slot de máximo 5 minutos retrasa el comienzo del lote 4. Destaca sobre todo que aun teniendo los autoclaves 1 y 2 listos, cree que la solución más conveniente es esperarse a que el esterilizador 5 termine. Esto es debido a la función a optimizar, para este caso el *solver* organiza los lotes a su manera para reducir el tiempo total de operación.

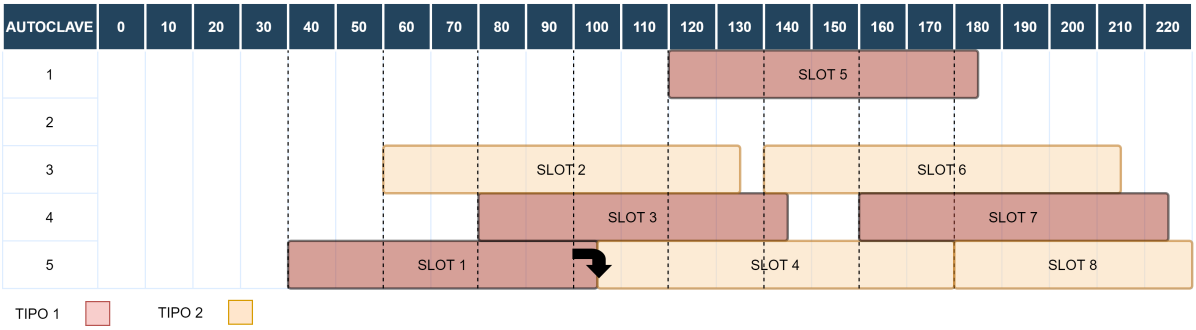

Figura 6.5: Diagrama de Gantt de las decisiones tomadas en Simio.

Slot	Tipo	Dur. (min)	Entr. (min)	Hora entr.	Hora salida	Auto.	Retraso (min)
1	1	65	40	00:40:00	01:45:00	5	0
2	2	75	60	01:00:00	02:15:00	3	0
3	1	65	80	01:20:00	02:25:00	4	0
4	2	75	100	01:40:00	02:55:00	5	5
5	1	65	120	02:00:00	03:05:00	1	0
6	2	75	140	02:20:00	03:35:00	3	0
7	1	65	160	02:40:00	03:45:00	4	0
8	2	75	180	03:00:00	04:15:00	5	0

Tabla 6.3: Resultados de las decisiones tomadas en Simio.

Con todo este se consigue una manera efectiva poder incluir una optimización matemática de *scheduling* dentro de un simulador que proporcione órdenes a la simulación en vista de conseguir unos objetivos globales.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

A través de este trabajo se ha logrado el objetivo principal de conseguir integrar en un simulador de eventos discretos (Simio), un lenguaje de programación de alto nivel como es Julia. Todo ello aplicándolo a un determinado caso de estudio industrial. Para poder obtener el resultado final, se ha tenido que seguir una serie de pasos.

El primero era tener un contexto que pudiera servir a modo de ejemplo del beneficio y utilidad del trabajo. Siendo este el estudio de la mejora del proceso de esterilizado de una empresa real de conservas, utilizando la optimización de *scheduling* en tiempo cortos de predicción. Algo que se puede compaginar con la digitalización y la industria 4.0 para la mejora de los sistemas MES.

El hecho de querer aplicar este tipo de optimizaciones exigía tener un conocimiento profundo del proceso industrial, sobre todo a la hora de crear un modelo matemático de Programación Lineal Mixta-Entera (MILP). La optimización de los dos modelos obtenidos daba como resultado el mejor *scheduling* posible para las funciones objetivo especificadas. Para lograrlo se codificaron en el lenguaje Julia, la cual facilitaba la implementación de este tipo de modelos y disponía de multitud de optimizadores como CPLEX o Cbc.

Una manera de simular la planta real era a través de un modelo de simulación en Simio. Aprovechando todas las ventajas que este tipo de *softwares*

proporcionan a los sistemas industriales. De esta manera, se elaboró una simulación que imitaba el comportamiento básico de la planta.

La parte final, una vez entendida la importancia que puede tener un simulador en un entorno industrial como este y la utilidad de aplicar una optimización matemática a la planificación de tareas de la planta, consistía en poderlos unir. Esto fue posible gracias a las plantillas proporcionadas por Simio y la capacidad de conectividad de ambos.

Por un lado, creando un *Step* o paso a través de C# que modificara la lógica o comportamiento de la simulación. Consiguiendo así, que cada vez que Simio lo considerara necesario se pudiera producir una optimización de *scheduling*, a través de Julia.

Por otro lado, integrando el lenguaje de programación Julia dentro del archivo desarrollado en C#. Estableciendo además una estrategia de comunicación entre ambos para poder enviar, recibir y guardar los datos que fueran necesarios para el caso de estudio del trabajo.

De esta manera se tiene como resultado la unión de las dos herramientas, la simulación y el lenguaje de programación, aplicado al estudio de la mejora de una planta industrial real. Con ello, se consigue incluir un entorno de cálculo matemático potente dentro de un simulador orientado a procesos industriales, logística, servicios... Aunque en el trabajo se ha utilizado para poder ver cómo sería la implantación de una optimización de *scheduling* en una planta de esterilizado. Se abre un gran abanico de posibilidades tanto a la hora de mejorar y probar nuevas técnicas en este sistema en concreto, como de utilizarse en multitud de casos que se pueden dar en la industria. Pudiendo probarse distintos algoritmos, métodos de cálculo o cualquier modificación que pueda necesitar de un largo estudio y garantías dentro de una planta industrial.

7.2 Trabajo futuro

La continuación de este trabajo pasaría de manera inmediata en mejorar el estudio del caso utilizado en el trabajo. Adaptando los modelos matemáticos a la particularidad de la planta, reajustando las variables y el modelo de simulación para ceñirse a la realidad y en definitiva, poder hacer una comparación mucho más realista. De este modo poder ver si el sistema de

planificación de tareas que se usa ahora mismo es mejor que el que se podría utilizar sin tener que hacer pruebas directamente en el proceso. Viendo así, cómo de rentable puede ser integrar una optimización de *scheduling* a cortos periodos de predicción dentro de la misma.

Por otro lado, se abre la posibilidad de poder experimentar con cualquier tipo de metodología que requiera un cálculo matemático aplicado a un proceso industrial sin tener que aplicarlo directamente en el mismo. Un simulador de eventos discretos como Simio facilitará de manera considerable el estudio y la capacidad de aplicación de nuevas técnicas que puedan estar en tendencia. La comunicación entre el lenguaje de programación Julia y Simio, crea una herramienta que puede ser de mucha ayuda para el estudio de mejoras de plantas reales, ya sea en el mundo industrial o en el de la investigación.

Referencias

Bezanson, Jeff y col. (2017). "Julia: A fresh approach to numerical computing". En: *SIAM review* 59.1, págs. 65-98 (vid. pág. 28).

Dehghanimohammadabadi, Mohammad y Thomas K Keyser (2017). "Intelligent simulation: Integration of SIMIO and MATLAB to deploy decision support systems to simulation environment". En: *Simulation Modelling Practice and Theory* 71, págs. 45-60 (vid. pág. 59).

G. Palacín, Carlos, Pablo Riquelme y César de Prada (2019). "Scheduling óptimo de procesos Batch de duración interdependiente". En: *XL Jornadas de Automática*. Universidade da Coruña, Servizo de Publicacións, págs. 560-567 (vid. págs. 18, 19).

Georgiadis, Georgios P, Apostolos P Elekidis y Michael C Georgiadis (2019). "Optimization-based scheduling for the process industries: from theory to real-life industrial applications". En: *Processes* 7.7, pág. 438 (vid. págs. 2, 4, 17).

Georgiadis, Georgios P, Chrysovalantou Ziogou y col. (2018). "Production scheduling of multi-stage, multi-product food process industries". En: *Computer Aided Chemical Engineering*. Vol. 43. Elsevier, págs. 1075-1080 (vid. pág. 7).

Gómez, Carlos, César de Prada y José Luis Pitarch (2018). "Ayuda al operario en la distribución óptima de carga entre equipos equivalentes". En: *Actas de las XXXIX Jornadas de Automática, Badajoz, 5-7 de Septiembre de 2018* (vid. pág. 18).

Harjunkoski, Iiro y col. (2014). "Scope for industrial applications of production scheduling models and solution methods". En: *Computers & Chemical Engineering* 62, págs. 161-193 (vid. págs. 13, 15).

Houck, Dan (2018). *Using the Simio API*. URL: https://www.simio.com/resources/events/2018-User-Group-Meeting/presentations/API.pdf (visitado 15-08-2020) (vid. pág. 57).

ISA (2010). *Enterprise-Control System Integration - Part 1: Models and Terminology*. Standard ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod). American National Standard (vid. pág. 15).

Julia (2020). *Embedding Julia*. URL: https://docs.julialang.org/en/v1/manual/embedding/#High-Level-Embedding-on-Windows-with-Visual-Studio (visitado 21-08-2020) (vid. pág. 63).

Méndez, Carlos A y col. (2006). "State-of-the-art review of optimization methods for short-term scheduling of batch processes". En: *Computers & chemical engineering* 30.6-7, págs. 913-946 (vid. págs. 17, 18).

Microsoft (2018). *Programación orientada a objetos (C#)*. URL: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/object-oriented-programming (visitado 19-08-2020) (vid. pág. 61).

Palacín, Carlos G y César de Prada (2019). "Optimal Coordination Of Batch Processes with Shared Resources". En: *IFAC-PapersOnLine* 52.1, págs. 826-831 (vid. pág. 12).

Simio applications in scheduling (2015). IEEE, págs. 4150-4159 (vid. pág. 41).

Introduction to Simio (2016). IEEE, págs. 3594-3603 (vid. págs. 40, 43).

Simio (2019a). *Simio API Reference Guide*. Version 11.197.19514.0. Simio LLC (vid. pág. 62).

— (2019b). *Simio Reference Guide*. Version 11.197.19514.0. Simio LLC (vid. págs. 59, 61).

Steriflow (2016). *Static Steriflow water cascading*. URL: https://www.steriflow.com/en/autoclave-sterilization-static (visitado 06-07-2020) (vid. pág. 9).

Vieira, António y col. (2014). "Comparison of SIMIO and ARENA simulation tools". En: (vid. pág. 43).

Vilas, Carlos y Antonio A Alonso (2018). "Real time optimization of the sterilization process in a canning industry". En: *Actas de las XXXIX Jornadas de Automática, Badajoz, 5-7 de Septiembre de 2018* (vid. págs. 10, 11).

Winston, Wayne L y Jeffrey B Goldberg (2004). *Operations research: applications and algorithms*. Vol. 3. Thomson/Brooks/Cole Belmontˆ eCalif Calif (vid. pág. 19).

Yang, Wenhe y Soemon Takakuwa (2017). "Simulation-based dynamic shop floor scheduling for a flexible manufacturing system in the industry 4.0 environment". En: *2017 Winter Simulation Conference (WSC)*. IEEE, págs. 3908-3916 (vid. pág. 2).

Anexos

A Anexo I: Archivos de Julia utilizados en el Scheduling

A.1 *Opt_prec_general.jl:*

```
using JuMP
using CPLEX
using DataFrames, Gadfly
using Dates

prob=Model(CPLEX.Optimizer);
#set_optimizer_attributes(prob,"CPXPARAM_TimeLimit",50)

M=500;
n=100;
tau=30;
the=15;
tmc=75;
h_r=100;
ts=rand(Float64,n)*200;

I=collect(1:n);
J=collect(1:7);
U=collect(1:5);

@variable(prob,X[I,J],Bin);
@variable(prob,Z[J,U],Bin);
@variable(prob,Y[J,J],Bin);

@variable(prob,0 <= te[J] <= 200);
@variable(prob,0 <= tf[J] <= 300);
@variable(prob,MK >= 0);

@constraint(prob,[i in I], sum(X[i,j] for j in J) <= 1);
@constraint(prob,[j in J], sum(X[i,j] for i in I) <= 9);
@constraint(prob,[j in J], sum(X[i,j] for i in I) >= 1);
@constraint(prob,[j in J], sum(Z[j,u] for u in U) == 1 );

@constraint(prob,[i in I, j in J], te[j]>=ts[i]-M*(1-X[i,j]));
@constraint(prob,[i in I, j in J], te[j]<=ts[i]+tau+M*(1-X[i,j]));
@constraint(prob,[i in I], ts[i] >= h_r-M*(sum(X[i,j] for j in J)));
@constraint(prob,[j in J], tf[j]==te[j]+(the+tmc));

@constraint(prob,[j in J, jp in J, u in U; j != jp ], te[jp]>=tf[j]-M*(1-Y[j,jp])-M*(2-
    Z[j,u]-Z[jp,u]));
@constraint(prob,[j in J, jp in J, u in U; j != jp ], te[j]>=tf[jp]-M*(Y[j,jp])-M*(2-Z[
    j,u]-Z[jp,u]));
```

```
@constraint(prob,[j in J], MK>=tf[j]);

println("Funcion objetivo: Maximizar nUmero de carros")
@objective(prob,MOI.MAX_SENSE,sum(X[i,j] for i in I for j in J));
optimize!(prob);


println("SLOTS ASOCIADOS A AUTOCLAVES")

println("Tabla Slots-Autoclaves")

t_inicio=collect(Float64,1:length(J))
t_final=collect(Float64,1:length(J))
t_heat=collect(Float64,1:length(J))
t_heat1=collect(Float64,1:length(J))

for j in 1:length(J)
            t_inicio[j]= value.(te)[j]
            t_final[j]= value.(tf)[j]
             t_heat[j]= value.(te)[j]+the
            t_heat1[j]= t_heat[j]+1
        end

j_u=collect(1:length(J))
for j in 1:length(J)
    for u in 1:length(U)
        if value.(Z)[j,u] > 0.5
            j_u[j]=u
        else
        end
    end
end



D1 = DataFrame(Slot = J,
    t_inicio= t_inicio,
    t_final = t_final,
    Autoclave = j_u
)



println("Grafica Slots-Autoclaves")


ylab=string.(collect(1:length(J)))
D2 = DataFrame(y = j_u,
    x = t_inicio,
    xend = t_final,
    ylab=ylab,
    x2=t_heat,
    x3=t_heat1
)

ylabdict = Dict(i=>D2[:ylab][i] for i in 1:length(j_u))

barras=(3/maximum(j_u))*10mm;
coord = Coord.cartesian(ymin=0, ymax=0.8+maximum(j_u), xmin=0, xmax=maximum(t_final))

p = plot(D2, coord,
    layer(x=:x2, xend=:x3, y=:y, yend=:y ,Geom.segment, Theme(default_color="black",
        line_width=barras, line_style=[:ldash])),
    layer(x=:x, xend=:xend, y=:y, yend=:y, color=Scale.default_discrete_colors(length(J
        )),Geom.segment, Theme(line_width=barras)),
    Scale.y_continuous(labels=i->get(ylabdict,i,"")),
    Guide.xlabel("Time"), Guide.ylabel("Autoclave"), Guide.title("Slots en cada
        autoclave"),
    Guide.manual_color_key("Slot",ylab),
    Theme(key_position=:none)
)
```

```
@constraint(prob, sum(X[i,j] for i in I for j in J)>=objective_value(prob));
println("Funcion objetivo: MK")
@objective(prob,MOI.MIN_SENSE,MK);
optimize!(prob);

println("SLOTS ASOCIADOS A AUTOCLAVES")

println("Tabla Slots-Autoclaves")

for j in 1:length(J)
            t_inicio[j]= value.(te)[j]
            t_final[j]= value.(tf)[j]
             t_heat[j]= value.(te)[j]+the
            t_heat1[j]= t_heat[j]+1
        end

j_u=collect(1:length(J))
for j in 1:length(J)
    for u in 1:length(U)
        if value.(Z)[j,u] > 0.5
            j_u[j]=u
        else
        end
    end
end



D3 = DataFrame(Slot = J,
    t_inicio= t_inicio,
    t_final = t_final,
    Autoclave = j_u
)


println("Grafica Slots-Autoclaves")

ylab=string.(collect(1:length(J)))
D4 = DataFrame(y = j_u,
    x = t_inicio,
    xend = t_final,
    ylab=ylab,
    x2=t_heat,
    x3=t_heat1
)

ylabdict = Dict(i=>D4[:ylab][i] for i in 1:length(j_u))

barras=(3/maximum(j_u))*10mm;
coord = Coord.cartesian(ymin=0, ymax=0.8+maximum(j_u), xmin=0, xmax=maximum(t_final))

p1 = plot(D4, coord,
    layer(x=:x2, xend=:x3, y=:y, yend=:y ,Geom.segment, Theme(default_color="black",
        line_width=barras, line_style=[:ldash])),
    layer(x=:x, xend=:xend, y=:y, yend=:y, color=Scale.default_discrete_colors(length(J
        )),Geom.segment, Theme(line_width=barras)),
    Scale.y_continuous(labels=i->get(ylabdict,i,"")),
    Guide.xlabel("Time"), Guide.ylabel("Autoclave"), Guide.title("Slots en cada
        autoclave con MK minimizado"),
    Guide.manual_color_key("Slot",ylab),
    Theme(key_position=:none)
)

p1

println("INFORMACION DE LOS CARROS")

t_entrada=collect(Float64,1:length(ts))
for i in 1:length(ts)
            #println(value.(tf)[i])
            global t_entrada[i]= ts[i]
```

```
        end

i_j=collect(1:length(I))
for i in 1:length(I)
    for j in 1:length(J)
        if value.(X)[i,j] > 0.5
            i_j[i]=j
        else
        end
    end
    if sum(value.(X)[i,j] for j in J)==0
        i_j[i]=0
    else
    end
end

auto_i=collect(1:length(I))
for i in 1:length(I)
    if i_j[i]==0
        auto_i[i]=0
    else
        auto_i[i]=j_u[i_j[i]]
    end

end

t_inicio_i=collect(Float64,1:length(I))
t_esperamax=collect(Float64,1:length(I))
t_final_i=collect(Float64,1:length(I))

for i in 1:length(I)
    if i_j[i]==0
        t_inicio_i[i]=0
        t_esperamax[i]=t_entrada[i]-h_r
        t_final_i[i]=0
    else
        t_inicio_i[i]= value.(te)[i_j[i]]
        t_esperamax[i]=t_inicio_i[i]-t_entrada[i]
        t_final_i[i]= value.(tf)[i_j[i]]
    end
end


D5 = DataFrame(Carros = I,
    Entrada= t_entrada,
    Espera=t_esperamax,
    Slot = i_j,
    Autoclave=auto_i,
    Inicio=t_inicio_i,
    Salida=t_final_i
)
```

A.2 *Opt_prec_general_vapor.jl:*

```
using JuMP
using CPLEX
using DataFrames, Gadfly
using Dates

prob=Model(CPLEX.Optimizer);
#set_optimizer_attributes(prob,"CPXPARAM_TimeLimit"=>50)

M=500;
n=100;
tau=30;
the=15;
thp=15;
tmc=75;
```

```
h_r=100;
ts=rand(Float64,n)*200;

I=collect(1:n);
J=collect(1:7);
U=collect(1:5);

@variable(prob,X[I,J],Bin);
@variable(prob,Z[J,U],Bin);
@variable(prob,Y[J,J],Bin);
@variable(prob,W[J,J],Bin);

@variable(prob,0 <= te[J] <= 200);
@variable(prob,0 <= th[J] <= 40);
@variable(prob,0 <= tf[J] <= 300);
@variable(prob,MK >= 0);
@variable(prob,hmax >= 0);

@constraint(prob,[i in I], sum(X[i,j] for j in J) <= 1);
@constraint(prob,[j in J], sum(X[i,j] for i in I) <= 9);
@constraint(prob,[j in J], sum(X[i,j] for i in I) >= 1);
@constraint(prob,[j in J], sum(Z[j,u] for u in U) == 1);

@constraint(prob,[i in I, j in J], te[j]>=ts[i]-M*(1-X[i,j]));
@constraint(prob,[i in I, j in J], te[j]<=ts[i]+tau+M*(1-X[i,j]));
@constraint(prob,[i in I], ts[i] >= h_r-M*(sum(X[i,j] for j in J)));
@constraint(prob,[j in J], tf[j]==te[j]+(th[j]+tmc));

@constraint(prob,[j in J, jp in J, u in U; j != jp ], te[jp]>=tf[j]-M*(1-Y[j,jp])-M*(2-
    Z[j,u]-Z[jp,u]));
@constraint(prob,[j in J, jp in J, u in U; j != jp ], te[j]>=tf[jp]-M*(Y[j,jp])-M*(2-Z[
    j,u]-Z[jp,u]));

@constraint(prob,[j in J, jp in J; j != jp ], te[j]+th[j]>=te[jp]-M*(1-W[j,jp])-M*(1-Y[
    j,jp]));
@constraint(prob,[j in J, jp in J; j != jp ], te[j]+th[j]<=te[jp]+M*(W[j,jp])+M*(1-Y[j,
    jp]));
@constraint(prob,[j in J, jp in J; j != jp ], te[jp]+th[jp]>=te[j]-M*(1-W[j,jp])-M*(Y[j
    ,jp]));
@constraint(prob,[j in J, jp in J; j != jp ], te[jp]+th[jp]<=te[j]+M*(W[j,jp])+M*(Y[j,
    jp]));
@constraint(prob,[j in J], th[j]==the+thp*(sum(W[j,jp] for jp in J)));
@constraint(prob,[j in J, jp in J], W[j,jp]==W[jp,j]);
@constraint(prob,[j in J], W[j,j]==0);

@constraint(prob,[j in J], MK>=tf[j]);
@constraint(prob,[j in J], hmax>=th[j]);

println("Funcion objetivo: Maximizar numero de carros")
@objective(prob,MOI.MAX_SENSE,sum(X[i,j] for i in I for j in J));
optimize!(prob)


println("SLOTS ASOCIADOS A AUTOCLAVES")

println("Tabla Slots-Autoclaves")

t_inicio=collect(Float64,1:length(J))
t_heat=collect(Float64,1:length(J))
t_heat1=collect(Float64,1:length(J))
t_final=collect(Float64,1:length(J))

for j in 1:length(J)
            t_inicio[j]= value.(te)[j]
            t_heat[j]= value.(th)[j]+t_inicio[j]
            t_heat1[j]= t_heat[j]+1
            t_final[j]= value.(tf)[j]
        end

j_u=collect(1:length(J))
for j in 1:length(J)
    for u in 1:length(U)
```

```
        if value.(Z)[j,u] > 0.5
            j_u[j]=u
        else
        end
    end
end



D1 = DataFrame(Slot = J,
    t_inicio= t_inicio,
    t_heating=t_heat,
    t_final = t_final,
    Autoclave = j_u
)

println("Grafica Slots-Autoclaves")

ylab=string.(collect(1:length(J)))
D2 = DataFrame(y = j_u,
    x = t_inicio,
    xend = t_final,
    ylab=ylab,
    x2=t_heat,
    x3=t_heat1
)

ylabdict = Dict(i=>D2[:ylab][i] for i in 1:length(j_u))

barras=(3/maximum(j_u))*10mm;
coord = Coord.cartesian(ymin=0, ymax=0.8+maximum(j_u), xmin=0, xmax=maximum(t_final))

p = plot(D2, coord,
    layer(x=:x2, xend=:x3, y=:y, yend=:y ,Geom.segment, Theme(default_color="black",
        line_width=barras, line_style=[:ldash])),
    layer(x=:x, xend=:xend, y=:y, yend=:y, color=Scale.default_discrete_colors(length(J
        )),Geom.segment, Theme(line_width=barras)),
    Scale.y_continuous(labels=i->get(ylabdict,i,"")),
    Guide.xlabel("Time"), Guide.ylabel("Autoclave"), Guide.title("Slots en cada
        autoclave"),
    Guide.manual_color_key("Slot",ylab),
    Theme(key_position=:none)
)


@constraint(prob, sum(X[i,j] for i in I for j in J)>=objective_value(prob))
println("Funcion objetivo: MK")
#@objective(prob,MOI.MIN_SENSE,MK+hmax);
@objective(prob,MOI.MIN_SENSE,MK);
optimize!(prob)


println("SLOTS ASOCIADOS A AUTOCLAVES")

println("Tabla Slots-Autoclaves")

for j in 1:length(J)
            t_inicio[j]= value.(te)[j]
            t_heat[j]= value.(th)[j]+t_inicio[j]
            t_heat1[j]= t_heat[j]+1
            t_final[j]= value.(tf)[j]
        end

j_u=collect(1:length(J))
for j in 1:length(J)
    for u in 1:length(U)
        if value.(Z)[j,u] > 0.5
            j_u[j]=u
        else
        end
    end
end
```

```
D3 = DataFrame(Slot = J,
    t_inicio= t_inicio,
    t_heating=t_heat,
    t_final = t_final,
    Autoclave = j_u
)

println("Grafica Slots-Autoclaves")

ylab=string.(collect(1:length(J)))
D4 = DataFrame(y = j_u,
    x = t_inicio,
    xend = t_final,
    ylab=ylab,
    x2=t_heat,
    x3=t_heat1
)

ylabdict = Dict(i=>D4[:ylab][i] for i in 1:length(j_u))

barras=(3/maximum(j_u))*10mm;
coord = Coord.cartesian(ymin=0, ymax=0.8+maximum(j_u), xmin=0, xmax=maximum(t_final))

p1 = plot(D4, coord,
    layer(x=:x2, xend=:x3, y=:y, yend=:y ,Geom.segment, Theme(default_color="black",
        line_width=barras, line_style=[:ldash])),
    layer(x=:x, xend=:xend, y=:y, yend=:y, color=Scale.default_discrete_colors(length(J
        )),Geom.segment, Theme(line_width=barras)),
    Scale.y_continuous(labels=i->get(ylabdict,i,"")),
    Guide.xlabel("Time"), Guide.ylabel("Autoclave"), Guide.title("Slots en cada
        autoclave con MK minimizado"),
    Guide.manual_color_key("Slot",ylab),
    Theme(key_position=:none)
)


println("INFORMACION DE LOS CARROS")

t_entrada=collect(Float64,1:length(ts))
for i in 1:length(ts)
            #println(value.(tf)[i])
            global t_entrada[i]= ts[i]
        end

i_j=collect(1:length(I))
for i in 1:length(I)
    for j in 1:length(J)
        if value.(X)[i,j] > 0.5
            i_j[i]=j
        else
        end
    end
    if sum(value.(X)[i,j] for j in J)==0
        i_j[i]=0
    else
    end
end

auto_i=collect(1:length(I))
for i in 1:length(I)
    if i_j[i]==0
        auto_i[i]=0
    else
        auto_i[i]=j_u[i_j[i]]
    end

end

t_inicio_i=collect(Float64,1:length(I))
```

```
t_esperamax=collect(Float64,1:length(I))
t_heating_i=collect(Float64,1:length(I))
t_fheating_i=collect(Float64,1:length(I))
t_final_i=collect(Float64,1:length(I))

for i in 1:length(I)
    if i_j[i]==0
        t_inicio_i[i]=0
        t_esperamax[i]=t_entrada[i]-h_r
        t_heating_i[i]=0
        t_fheating_i[i]=0
        t_final_i[i]=0
    else
        t_inicio_i[i]= value.(te)[i_j[i]]
        t_esperamax[i]=t_inicio_i[i]-t_entrada[i]
        t_heating_i[i]= value.(th)[i_j[i]]
        t_fheating_i[i]= value.(th)[i_j[i]]+t_inicio_i[i]
        t_final_i[i]= value.(tf)[i_j[i]]
    end
end


D5 = DataFrame(Carros = I,
    Entrada= t_entrada,
    Espera=t_esperamax,
    Slot = i_j,
    Autoclave=auto_i,
    Inicio=t_inicio_i,
    Dur_heat=t_heating_i,
    Final=t_fheating_i,
    Salida=t_final_i
)
```

B Anexo II: Archivos del Visual Studio utilizados en la conexión Simio Julia

B.1 UserElement.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using SimioAPI;
using SimioAPI.Extensions;

namespace UserDefinedStepAndElement3
{
    class UserElementDefinition : IElementDefinition
    {
        #region IElementDefinition Members

        /// <summary>
        /// Property returning the full name for this type of element. The name should
            contain no spaces.
        /// </summary>
        public string Name
        {
            get { return "UserElement"; }
        }

        /// <summary>
        /// Property returning a short description of what the element does.
        /// </summary>
        public string Description
        {
            get { return "Description text for the 'UserElement' element."; }
        }

        /// <summary>
        /// Property returning an icon to display for the element in the UI.
        /// </summary>
        public System.Drawing.Image Icon
        {
            get { return null; }
        }

        /// <summary>
```

```
        /// Property returning a unique static GUID for the element.
        /// </summary>
        public Guid UniqueID
        {
            get { return MY_ID; }
        }
        public static readonly Guid MY_ID = new Guid("{430000d2-bf56-456e-aef6-30
            fe47b0bfd5}");

        /// <summary>
        /// Method called that defines the property, state, and event schema for the
            element.
        /// </summary>
        public void DefineSchema(IElementSchema schema)
        {
            // Example of how to add a property definition to the element.
            IPropertyDefinition pd;
            pd = schema.PropertyDefinitions.AddExpressionProperty("MyExpression", "0.0"
                );
            pd.DisplayName = "My Expression";
            pd.Description = "An expression property for this element.";
            pd.Required = true;

            // Example of how to add a state definition to the element.
            IStateDefinition sd;
            sd = schema.StateDefinitions.AddState("MyState");
            sd.Description = "A state owned by this element";

            // Example of how to add an event definition to the element.
            IEventDefinition ed;
            ed = schema.EventDefinitions.AddEvent("MyEvent");
            ed.Description = "An event owned by this element";
        }

        /// <summary>
        /// Method called to add a new instance of this element type to a model.
        /// Returns an instance of the class implementing the IElement interface.
        /// </summary>
        public IElement CreateElement(IElementData data)
        {
            return new UserElement(data);
        }

        #endregion
    }

    class UserElement : IElement
    {
        IElementData _data;

        public UserElement(IElementData data)
        {
            _data = data;
        }

        #region IElement Members

        /// <summary>
        /// Method called when the simulation run is initialized.
        /// </summary>
        public void Initialize()
        {
        }

        /// <summary>
        /// Method called when the simulation run is terminating.
        /// </summary>
        public void Shutdown()
        {
        }

        #endregion
```

```
    }
}
```

B.2 UserStep.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading.Tasks;
using System.Runtime.InteropServices;
using SimioAPI;
using SimioAPI.Extensions;

namespace Simio_opt
{
    class UserStepDefinition : IStepDefinition
    {
        #region IStepDefinition Members

        /// <summary>
        /// Property returning the full name for this type of step. The name should
            contain no spaces.
        /// </summary>
        public string Name
        {
            get { return "Simio_Julia"; }
        }

        /// <summary>
        /// Property returning a short description of what the step does.
        /// </summary>
        public string Description
        {
            get { return "Step que indica a Simio el servidor al que tiene que ir la
                entidad a traves de una optimizacion en Julia. Se le debe proporcionar
                una variable estado que contenga el tipo de entidad, el peso del path
                que forme el camino y la ruta al archivo de Julia."; }
        }

        /// <summary>
        /// Property returning an icon to display for the step in the UI.
        /// </summary>
        public System.Drawing.Image Icon
        {
            get { return null; }
        }

        /// <summary>
        /// Property returning a unique static GUID for the step.
        /// </summary>
        public Guid UniqueID
        {
            get { return MY_ID; }
        }
        static readonly Guid MY_ID = new Guid("{e216ff81-f981-4951-b3c4-8b6fea952fc5}")
            ;

        /// <summary>
        /// Property returning the number of exits out of the step. Can return either 1
             or 2.
        /// </summary>
        public int NumberOfExits
        {
            get { return 1; }
        }
```

```
        /// <summary>
        /// Method called that defines the property schema for the step.
        /// </summary>
        public void DefineSchema(IPropertyDefinitions schema)
        {
            // Propiedades que va a tener el Step
            IPropertyDefinition pd1;
            IPropertyDefinition pd2;
            IPropertyDefinition pd3;
            IPropertyDefinition pd4;
            IPropertyDefinition pd5;
            IPropertyDefinition pd6;
            IPropertyDefinition pd7;
            IPropertyDefinition pd8;

            // Informacion de las propiedades que va a tener el bloque Step
            pd1 = schema.AddStateProperty("EntityType");
            pd1.DisplayName = "Tipo de entidad";

            pd2 = schema.AddStateProperty("tiempo");
            pd2.DisplayName = "Tiempo de simulacion";
            pd2.Description = "Tiempo de simulacion en segundos";

            pd3 = schema.AddStateProperty("Peso1");
            pd3.DisplayName = "Peso del path 1";

            pd4 = schema.AddStateProperty("Peso2");
            pd4.DisplayName = "Peso del path 2";

            pd5= schema.AddStateProperty("Peso3");
            pd5.DisplayName = "Peso del path 3";

            pd6 = schema.AddStateProperty("Peso4");
            pd6.DisplayName = "Peso del path 4";

            pd7 = schema.AddStateProperty("Peso5");
            pd7.DisplayName = "Peso del path 5";

            pd8 = schema.AddStringProperty("Path", string.Empty);
            pd8.DisplayName = "Path";
            pd8.Description = "Ruta del archivo de Julia (.jl) al que se quiere llamar
                de la forma: C:/Users/Usuario/Documents/fichero.jl";

        }

        /// <summary>
        /// Method called to create a new instance of this step type to place in a
            process.
        /// Returns an instance of the class implementing the IStep interface.
        /// </summary>
        public IStep CreateStep(IPropertyReaders properties)
        {
            return new UserStep(properties);
        }

        #endregion
    }


    unsafe class UserStep : IStep
    {
        [DllImport("kernel32.dll")]
        public static extern bool SetDllDirectory(string pathName);

        [DllImport("libjulia.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern void jl_init__threading(string julia_home_dir);

        [DllImport("libjulia.dll", CallingConvention = CallingConvention.Cdecl)]
        private static extern IntPtr jl_eval_string(string str);

        [DllImport("libjulia.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern void jl_atexit_hook(int a);
```

```
[DllImport("libjulia.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void* jl_array_ptr(IntPtr array);

IPropertyReaders _properties;
IPropertyReader _PathName;

public UserStep(IPropertyReaders properties)
{
    _properties = properties;
    _PathName = _properties.GetProperty("Path");

}

#region IStep Members

/// <summary>
/// Method called when a process token executes the step.
/// </summary>
public ExitType Execute(IStepExecutionContext context)
{
    // Obtener la propiedad de una determinada variable de estado (States)
    IStateProperty stateProp1 = _properties.GetProperty("EntityType") as
        IStateProperty;
    IState state1 = stateProp1.GetState(context);

    IStateProperty stateProp2 = _properties.GetProperty("tiempo") as
        IStateProperty;
    IState state2 = stateProp2.GetState(context);

    IStateProperty stateProp3 = _properties.GetProperty("Peso1") as
        IStateProperty;
    IState state3 = stateProp3.GetState(context);

    IStateProperty stateProp4 = _properties.GetProperty("Peso2") as
        IStateProperty;
    IState state4 = stateProp4.GetState(context);

    IStateProperty stateProp5 = _properties.GetProperty("Peso3") as
        IStateProperty;
    IState state5 = stateProp5.GetState(context);

    IStateProperty stateProp6 = _properties.GetProperty("Peso4") as
        IStateProperty;
    IState state6 = stateProp6.GetState(context);

    IStateProperty stateProp7 = _properties.GetProperty("Peso5") as
        IStateProperty;
    IState state7 = stateProp7.GetState(context);

    // Obtener la cadena de caracteres de la ruta del archivo de Julia (.jl)
    String varPath = _PathName.GetStringValue(context);

    string juliaDir = "C:\\Users\\Usuario\\AppData\\Local\\Programs\\Julia\\
        Julia-1.4.2\\bin";
    SetDllDirectory(juliaDir);

    // Abrir Julia
    jl_init__threading(juliaDir);

    // Indicar el tiempo de simulacion en segundo a Julia
    string time = "time=" + state2.StateValue.ToString();
    jl_eval_string(time);

    // Indicar el tipo de lote por el que se ha producido el evento y definirlo
         en Julia
    string tipo = "tipo_1=" + state1.StateValue.ToString();
    jl_eval_string(tipo);

    // Ejecutar el fichero .jl
    string pathname = "include(\"" + varPath + "\")";
    jl_eval_string(pathname);
```

```
            // Extraer el autoclave seleccionado
            IntPtr autoclave_sel = jl_eval_string("autoclave_sel");

            int* res_auto = (int*)autoclave_sel;

            // Cerrar julia
            jl_atexit_hook(0);


            // Activar el path de autoclave_sel y dejar los otros caminos con peso 0.
            if (*res_auto == 1)
            {
                state3.StateValue = 1;
                state4.StateValue = 0;
                state5.StateValue = 0;
                state6.StateValue = 0;
                state7.StateValue = 0;
                MessageBox.Show("Autoclave 1");
            }
            else if (*res_auto == 2)
            {
                state3.StateValue = 0;
                state4.StateValue = 1;
                state5.StateValue = 0;
                state6.StateValue = 0;
                state7.StateValue = 0;
                MessageBox.Show("Autoclave 2");
            }
            else if (*res_auto == 3)
            {
                state3.StateValue = 0;
                state4.StateValue = 0;
                state5.StateValue = 1;
                state6.StateValue = 0;
                state7.StateValue = 0;
                MessageBox.Show("Autoclave 3");
            }
            else if (*res_auto == 4)
            {
                state3.StateValue = 0;
                state4.StateValue = 0;
                state5.StateValue = 0;
                state6.StateValue = 1;
                state7.StateValue = 0;
                MessageBox.Show("Autoclave 4");
            }
            else
            {
                state3.StateValue = 0;
                state4.StateValue = 0;
                state5.StateValue = 0;
                state6.StateValue = 0;
                state7.StateValue = 1;
                MessageBox.Show("Autoclave 5");
            }

            return ExitType.FirstExit;
        }

        #endregion
    }
}
```

B.3 julia_simio.jl

```
using JuMP
using Cbc
using JSON
using Dates
ahora=DateTime(2020,9,9,0,0,0,0) + Dates.Second(time);

#Informacion=Dict("Esterilizador_oc"=>[0,0,0,0,0])
#open("basedatos.json","w") do j
#     write(j,JSON.json(Informacion))
#end

open("basedatos.json", "r") do f
    global Informacion_lec
    Informacion_lec=JSON.parse(f);
end

ocupados=Informacion_lec["Esterilizador_oc"];

prob=Model(Cbc.Optimizer);
set_optimizer_attribute(prob, "threads", 4);

#Frecuencia slot
if tipo_1==1;
    frec_slot_a=collect(StepRange(0, 40, 180));
    frec_slot_b=collect(StepRange(20, 40, 180));
else
    frec_slot_a=collect(StepRange(20, 40, 180));
    frec_slot_b=collect(StepRange(0, 40, 180));
end;

ts=collect(1:(length(frec_slot_a)+length(frec_slot_b)));
t_t=collect(1:(length(frec_slot_a)+length(frec_slot_b)));

pos=0;
for i in collect(1:length(frec_slot_a))
        if frec_slot_a[i]<frec_slot_b[i]
                ts[i+pos]=frec_slot_a[i]
                ts[i+1+pos]=frec_slot_b[i]
                t_t[i+pos]=1
                t_t[i+1+pos]=2
    else
                ts[i+pos]=frec_slot_b[i]
        ts[i+1+pos]=frec_slot_a[i]
        t_t[i+pos]=2
        t_t[i+1+pos]=1
    end
    global pos=i;
end

M=500;
tau=15;
the=15;
J=collect(1:8);
U=collect(1:5);
tipo=collect(1:2);
tmc=[50,60];
@variable(prob,Z[J,U],Bin);
@variable(prob,Y[J,J],Bin);
@variable(prob,T[J,tipo],Bin);
@variable(prob,0 <= te[J] <= 200);
@variable(prob,0 <= t_tipo[J] <= 100);
@variable(prob,0 <= tf[J] <= 300);
@variable(prob,MK >= 0);
@constraint(prob,[j in J], sum(Z[j,u] for u in U) == 1 );
@constraint(prob,[j in J], sum(T[j,t] for t in tipo) == 1);
@constraint(prob,[j in J], sum(t*(T[j,t]) for t in tipo)==t_t[j]);
@constraint(prob,[j in J], tf[j]==te[j]+(the+t_tipo[j]));
@constraint(prob,[j in J, jp in J, u in U; j != jp ], te[jp]>=tf[j]-M*(1-Y[j,jp])-M*(2-
    Z[j,u]-Z[jp,u]));
```

```
@constraint(prob,[j in J, jp in J, u in U; j != jp ], te[j]>=tf[jp]-M*(Y[j,jp])-M*(2-Z[
    j,u]-Z[jp,u]));
@constraint(prob,[j in J], te[j]>=ts[j]);
@constraint(prob,[j in J], te[j]<=ts[j]+tau);
@constraint(prob,[j in J], t_tipo[j]==sum(tmc[t]*(T[j,t]) for t in tipo));
@constraint(prob,[j in J], MK>=tf[j]);
@constraint(prob,[j in J], te[1]<=5);

for i in 1:length(U)
    if ocupados[i] != 0
       ocupados[i]=(Dates.value((Dates.DateTime(ocupados[i]))-ahora))/60000

    end
end

for u in 1:length(U)
    if ocupados[u]>0
        @constraint(prob,[j in J,u in U], te[j]>=ocupados[u]*Z[j,u]);
    else
        ocupados[u]=0
    end
end

@objective(prob,MOI.MIN_SENSE,MK);
optimize!(prob)


for u in 1:length(U)
    if value.(Z)[1,u]>0.5
      global autoclave_sel=u
    else
    end
end
ocupados[autoclave_sel]=round(value.(tf)[1],digits=3);
for u in 1:length(U)
    if ocupados[u]!=0
        ocupados[u]=ahora+Dates.Millisecond(ocupados[u]*60000)
    end
end

Informacion=Dict("Esterilizador_oc"=>ocupados)
open("basedatos.json","w") do j
    write(j,JSON.json(Informacion))
end
```