UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de software auto-adaptativo. Una aplicación industrial práctica

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Miguel García Valero

Tutor: Joan Fons i Cors

Curso 2019-2020

Resumen

Los sistemas informáticos actuales son entidades cada vez más complejas debido a las tecnologías y arquitecturas modernas. Ya es habitual que se articulen en forma de ecosistemas de soluciones, creando dependencias entre este gran número de diferentes sistemas que constituyen las soluciones. Esta complejidad se agrava cuando se involucran procesos físicos que hay que integrar junto a estos sistemas informáticos, debido a la naturaleza dinámica y heterogénea de estos sistemas, entre otros. Por ejemplo, en el ámbito de la industria, se desarrollan sistemas informáticos que son capaces de dar soporte y estar sincronizados con los propios procesos de producción.

Se requiere que estos sistemas funcionen ininterrumpidamente, y que sean capaces de ser conscientes de las diferentes situaciones que ocurran en su contexto (por ejemplo, en el ámbito productivo), para operar de manera resiliente y lo más óptimo posible bajo parámetros operativos cambiantes. Sin embargo, en la actualidad no es habitual diseñar y desarrollar software con capacidades de computación autónoma, que pueda adaptarse a estos entornos dinámicos. La adaptatividad es la capacidad de un sistema de cambiar, mutar o reconfigurarse para funcionar de manera óptima en función de su contexto operativo.

En este proyecto se pretenden integrar, desde un punto de vista práctico, técnicas que provienen de la teoría de control, utilizando bucles de control para incorporar capacidades de adaptación a los sistemas. En concreto, se usará la aproximación FADA (desarrollada por el grupo TaTami del Centro de investigación PROS) que propone enfoque práctico para desarrollar software auto-adaptativo aplicando los conceptos de la computación autónoma a través de bucles de control MAPE-K. Se analizará un problema proveniente de un ámbito industrial, y se diseñará una solución auto-adaptativa para este problema.

Como resultado, el proyecto propondrá tanto un diseño software que incorpore estas capacidades de adaptación, y desarrollará un prototipo funcional (usando el framework de implementación FADA) que siga la propuesta realizada.

Palabras clave: Computación Autónoma, Software auto-adaptativo, Bucles de Control, MAPE-K, Fábricas del Futuro

Abstract

Contemporary information systems are increasingly complex entities due to modern technologies and architectures. Often, they are articulated as ecosystems of solutions, where a number of dependencies appear between this many different systems that constitute the solutions. This complexity is being aggravated when physical processes are involved in integrating this systems, on account of dynamic, heterogeneous nature of these systems, among others. For example, at the industry field, systems capable of providing support and syncronize with production processes are being developed.

It is required that these systems run continuously, and they must be aware of different situations that may occur around them (for example, at a production scope) in order to operate in a flexible way, and stay optimized under changing operative parameters. Nevertheless, we usually do not design and develop software with autonomic computing capabilities, which can adapt to this dynamic environments. Adaptativity is the ability of a system for changing, mutating or reconfigurating in order to work at peak performance in correlation to its operating context.

This proyect aims at integrating, from a practical perspective, techniques that come from control theory, using control loops for incorporating adaptation capabilities to systems. Specifically, we will use FADA approach (developed by group TaTami form PROS investigation centre), that provides a practical standpoint for developing self-adaptative software applying autonomous computing concepts through MAPE-K control loops. We will analyse a problem from a industry field, and we will design an auto-adaptative solution for this problem.

Como resultado, el proyecto propondrá tanto un diseño software que incorpore estas capacidades de adaptación, y desarrollará un prototipo funcional (usando el framework de implementación FADA) que siga la propuesta realizada.

As result, this proyect will propose a software design that incorporates this adaptation capabilities, and will develop a functional prototype (making use of FADA framework) that follows the conducted proposal.

Key words: Autonomic Computing, Auto-adaptative software, Control Loops, MAPE-k, Factories of the future

Índice general

Índice de figuras

Índice de tablas

Índice de fragmentos de código

CAPÍTULO 1
Introducción

En ambientes industriales, donde diferentes máquinas de producción, operarios, dispositivos, sensores, etc. colaboran en la producción, es vital que el software que da soporte a estos procesos sea consciente del contexto productivo existente. Es decir, el software debe ofrecer mecanismos para adaptarse a las diferentes situaciones que puedan ocurrir en la producción (falta de materia prima, problemas en la cadena de producción, operarios no disponibles para colaborar, etc.), de manera que en todo momento sea capaz de (auto-)configurarse para atender a estas situaciones.

Dicho de otro modo, estos sistemas o entornos no están exentos de fallos, dada la complejidad inherente a los mismos. Los sistemas informáticos deben estar preparados para afrontar estas situaciones, ofreciendo en cada momento el contexto operacional más favorable, adaptando la configuración de los mismos para amoldarse al entorno en el que se encuentran. Igualmente, en un entorno industrial de producción, será vital que la gestión de las incidencias se resuelva de la manera más rápida, ágil y menos intrusiva, y, a ser posible, que se realice automáticamente o de manera autónoma.

Este tipo de software requiere integrar dispositivos, procesos y recursos físicos con los sistemas software de gestión de los procesos digitales, por lo que suelen ser desarrollos complejos. Además del reto de integración que acabamos de mencionar, existe otro tipo de dificultades en estos sistemas de soporte: en estos escenarios es común que algún recurso necesario para completar una tarea no esté disponible, que se esté operando con niveles altos de estrés, que las lecturas repostadas por los sensores o dispositivos físicos sean erróneas, etc.

Todo esto nos lleva a diseñar entornos de producción que implementen ciertas capacidades de computación autónoma, donde el sistema, por ejemplo una fábrica, sea capaz de organizar, monitorizar y planificar tareas y procesos productivos por sí mismo. Los operarios son recursos clave en estas planificaciones, y el sistema autónomo es capaz de distribuir tareas entre estos. En este tipo de sistemas se debe cuidar la comunicación entre los recursos físicos y digitales de la fábrica, con sus operarios, tratando de conseguir en todo momento un entorno de colaboración consciente entre ambos: la fábrica sabe que puede contar con el operario para realizar las diferentes tareas, y el operario entiende las tareas que la fábrica requiere.

Estas cuestiones son complejas de gestionar, y deben abordarse desde un punto de vista ingenieril. Para ejemplificar el trabajo desarrollado en este proyecto, nos centramos en una parte de la producción de una empresa que trabaja en la elaboración de cantoneras de cartón a partir de bobinas de papel. Esta línea de la producción consiste en máquinas operadas por un trabajador, que cortan y doblan el papel para elaborar las cantoneras. Las máquinas tienen la capacidad de reportar a qué velocidad están produciendo. La cantidad de papel desechado también es comunicado, midiendo el nivel del contenedor de desechos.e

El sistema de soporte a la producción debe integrar, por una parte, la información física generada por las máquinas de producción con los sistemas software de gestión de recursos de producción. Por otra parte, debe asistir, utilizando diferentes tipos de interfaces de usuario a los operarios. Y todo esto, en un entorno industrial, donde los procesos físicos no están exentos de incidencias, fallos o anomalías productivas.

Nos centraremos en diseñar y prototipar un pequeño sistema con capacidades de computación autónoma que sea capaz de facilitar la interacción entre las máquinas de la línea de producción de las cantoneras y los operarios, de manera que, en todo momento, además de recoger datos de producción, el sistema pueda atender a las posibles incidencias, tanto físicas como digitales, que ocurran durante esta, de manera autónoma, sin requerir que un operario se encargue de solucionar estos problemas. De esta forma, conseguiremos que el sistema se adapte al contexto de la producción, ahorrándonos estas intervenciones humanas, que son, claramente, más costosas en tiempo invertido para solucionar las situaciones anómalas.

Como se ha comentado antes, el contexto de producción deberá ofrecer mecanismos para que los operarios de cada línea sean conscientes de la situación de producción real y puedan saber (ser conscientes), de manera natural e intuitiva, de incidencias que están aconteciendo (y de qué tipo), o de que todo está funcionando como se espera.

Así pues, el objetivo es dotar de capacidades de auto-adaptación a un semáforo que comunica al operador de una máquina de producción este contexto productivo. Se decide usar la metáfora del semáforo como mecanismo de interacción intuitivo, y que a la vez, ofrece una interfaz visual fácilmente accesible en un entorno real.

Para ello, tendremos que integrar la información del software de gestión de recursos, que reporta las velocidades de producción esperadas o necesarias para cumplir los objetivos de producción, con la información física que reporta la máquina, entre otros datos disponibles en el entorno de la línea de producción. El semáforo indicará, mediante combinaciones de valores, el estado de la producción.

Pero también debemos abordar los problemas que puedan surgir en la línea de producción desde el punto de vista de la computación autónoma.

Estos problemas o situaciones que se deberían tener en cuenta pueden ser de diferente índole. Por ejemplo, podríamos clasificar en errores físicos: agotamiento de materia prima, fallo en la máquina de producción, etc. Otra división sería de errores digitales, causados por fallos en el software: fallos en las conexiones,

reconfiguración... Pueden existir más tipos de situaciones, como las relacionadas con los operarios: ausencia o presencia de los mismos, cambios en la conducta, cambios en el nivel de distracción, etcétera. Estos no son todos los tipos de problemas que se pueden encontrar en un entorno de producción, pero estos ejemplos nos muestran cómo, en estos contextos, existe una alta complejidad y hay una gran cantidad de situaciones que debemos abordar mediante técnicas de computación autónoma.

Uno de los errores o situaciones que queremos abordar son, por ejemplo, los fallos de conexión con el resto de piezas que intervienen en la integración del software de soporte. Típicamente, estas situaciones supondrían la intervención de un humano, que reconfigurase la solución, la reiniciase, etc., provocando así una interrupción en el servicio que este software otorga a la línea de producción. A priori, puede parecer que estas mediaciones humanas no sean muy costosas, pero, como ya hemos comentado, en los entornos industriales, estos fallos son comunes. Y no solo se invierte tiempo en reconfigurar manualmente la solución, si no que el servicio es interrumpido hasta que una persona advierta esta situación anómala, produciendo así confusiones/malentendidos si el software de soporte queda en un estado no consistente con la realidad. Esto puede llegar a causar frustración en los usuarios perjudicados, que podrían incluso desechar o descartar estas soluciones, debido a que producen más inconvenientes que beneficios, por lo que es necesario minimizar el impacto de estos problemas. Abogamos por el uso de la computación autónoma para ello, logrando así que el sistema de soporte se auto-configure o auto-adapte a las situaciones irregulares que puedan ocurrir en el entorno de producción.

En este desarrollo, abordaremos la cuestión de la auto-adaptabilidad mediante un bucle de adaptación MAPE-k, que será descrito en detalle en un capítulo posterior. Los bucles MAPE-k nos permiten que los sistemas a los que los aplicamos sean capaces de auto-configurarse, cambiando atributos internos o poniendo y quitando componentes completos en tiempo de ejecución, en base a un conjunto de reglas y conocimiento establecidos. El reto en este tipo de soluciones es controlar esas reglas que alteran la configuración del sistema, para que en todo momento se ofrezca un estado del sistema que sea lo más operacional posible, y que esas reglas no «interfieran» con otras. El uso de estos bucles de adaptación no estará restringido a los semáforos informativos, cualquier pieza de software es susceptible de ser auto-adaptable. Nos centraremos en el desarrollo y auto-adaptabilidad del semáforo, pero sería posible, aplicar un bucle de adaptación a la máquina de producción, o cualquier otro sistema de la fábrica, de forma simultánea. Es decir, bucles se pueden desplegar deforma independiente, ortogonal y orquestada en distintos sistemas software de un mismo entorno.

1.1 Motivación

La alta complejidad de los sistemas informáticos, especialmente en entornos industriales, ha desembocado en un alto coste de configuración y mantenimiento de los mismos. La computación autónoma surge como respuesta a esta complejidad, proponiendo sistemas que se auto-configuran y se autoreparan, eliminando

así el tiempo dedicado al despliegue, configuración y reconfiguración de los sistemas.

En esta memoria se abordan conceptos sobre la computación autónoma y se detalla el proceso para refactorizar una solución existente en la industria e introducir en ella capacidades auto-adaptativas. Se persigue obtener un nuevo prototipo de sistema que tenga capacidades de auto-gestión a través de la aplicación de estos conceptos de computación autónoma.

1.2 Problemática

En esta sección explicaremos los requisitos de adaptación que se pueden encontrar en estos entornos. Estos requisitos son usuales en los entornos industriales de producción en general, y son susceptibles de verse afrontados mediante capacidades de auto-adaptación.

Para empezar, se requiere que los sistemas de soporte a la producción sean conscientes del estado de las conexiones con el resto de componentes software con los que se integra. La disponibilidad de los datos que proporcionan los diferentes procesos de una fábrica es esencial para el correcto funcionamiento de los sistemas de soporte, ya que la ausencia de alguno de ellos puede conducir a errores o fallos en las estimaciones de producción, por ejemplo.

Llevando más allá el aspecto de las conexiones, puede ser recomendable que existan varias formas de comunicación con el sistema, y que estas se intercambien según el contexto, de forma que se aumente el dominio de casos en los que existe comunicación con el sistema.

Si se da el caso de que las conexiones no se pierden, pero no se recibe información, ya sea de los sistemas físicos, o bien de los sistemas software con los que se integra el sistema de soporte, este podría quedar en un estado inconsistente con la realidad, pudiendo confundir a los operarios con los que interactúa. El sistema debe configurarse de forma que los operarios sean conscientes de la situación, y no muestre información incoherente, potencialmente inadecuada a la realidad de producción.

Otra situación que se puede dar es que las interfaces físicas del sistema dejen de funcionar. Esto supondría la perdida de comunicación con los operarios, quedando el sistema inhabilitado. El software debe adaptarse a esta situación, utilizando otra interfaz para interactuar con los operarios, ya sea mediante el uso de una interfaz de respaldo, o bien adaptando la comunicación a otros medios, por ejemplo, usando interfaces sonoras.

Generalizando la situación anterior, los sistemas deben soportar errores en los dispositivos físicos, así como la ausencia de los mismos. Las máquinas pueden estar reportando información ruidosa, o información no coherente, por ejemplo, si una máquina se atasca pero sigue publicando datos que están, probablemente, desvinculados de la realidad de la producción.

También debemos tener en cuenta los posibles cambios en el comportamiento de los operarios. Las máquinas de producción son herramientas que consumen grandes cantidades de energía, y siempre deberíamos apuntar hacia la eficiencia

energética. Cuando un operario se ausenta y deja de utilizar la máquina, esta debe entrar en un estado de ahorro energético, o apagarse, con el propósito de no desperdiciar energía. Esta cuestión es compleja de manejar, ya que el arranque de las maquinarias industriales pueden consumir mucho tiempo, o incluso suponer un gasto energético mayor al que se ahorraría apagándola, por lo que este caso debe ser considerado en cada entorno de producción, para decidir si es viable o no.

Siguiendo la línea del caso anterior, en líneas de producción en las que las máquinas puedan suponer un peligro para el operario si este no la usa debidamente, por ejemplo, si no presta la suficiente atención, las máquinas deberían bloquearse o interponer barreras que impidan que el operario sea herido. La fábrica debería ser capaz de reconocer este comportamiento y reaccionar adecuadamente, como ya se ha dicho, bloqueando la máquina, apagándola, etc.

Continuando en las cuestiones relacionadas con los operarios, la fábrica debe ser consciente del estado de ánimo de sus operarios con el objetivo de maximizar la producción y minimizar los desperdicios. Es común que en las líneas de producción cambien las exigencias de rendimiento en la fabricación. Los operarios se adaptan a estas exigencias, pero ante altos niveles de exigencia, pueden llegar a sentirse estresados. Cuando una persona opera una máquina, puede cometer errores que conduzcan al desecho de la materia prima. Esto se agrava cuando se trabaja bajo niveles de estrés altos. Los sistemas de soporte a la producción deben ser conscientes de estos cambios de actitud de los operarios y cambiar su configuración para minimizar la cantidad de desperdicio generada y disminuir el estrés del operario.

Si concretamos con el escenario específico de este proyecto, el sistema de soporte de la línea de producción de cantoneras de papel, los aspectos de adaptación que encontramos no distan de los que acabamos de explicar. Encontramos necesidad de soportar adaptación en las conexiones con el resto de componentes de la fábrica, posibilidad de fallo de la interfaz, errores en los datos reportados por la máquina de producción y cambios en el estado de estrés del operario. Puede ser que existan más requisitos de adaptación, pero estos se detectan a lo largo del ciclo de vida de la solución, y se pueden encontrar aún más cuando el sistema está desplegado. No obstante, estos requisitos son un buen punto de partida para dotar de capacidades de auto-adaptación a esta línea de producción y desplegar un prototipo para empezar a hacer pruebas en la fábrica.

1.3 Objetivos

Como ya hemos comentado, nos centraremos en desarrollar un sistema de asistencia a la producción de cantoneras de papel, que consistirá en un semáforo que informará sobre el estado de la producción a los operarios de las máquinas. Este sistema deberá integrarse con los recursos software de gestión de recursos de la empresa, con varios recursos físicos, como la información sobre la producción de las máquinas, el nivel de desechos generados, entre otros.

Este entorno industrial de producción, como la mayoría de ellos, no está exento de fallos, problemas o situaciones que pueden hacer que los sistemas software

no se comporten como deberían. Tradicionalmente, los sistemas asumen la disponibilidad de recursos que requieren para finalizar sus tareas, dan por sentado la ausencia de errores o cambios en el entorno. Como ya hemos dicho, los entornos industriales son dinámicos y propensos a fallos (errores mecánicos, caída de las conexiones, etc.), por lo que debemos tener en cuenta estos aspectos en el desarrollo de los sistemas software, con el objetivo de cumplan sus objetivos de la forma más óptima posible en contextos operacionales diversos, en los que no podemos asumir la ausencia de fallos o el correcto funcionamiento de todos los componentes necesarios para integrar las soluciones.

Los avances tecnológicos e ingenieriles a lo largo de los años nos han permitido desarrollar soluciones que, a pesar de su complejidad, adquieren estructuras y arquitecturas bien definidas que simplifican la programación, la reusabilidad y la mantenibilidad. Pero estos avances han traído consigo, o propiciado, las dificultades de las que hablamos en este discurso. Por lo tanto, nos encontramos con sistemas software que ya son complejos por sí mismos, que además se encuentran en entornos o contextos también complejos, dinámicos y cambiantes, para los que los sistemas no están preparados, causando grandes complicaciones al momento de desplegarlos, mantenerlos en funcionamiento, o simplemente imposibilitando la capacidad de cumplir los requisitos para los que fueron diseñados.

Debemos dotar a los sistemas de capacidades para amoldarse a estas dificultades, para ofrecer soluciones viables, que tengan en cuenta estos aspectos, que no asuman la ausencia de fallos en los componentes, y que tengan en cuenta el contexto operacional en el que se encuentran. Para ello, debemos desarrollar sistemas con capacidades de computación autónoma. Del mismo modo que el resto de aspectos del software, la adaptación del mismo debe plantearse de forma ingenieril, separando aspectos funcionales de aspectos de adaptación. De no ser así, incurriríamos en los problemas que tienen las soluciones que no hacen uso (o uso no adecuado) de la ingeniería del software: dificultad en la mantenibilidad, expansión, reusabilidad… Además, si mezclamos los aspectos funcionales con los de adaptación, estaríamos 'contaminando' las partes funcionales, a las que normalmente si se les aplica ingeniería.

Por lo tanto, para ofrecer soluciones profesionales cuando nos enfrentamos a esta problemáticas, tenemos que, como ya hemos dicho, aplicar computación autónoma. Esto nos permitirá que los componentes software desarrollados se auto-configuren y sean conscientes del entorno en el que se encuentran, adaptándose a los posibles cambios del mismo en base a unos objetivos impuestos por los desarrolladores o administradores. La computación autónoma usa ingeniería para que estas capacidades se materialicen en forma de componentes o piezas mantenibles, reusables, extensibles, separando los aspectos funcionales de los aspectos de adaptación.

En nuestro caso, aplicaremos computación autónoma a los sistemas de la línea de producción de cantoneras de papel, que, como ya se ha comentado, es un entorno industrial cambiante y propenso a fallos, en el que podremos realizar un desarrollo que esclarezca las capacidades y beneficios del uso de computación autónoma en los sistemas software. Deberemos seguir las fases de los desarrollos de software: análisis, diseño e implementación, centrándonos siempre en las

capacidades de computación autónoma que se desean aplicar a la línea de producción.

Para finalizar esta sección, resumimos a continuación los objetivos de este proyecto:

- Identificar y analizar los principales aspectos del entorno de la línea de producción de cantoneras de papel que puedan cambiar o fallar, y por lo tanto son susceptibles de verse afectados por computación autónoma.
- Diseñar un sistema que afronte los aspectos identificados, desde el punto de vista de la computación autónoma.
- Implementar un prototipo de la solución de computación autónoma diseñada, que muestre como los aspectos de adaptabilidad del software son afrontables de forma ingenieril.

1.4 Metodología

Se ha utilizado un modelo en cascada, dividiendo el desarrollo del proyecto en las siguientes cinco fases:

- Fase 1. Definición del proyecto:

  Definición del proyecto, propuesta de caso de estudio y objetivos a alcanzar.
- Fase 2. Estudio de conceptos y tecnologías:

  Aprendizaje de los conocimientos sobre computación autónoma y bucles de adaptación, así como de las tecnologías utilizadas para el desarrollo: Java OSGi y el framework FADA.
- Fase 3. Análisis del problema y diseño de solución:

  Análisis de los escenarios de adaptación del caso de estudio y diseño de los componentes y del bucle de adaptación.
- Fase 4. Implementación:

  Implementación de la solución diseñada mediante el framework FADA.
- Fase 5. Prueba de funcionamiento:

  Preparación de un escenario de pruebas y realización de pruebas de autoadaptación.

A continuación, mostramos un desglose de las fases y una planificación temporal mediante un diagrama de Gantt 1.1.

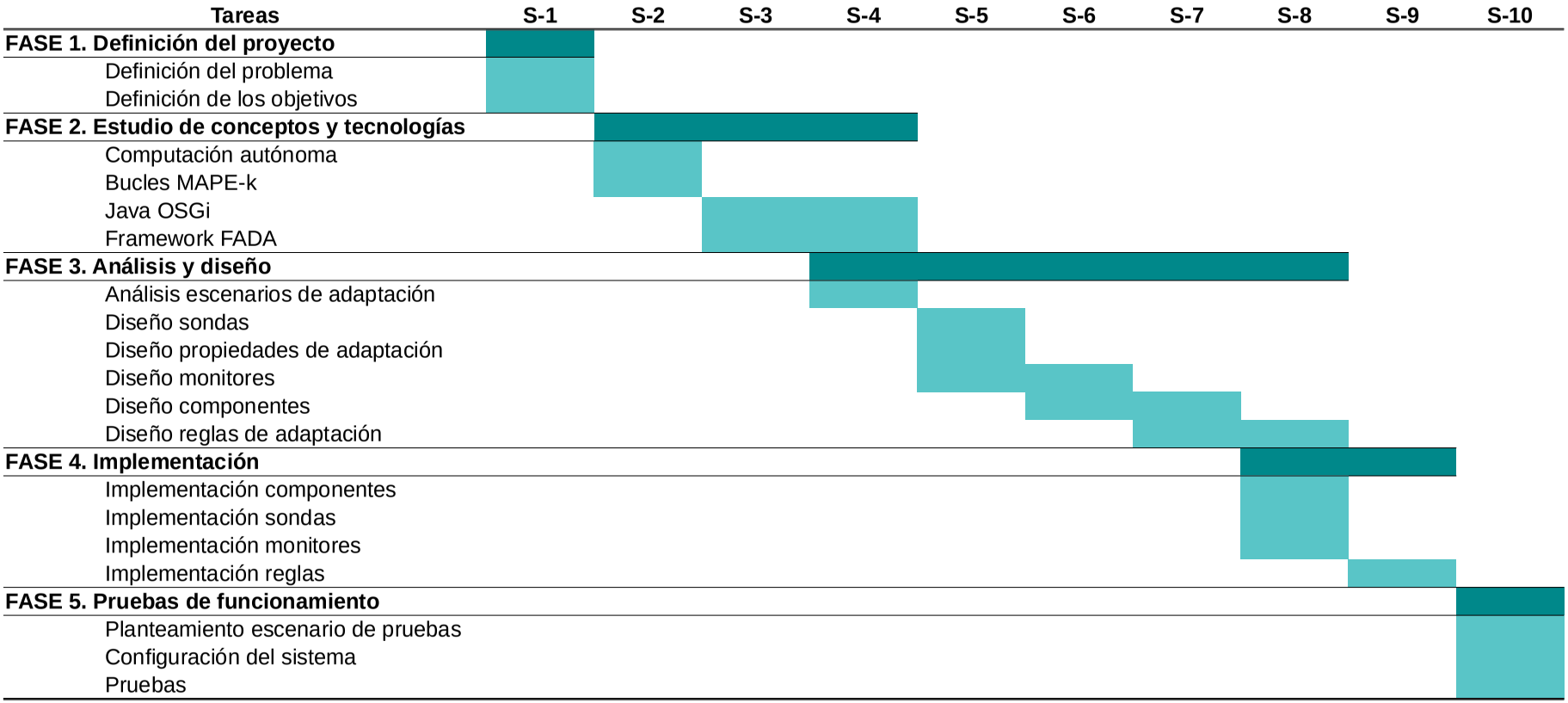

Figura 1.1: Diagrama de Gantt

CAPÍTULO 2
Contexto tecnológico

En este capítulo, introduciremos los conceptos teóricos sobre computación autónoma. Después explicaremos las tecnologías que se han utilizado en el proyecto para otorgar a la solución de capacidades de auto-adaptación. Para finalizar, también presentaremos algunas otras alternativas a estas tecnologías.

2.1 Computación autónoma

Los sistemas informáticos exponen una creciente complejidad, integran entornos heterogéneos, aumentando cada vez más las cantidad de interconexiones. Esto está provocando que la gestión de los mismos sea una tarea ardua y que requiere de personal altamente cualificado para su instalación, configuración, puesta a punto y manutención. Además, el progresivo aumento de la complejidad de los sistemas parece estar llegando al límite de la capacidad humana, exigiendo que estas tareas sean delegadas a las máquinas, para que se realicen automáticamente, con la mínima intervención humana posible. La computación autónoma ayuda a abordar esta complejidad usando la tecnología para gestionar la tecnología.

Computación autónoma se refiere a las características de gestión automática de los recursos computacionales. Paul Horn escogió deliberadamente un término con connotación biológica, ya que el sistema nervioso autónomo controla nuestro ritmo cardíaco y la temperatura de nuestro cuerpo, liberando a la parte consciente del cerebro de la necesidad de tratar con estas y otras cuestiones de bajo nivel. Es decir, estas tareas no necesitan de conciencia o esfuerzo para ser realizadas, y esto se desea trasladar a los sistemas informáticos.

La esencia de los sistemas de computación autónoma es la auto-gestión, cuyo propósito es liberar a los administradores de sistemas de los detalles de operación del sistema y de su mantenimiento, proporcionando a los usuarios un sistema/-máquina que funciona a máximo rendimiento constantemente.

Gracias a la computación autónoma, los ingenieros pueden delegar las tareas de gestión al propio sistema o sistemas, en conformidad con las políticas o reglas que ellos establezcan. [6, 2]

Según [8], la complejidad de los sistemas software conduce a los ingenieros a investigar nuevas vías de desarrollo, despliegue, gestión y evolución de los siste-

mas y servicios. Aseguran que, desde un punto de vista ingenieril, los bucles de control se deben convertir en entidades de primera clase a lo largo del ciclo de vida del software auto-adaptativo. Los bucles de control, o bucles de adaptación, son el método más común para abordar el software auto-adaptativo. Haremos una explicación sobre ellos en la siguiente sección.

2.1.1. Bucles de adaptación

Según [3], los bucles de control, típicamente, involucran cuatro actividades esenciales: coleccionar, analizar, decidir y actuar.

El ciclo comienza recolectando datos relevantes sobre el contexto del sistema, mediante sensores u otras fuentes de datos. Esto plantea una serie de cuestiones sobre las que se debe reflexionar, como la frecuencia de muestreo de las sondas, la fiabilidad, entre otras. Normalmente, los datos recogidos son filtrados y agrupados antes de ser almacenados, para no «ensuciar» el sistema.

A continuación, el bucle analiza los datos obtenidos. Existen varias aproximaciones para ello, por ejemplo el uso de modelos, teorías, reglas, etc. Aquí se debe decidir una manera de inferir el estado actual del sistema, y se debe considerar la viabilidad de las reglas o modelos a lo largo del tiempo.

Después, se deben tomar decisiones para saber cómo adaptar el sistema para lograr el estado deseado. Es importante conocer el estado del sistema y tener en cuenta otras políticas para el análisis de riesgo, así como la prioridad de los bucles, cuando se integran varios bucles en una solución.

El bucle finaliza tomando acciones sobre el sistema, haciendo uso de efectores. En la figura podemos ver cada una de las fases de un bucle de control típico.

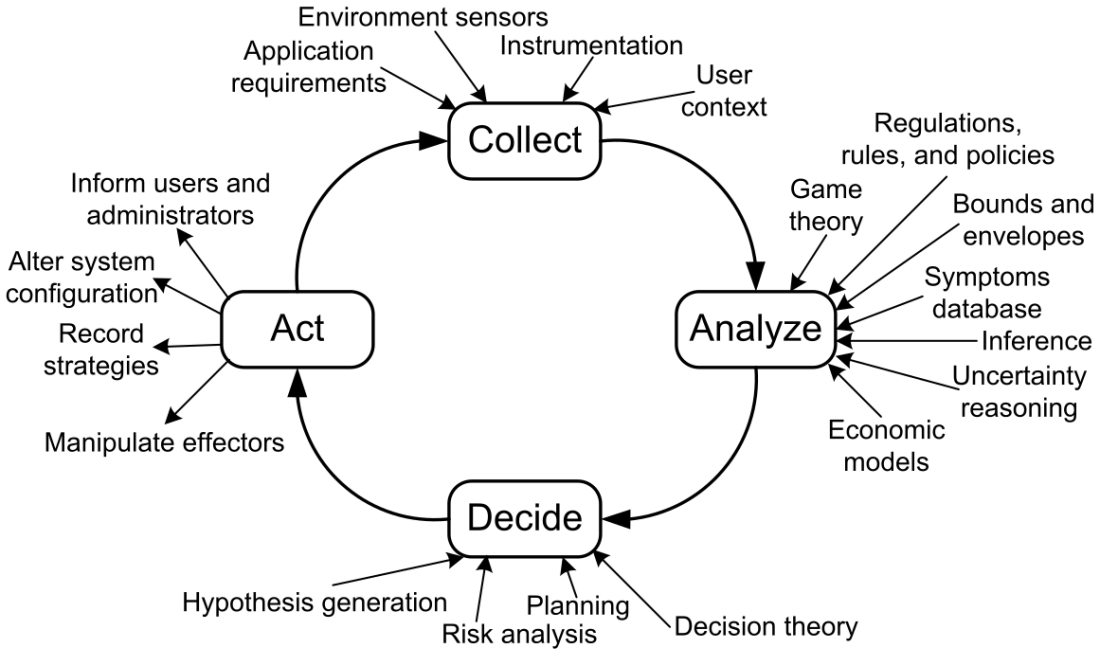

Figura 2.1: Bucle de control [3]

2.1.2. El patrón MAPE-k

Ahora que ya hemos introducido la computación autónoma y los bucles de adaptación, vamos a explicar un patrón de diseño propuesto por IBM [2], que permite hacer ingeniería en los sistemas auto-adaptativos. Presentan estos conceptos como una aproximación a la estandarización de los sistemas autónomos, y reconocen el valor de la computación autónoma, afirmando que acelera los procesos de configuración y adaptación, para los que profesionales de la informática son necesarios, ahorrando tiempo que podrán invertir en tareas de mayor provecho.

Este patrón será la manera en la que aplicaremos computación autónoma a la solución de este proyecto, por lo que debemos comprenderlo con claridad. Se trata de un bucle de adaptación que llamado MAPE-k. Este nombre nace de las iniciales de las fases o componentes que lo forman: Monitorizar, Analizar, Planificar y Ejecutar sobre un Conocimiento compartido.

En este patrón, se hace una separación entre las partes del bucle, las ya mencionadas, y las partes de pertenecen al sistema manejado, que permiten integrar el bucle en las soluciones: los sensores y los efectores. De este modo, se consigue una arquitectura en la que existe una capa de auto-adaptación, que se encarga de monitorizar los cambios del entorno y distinguir qué cambios se deben aplicar, y otra capa del sistema gestionado: sensores, que extraen datos del sistema y del entorno, y efectores, que saben como aplicar los cambios que el bucle de control planee.

Pasaremos ahora a explicar cada uno de las fases del bucle MAPE-k.

Monitorizar. La función de monitorizar permite recoger, agregar y filtrar los detalles o medidas que les proporcionan los sensores para traducirlos a síntomas que puedan ser analizados.

Analizar. La función de analizar permite observar los síntomas o situaciones para determinar si es necesario realizar algún cambio. Si algún cambio es necesario, se le pasa una petición de cambio a la siguiente fase.

Planificar. Crea un plan de actuación para conseguir la configuración del sistema que se desea, y se entrega este plan a la función de ejecución.

Ejecutar. Utiliza los efectores del sistema manejado para llevar a cabo el plan de cambio o plan de adaptación. Traduce el plan que le proporciona la función de planificación en acciones que los efectores deben realizar para lograr la configuración deseada del sistema.

Además, este patrón utiliza una fuente o almacén de conocimiento, que cada uno de estas fases comparten, en el que se almacena información relacionada con los síntomas del sistema, políticas, peticiones de cambio, planes de adaptación.

Ahora, las partes que sirven para integrar el bucle con los recursos manejados son los sensores, que permiten extraer datos o mediciones del sistema manejado o de su entorno, y los efectores, que son componentes que efectúan los cambios pertinentes sobre el sistema.

IBM aboga por el uso de interfaces para que diferentes componentes puedan ser compuestos de forma transparente a los recursos manejados. Es decir, los

componentes o piezas que formas los sistemas no tienen que ser conscientes de las capacidades de auto-adaptación del mismo, solo deben ser conscientes de sí mismos y de la funcionalidad que ofrecen, consiguiendo así separar los aspectos relativos a la funcionalidad de los aspectos de auto-adaptación.

En la figura 2.2 podemos ver las capas en las que se separa el software en este patrón, así como las diferentes fases del bucle de adaptación.

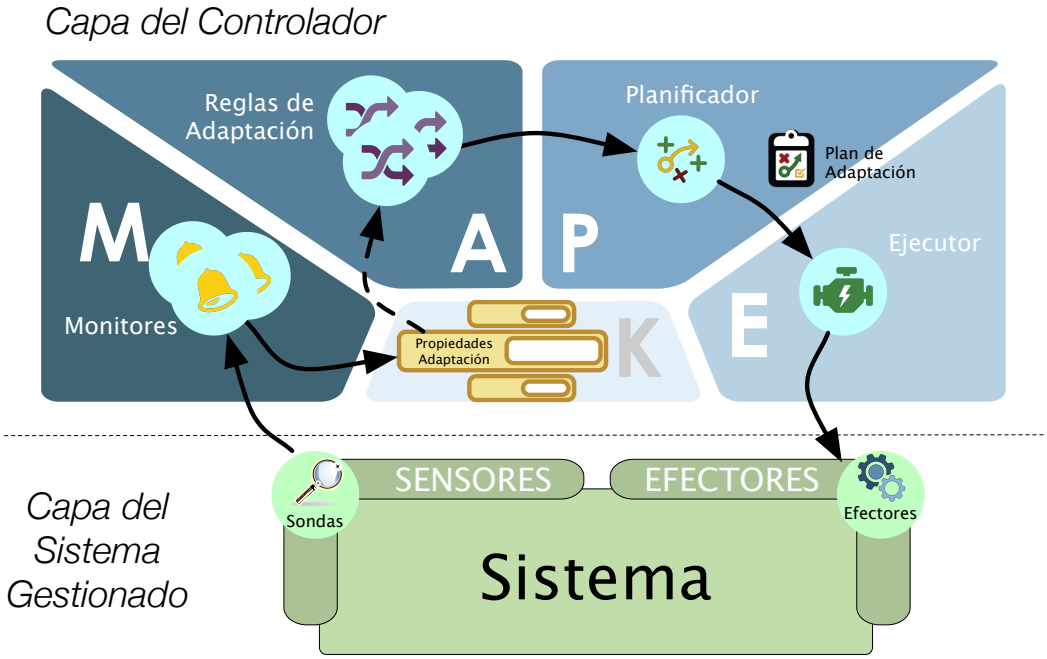

Figura 2.2: Bucle MAPE-k

2.2 Tecnologías utilizadas

Presentamos a continuación las tecnologías utilizadas para el desarrollo de este proyecto.

2.2.1. Librería MAPE-k lite

La librería MAPE-k lite surge como una simplificación del *framework* FADA, resultado de un proyecto de investigación del grupo TATAMI del centro PROS de la UPV [13].

Estas herramientas asisten el desarrollo de software con capacidades de auto-adaptación, que es el principal objetivo de este proyecto.

Proporciona herramientas para crear componentes capaces de ser adaptados, para hacer sondas y monitores. También proporciona una implementación de las diferentes fases de un bucle MAPE-k sencillo, facilitando la tarea de integrar las capacidad de computación autónoma a las soluciones. Se explicará con mayor profundidad en el capítulo de implementación de esta memoria (6).

2.2.2. OSGi

OSGi [11] es un *framework* que permite el desarrollo de una arquitectura modular dinámica. Introduce la capacidad de que los componentes desarrollados sean instalados y desinstalados, iniciados y detenidos o actualizados en tiempo de ejecucción. Proporciona un sistema de registros, en el que los componentes pueden inscribirse para ofrecer su funcionalidad, o eliminarse del mismo para dejar de ofrecerla. El *framework* FADA está implementado sobre esta tecnología, y, por lo tanto, lo está la librería MAPE-k lite que utilizamos en este proyecto.

2.3 Otras tecnologías

En esta sección mostramos algunas tecnologías alternativas a las utilizadas en el proyecto.

2.3.1. FESAS

FESAS [7] es un *framework* utilizado para construir componentes reusables auto-adaptables. Permite el desarrollo de componentes MAPE (véase 2.1.2). Ofrece un alto grado de flexibilidad y reusabilidad.

2.3.2. SASSY

SASSY [9] es un *framework* que ofrece generación automática de arquitecturas orientadas a servicios en las que se integran capacidades de computación autónoma. Se centra en generar una arquitectura optimizada para aproximarse a los requisitos de calidad de servicio impuestos por los administradores de sistemas, y aplicarla automáticamente, reconfigurando los sistemas en tiempo de ejecucción.

2.3.3. Genie

Por último, Genie [1] es un *framework* que soporta el modelado, generación y operación de sistemas basados en componentes altamente reconfigurable mediante capacidades de auto-adaptación. Su propósito inicial era construir interfaces web para aplicaciones de bases de datos, pero puede ser utilizado para desarrollar aplicaciones web de todo tipo.

CAPÍTULO 3
Caso de estudio

En este capítulo, después de haber explicado los conceptos de computación autónoma y bucles de adaptación, introduciremos el caso de estudio de la empresa Embalpack, donde se desea aplicar computación autónoma.

3.1 Introducción a la problemática

Este proyecto surge ante la problemática existente en las líneas de producción de la empresa Embalpack. Se trata de una línea de producción de cantoneras de papel de la empresa. Concretamente, fijamos la atención en un sistema de apoyo a la producción que está actualmente operativo en las línea de producción. El sistema consiste en un semáforo que informa sobre el estado de la producción a los operarios de las máquinas de cantoneras, que integra tanto información del sistema de gestión de recursos empresariales, como datos que le proporcionan las propias máquinas, relacionados con el proceso físico de fabricación de las cantoneras. Además, el sistema se comporta de diferentes maneras según el contexto operacional, por ejemplo, dependiendo del estado de los operarios puede llegar a mostrar una información u otra diferente, es decir, tiene diversas formas de funcionar, o diversos algoritmos, que se deben aplicar en diferentes circunstancias que no dependen de el mismo semáforo, si no que son dadas por el entorno y sus cambios.

El sistema de apoyo se despliega en un entorno industrial, en un contexto cambiante en el que los fallos pueden ser habituales. Actualmente, este sistema esta diseñado para tolerar algunos de los cambios de comportamiento o fallos que pueden ocurrir, pero no se ha aplicado computación autónoma a él, por lo que nos encontramos frente a un sistema que tiene amalgamados aspectos funcionales y aspectos de adaptación. Nos centraremos en aplicar computación autónoma a este sistema, identificando los principales contextos de cambio y errores que puedan darse en este entorno industrial, con el objetivo de refactorizar la solución, consiguiendo un sistema con capacidades de auto-adaptación en el que haya una separación entre los aspectos referentes a funcionalidad del semáforo, y los aspectos referentes a la adaptación del mismo. Nuestro objetivo es desarrollar un prototipo de este sistema, que tenga capacidades de computación autónoma, con la que se puedan realizar pruebas en la empresa.

3.2 El sistema de apoyo a la producción

Vamos a profundizar en la problemática que vamos a afrontar. En la fábrica, nos encontramos con líneas de producción que están formadas por las propias máquinas de producción, en las que trabajan los operarios. Al lado de cada máquina, hay un semáforo, que es la interfaz del sistema de apoyo a la producción. Podemos ver en la figura 3.1 que el semáforo integra datos proporcionados tanto por la máquina como por el sistema de gestión de recursos empresariales. Esto se realiza mediante un agente de mensajería, al que la máquina y el software de gestión envían datos que el semáforo recibirá.

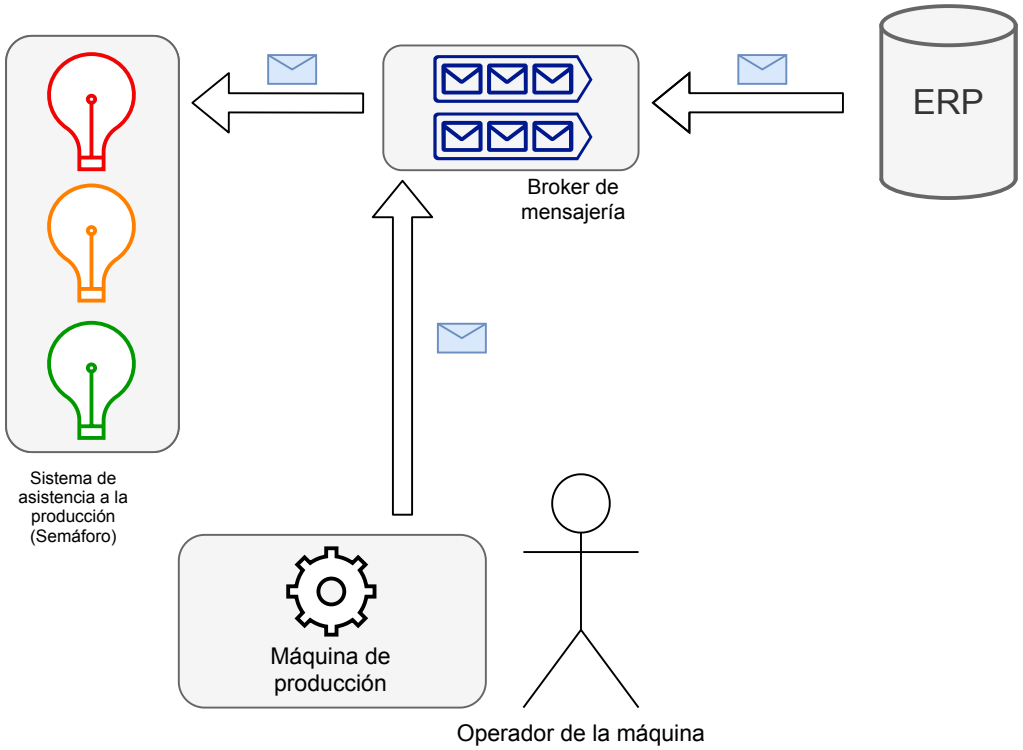

Figura 3.1: Línea de producción

Resulta ahora más fácil identificar los posibles puntos en el que el sistema puede fallar. Enfocándonos al sistema de apoyo: la posibilidad de que el agente de mensajería caiga, debido a un fallo en ese servidor, a caída de la red eléctrica, fallos en la conexión física, como obstrucción del medio aéreo (podemos imaginar a una persona trasportando material de metal, que dificulta la propagación de las señales inalámbricas) o ruido en los cables...

A continuación vamos a describir los problemas a los que está sometida la actual solución, así como las ventajas que obtendremos al aplicar computación autónoma.

Las caídas de la conexión, necesarias para integrarse con las máquinas y el sistema de gestión. Dada la naturaleza cambiante de las fábricas, y los diferentes métodos existentes para realizar una conexión física, por ejemplo, de forma inalámbrica o mediante cable, con una diversidad de medios físicos, todos ellos susceptibles de fallos, desde la presencia de ruido eléctrico en los cables de cobre, hasta la obstrucción de las redes inalámbricas, es imposible garantizar que

las mismas se encuentren operativas en todo momento. A esto debemos añadir la posibilidad de que los propios servidores caigan. Por ello, el sistema de asistencia a la producción debe ser consciente del estado de las conexiones, para no mostrar información errónea. Por ejemplo, no mostrar la información relativa a los últimos datos recibidos antes de una caída de la conexión, ya que esto puede ser incoherente con la realidad. También debe ser capaz de volver a ponerse en funcionamiento en cuanto las conexiones se restablezcan, de forma automática, sin la necesidad de intervención humana.

La máquina de producción puede dejar de reportar datos. Para el correcto funcionamiento de la línea de producción, los operarios deben saber que existe una anomalía y, ante todo, el semáforo debe procurar no quedar en un estado inconsistente con la realidad de producción. Siguiendo esto, el sistema debe intentar auto-curarse (auto-reconfigurarse para volver a funcionar), o bien mostrar de una forma clara para el operario que está ocurriendo una irregularidad, con el fin de que este sea consciente de lo que está ocurriendo en el sistema. Debemos procurar que el sistema, en la medida de lo posible, sea coherente con la realidad productiva de la fábrica, y que no muestre una información que pueda confundir a los operarios.

Los cambios de estado emocional en los operarios de las máquinas. El sistema debe reaccionar de forma distinta, a datos de entrada idénticos, dependiendo del estado del operario. Es decir, el estado del operario condiciona el funcionamiento del semáforo. Concretamente, nos enfrentamos al estrés del trabajador. Se desea que el sistema de apoyo a la producción reaccione de manera más leve en caso de que se detecte que el operario se encuentre estresado. Esto se propone como medida para reducir el estrés de los trabajadores, y así reducir la cantidad de desperdicios que generan las líneas de producción, que suele ser proporcional al nivel de estrés de los operarios. Mediante aplicación de auto-adaptabilidad a los sistemas, se puede conseguir que los algoritmos cambien automáticamente, en tiempo de ejecución, en relación a variables del entorno, que, esencialmente, es el problema que acabamos de describir.

Conociendo la problemática del sistema actual, vamos ahora a definir los objetivos que se desean lograr al refactorizar este sistema.

Diseñar e implementar capacidades de auto-adaptación para la auto-conexión y auto-reconfiguración de las conexiones del sistema.

Diseñar e implementar capacidades de auto-adaptación para reaccionar a caídas o fallos en las máquinas de producción.

Diseñar e implementar capacidades de auto-adaptación para cambiar el comportamiento del semáforo en función del nivel de estrés del operario.

CAPÍTULO 4
Análisis del problema

En este capítulo analizaremos cómo abordar el caso de estudio planteado, centrándonos en integrar en él sistema un bucle de adaptación para abordar la problemática descrita en el capítulo anterior.

4.1 Introducción

Habiendo expuesto los problemas que encontramos en el sistema de apoyo a la producción de las líneas de fabricación de la empresa, podemos empezar a analizar como resolver estas cuestiones. Nuevamente, nos posicionamos desde el punto de vista de la computación autónoma, ya que el objetivo es dotar a la solución de capacidades de auto-adaptación.

En primer lugar, explicaremos el diagrama de clases que hemos elaborado. En él se plantean las diferentes entidades que existirán en la solución del sistema de apoyo.

Después, describiremos cómo debemos afrontar los aspectos de auto-adaptación del sistema, en la sección de ingeniería del bucle de control.

4.2 Diagrama de clases

El problema del caso de estudio del proyecto se puede modelar como se detalla en la figura 4.1. Se trata de un modelado que se basa en la división de responsabilidades del sistema de apoyo, que explicaremos a continuación.

Existen cuatro clases principales:

Light: representa el concepto de bombilla, que tiene un color específico y puede ser encendida o apagada. Cada bombilla pertenecerá a un único semáforo.

TrafficLight: representa un semáforo como un conjunto de tres bombillas. Guarda el estado del mismo mediante la enumeración *State*. Tiene la capacidad de establecer qué bombilla está encendida, pero la decisión sobre cuál debe encenderse pertenece a la siguiente clase.

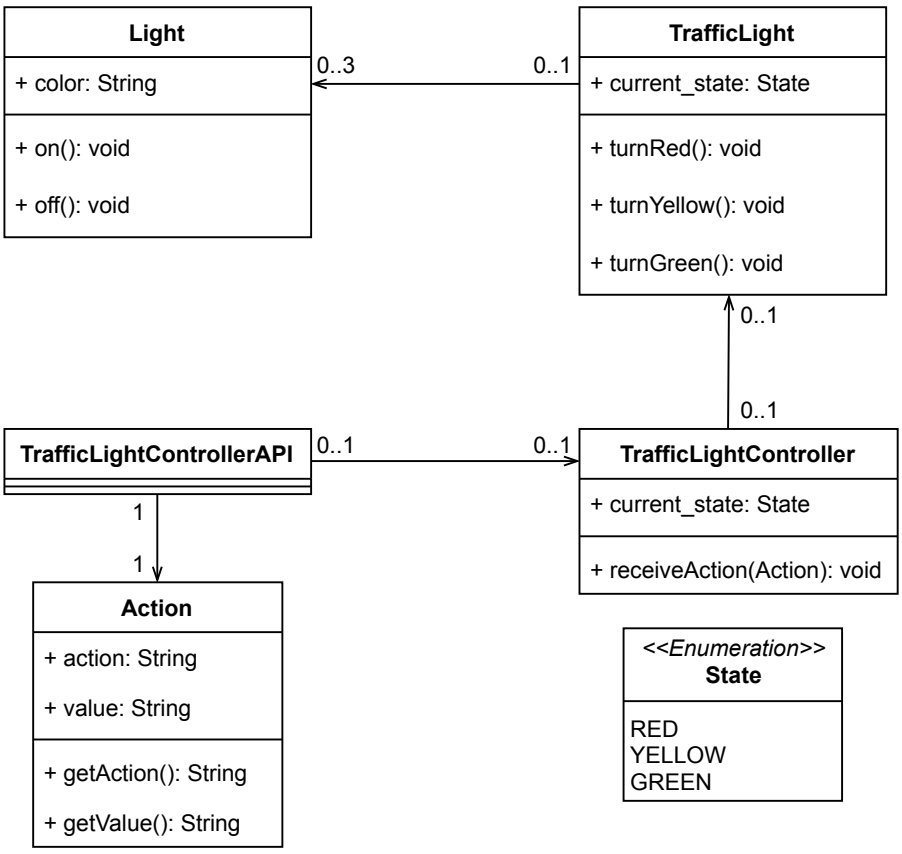

Figura 4.1: Diagrama de clases

TrafficLightController: decide, a partir de la información recibida por el conector, la bombilla que debe encenderse. También guarda el estado mediante la enumeración *State*.

TrafficLightControllerAPI: es el punto de conexión con el resto de elemento de la línea de producción. Mediante esta clase, se reciben los datos y estos son comunicados al controlador mediante el método *receiveAction(Action)*. Cada acción recibida se representa mediante la clase Action, y, mediante una referencia a esta clase, la API del controlador guarda el estado, que en este caso sería la última acción recibida.

Una vez modelado el sistema, podemos pasar a determinar cuestiones de auto-adaptación en la siguiente sección.

4.3 Ingeniería del bucle de control

Partiendo de la base de los bucles de control (véase la figura 2.1), vamos a explicar, en nuestro caso concreto de estudio, cómo se desarrollaría cada una de las fases del bucle de la solución.

4.3.1. Collect

Para comenzar con el bucle, como ya hemos visto, la primera fase consiste en recoger y filtrar los datos que nos permitan conocer el estado del sistema y del contexto del mismo, para poder reaccionar a los cambios de los mismos en siguientes fases del bucle.

Deberemos recoger datos sobre el estado de las conexiones con los agentes de mensajería. Se puede lograr mediante una sonda que consiste en un cliente del agente de mensajería, que es capaz de reportar el estado de conexión o desconexión con el mismo. Esta información permite al sistema ser consciente de si el agente de mensajería esta disponible o no.

Para detectar la presencia de la máquina de producción, podemos seguir una estrategia similar a la anterior, ya que la máquina de producción también envía mensajes al agente de mensajería, por lo que, la presencia de mensajes significa la presencia de la máquina que los origina. Debemos establecer un tiempo máximo sin recibir un mensaje de la máquina. Deberá ser un tiempo prudencial, en el que suponemos que transcurrido totalmente, la máquina ha sufrido una caída, un error, o el operario ha dejado de usarla. Así, concedemos al sistema de conciencia sobre la disponibilidad, o ausencia, de la máquina de producción.

Continuando con los datos a detectar, respecto al estrés del operario de la máquina, sabemos que existe un mecanismo mediante el que la máquina realiza una estimación de estrés en relación al desecho generado, y esto se publica en el agente de mensajería. Leyendo estos mensajes y filtrándolos a niveles de estrés altos, podemos conseguir esta medición de forma sencilla. Mediante esta sonda, el sistema adquiere conciencia sobre la necesidad de activar o desactivar el modo de funcionamiento de estrés.

Para finalizar, mencionaremos que también es deberemos detectar el posible fallo en el funcionamiento de las luces del sistema, tanto como si se implementan de forma física o de forma simulada mediante interfaces de ordenador. Como en este punto del desarrollo del proyecto no conocemos como funcionarán las luces, dejaremos esta cuestión a resolver más adelante, pero asumimos que el sistema tendrá consciencia sobre la disponibilidad de las luces.

4.3.2. Analyze

A partir de los datos recogidos y la información de la que se tiene consciencia, mantendremos registros sobre el estado del sistema y su contexto. Esto es, se registrará si el agente de mensajería esta disponible o no, si la máquina se encuentra en funcionamiento o no, etc. De esta forma, podremos observar los cambios que ocurren en estos registros para poder analizar la necesidad de aplicar cambios sobre la configuración del sistema.

Estos cambios los definiremos como una serie de acciones que se deberían aplicar al sistema para reaccionar a los cambios del contexto representados por los registros observados. Entonces, explicaremos a continuación los cambios que deberá soportar el sistema.

Ante la detección de disponibilidad del agente de mensajería, los pasos a seguir serían conectar al mismo mediante un controlador de mensajería, que se enlace con un posible controlador de semáforo. Por el contrario, si se detecta la no disponibilidad del agente de mensajería, se debería replegar el controlador de mensajería y desplegar un controlador de semáforo a modo de alerta, para advertir que se ha desconectado.

Respecto a la disponibilidad de la máquina de producción, cuando se observe que ésta existe, el sistema debería adaptarse a la existencia de un controlador que procese los mensajes que generen las máquinas. Esto supone eliminar de la configuración otros controladores existentes, además de enlazar el nuevo controlador con el resto de entidades correspondientes, controlador de comunicaciones y semáforo. En el caso de observar que la disponibilidad de la máquina desaparece, se deberían revertir los cambios anteriores.

Para continuar, cuando se observe sobre los registros que el modo de estrés se ha alcanzado, se seguiría una estrategia similar a la anterior. Habría que intercambiar el controlador de semáforo existente con el controlador que tiene el cuenta el estrés medido en el operario de la máquina, eliminando el controlador anterior y sus respectivos enlaces, para así colocar el nuevo controlador y enlazarlo con el semáforo y las comunicaciones.

Finalizando esta sección, en lo que respecta a las luces del sistema de apoyo, si se observa que funcionan correctamente, deberíamos desplegar sobre la solución los controladores de las mismas, que son las **Light** en el diagrama 4.1, y en caso contrario, eliminarlas.

4.3.3. Decide

En esta fase del bucle de adaptación tomará como base el análisis que se ha realizado en la fase anterior y generará un plan de actuación sobre el sistema. Debe tener en cuenta el estado actual del sistema, anterior a los diagnósticos, para compararlo con el análisis hecho. Por ejemplo, si el análisis estima se debe un cambio en el sistema tal que debe existir en él un componente A, pero este componente ya existe, el plan no contemplará el despliegue de este componente, evitando así pasos de ejecución innecesarios y duplicidad en las entidades del sistema.

4.3.4. Act

Finalmente, el plan generado en la fase anterior del bucle debe ser aplicado sobre la solución. El bucle de adaptación deberá saber como manejar el sistema para aplicar cambios sobre él, hacer que unas partes aparezcan o desaparezcan, modificar los enlaces entre entre las posibles entidades o cambiar parámetros de las mismas. Así mismo, el sistema debe saber cómo reaccionar cuándo el bucle solicite alguno de estos cambios, para así satisfacer los requisitos de adaptación. Por ejemplo, se da por supuesto que un controlador de comunicaciones debe establecer canales de comunicación para funcionar, entonces, al desplegarlo, se deben establecer los mismos, y esto es responsabilidad del controlador, no del bucle.

CAPÍTULO 5
Diseño de la solución

Una vez analizada la problemática que nos encontramos en la línea de producción y en el sistema actual, nos encontramos en posición de empezar a diseñar el nuevo prototipo, incorporando aspecto de computación autónoma al mismo. Debemos tener siempre en cuenta que se trata de una refactorización, en el que cambiaremos el enfoque de una solución existente para diseñarla desde el punto de vista de la computación autónoma.

5.1 Interfaces de la solución

Primeramente, se debe rediseñar la solución para facilitar la integración de un bucle de control en ella. Para ello, se decide usar un diseño modular, en el que existirán varios componentes que cumplirán unas interfaces, que explicaremos a continuación, para que el bucle de adaptación sea capaz de intercambiar los componentes entre sí.

Se han extrapolado las funcionalidades del sistema en una serie de interfaces, que son:

- Las luces. Cada luz o bombilla será un componente que cumple la interfaz **ILight**, ofreciendo la posibilidad de encender o apagar la luz.

- El semáforo. Se usa el concepto de semáforo como un componente que es capaz de establecer un estado de luces, implementando la interfaz **ITrafficLight**. Es decir, un semáforo puede ponerse en verde, en rojo o en amarillo, o en cualquier combinación de colores que se programe. Para ello, el semáforo debe tener una referencia a cada luz (ILight) y conocer el estado de las luces, para, por ejemplo, poder apagar el color que no corresponda con el estado que se quiera establecer.

- El controlador. Se encarga de procesar los diferentes datos que integra el sistema para decidir en qué estado debe estar el semáforo, por lo que tiene una referencia a un semáforo (ITrafficLight). Cumple la interfaz **ITrafficLightController**. El objetivo es separar la funcionalidad que se espera en cada situación en diferentes controladores, y que estos se intercambien, automáticamente, cuando el sistema detecte que es necesario.

- El conector. Se trata de un componente que recogerá o recibirá datos para dárselos al controlador de semáforo, de modo que debe tener una referencia a un controlador.

En la figura 5.1 podemos observar las diferentes interfaces del sistema y sus relaciones.

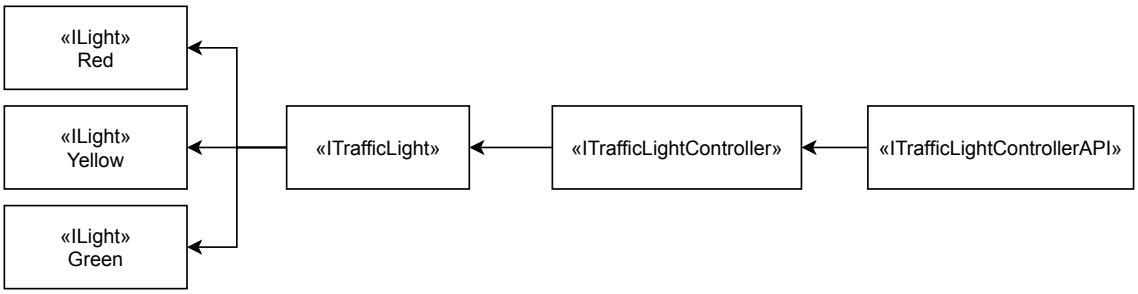

Figura 5.1: Interfaces del sistema de apoyo.

5.2 Diagrama de componentes

Antes de empezar, cabe destacar que en este proyecto no se hará una implementación física de las luces. Usaremos una interfaz de usuario virtual mediante una aplicación web para ver el estado de las mismas. Se usará un agente de mensajería para este propósito.

En primer lugar, mostramos en las figuras 5.2 y 5.3 el diagrama de componentes que hemos diseñado, donde podemos ver todos los componentes necesarios para cubrir las necesidades del sistema. Estos diagramas consisten en un rectángulo que representa cada componente y, de cada componente, nacen unas líneas. Estas líneas acaban en una circunferencia completa (en la parte superior), mostrando la interfaz de funcionalidad que ofrecerá el componentes, o en una semicircunferencia (en la parte inferior), mostrando las interfaces que el componente requiere para funcionar.

Ya hemos explicado la funcionalidad de los componentes, ya que estos serán empleados para implementar las interfaces, explicadas en 5.1, por lo que nos centramos en los componentes que hemos especializado, que son los controladores de semáforo.

A continuación, explicamos los componentes que actuarán como controladores en la solución. Hemos dividido las diferentes funcionalidades que se esperan del sistema en distintos controladores de semáforo. De esta forma, el bucle de adaptación podrá intercambiar estos componentes en tiempo de ejecución, cuando los requisitos de adaptación lo requieran. Los controladores existentes en el diseño se pueden ver en la figura 5.3, y a continuación los explicamos:

- Controlador de velocidad de referencia.

  Se trata de un controlador que reaccionará a la información proporcionada por la máquina y por el sistema de ERP, cuando el operador se encuentra

bajo condiciones normales. El sistema de ERP proporcionaría una velocidad de referencia estimada a la que sería óptimo producir, desde el punto de vista de la gestión de recursos. La máquina de producción proporcionaría la velocidad real de producción. Este controlador de semáforo integraría estos datos para decidir en qué estado debe estar el semáforo.

- Controlador de velocidad de referencia bajo estrés.

  Introduciendo un nuevo dato que proporcionaría la máquina de producción, la cantidad de deshechos producidos, el sistema de asistencia a la producción puede estimar si el operador se encuentra bajo estrés. En esta situación, el sistema funcionaría de forma similar a la descrita en el anterior controlador, integrando la misma información, pero reaccionando a ella de forma diferente, con el objetivo de salvar la situación de estrés del operario y que se disminuya la cantidad de desperdicio generado.

- Controlador en bucle.

  Este controlador no procesará datos, se usará cuando hayan fallos de conexión con algún punto del sistema. Hará parpadear las luces en bucle, para indicar al operario que ha ocurrido un fallo.

Presentamos ahora una tabla 5.1 a modo de resumen, donde aparecen todos los componentes, agrupados por la interfaz que implementan.

Interfaz	**Componentes**
ILight	Light
ITrafficLight	TrafficLight
ITrafficLightController	TrafficLightController TrafficLightControllerReferenceSpeed TrafficLightControllerReferenceSpeedStress TrafficLightControllerLooping
ITrafficLightControllerAPI	TrafficLightControllerAPI

Tabla 5.1: Componentes del sistema y sus interfaces

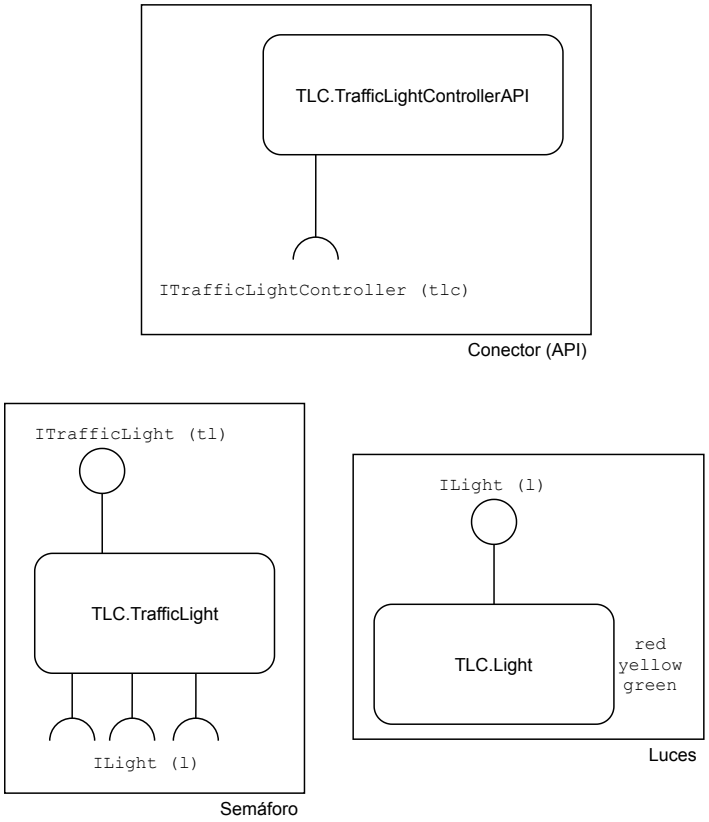

Figura 5.2: Diagrama de componentes (a)

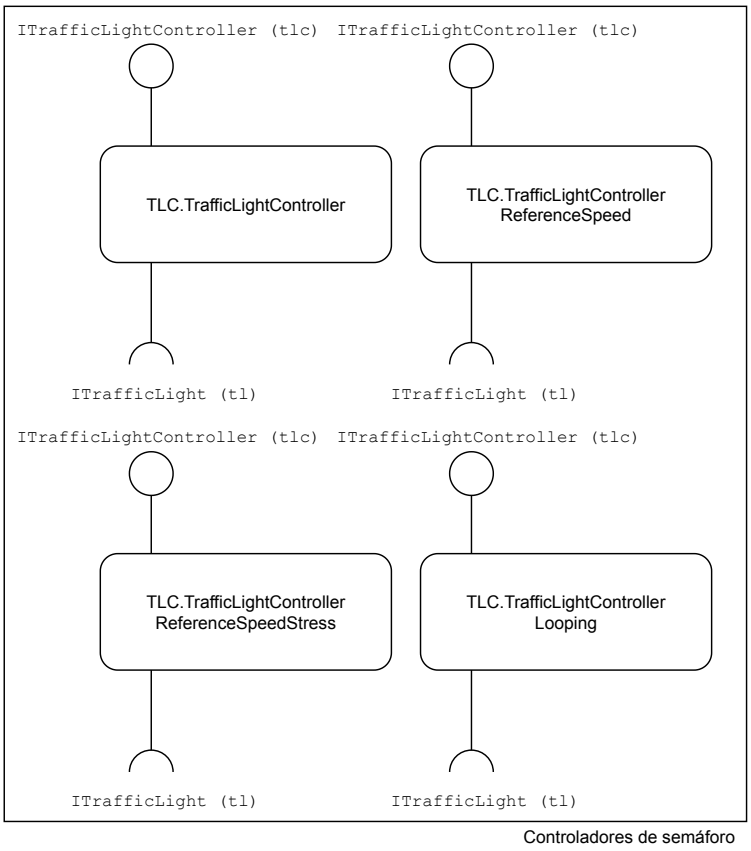

Figura 5.3: Diagrama de componentes (b)

5.3 Diseño del bucle de control MAPE-k

Queremos otorgar capacidades de auto-adaptación al sistema haciendo uso del patrón MAPE-k que se explicó en la sección 2.1.2. Explicaremos a continuación las diferentes partes del bucle MAPE-k que integraremos en el sistema para concederle capacidad de auto-adaptación. Tomamos la propuesta del grupo Tatami [13] como base para especificar los elementos del bucle.

5.3.1. Sondas

Definiremos en primer lugar las sondas que se usarán para tomar medidas del sistema y su contexto.

Sonda: BrokerAliveProbe.
Descripción: identifica si la conexión al agente de mensajería está disponible.
Monitor: BrokerAliveMonitor.
Datos: booleano, broker-alive.

Sonda: BrokerLightsAliveProbe.
Descripción: identifica si la conexión al agente de mensajería de las luces está disponible.
Monitor: BrokerLightsAliveMonitor.
Datos: booleano, lights-alive.

Sonda: MachineAliveProbe.
Descripción: identifica si se están recibiendo datos desde la máquina de producción.
Monitor: MachineAliveMonitor.
Datos: booleano, machine-alive.

Sonda: StressProbe.
Descripción: recoge los datos de deshechos generados en la máquina de producción. Se estima que estos datos están relacionados con el estrés del operario.
Monitor: StressMonitor.
Datos: número real, stress.

5.3.2. Propiedades de adaptación

Como ya hemos descrito, el conocimiento o las propiedades de adaptación son el medio de comunicación entre las diferentes partes del bucle de adaptación. Los monitores actualizarán estas propiedades cuando convenga, haciendo que la fase

de análisis reaccione para amoldar el sistema a los cambios que hayan ocurrido en el conocimiento.

Propiedad: is-broker-alive.
Descripción: variable que representa si el agente de mensajería está disponible.
Tipo de dato: booleano.

Propiedad: is-broker-lights-alive.
Descripción: variable que representa si el agente de mensajería de las luces está disponible.
Tipo de dato: booleano.

Propiedad: is-machine-alive.
Descripción: variable que representa si la máquina está disponible y se reciben datos de ella.
Tipo de dato: booleano.

Propiedad: stress-exceeded.
Descripción: variable que almacena si la cantidad de estrés medida ha superado al máximo establecido.
Tipo de dato: booleano.

5.3.3. Monitores

Los monitores serán un mecanismo de filtración y agregación de los datos reportados por las sondas.

Monitor: BrokerAliveMonitor.
Descripción: Monitoriza el estado de la conexión con el agente de mensajería que usa el conector de la solución.
Afecta a las propiedades: is-broker-alive.
Acciones:

ACTUALIZA-KNOWLEDGE is-broker-alive = broker-alive

Monitor: BrokerLightsAliveMonitor.
Descripción: Monitoriza el estado de la conexión con el agente de mensajería que usan las luces del semáforo.
Afecta a las propiedades: is-broker-lights-alive.
Acciones:

ACTUALIZA-KNOWLEDGE is-broker-lights-alive = lights-alive

Monitor: MachineAliveMonitor.
Descripción: Monitoriza la presencia de mensajes de la máquina de producción.
Afecta a las propiedades: is-machine-alive.
Acciones:

ACTUALIZA-KNOWLEDGE is-machine-alive = machine-alive

Monitor: StressMonitor.
Descripción: Monitoriza si se ha alcanzado el límite de estrés del operario.
Afecta a las propiedades: stress-exceeded.
Acciones:

SI stress $>$ 20.0
ACTUALIZA-KNOWLEDGE stress-exceeded = true

SI stress $\leq$ 20.0
ACTUALIZA-KNOWLEDGE stress-exceeded = false

5.3.4. Reglas de adaptación

Definimos las reglas de adaptación como método de descripción de los cambios de deben ocurrir en el sistema cuando sea necesario. Describiremos las reglas mediante diagramas de especificación de cambios. En los diagramas, la ocurrencia de un componente con fondo blanco significará que ese componente debe estar en la configuración del sistema. Por el contrario, los componentes con fondo rojo denotarán que el componente debe desaparecer de la configuración. Así mismo, las flechas que representan los enlaces de componentes implicarán que el enlace debe establecerse, en caso de que sean negras, o que debe eliminarse, en caso de que sean rojas.

Mostramos, al final de esta sección, las diferentes reglas diseñadas, agrupadas por el requisito de adaptación que satisfacen:

- Reglas que controlan la disponibilidad del agente de mensajería principal, en las figuras 5.4 y 5.5
- Reglas que controlan la presencia de la máquina de producción en la línea de la empresa, en las figuras 5.6 y 5.7
- Reglas que controlar el nivel de estrés del operario, en las figuras 5.8 y 5.9
- Reglas que controlan la disponibilidad del agente de mensajería usado por las luces de la solución, en las figuras 5.10 y 5.11

Regla: BrokerAliveRule.
Descripción: Establece el modo de funcionamiento manual, en el que existe conexión al agente de mensajería.
Condición: is-broker-alive = true.
Cuerpo:

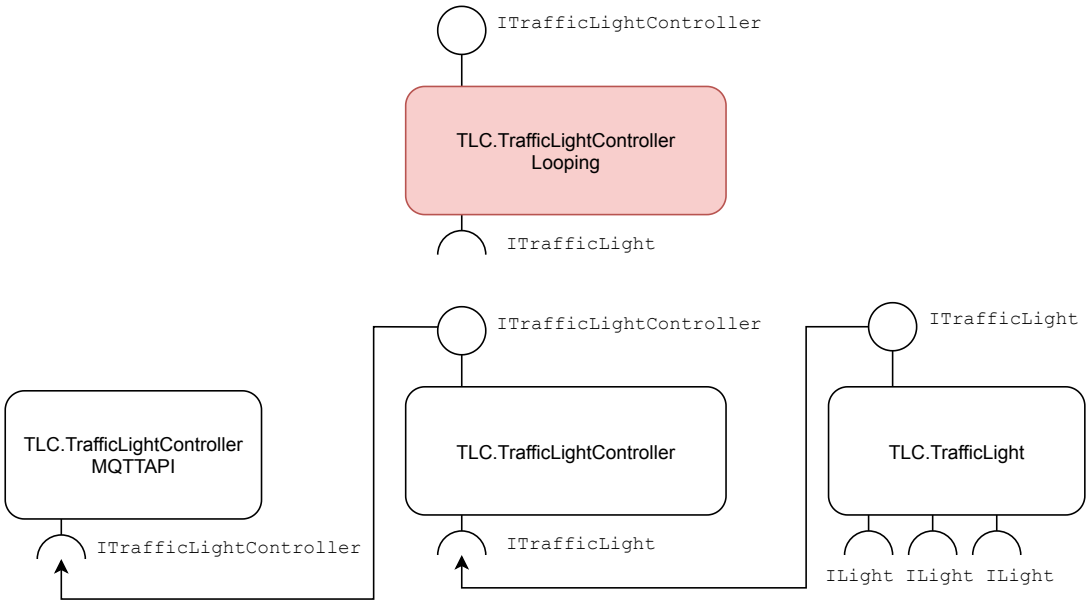

Figura 5.4: Regla broker activo

Esta regla establece el modo de funcionamiento manual de la solución, desplegando el componente TrafficLightController y fijando sus enlaces con el resto de componentes. A su vez, se define que el controlador usado en caso de errores debe ser replegado, ya que, llegado el punto en el que se ejecuta esta regla, el agente de mensajería ya ha sido restaurado y el sistema de apoyo puede volver a funcionar.

Regla: BrokerDeadRule.
Descripción: Establece el modo de funcionamiento de alerta, en el que no existe conexión al agente de mensajería.
Condición: is-broker-alive = false.
Cuerpo:

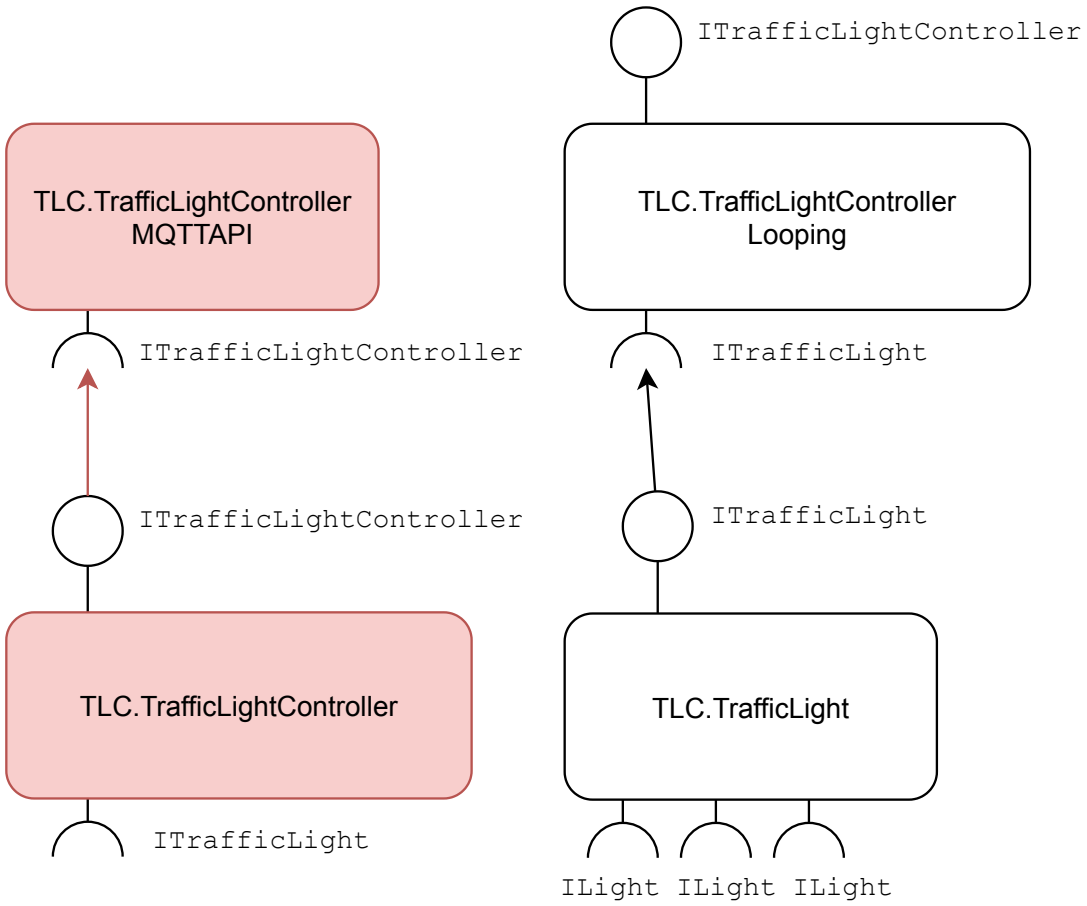

Figura 5.5: Regla broker inactivo

La regla descrita será tomada en cuenta cuando ocurra un error en la conexión con el agente de mensajería de la línea de producción. Se eliminarán de la configuración del sistema tanto el componente que actúa como conector, como el controlador de semáforo. Además, para advertir esta situación anómala en la fábrica, se acoplará a la configuración un nuevo controlador de semáforo que hace parpadear las luces en bucle.

Regla: MachineAliveRule.
Descripción: Establece el modo de funcionamiento de integración de datos, en el que la máquina de producción envía datos y se reciben correctamente.
Condición: is-machine-alive = true.
Cuerpo:

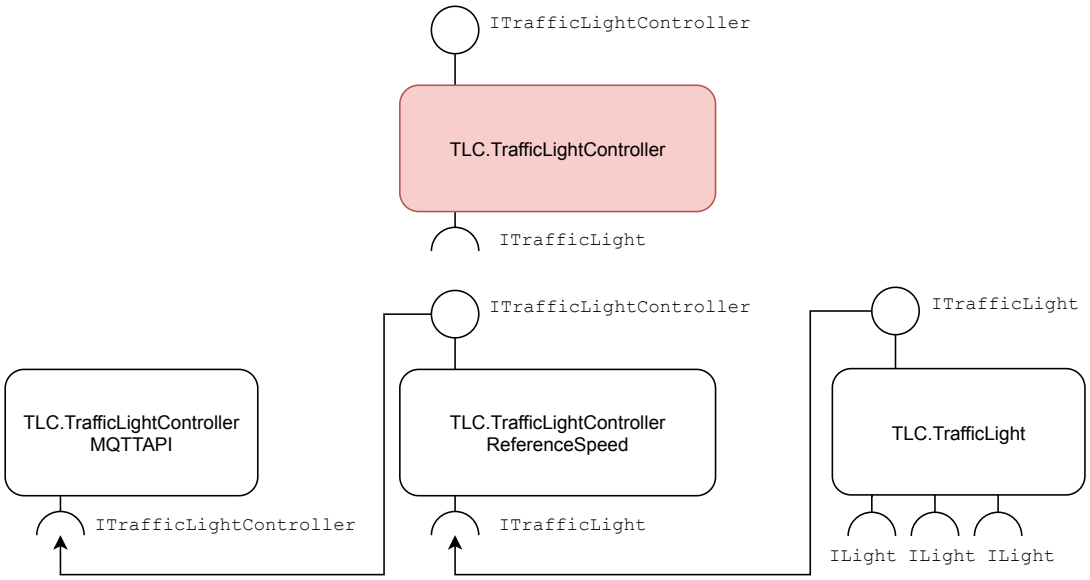

Figura 5.6: Regla máquina activa

La regla recién presentada se ha pensado para establecer el modo de integración de datos, cuando en la línea se detecte la presencia de la máquina de producción. Las acciones que realiza la regla son, en esencialmente, reemplazar el controlador de semáforo de la configuración por el controlador de integración de datos.

Regla: MachineDeadRule.
Descripción: Retorna al modo de funcionamiento manual.
Condición: is-machine-alive = false.
Cuerpo:

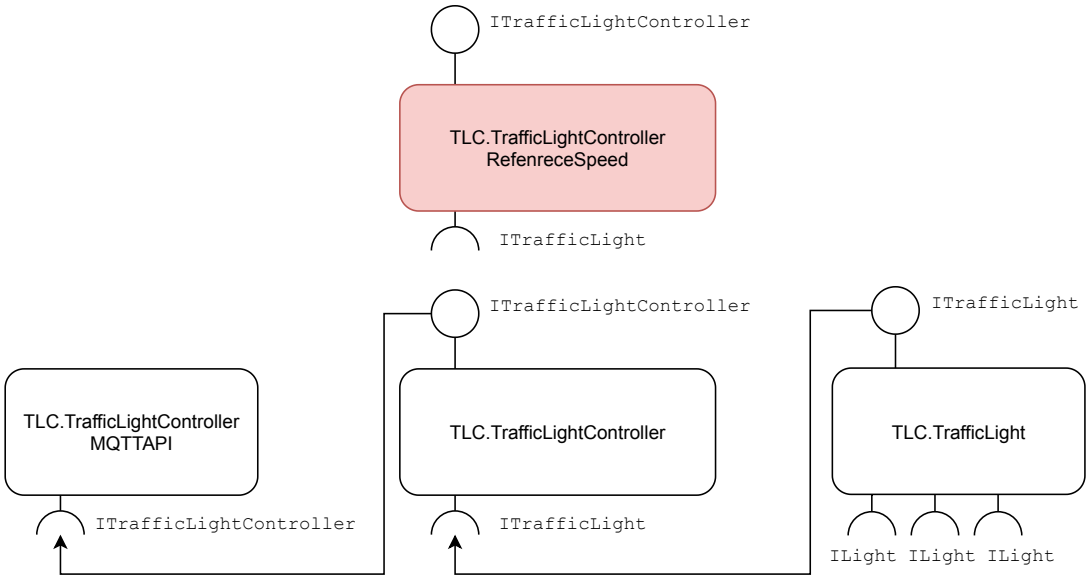

Figura 5.7: Regla máquina inactiva

En contraposición a la regla anterior, esta regla se concibe para revertir los cambios originados, cuando se determine la caída de la máquina de producción.

Regla: StressEnabledRule.
Descripción: Establece el modo de funcionamiento de estrés, en el que se estima que el estrés del operario supera el umbral establecido.
Condición: stress-exceeded = true.
Cuerpo:

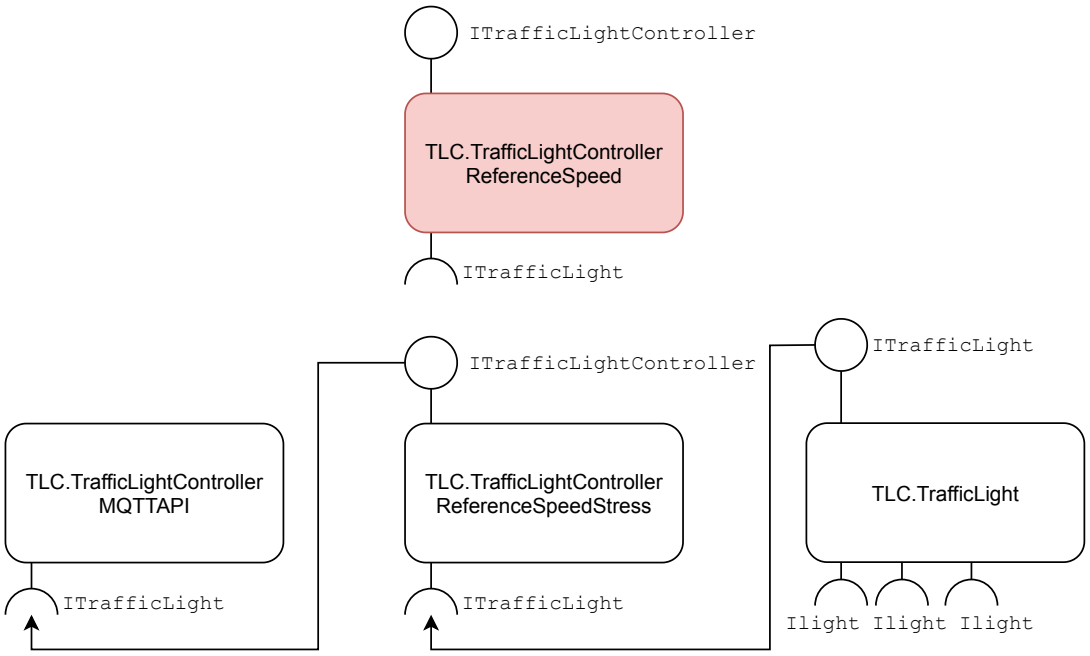

Figura 5.8: Regla estrés activo

Nuevamente, esta regla intercambia los controladores de semáforo de la configuración, para usar el controlador que tiene en mayor consideración el posible nivel de estrés que presente el operario de la máquina de producción. Será ejecutada cuando se superen los niveles de estrés que se establezcan.

Regla: StressDisabledRule.
Descripción: Retorna modo de funcionamiento de integración de datos normal.
Condición: stress-exceeded = false.
Cuerpo:

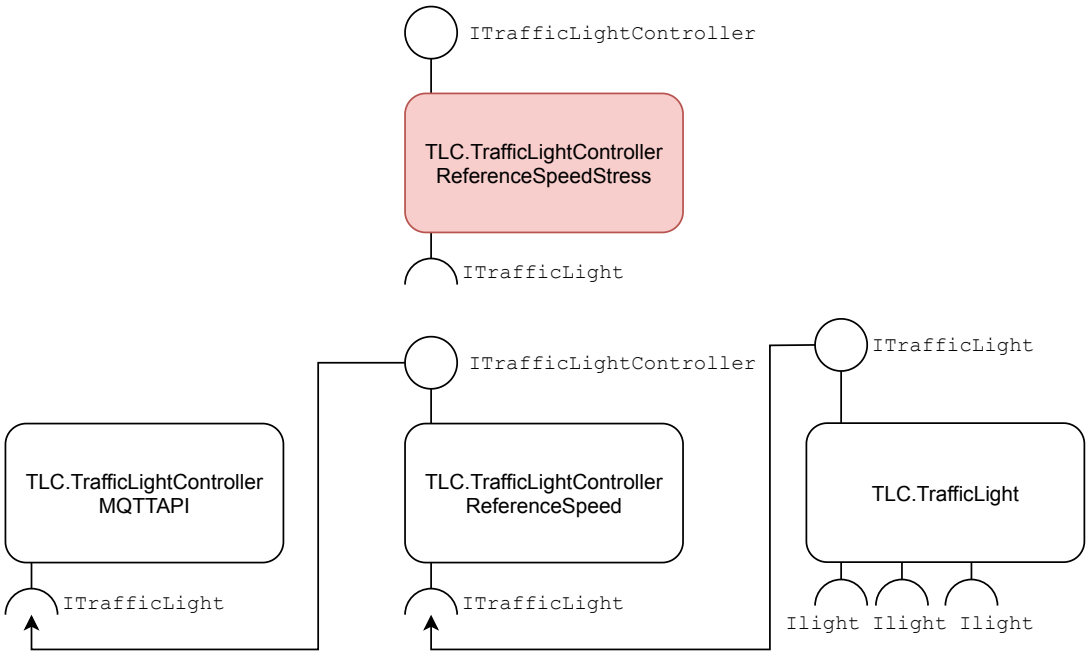

Figura 5.9: Regla estrés inactivo

Este regla es la pareja de la anterior, servirá para revertir los cambios en el caso de que se detecte que el estrés del operario desaparece o se reduce a los niveles aceptables.

Regla: BrokerLightsAliveRule.
Descripción: Despliega las luces, el semáforo y los enlaza.
Condición: is-broker-lights-alive = true.
Cuerpo:

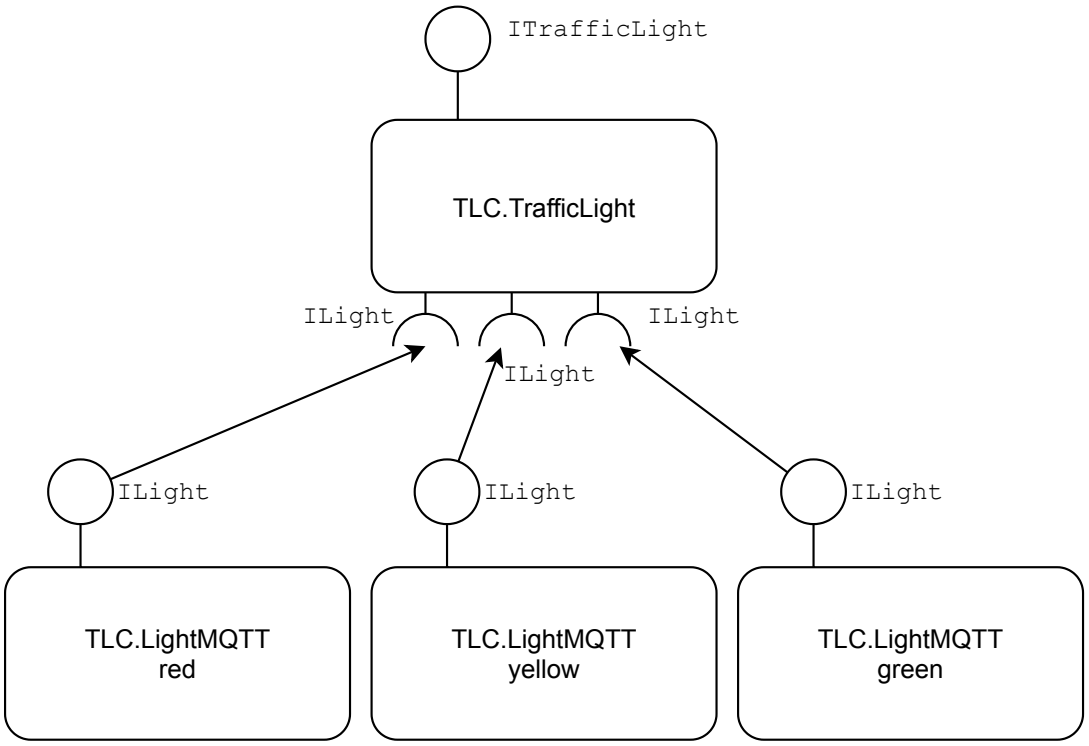

Figura 5.10: Regla luces activas

Definimos esta regla para desplegar las luces del sistema, cuando se detecte que el agente de mensajería que estas usan se encuentra disponible para el envío de mensajes. Se enlazará cada luz con el semáforo.

Regla: BrokerLightsDeadRule.
Descripción: Repliega las luces.
Condición: is-broker-lights-alive = false.
Cuerpo:

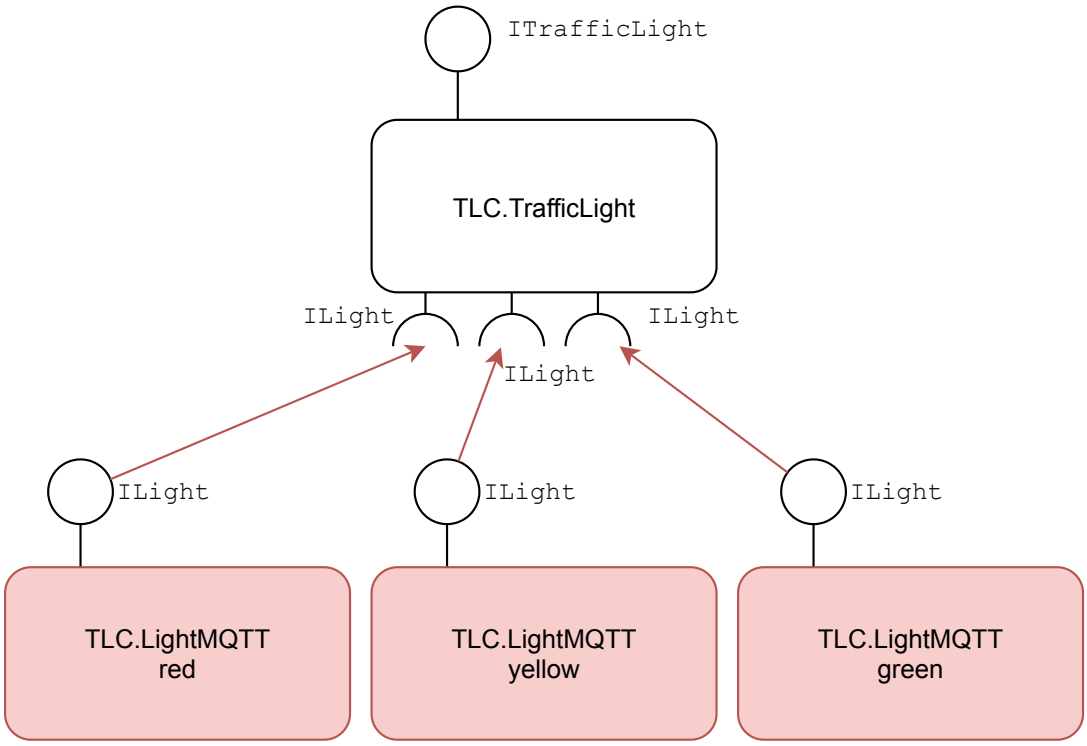

Figura 5.11: Regla luces inactivas

Por último, determinamos la regla BrokerLightsDeadRule, que será efectuada cuando se detecte una caída en el agente de mensajería de las luces. La regla eliminará de la configuración del sistema cada una de las luces presentes, así como su enlace con el semáforo.

5.4 Diseño de la solución

En esta sección reunimos los componentes que compondrán el bucle de adaptación que vamos a integrar en la solución. La figura 5.12 concentra e identifica cada unos de los componentes. Esta figura es especialmente importante, debido a que será utilizada de nuevo en las diferentes secciones del siguiente capítulo, y nos servirá para identificar en qué punto de la fase de implementación nos encontramos.

Antes de finalizar el capítulo, incluimos la tabla 5.2 a modo de resumen, relacionando cada parte del bucle con el requisito de adaptación que afronta. Mediante esta tabla y la figura 5.12, tenemos una imagen global del sistema diseñado.

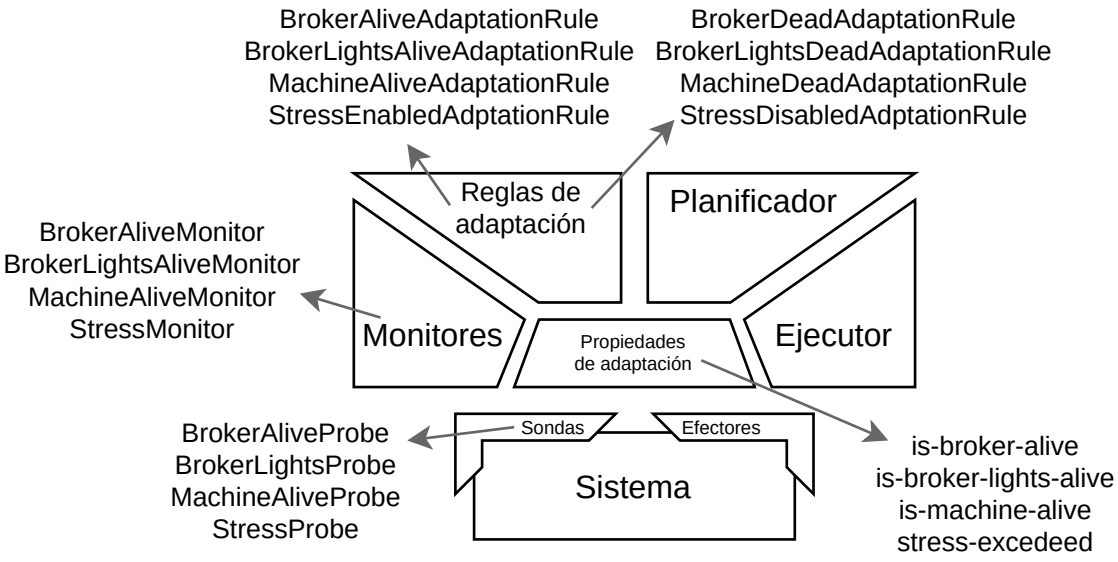

Figura 5.12: Bucle de adaptación de la solución

Requisito de adaptación	**Componente**	**Explicación**
Conexión con el agente de mensajería	BrokerAliveProbe	Mide la disponibilidad del agente.
	BrokerAliveMontior	Actualiza la propiedad de adaptación is-broker-alive.
	BrokerAliveRule	Especifica los cambios para conectar al agente.
	BrokerDeadRule	Especifica los cambios para desconectar del agente.
Fallo en la máquina de producción	MachineAliveProbe	Mide la presencia de la máquina de producción.
	MachineAliveMonitor	Actualiza la propiedad de adaptación is-machine-alive.
	MachineAliveRule	Especifica los cambios para integrar los datos de la máquina.
	MachineDeadRule	Especifica los cambios para volver al modo manual.
Fallo en las luces del semáforo	BrokerLightsAliveProbe	Mide la disponibilidad del agente de mensajería de las luces.
	BrokerLightsAliveMonitor	Actualiza la propiedad de adaptación is-broker-lights-alive.
	BrokerLightsAliveRule	Especifica los cambios para desplegar las luces remotas.
	BrokerLightsDeadRule	Especifica los cambios para replegar las luces remotas.
Estado de estrés del operario	StressProbe	Mide el nivel de estrés del operario.
	StressMonitor	Actualiza la propiedad de adaptación stress-exceeded.
	StressEnabledRule	Especifica los cambios para activar el modo de estrés.
	StressDisabledRule	Especifica los cambios para desactivar el modo de estrés.

Tabla 5.2: Componentes del bucle MAPE-k

5.5 Otros aspectos considerados

Uno de los problemas a los que nos hemos enfrentado en este proyecto, a nivel de diseño, es cómo actuar cuándo un componente no dispone de algún otro componente que este requiere para funcionar. Es un problema común en los sistemas auto-adaptativos, es una cuestión importante a decidir ya que es frecuente que se den estas situaciones en los entornos auto-adaptativos.

La solución que hemos decidido en este proyecto es la siguiente: los componentes comprobarán, en el momento que vayan a usar un componente que requieren, si este está disponible, y solo actuarán si este se encuentra activo y enlazado. Sin embargo, los componentes guardarán el estado en el que se encuentren, para poder aplicar ese estado inmediatamente a los componentes que se enlacen con él, cuando corresponda.

Siguiendo este planteamiento, el funcionamiento de los controladores de semáforo debería:

- guardar el estado en el que se encuentra, es decir, según los parámetros recibidos, debe guardar si quiere poner el semáforo en verde, amarillo o rojo,
- comprobar, cuando lo vaya a usar, si tiene enlazado un semáforo, al que se le pasará la orden si está enlazado, y se ignorará si no lo está,
- cuando se enlace un semáforo con el controlador, el controlador le pasará la orden correspondiente inmediatamente después de ser enlazado, de manera que el estado del semáforo se actualizará lo antes posible.

De esta forma, conseguiríamos que el estado se propague en cuanto los componentes se enlazan. Si no fuera así, podríamos encontrarnos con estados de luces que no son representativas de la realidad, cuando, funcionalmente, todos los componentes ya están disponibles, enlazados y capaces de mostrar el contexto de producción real. Así, se debería esperar a un cambio en el contexto de producción para que cambiara el estado del semáforo. Con la solución propuesta, no sería necesario esperar ese nuevo cambio, si no que, al enlazar los componentes, se lanzan las órdenes correspondientes, consiguiendo así propagación del estado, con lo que estaríamos más cerca de un estado del semáforo que representara la realidad de las lineas de producción.

CAPÍTULO 6
Implementación

La implementación de este proyecto se ha realizado mediante el lenguaje de programación Java, usando la librería MAPE-k lite proporcionada por el tutor de este trabajo [13]. La librería consiste en un conjunto de herramientas que facilitan el desarrollo de una solución auto-adaptativo bajo el modelo MAPE-k [2] que hemos explicado.

La librería MAPE-k lite consiste en una versión reducida del framework FADA, que proporciona herramientas que nos permiten integrar un bucle MAPE-k en nuestra herramienta. Además, ofrece una implementación para las fases de planificación y ejecución, por lo que nos centraremos en el desarrollo del resto de fases y del propio sistema.

6.1 Capa del sistema manejado

Antes de comenzar, mostramos la figura 6.1 de la solución del sistema diseñado, donde se resalta en verde la pieza que estamos implementado. Empezamos la implementación por la capa del sistema manejado.

Para comenzar con la implementación de la capa del sistema, creamos un paquete que contiene las diferentes interfaces del sistema, que permitirán la sustitución de unos componentes por otros. Las interfaces existentes ya han sido explicadas, nos limitamos ahora a enumerarlas: ILight, ITrafficLight, ITrafficLightController y ITrafficLightControllerAPI. Las interfaces solo son una declaración de la funcionalidad que implementarán los componentes, por lo que esta tarea no tiene un gran peso en la implementación, sin embargo sí lo tuvo en la fase de diseño de la solución.

A continuación, se implementaron los diferentes componentes. El uso de la librería MAPE-k lite requiere un desarrollo basado en componentes, consistiendo cada componente en un paquete OSGi [12], ya que, la librería, toma el sistema de módulos dinámicos de este. El trabajo realizado en este momento consistió en refactorizar la solución existente, dividiendo sus funcionalidades en diferentes componentes. Por ejemplo, los diferentes modos de funcionamiento del semáforo (modo manual, modo normal y modo estrés) fueron divididos en tres componentes, cada uno de ellos siendo un controlador de semáforo que implementa

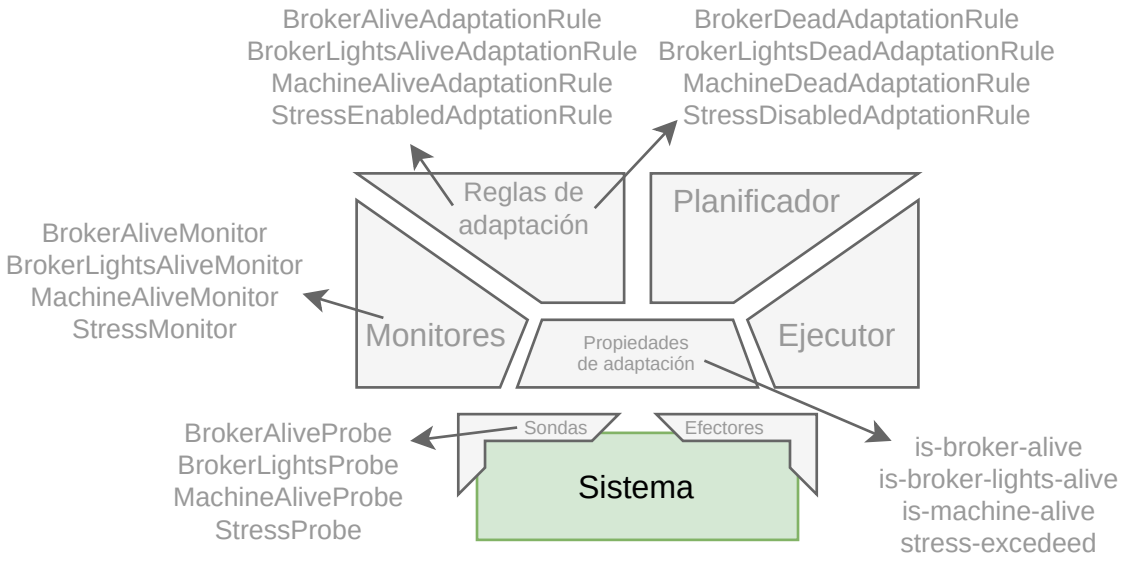

Figura 6.1: Diagrama de la solución: sistema

la interfaz ITrafficLightController. De este modo, los tres controladores son fácilmente sustituibles entre sí, y el bucle de control será capaz de intercambiarlos.

El componente que actuará como conector, que recibirá los datos del agente de mensajería, está implementado mediante la librería Eclipse Paho [4], que implementa el protocolo de mensajería MQTT [10]. El conector recibe mensajes en notación JSON [5], que serán interpretados y entregados al controlador de semáforo. Así mismo, los componentes «luz» también usan esta librería, pero esta vez no reciben datos, los publicarán, para que el simulador de interfaz de semáforo los interprete y pueda encender y apagar luces.

En este punto del desarrollo solo hemos amoldado la solución para facilitar la introducción del bucle de control en ella, pero todavía no hemos implementado ningún componente del bucle, ni hemos utilizado las herramientas que proporciona la librería MAPE-k lite. Tan solo hemos implementado los componentes del sistema, que ofrecen la funcionalidad que esperamos. Pero estos componentes por sí solos no poseen capacidad de auto-adaptación, esta viene dada por el bucle de adaptación y, para que éste sea capaz de manejar los componentes del sistema, tenemos que otorgar a los mismos una interfaz mediante la que el bucle logre su propósito. En otras palabras, tenemos que hacer que nuestros componentes, hasta el momento convencionales, estén preparados para ser adaptados.

Para ello, implementamos la interfaz *AdaptativeReadyComponent*, incluida en la librería. Esta interfaz se utiliza para que los componentes del sistema estén preparados para ser adaptados y, por tanto, que el bucle de adaptación sea capaz de manejarlos. Un componente está preparado para ser adaptado cuando implementa las siguientes funcionalidades:

- Función de despliegue, que configurará el componente para que sea utilizable, por ejemplo, estableciendo las conexiones pertinentes.

- Función de repliegue, dónde se realizarán las acciones necesarias para que el componente desaparezca de la configuración del sistema, por ejemplo, cerrando las conexiones que tenga establecidas.
- Función de enlace, en la que se establece la conexión entre componentes. Un componente que requiera de otro se enlazará con este, no al revés.
- Función de desenlace, en la que se revierte el proceso anterior.
- Función de cambio de propiedad, donde se introduce la capacidad de cambiar un atributo del componente.
- Función de obtención del componente, que devuelve la instancia del componente que ofrece la funcionalidad.

La implementación de esta interfaz se ha realizado envolviendo cada componente del sistema en un componente «preparado para adaptarse». De esta forma, existe un componente *adaptative ready*, que contiene al componente «normal» e implementa la interfaz explicada, concediendo a cada componente la capacidad de ser adaptado por el bucle de control.

A continuación, en el fragmento de código 6.1 situamos las partes esenciales de la implementación de la interfaz. Se puede ver como la clase TrafficLightARC envuelve a la clase TrafficLight, es decir, posee una referencia a ésta, exponiéndola al resto del sistema. Además, también se muestran las operaciones de enlace y desenlace con los componentes de tipo Light.

```
1  public class TrafficLightARC extends AdaptiveReadyComponent implements
       IAdaptiveReadyComponent {
2
3    protected TrafficLight the_tl = null;
4
5    public TrafficLightARC(BundleContext context) {
6      super(context, context.getBundle().getSymbolicName());
7      the_tl = new TrafficLight();
8    }
9
10   @Override
11   public IAdaptiveReadyComponent bindService(String req, Object value)
       {
12     String r = req.toLowerCase();
13     switch(r) {
14       case "l.red": this.the_tl.setRojo((ILight) value); break;
15       case "l.yellow": this.the_tl.setAmarillo((ILight) value); break;
16       case "l.green": this.the_tl.setVerde((ILight) value); break;
17     }
18     return this;
19   }
20
21   @Override
22   public IAdaptiveReadyComponent unbindService(String req, Object value
       ) {
23     String r = req.toLowerCase();
24     switch(r) {
25       case "l.red": this.the_tl.setRojo(null); break;
26       case "l.yellow": this.the_tl.setAmarillo(null); break;
```

```
27        case "l.green": this.the_tl.setVerde(null); break;
28      }
29      return this;
30    }
31    ...
```

Código 6.1: Fragmento de un *AdaptativeReadyComponent*

Una vez implementados los componentes, y habiéndolos preparados para ser adaptados, podemos empezar a implementar las partes del bucle de control pertenecientes al sistema manejado, es decir, las sondas y los efectores. Estos últimos vienen facilitados por la librería utilizada, por lo que nos centramos en la implementación de las sondas. En la figura 6.2 especificamos en que fase del desarrollo nos encontramos.

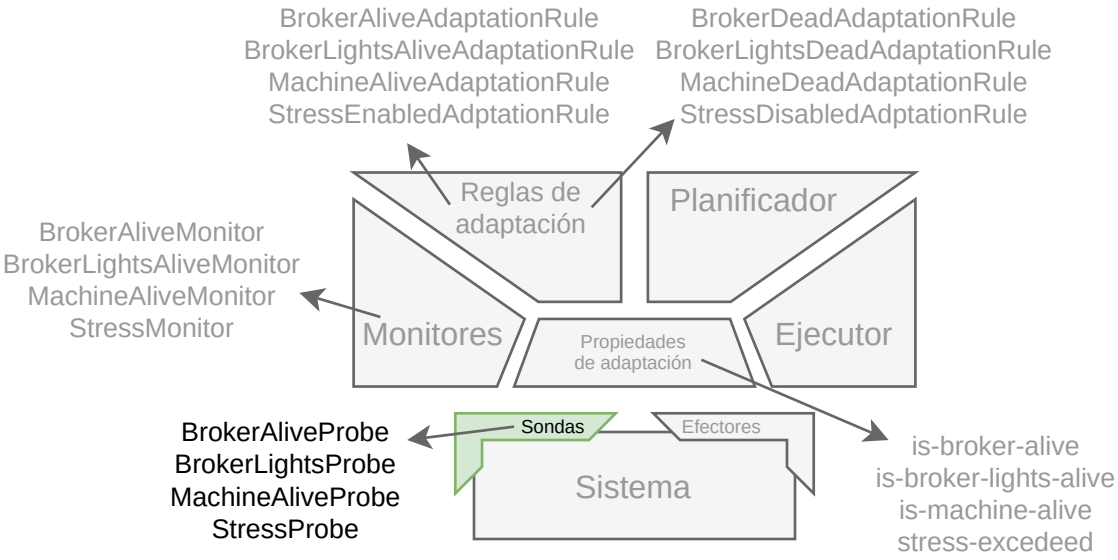

Figura 6.2: Diagrama de la solución: sondas

La librería MAPE-k lite proporciona una clase genérica de la cual podemos extender para que la implementación de las sondas sea más sencilla. Se implementan las sondas mediante un cliente MQTT, de forma similar al conector del sistema. Así, teniendo una conexión al agente de mensajería, podremos medir si hay conexión, en el caso de las sondas BrokerAliveProbe y BrokerLightsAliveProbe, o podremos recibir los mismos mensajes que recibiría el sistema, midiendo así la presencia de la máquina de producción, para MachileAliveProbe, o el nivel de estrés reportado, para StressProbe.

En el fragmento de código 6.2 podemos ver cómo se ha implementado la sonda StressProbe. Se trata de un cliente MQTT que reporta a su monitor el nivel de estrés medido utilizando el método reportMeasure(), facilitado por la clase Probe de la herramienta MAPE-k lite.

```
1 public void reportStress(Double s) {
2   this.reportMeasure(s);
3 }
4 ...
```

```
client.subscribe(topic,0,(topic,message) -> {
  Double stress = JSONStress.parse(message.toString());
  if(stress != null)
    this.reportStress(stress);
});
```

Código 6.2: Fragmento de una sonda

Hasta ahora hemos implementado la capa del sistema manejado, podemos pasar a implementar la capa de control.

6.2 Capa de control

Habiendo finalizado la implementación del sistema manejado, falta implementar las diferentes fases del bucle de auto-adaptación. Esta fase de implementación, nuevamente, es asistida por la librería MAPE-k lite, que proporciona módulos que implementan las diferentes fases del bucle, dejándonos con la tarea de implementar los diferentes monitores y las diferentes reglas, específicas a nuestra solución, así como definir las propiedades de adaptación.

6.2.1. Monitores

Empezamos a completar la capa de control implementando los monitores. De nuevo, mostramos la figura 6.3 de la solución para identificar en qué fase nos encontramos.

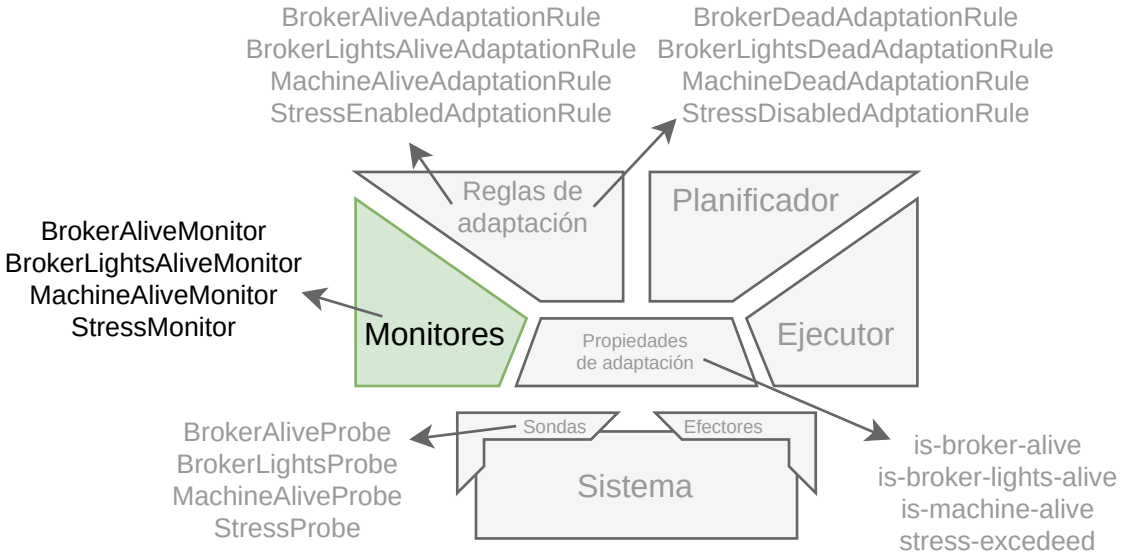

Figura 6.3: Diagrama de la solución: monitores

La implementación de los monitores se realiza extendiendo la clase «Monitor», donde escribiremos la lógica a seguir cuando una sonda reporta un valor al monitor. Esta lógica va desde comprobar que el dato es correcto hasta interpretarlo para decidir si actualizar las propiedades de adaptación o no. Cabe destacar

que en todos los monitores se realiza una comprobación final, que consiste en verificar que el valor de la propiedad que se decide actualizar no es el valor ya existente, evitando así una reactivación de las reglas cuando no es necesario.

Podemos observar una parte representativa de la implementación del monitor StressProbe, que es el encargado de mantener la propiedad de adaptación «stress-exceeded». En la línea número cinco vemos como el monitor filtra las mediciones en el valor 20.0 y, a continuación, se lee el valor de la propiedad de adaptación, para actualizarla en caso de que el nuevo valor difiera del ya existente, y así, la fase de análisis del bucle podrá decidir si es necesario realizar cambios sobre la configuración del sistema.

```
1  @Override
2  public IMonitor report(Object measure) {
3    try {
4      Double reported_stress = (Double) measure;
5      Boolean exceeded_stress = 20.0 > reported_stress;
6      IKnowledge knowledge = this.getTheMonitoringModule().
           getTheKnowledgeModule().getTheKnowledge();
7      IKnowledgeProperty kp = knowledge.getKnowledgeProperty("stress-
           exceeded");
8      // solo actualizamos si el valor es diferente
9      if ( kp.getValue() != exceeded_stress ) {
10       logger.debug(String.format("Updating Knowledge Property %s TO %
             s", kp.getId(), exceeded_stress));
11       kp.setValue(exceeded_stress);
12     }
```

Código 6.3: Fragmento de un monitor

6.2.2. Reglas de adaptación

La última fase del bucle que tenemos que implementar, la fase de análisis del mismo, se implementa mediante un conjunto de reglas de adaptación. En la figura 6.4 destacamos la fase de implementación en la que nos encontramos.

Del mismo modo, las reglas se implementan extendiendo la clase «AdaptationRule», donde los diagramas de especificación de cambios que se realizaron a nivel de diseño se traducen a código. En esta clase, la librería proporciona un objeto que representa la próxima configuración del sistema, donde se añade la especificación de componentes y enlaces que se deben añadir o quitar en la misma.

Para ejemplificar la implementación de una regla de adaptación, incluimos el fragmento de código 6.4 de la regla que activa el modo de estrés del sistema. En él podemos ver:

- En la línea 5, cómo se obtienen referencias a las especificaciones de componentes del sistema, ajenos a la regla.
- En la línea 7, cómo se especifica que se debe replegar un componente del sistema.
- En la línea 9, cómo se especifica que se debe eliminar un enlace entre componentes.

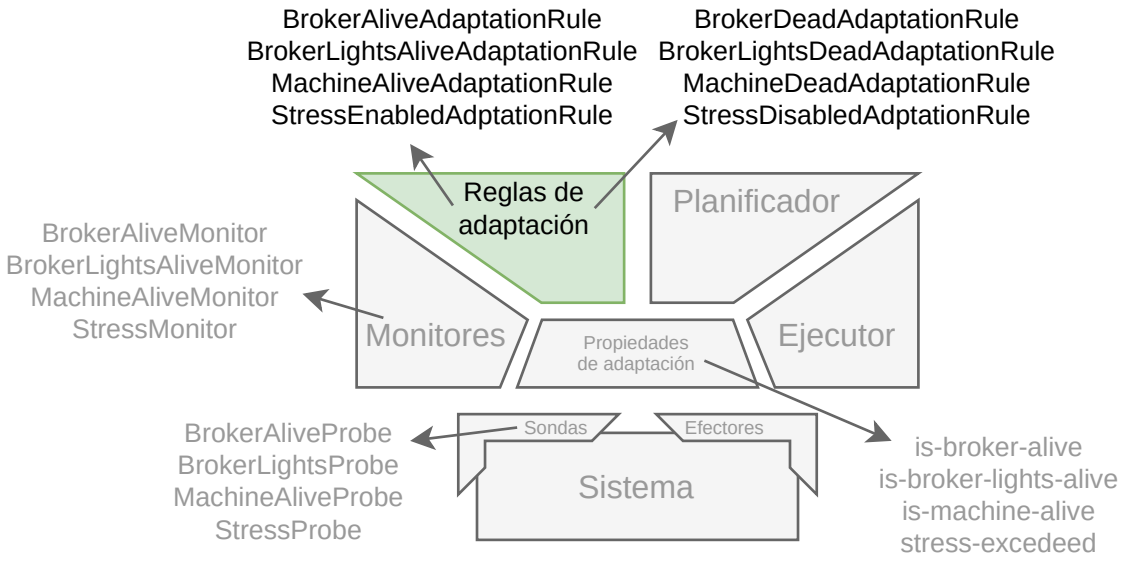

Figura 6.4: Diagrama de la solución: reglas de adaptación

- En la línea 12, cómo se especifica que se desplegar un nuevo componente en el sistema.

- En la línea 14, cómo se especifica que se debe establecer un enlace entre componentes.

```
1  @Override
2  public IRuleSystemConfiguration onExecute(IKnowledgeProperty property)
       throws RuleException {
3    ...
4    IRuleComponentsSystemConfiguration nextSystem =
         RuleComponentsSystemConfiguration.build(this.getId() + "_" +
         ITimeStamped.getCurrentTimeStamp(), REFERENCE_MODEL);
5    IComponentSpecification theStressTLC = ComponentSpecification.build("
         TLC.TrafficLightControllerReferenceSpeedStress", "1.0.0");
6    ...
7    nextSystem.componentToRemove(theTLCreference);
8
9    nextSystem.bindingToRemove(BindingSpecification.build(
         BindingEndpointSpecification.build(theTLCreference, "tl"),
10        BindingEndpointSpecification.build(theTL, "tl")));
11   ...
12   nextSystem.componentToAdd(theStressTLC);
13   ...
14   nextSystem.bindingToAdd(BindingSpecification.build(
         BindingEndpointSpecification.build(theStressTLC, "tl"),
15        BindingEndpointSpecification.build(theTL, "tl")));
16   ...
17   return nextSystem;
18 }
```

Código 6.4: Fragmento de una regla

6.2.3. Método de arranque

Para finalizar, se fue implementado, progresivamente conforme se añadían componentes al bucle, un método de arranque de la solución, que instancia los módulos necesarios para el funcionamiento del bucle y los entrelaza entre sí: las sondas son enlazadas con su respectivo monitor, cada uno de los monitores es enlazado con el módulo de monitorización de la librería MAPE-k lite, que a su vez se enlaza con el módulo de análisis. A este último, se le asociaron cada una de las reglas implementadas, y se enlaza con el módulo de planificación, este al de ejecución y por último, se engancha el módulo de efectores, completando la solución con el bucle de adaptación integrado.

También es en este punto donde se definen las propiedades de adaptación. Lo podemos ver en el fragmento 6.5, en la línea 10, dónde se crea la propiedad «is-broker-alive». También podemos ver, a partir de los comentarios en el código, cómo se inicializa un monitor, una regla de adaptación y una sonda, y cómo se enlazan a los módulos correspondientes, tal y como se ha descrito anteriormente.

```
1  public static void start(BundleContext bundleContext) {
2    ...
3    // MONITORS
4    IAdaptiveReadyComponent theBrokerAliveMonitorARC = new MonitorARC(
         bundleContext, new BrokerAliveMonitor(bundleContext));
5    theBrokerAliveMonitorARC.start();
6    theBrokerAliveMonitorARC.bindService(MonitorARC.
         REQUIRE_MONITORINGMODULESERVICE, theMonitoringModuleARC.
         getServiceSupply(MonitoringModuleARC.
         SUPPLY_MONITORINGMODULESERVICE));
7    theBrokerAliveMonitorARC.deploy();
8    ...
9    // ADAPTATION PROPERTIES
10   IKnowledgeProperty kp_brokerAlive = theKnowledge.
         createKnowledgeProperty("is-broker-alive");
11   ...
12   // ADAPTATION RULES
13   IAdaptiveReadyComponent theBrokerAliveRuleARC = new AdaptationRuleARC
         (bundleContext, new BrokerAliveAdaptationRule(bundleContext));
14   theBrokerAliveRuleARC.start();
15   theBrokerAliveRuleARC.bindService(AdaptationRuleARC.
         REQUIRE_ANALYZINGMODULESERVICE, theAnalyzingModuleARC.
         getServiceSupply(AnalyzingModuleARC.SUPPLY_ANALYZINGMODULESERVICE
         ));
16   theBrokerAliveMonitorARC.deploy();
17     ...
18   // PROBES
19   //broker-alive-probe
20   IAdaptiveReadyComponent brokeraliveProbeARC = new ProbeARC(
         bundleContext, new BrokerAliveProbe(bundleContext));
21   brokeraliveProbeARC.start();
22   brokeraliveProbeARC.bindService(ProbeARC.REQUIRE_MONITORSERVICE,
         theBrokerAliveMonitorARC.getServiceSupply(MonitorARC.
         SUPPLY_MONITORSERVICE));
23   brokeraliveProbeARC.deploy();
24   ...
25 }
```

Código 6.5: Fragmento del método de arranque

A partir de este momento, ya se encontraba el prototipo funcional preparado para hacer pruebas de adaptación con él. Así, en el siguiente capítulo, se detalla una prueba de funcionamiento de las capacidades de auto-adaptación del prototipo implementado.

CAPÍTULO 7
Pruebas de adaptación

Es este capítulo realizaremos pruebas de adaptación del sistema bajo un entorno controlado.

Hemos preparado un entorno en el que hemos desplegado un agente de mensajería, similar al utilizado en la fábrica, donde ejecutaremos la solución junto a un simulador de la máquina de producción y un simulador web de luces de semáforo, que funciona mediante otro agente de mensajería replicado.

La librería MAPE-k lite proporciona un repositorio de modelos en ejecución, que consiste en una serie de archivos en formato JSON que describen las diferentes configuraciones por las que pasa la solución. Usaremos esta herramienta para conocer qué componentes están desplegados en cada momento, así como las relaciones entre ellos.

7.1 Configuración inicial de la solución

Para comenzar con las pruebas, dejamos en funcionamiento tanto el agente de mensajería del controlador de semáforo como el agente de mensajería que utilizan las luces. De esta forma, al arrancar la solución, se deberían ejecutar las reglas BrokerAliveRule (figura 5.4) y BrokerLightsAliveRule (figura 5.10) El bucle de adaptación debería configurar los componentes, de manera que podremos visualizar el estado de las luces mediante el simulador web y podremos enviar mensajes «manuales» al agente de mensajería.

Una vez ejecutada la solución, exploramos el repositorio de modelos y vemos que la primera regla que se ha tenido en cuenta es BrokerAliveRule, ya que la sonda BrokerAliveProbe reporta que el agente mensajería está disponible. El sistema queda en el siguiente estado tras la ejecución de la regla:

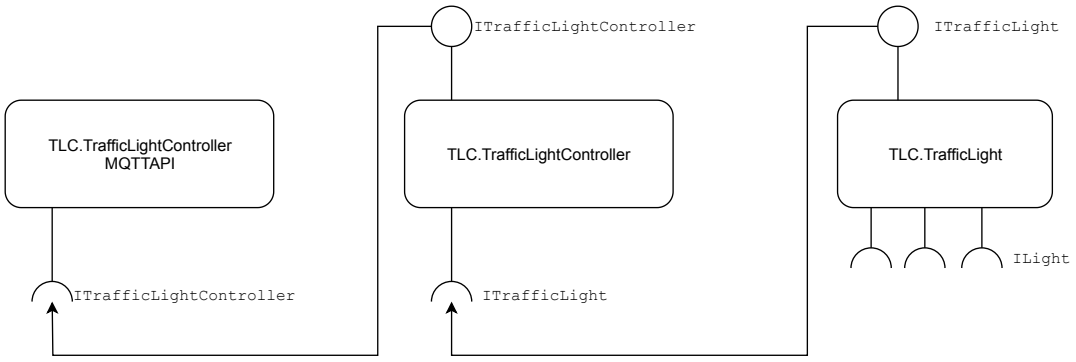

Figura 7.1: Primer paso de adaptación

Como podemos ver, se despliegan tres componentes: el conector, el controlador de semáforo y el semáforo. Sin embargo, esta configuración del sistema todavía no es funcional, no se han desplegado luces, por lo que no «existe» una interfaz con la que ver el estado de la producción de la fábrica.

A continuación, inmediatamente después de este paso de adaptación, se reporta al bucle de MAPE-k la presencia del agente de mensajería que utilizan las luces de la solución, provocando la ejecución de la siguiente regla, BrokerLightsAliveRule, quedando el la configuración del sistema de esta forma:

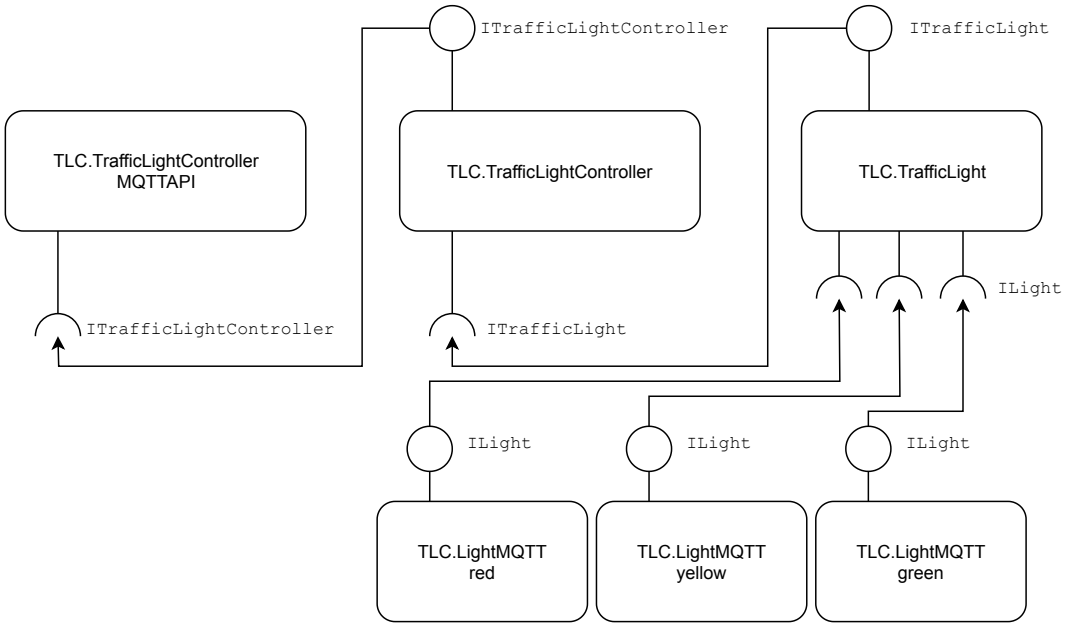

Figura 7.2: Configuración inicial del sistema

En esta configuración, que llamaremos inicial, puesto que es la primera configuración del sistema que es funcional, podemos enviar órdenes manuales al agente de mensajería, y vemos el cambio de estado reflejado en el simulador web.

Por ejemplo, emitiendo una orden de encender luz verde, vemos como cambia el estado del semáforo en el simulador:

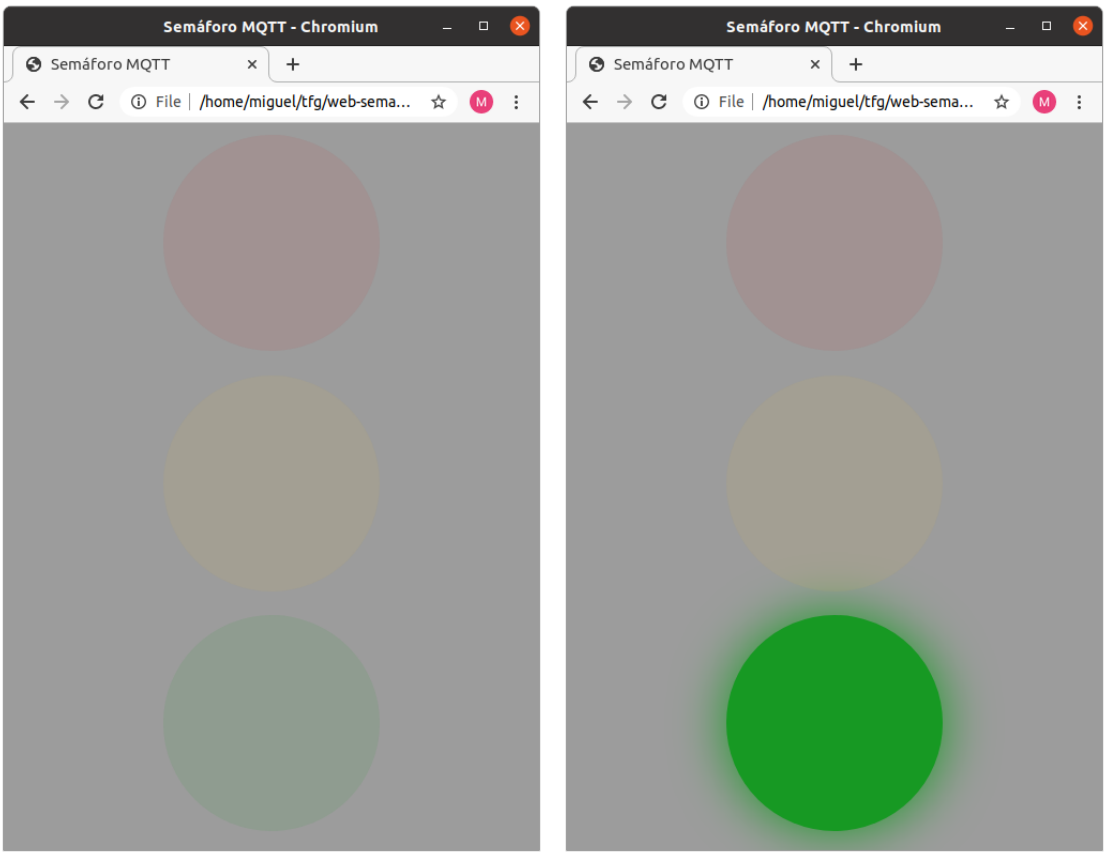

Figura 7.3: Cambio de estado en el simulador web

7.2 Configuración en presencia de máquina

Prosiguiendo con las pruebas de funcionamiento, ejecutamos el simulador de máquina de producción, que reportará datos de producción al agente de mensajería, de forma que la sonda MachineAliveProbe captará su presencia, causando la ejecución de la regla MachileAliveRule. Podemos ver a continuación como queda la configuración del sistema.

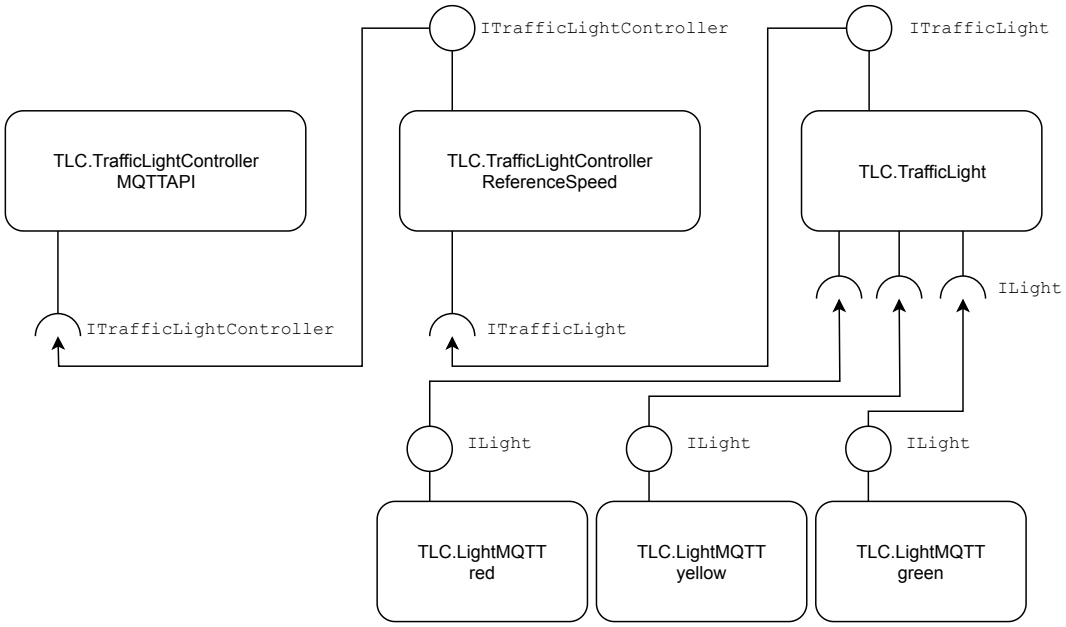

Figura 7.4: Configuración con máquina de producción

Observamos que, efectivamente, se han llevado a cabo los cambios que proponía la regla: se ha eliminado de la configuración el controlador «manual» (TrafficLightController), y se ha desplegado el controlador que integra datos de la máquina de referencia (TrafficLightControllerReferenceSpeed), y se han establecido los enlaces con los componentes correspondientes.

En este momento, el sistema reaccionaría a los datos reportados por la máquina de producción (en este caso, del simulador). Ahora, para realizar la siguiente prueba, paramos la ejecución del simulador, causando la ejecución de la regla MachineDeadRule, y volviendo a la configuración inicial del sistema (7.2).

7.3 Configuración con conexión perdida

La siguiente prueba de adaptación que vamos a realizar consiste en apagar el servicio del agente de mensajería, simulando de este modo una caída en la conexión con el mismo.

Recordemos que el sistema se encuentra en la configuración inicial (7.2). Al apagar el agente de mensajería, BrokerAliveProbe reportará la ausencia del mismo, causando que entre en funcionamiento la regla BrokerDeadRule, que debe replegar el componente que actúa como conector, así como replegar el controlador «manual», para desplegar y enlazar el controlador en bucle, que se usa en caso de problemas en las conexiones. Podemos presenciar que esto ocurre en la siguiente figura, representativa de la nueva configuración del sistema:

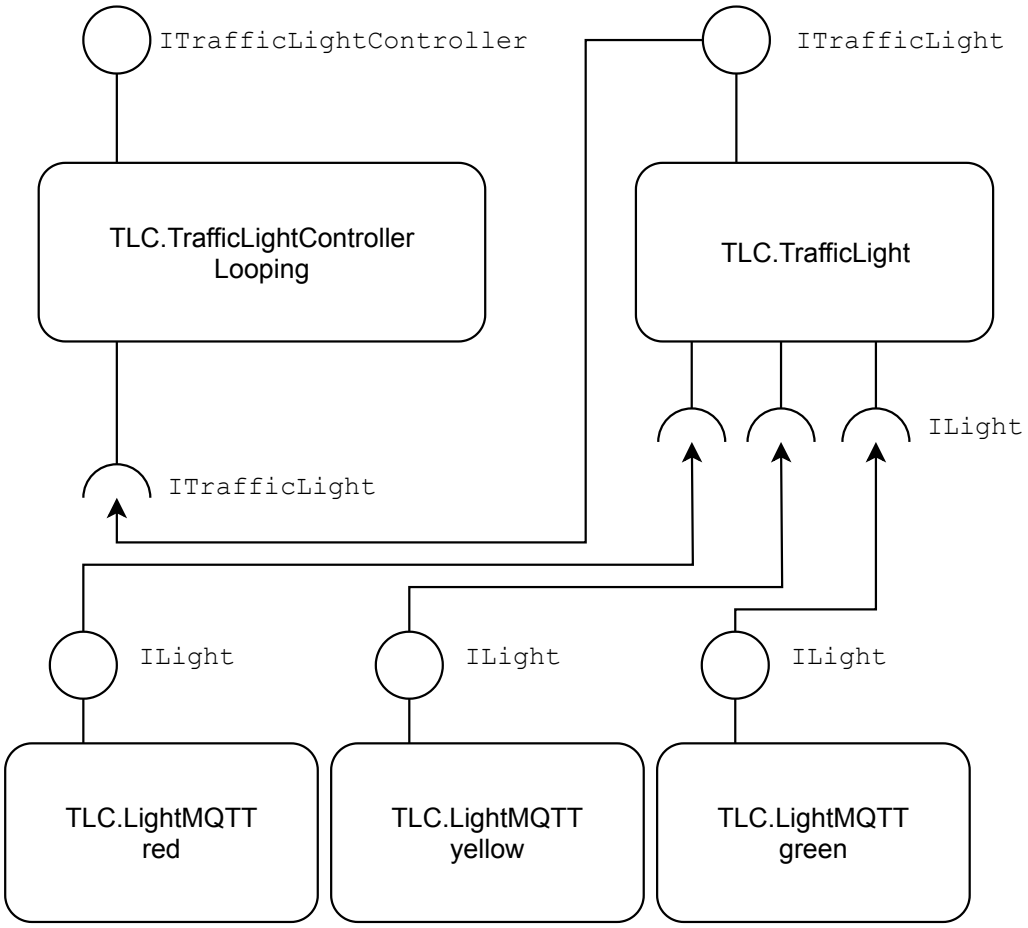

Figura 7.5: Configuración con fallo de conexión

En este estado, las luces parpadearían en bucle de la siguiente forma:

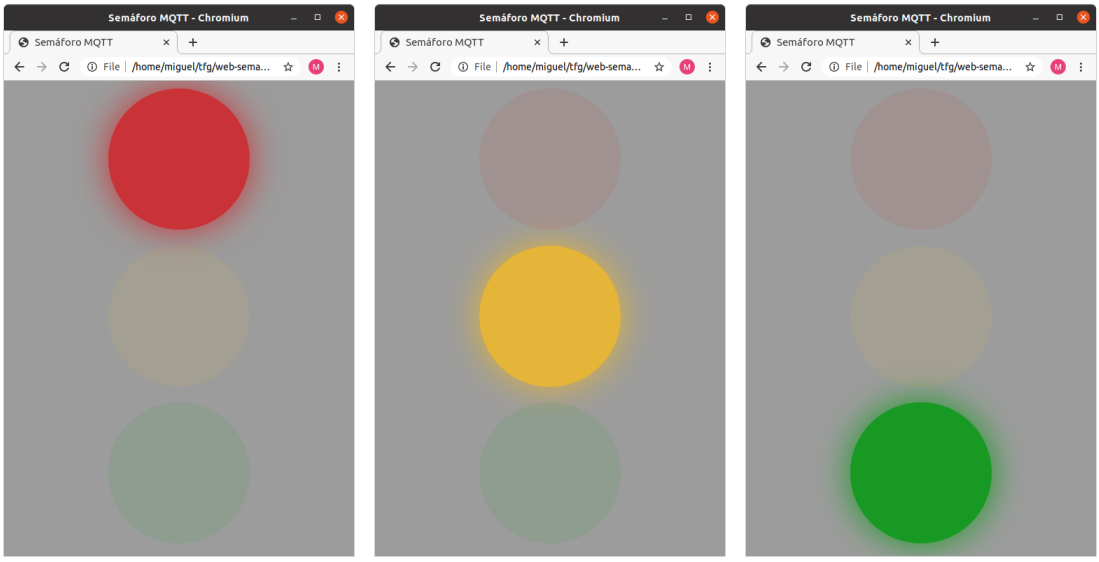

Figura 7.6: Simulador web con fallo de conexión

Aunque durante la fase de pruebas hemos comprobado el funcionamiento de todos los requisitos de adaptación, detenemos en este punto la descripción de los mismos, ya que los ejemplos de pasos de adaptación detallados en el capítulo son representativos del trabajo realizado en esta fase del proyecto. Dejamos paso al capítulo que concluye la memoria del proyecto.

CAPÍTULO 8
Conclusiones

Las sistemas informáticos son cada vez más complejos, y habitualmente su correcto comportamiento depende de otras soluciones o recursos que no siempre estarán disponibles. Normalmente se diseñan sistemas que no tienen en cuenta estos aspectos, pero, en entornos industriales como es el caso, estas cuestiones son imprescindibles para la viabilidad de las soluciones.

En este proyecto se ha detallado el proceso de diseño e implementación del prototipo de una solución industrial, moviéndola a un enfoque de computación autónoma, para lograr un sistema auto-adaptativo, capaz de reaccionar a cambios en su entorno.

El desarrollo del proyecto a supuesto una serie de retos, tales como el aprendizaje teórico sobre los conceptos de computación autónoma, bucles de control y bucles MAPE-k, así como el aprendizaje de las herramientas que nos han permitido implementar la solución. Pero todo ello ha permitido que se cumplan los objetivos que se establecieron al principio del proyecto: identificar los requisitos de adaptación, diseñar e implementar una solución, siempre desde el punto de vista de la computación autónoma.

A pesar de haber logrado los objetivos propuestos, podríamos mejorar la solución, utilizando el framework FADA, en vez de su simplificación, la librería MAPE-k lite, que ofrece mayor flexibilidad en desarrollos más amplios, como podría resultar éste, una vez implantado en las líneas de producción e identificados posibles nuevos requisitos de adaptación.

Para finalizar, durante este grado se han impartido una diversidad de materias que han otorgado el conocimiento necesario para realizar este proyecto. Destacamos especialmente programación orientada a objetos, ingeniería del software, integración de aplicaciones y tecnologías en red, aunque todas ellas han aportado una pequeña parte que ha permitido culminar el grado con este proyecto.

Bibliografía

[1] Nelly Bencomo, Paul Grace, Flores-Cortes Carlos, Danny Hughes, y Gordon Blair. Genie: supporting the model driven development of reflective, component-based adaptive systems. págs. 811–814, 01 2008.

[2] An Architectural Blueprint for Autonomic Computing. Technical report, IBM, June 2005.

[3] Y. Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, y P. Pezze. *Engineering self-adaptive systems through feedback loops*. 01 2009.

[4] Cliente MQTT para Java. https://www.eclipse.org/paho/clients/java/. Recuperado julio 2020.

[5] Formato de intercambio de datos JSON. https://www.json.org/json-es.html. Recuperado julio 2020.

[6] Jeffrey Kephart y D.M. Chess. The Vision Of Autonomic Computing. *Computer*, 36:41 – 50, 02 2003.

[7] Christian Krupitzer, Felix Roth, Christian Becker, Markus Weckesser, Malte Lochau, y Andy Schürr. FESAS IDE: An Integrated Development Environment for Autonomic Computing. págs. 15–24, 07 2016.

[8] Rogério Lemos, Holger Giese, Hausi Müller, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, y Jochen Wuttke. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*, págs. 1–32. 01 2013.

[9] Daniel Menascé, Hassan Gomaa, Sam Malek, y Joao Sousa. SASSY: A Framework for Self-Architecting Service-Oriented Systems. *Software, IEEE*, 28:78 – 85, 01 2012.

[10] Protocolo de mensajería MQTT. http://mqtt.org/. Recuperado julio 2020.

[11] Framework OSGi. https://www.osgi.org/developer/what-is-osgi/. Recuperado julio 2020.

[12] Arquitectura de OSGi. https://www.osgi.org/developer/architecture/. Recuperado julio 2020.

[13] Grupo de investigación Tatami del centro PROS de la UPV. http://www.pros.webs.upv.es/pros-center/. Recuperado julio 2020.