UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Adaptación de maquinaria para conformado plástico al entorno de Industria 4.0

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Manuel José Martínez Baños

Tutor: José Enrique Simó Ten

Curso 2018-2019

Resum

Este projecte va començar amb la idea de poder oferir un sistema que afavorisca la disponibilitat de dades i suport programari multiplataforma per a visualització de maquinàries i plantes de producció, però a causa de la retroalimentació durant el desenrotllament, es detectà que la proposta en si, ténia una certa analogia amb noves i revolucionàries idees d'automatització de sistemes de producció, com les que cobrix una de les branca d'I4.0. En conseqüència s'afig la idea addicional d'intentar aportar una part de la implementació i metodologia de la Indústria 4.0, al sector del plàstic en general. Amb una sèrie de clars objectius:

1. Conéixer correctament les màquines utilitzades en el sector, per a poder extraure i generar dades de manera persistent.
2. Creació d'un entorn software capaç de mostrar de manera adequada i ordenada la informació, per a distints tipus d'usuaris o rols.
3. Sistema de notificacions i visualitzat detallat d'informació personalitzada.
4. Documentar i analitzar totes les implicacions que l'entorn Indústria 4.0 implica.
5. Implementar els coneixements obtinguts, per a una correcta representació i virtualización dels actius, que al seu torn, són part del sistema de producció d'una empresa.

Per tant, aquest projecte està realitzat de tal forma, que permeta o facilite en un futur, ser part d'un sistema adaptat a l'entorn Indústria 4.0. Aquesta adaptació deurà ser escalonat i ha de conviure amb l'entorn actual de l'empresa, sense perjudicar ni incumbir en l'entorn software ja funcional.

Hui en dia, cada vegada se sol·licita una major adaptabilitat en els processos de producció i una assignació més eficient de recursos, per mitjà de fàbriques més intel·ligents on les instal·lacions han d'aconseguir autogestionar-se. A causa d'aquestes i altres sol·licituds, naix el paradigma d'Indústria 4.0. Un nou model d'organització i control de la cadena de producció, emprant tecnologies de la informació per a dur-ho a terme.

En particular, en este projecte es realitza una aportació i adaptació de les parts més rellevants d'Indústria 4.0, com és l'estructuració, representació i comunicació d'actius, acotant al sector plàstic. En estar en contacte amb empreses d'este sector, s'han pogut analitzar les seues necessitats i afrontar-les, mantenint el concepte d'indústries 4.0, el màxim possible. Fent front, a diverses sol·licituds, amb el maneig de tecnologies WEB, com API REST i Ionic per al desenrotllament d'una interfície funcional multiplataforma, motor de base de dades, SQLSERVER i servicis com IIS. Sent tot això, la idea original del projecte el qual s'ha adaptat per a brindar un gra d'arena a tot l'ampli concepte que Indutria 4.0 comprén.

Paraules clau: Industria 4.0, Sistema distribuït, Multiplataforma, Android, Web, Plàstic

Resumen

Este proyecto empezó con la idea de poder ofrecer un sistema que favorezca la disponibilidad de datos y soporte software multiplataforma para visualización de maquinarias y plantas de producción, pero debido a la retroalimentación durante el desarrollo, se detecto que la propuesta en si, tenia una cierta analogía con nuevas y revolucionarias ideas de automatización de sistemas de producción, como las que cubre una de las rama de I4.0. En consecuencia se añade la idea adicional de intentar aportar una parte de la implementación y metodología de la Industria 4.0, al sector del plástico en general. Con una serie de claros objetivos:

1. Conocer correctamente las máquinas utilizadas en el sector, para poder extraer y generar datos de manera persistente.
2. Creación de un entorno software capaz de mostrar de manera adecuada y ordenada la información, para distintos tipos de usuarios o roles.
3. Sistema de notificaciones y visualizado detallado de información personalizada.
4. Documentar y analizar todas las implicaciones que el entorno Industria 4.0 implica.
5. Implementar los conocimientos obtenidos, para una correcta representación y virtualización de los activos, que a su vez, son parte del sistema de producción de una empresa.

En consecuencia, este proyecto esta realizado de tal forma que permita o facilite en un futuro, ser parte de un sistema adaptado al entorno Industria 4.0. Esta adaptación deberá de ser escalonado y debe convivir con el entorno actual de la empresa, sin perjudicar ni incumbir en el entorne software ya funcional.

Hoy día, cada vez se solicita una mayor adaptabilidad en los procesos de producción y una asignación más eficiente de recursos, mediante fábricas más inteligentes donde las instalaciones deben lograr auto-gestionarse. Debido a estas y otras solicitudes, nace el paradigma de Industria 4.0. Un nuevo modelo de organización y control de la cadena de producción, empleando tecnologías de la información para llevarlo a cabo.

En particular, en este proyecto se realiza una aportación y adaptación de las partes más relevantes de Industria 4.0, como es la estructuración, representación y comunicación de activos, acotando al sector plástico. Al estar en contacto con empresas de este sector, se han podido analizar sus necesidades y afrontarlas, manteniendo el concepto de Industrias 4.0. Haciendo frente a varias solicitudes, con el manejo de tecnologías WEB, como API REST y Ionic para el desarrollo de una interfaz funcional multiplataforma, motor de base de datos, SQLSERVER y servicios como IIS. Es esto por consiguiente, la idea original del proyecto, el cual se ha adaptado para brindar un granito de arena a todo el amplio concepto que Industria 4.0 abarca.

Palabras clave: Industria 4.0, Sistema distribuido, Multiplataforma, Android, Web, Plástico

Abstract

This project began with the idea of being able to offer a system that favors the availability of data and support multiplatform software for visualization of machinery and production plants, but due to the feedback during the development, it is defective that the proposal itself had a certain analogy with revolutionary new ideas of production system automation, such as those covered by one of the branches of I4.0. Consequently, the additional idea of trying to contribute a part of the implementation and methodology of Industry 4.0 to the plastics sector in general is added. With a series of clear objectives:

1. Know the machines used in the sector correctly, in order to generate data persistently.
2. Creating a software environment capable of displaying information in an appropriate and orderly manner, for different types of users or roles.
3. Notification system and detailed visualization of personalized information.
4. Document and analyze all the implications that the Industry 4.0 environment implies.
5. Implement specific knowledge, for a correct representation and virtualization of assets, which in turn, are part of a company's production system.

Nowadays, a greater adaptability in the production processes and a more efficient allocation of resources is requested, through smarter factories where the facilities must achieve self-management. This adaptation must be staggered and must coexist with the current environment of the company, without harming or incumbent on the already functional software environment.

Increasingly, adaptability in production processes and a more efficient allocation of resources are requested, through smarter factories where must achieve self-management. Due to these and other requests, the Industry 4.0 paradigm is born. Due to these and other requests, the Industry 4.0 paradigm is born. A new model of organization and control of the production chain, using information technologies to carry it out.

In particular, this project will make a contribution and adaptation of the most relevant parts of Industry 4.0, such as structuring, representation and communication of assets, limiting the plastic sector. By being in contact with companies in this sector, they have been able to analyze their needs and face them, to have the concept of 4.0 industries, as much as possible. In the face of several requests, with the management of WEB technologies, such as REST and Ionic APIs for the development of a cross-platform functional interface, database engine, SQLSERVER and services such as IIS. Being all this, the original idea of the project, which has been adapted to provide a grain of sand all the broad concept that Indutria 4.0 covers.

Key words: Industry 4.0, Distributed System, Multiplatform, Android, Web, Plastic

Índice general

Índice de figuras

Índice de tablas

Glosario

AAS Asset Administration Shell. 1, 26

AAS centralizado AAS para un único activo. Todos los elementos de datos y servicios están en una ubicación central con un único punto de entrada (por ejemplo, una dirección IP).. 1, 36

ADAM-4520 Convertidor de señal RS-232 en señales aisladas RS-422 o RS-485, para una comunicación de larga distancia en un grado Industrias.. 1

Angular Framework de código abierto utilizado para crear aplicaciones web, en el contexto de angular, el modelo es el framework, mientras las vista es HTML y el control es TypeScript/JavaScript.. 1

Componente Dashboard Descendiente de la clase Dashboard ofrecida por DevExpress. Es creado y editado en Visual Studio designer o con la opcion de edicion que contiene la propia clase en un WebForms y puede ser visualizada con Dashboard Viewer's. Se puede almacenar el componente dashboard como un fichero XML. 1

Componente Dashboard Es un superconjunto (superset) de JavaScript que se compila a simple JavaScript. Denominado superconjunto debido a que la tecnología TypeScript puede ejecutar programas de JavaScrip. 1, 25

Dashboard Un componente que utiliza diferentes tipos de elementos para la visualización de datos y permite a los usuarios interactuar con ellos.. 1, 33

Dashboard Viewer Control que permite visualizar los Dashboards creados en Escritorio o la aplicación Web.. 1

DevExpress Este es un almacén de componentes de UI para el desarrollo en todas las plataformas de .NET y C# como Windows Forms, ASP.NET o Angular.. 1, 18, 33

Digital Factory Sus estándares definen métodos y modelos de información, que son necesarios para la auto-descripción de los componentes I40. IEC TS 62832 define métodos que permiten validar o planificar conexiones entre activos.. 1

IIS Internet Information Services, Software propietario formado por una serie de servicios para el sistema operativo de Microsoft Windows, para la creación de un servidor web.. 1, 30

Ionic Es un SDK de código abierto para desarrollar Progressive Web Apps (PWA) y apps nativas con JavaScript, CSS y HTML5. El framework del que dispone es altamente poderosos debido a que mezcla dos frameworks importantes Angular y Cordova.. 1, 24

JSON JavaScript Object Notation, es un formato estándar ligero de intercambio de datos. 1, 30

Listado (DataGridView) Componente de DevExpress que representa datos de una fuente local o remota en forma de una cuadrícula, com muchas funcionalidades extras para el cliente.. 1, 21

Microsoft ASP.NET Plataforma web madura que proporciona todos los servicios para crear aplicaciones web basadas en servidor de clase empresarial utilizando .NET en Windows. Funciona sobre servidores tanto en IIS como en Apache. Las páginas web generadas en ASP.NET, suelen ser conocidas como "web forms", es el principal medio de construcción pero no el único, Microsoft brinda varias opciones. Actualmente ASP.NET soporta tres modelos de programación: ASP.NET Web Forms, ASP.NET MVC y ASP.NET Web Pages. 1, 29

Objeto Observable Es objeto obervador, lo que quiere decir que se registra y actúa en consecuencia. Proporciona soporte para mensaje bajo subscripciñon y aporta facilidades para el manejo de eventos o programación asíncrona . 1, 31

REST Representational State Transfer, estilo de desarrollo web que se apoya totalmente en el estándar HTTP y centrado en recursos. REST ofrece siempre el mismo interfaz de acceso a recursos a diferencia de SOAP. 1, 26

REST Arquitectura Orientada a Servicios, formada por numerosos servicios débilmente acoplados y interoperables.. 1, 26

RESTFul web service Es la utilización de la tecnología y arquitectura REST bajo un servicio web. Para la implementación de operaciones CRUD mediante la semántica de operaciones HTTP.. 1

RS-485 Estándar de transmisión en bus de la capa física, es ideal para transmitir a altas velocidades sobre largas distancias, incluso a través de canales ruidosos. Por ejemplo, algunas unidades de control del pasajero lo utilizan.. 1, 15

TDS-712 Es un servidor de dispositivo RS-232/RS-485 a Ethernet y eliminar la limitación de utilizar cables en serie. pueden utilizarse para transmitir datas a través de TCP/IP entre dos dispositivos en serie.. 1

Vistas SQL bjeto que se puede buscar en una base de datos definida mediante una consulta. Aunque una vista no almacena datos, algunos se refieren a las vistas como "tablas virtuales". 1, 22

Web API Es una API a través de la web a la que se puede acceder mediante el protocolo HTTP. Es un concepto y no una tecnología. 1, 23

Wep API ASP.NET Es un framework de codigo abierto para la creación de servicios basadosen HTTP al que se puede acceder en diferentes aplicaciones en diferentes plataformas.. 1

XML Metalenguaje que nos permite definir lenguajes de marcado adecuados.. 1, 33

CAPÍTULO 1
Introducción

Hoy en día, ya se han empezado a establecer en diferentes industrias una monitorización de sistema de producción o de control. Cada vez se solicita una mayor adaptabilidad a las necesidades y a los procesos de producción, además de una monitorización global y una asignación más eficiente de recursos. Todo recae, finalmente, en el concepto de crear fábricas más inteligentes, las instalaciones deben lograr auto-gestionarse de forma más autónoma. Debido a estas y otras necesidades, nace el paradigma de Industria 4.0. Este término se refiere a un nuevo modelo de organización y control de la cadena de valor o producción, a lo largo de los sistemas de fabricación y producción, empleando las tecnologías de la información para llevarlo a cabo.

Este proyecto empezó con la idea de poder ofrecer un sistema software multiplataforma para la visualización de maquinarias y plantas de producción, pero debido a la retroalimentación durante el desarrollo, se detectó que la propuesta en sí, tenía una cierta analogía con nuevas ideas revolucionarias de automatización de sistema de producción, como las que cubre una de las ramas de I4.0. En consecuencia se añade el objetivo adicional de intentar aportar una parte de la implementación y metodología de la Industria 4.0, al sector plástico en general. Todo ello será especificado a continuación en los objetivos principales del proyecto.

En particular, en este proyecto se procede a realizar, una aportación y adaptación, de un fragmento base de Industria 4.0, como la estructuración, la representación y la comunicación de activos y sus datos, acotando a un tipo de sector, el plástico. Al estar en contacto con empresas de este sector, se han podido analizar sus necesidades y afrontarlas, manteniendo el concepto de Industria 4.0, lo máximo posible. Además, se aplican tecnologías WEB (API REST y Ionic) y base de datos SQLSERVER, todo desplegado en servicios IIS. Se procede a resolver las demandas del sector y atacar el objetivo original del proyecto, que una vez resueltas, se recrean varios ejemplos cavle para la incursión y implementación hacia metodologías del ámbito Industria 4.0.

1.1 Motivación

Al inicio el motivo principal fue la solución por medio de tecnología web de unas demandas de clientes del sector plástico, para resolver problemas de escalabilidad, disponibilidad y tolerancia a fallos, pero durante el desarrollo el motivo principal quedo relevado a secundario debido a Industria 4.0. Por consiguiente, el motivo principal ahora afrontar este proyecto es aprender y dar a conocer más sobre esa supuesta revolución 4.0 que siempre es noticia. Además de, alcanzar a crear una base que sirva como sustento

para implementar el paradigma Industria 4.0 en futuros desarrollos. Conjuntamente con el motivo principal, también existe la motivación de cumplir los requerimientos demandados a través de tecnologías aprendidas sobre el curso

1.2 Objetivos

El objetivo principal de este proyecto es ofrecer un software funcional multiplataforma, con el propósito de mostrar información personalizada, para empresas del sector plástico, en concreto para la visualización del estado de máquinas y unidades de fabricación. Estableciéndose unos objetivos claros:

- Elaboración de un servicio capaz de ofrecer mediante servicios REST, una representación jerárquica de la información.
- Creación de un entorno software capaz de mostrar de manera adecuada y ordenada la información ofrecida.
- Software capaz de modificar el aspecto o información ofrecida, para distintos tipos de usuarios o roles.
- Sistema de visualizado de datos detallada y personalizada.
- Ofrecer el software creado en cualquier plataforma.

Todos estos objetivos se han ido creando y perfeccionando durante el desarrollo en espiral del proyecto. Debido a la auto realimentación del enfoque cíclico, se observan muchos de los objetivos similares a los de Industria 4.0. Objetivos para alcanzar "La atomización y desarrollo de fábricas auto-gestionables"". Por tanto, se añaden nuevos objetivos adicionales al proyecto, asumiendo los riesgos que ahora se proponen con seguridad, gracias a la evaluación de riesgos el modelo en espiral. Dichos objetivos adicionales son los siguientes:

- Documentar y analizar todas las implicaciones que el entorno Industria 4.0 en el entorno de desarrollo donde se sitúa el proyecto.
- Implementar los conocimientos obtenidos, dando como resultado, una representación y virtualización de los activos, que a su vez, son parte del sistema de producción de una empresa.
- Poder ofrecer una comunicación mínima en base al protocolo OPC UA, ofreciendo unos mínimos datos del activo.

La finalidad de estos objetivo, es aportar e implementar una estructura del modelo Automation UML, utilizar submodelos con identificación basados en URI (ID única)o IRDI y implementar un protocolo de comunicación OPC UA. Todo ello, apoyado por los estándares y modelos de referencia RAMI 4.0, de Industria 4.0. Logrando una representación con una semántica precisa y consensuada, facilitando una interpretación sencilla de los datos, existentes en CDD (Common Data Dictionary). Ello permite la interoperabilidad de activos, con ayuda y apoyo de estándares consensuados por Industria 4.0, como el IEC 61360, proporcionando referencias a la estructura, contenido y especificación de datos del diccionario (CDD).

Todo ello, se realiza de tal manera que permita y facilite en un futuro, ser parte de un

sistema adaptado al entorno de Industria 4.0. Esta adaptación deberá de ser escalonada y debe convivir con el entorno actual de la empresa, sin perjudicar ni incumbir en el entorno software actual.

1.3 Metodología

Durante los comienzos del proyecto, se fijan unas necesidades y objetivos, algunos de estos objetivos surgen de la comunicación y demanda de varias empresa del sector plástico.

Una vez, establecidas las mínimas necesidades, se verifica que dichos requerimientos a cubrir sean posibles de realizar mediante los medios ofrecidos y que las funciones futuras puedan ser soportadas. Para cubrir los objetivos principales presentados, se descompone el desarrollo en varias partes diferenciadas:

1. Una página web para poder generar y visualizar Dashboards o listados, además de añadir una mínima herramienta administrativa, para resolución de errores.
2. Un Servicio API REST, para ofrecer y controlar de manera jerárquica, toda la información referente a la empresa, incidiendo en la planta de máquinas.
3. Aplicación multiplataforma para la visualización de maquinarias y plantas de producción, junto a datos relativos a la empresa y Dashboards o listados generados
4. Implementación de un AAS funcional.

Para el desarrollo software de este proyecto entre todas las metodologías ágiles, se ha seguido un desarrollo en espiral con uso de prototipos.Dicho modelo tiene un esfuerzo es iterativo conforme termina una etapa o meta prevista del desarrollo, al mismo tiempo que comienza otro. Cada ejecución del desarrollo se divide en varia etapas principales: Requisitos, Criterio de diseño, Diseño, Implementación, Pruebas.

Con cada iteración de la espiral surgen más versiones del software, cada vez más completas hasta alcanzar una versión software totalmente funcional. Este modelo ha sido escogido debido a una serie de ventaja que aporta al desarrollo del proyecto:

- No requiere un conocimiento exacto de todas las propiedades del programa en cuestión, para empezar con la funcionalidad.
- Los retrasos tiene un riesgo menor, ya que los motivos pueden revisarse a tiempo gracias a los prototipos.
- Se van aportando versiones continuas y mejoras.

Junto a este modelo, es utilizada una herramienta para un control de versiones, como Git y otra herramienta Trello para una gestión del proyecto, ofreciendo la posibilidad de crear rutinas o tareas organizadas y visibles, dando la posibilidad de priorizar y ofrecer tiempos previos a la tareas.

1.4 Estructura de la memòria

El documento se divide en diferentes fases de desarrollo, correspondientes a la metodología ágil elegida. Primero se determina el alcance y los requisitos mínimos para

un software funcional que las empresas requieran. A continuación, se empieza con un proceso de diseño donde se determina el esqueleto para la futura implementación de la solución, exponiendo que moldes y organización se ha establecido para realizar la siguiente fase de Implementación.

En dicha fase, se muestran diferentes tecnologías utilizadas, fragmentos de código y referencias a estándares utilizados. Seguido de la implementación, se procederá a mostrar una o varias pruebas donde se presentan las distintas implementaciones funcionando.

Toda esta estructura mantiene un plan de trabajo con ayuda de la herramienta Trello. Se adjunta una captura de pantalla que representa en una de las etapas del trabajo durante el desarrollo.

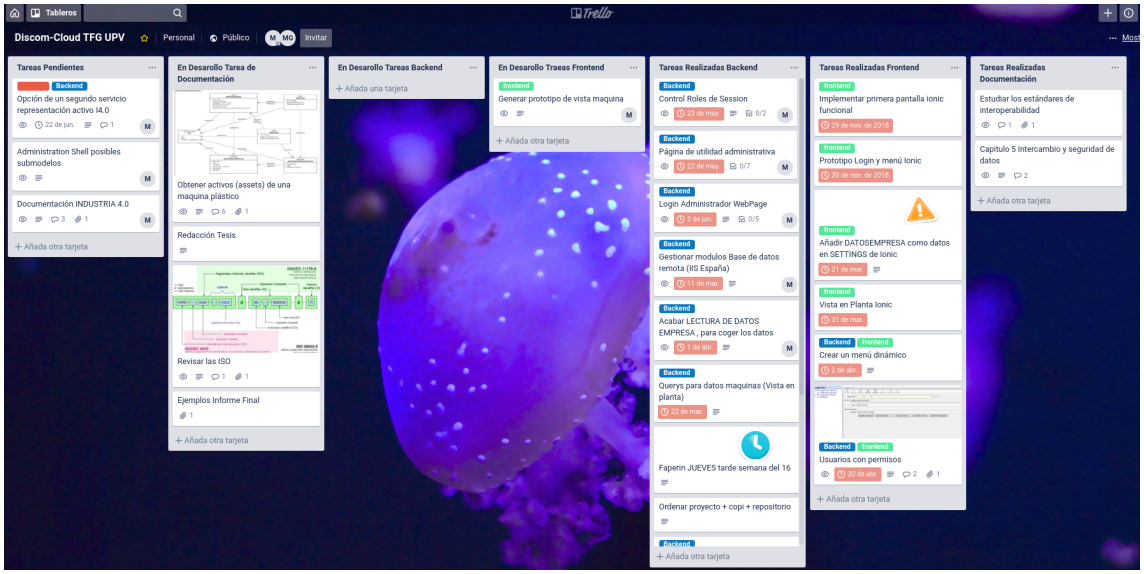


Imagen 1.1: Captura de la web Trello

Para este proyecto, la estructura está divida entre varias categorías, donde cada tarea transita por 3 fases del desarrollo. Estas fases se dividen en:

1. Pendientes: Tareas pendientes de realizar
2. Desarrollo: Tareas que se están realizando actualmente.
3. Realizadas: Tareas terminadas.

Para el Trello se ha dividido el proyecto en 3 categorías, para un mejor control y orden, identificadas en Trello con una etiqueta. Estos categorías son:

- Backend: Debe contener todo lo referente al servidor, servicios web, base de datos etc..
- Frontend: Se incluyen aspectos como aplicación cliente, interfaz, diseño, producto final.
- Documentación: Contiene tareas de documentación, investigación y análisis.

Además, todas las tareas de backend, frontend y de documentación fueron etiquetadas. Se utilizaron colores distintos:

(a) Tarea Backend. **(b)** Tarea Frontend. **(c)** Tarea Documentación.

Figura 1.2: Tipos de tarea

Las tareas son etiquetadas y se pueden agregar aclaraciones o comentarios sobre el estado o problemas de la tareas. Los cambios de estado de cada tareas, junto con cualquier comentario, son registrados en el detalle de cada una de las tareas. Se incluyen, además fechas de vencimiento aproximadas en tareas que son primordiales que se lleven a cabo en el proyecto. Al finalizar el seguimiento de las tareas se tiene una estimación más exacta del tiempo utilizado para el desarrollo del backend, frontend y documentación.

CAPÍTULO 2
Estado del Arte

Existe una gran demanda por el control de flujo de datos y automatización en el proceso de producción. Por ende, es necesario que este flujo permita un amplio acceso. Cabe destacar que existen muchas herramientas software para el control o visualización de plantas o máquinas y todas estas tienen gráficos e interfaces predeterminadas, orientadas a grandes empresas. Algunos ejemplos de empresas que ofrecen software que se rigen, en parte, por estándares I4.0 son Tibco[1] o Solmicro[2].

Asimismo, este control y representación es esencial para el crecimiento de la Industria 4.0. Esta es una realidad que hace referencia a un nuevo modelo de control y organización que muchas empresas y fábricas han implementado. Un modelo que está siendo tendencia a nivel mundial y revistas especializadas como Forbes constantemente publican artículos con titulares como "Beyond Industry 4.0: Getting To Tomorrow's Manufacturing Solutions Today"[3]. También artículos recientes en prensa especializada nacional: "La realidad aumentada impone su eficacia en la industria 4.0"[4], "El Gobierno advierte que el salto a Industria 4.0 no es una "opción" sino "necesidad" "[5]. Entre las empresas más comprometidas a nivel internacional como Siemens, con su programa software TIA Portal, portal de automatización totalmente integrado. Otro ejemplo sería la empresa Kuka, cuyo objetivo está basado en la implicación de los robots como elemento clave de la Industria 4.0.

En la actualidad, el paradigma Inudstria 4.0 se rige bajo organismos internacionales de normalización IEC e ISO y sus miembros nacionales como UNE. Estos organismos son necesarios para garantizar la interoperabilidad que la Industria 4.0 requiere, comprendiendo la aplicación de uso de tecnologías habilitadoras como: realidad virtual, big data, inteligencia artificial, IoT, sistema de objetos distribuidos etc, de una manera estandarizada y consensuada entre distintos modelos y activos.

En el futuro, la fabricación está orientada a la disponibilidad de la información necesaria en tiempo real mediante la conexión de todos los elementos o activos que participan. Estos necesitan un nivel sin precedentes de interoperabilidad e integración de la información de todos los dominios, y si se desea lograr esta meta, el flujo de información debe

[1] consultado el 07/06/2019 en https://www.tibco.com/es/node/10466

[2] consultado el 07/06/2019 en https://www.solmicro.com/noticias-erp/noticias-erp/

[3] consultado el 10/06/2019 en https://www.forbes.com/sites/jimvinoski/2018/11/25/beyond-industry-4-0-getting-to-tomorrows-manufacturing-solutions-today/#4ecb723d122b

[4] consultado el 11/06/2019 https://www.eleconomista.es/tecnologia/noticias/9533029/11/18/La-realidad-aumentada-impone-su-eficacia-en-la-industria-40.html

[5] consultado el 11/06/2019 https://www.elconfidencial.com/ultima-hora-en-vivo/2018-09-26/gobierno-advierte-que-salto-a-industria-4-0-no-es-opcion-sino-necesidad_1628157/

ser continuo, uniforme y hacerse a través de interfaces normalizadas.

A modo de conclusión, la potencia y dificultad de adaptar un modelo Industria 4.0, reside en el consenso y uso de los estándares que se van definiendo para todos los modelos y ámbitos que pueden cubrir dichos modelos. Para lograr un consenso global, donde se facilite la interoperabilidad de componentes indiferentemente del tipo o lugar donde reside.

2.1 Crítica al estado del arte

Al examinar los diversos trabajos y herramientas software relacionadas con los principales objetivos del proyecto, son pocos los que ofrecen una solución software con un soporte multiplataforma y ninguno de ellos, permite una interfaz dinámica, que ofrezca una visualización e interacción con gráficos y estadísticas, creados por parte del cliente o empresa. Este número de similitudes disminuye si solo de limita al sector plástico. Además, el abanico de posibilidades que ofrece el software front-end de este proyecto se moldea a cada usuario dependiendo de los permisos asignados.

Por otra parte, no existe nada relacionado con la parte o fragmento de Industria 4.0 que cubre este proyecto. Como por ejemplo, el análisis y implementación de la parte representativa y funcional de activos integrados en maquinaria para industria plástica, sin por ello, afectar al desarrollo de la herramienta software propuesta. Por tanto, se consigue dar unos primeros paso hacia un nuevo paradigma, aportando las primeras pautas a la inclusión hacia la automatización y auto-gestión de sistemas de producción.

2.2 Propuesta

Este proyecto, aporta una mejora a nivel de disponibilidad y personalización para el apartado de visualización y interacción de la información con respecto a trabajos o herramientas relacionadas. Permitiendo que cada usuario o cliente disponga de una una versión diferente, debido a los permisos asignados y "dashboard" o listados creados. Junto al apartado anterior, se propone realizar una identificación y representación de activos, bajo los estándares de Industria 4.0. Pudiendo brindar una pequeña aportación a la forma en que los sistemas industriales deben evolucionar, sin afectar al desarrollo principal del proyecto, debido a la separación de servicios. Como consecuencia se fomenta el uso de una semántica común, seguimiento de patrones y la interoperabilidad. Toda la información recopilada y investigación realizada ha tenido como resultado una documentación extensa por parte del alumno. Todos los elementos referentes a Industria 4.0 se encuentra explicado en apéndice A

Por último, destacar la escalabilidad del proyecto, debido a la utilización de diferentes servicios independientes y su fácil despliegue, donde además, se pueden sumar nuevos servicios, ampliando la funcionalidad de la herramienta, sin tener que modificar el código ya existente.

2.3 Requerimientos

En la propuesta del trabajo se ha tenido en cuenta la demanda de los clientes del sector y todos los requerimientos fruto de la incursión a Industria 4.0.

2.3.1. Requisito funcionales

Requisito	Descripción	Dependencias
RF 1.0	Permitir ofrecer tablero de creados de forma remota por el cliente. Su visualización e interacción debe estar disponible en cualquier plataforma.	
RF 1.1	Ofrecer una Página web para funciones administrativas orientadas a la resolución de errores. La página del administrador.	RF 1.0
RF 1.2	Login de acceso a la página web del Administrador	RF 1.0
RF 1.3	Página principal del Administrador, con la mínima funcionalidad para detectar fallo de configuración, insuficiencia de datos o comprobación de permisos de usuario. Así como pruebas para determinar posibles errores en los tableros generados por los clientes.	
RF 2.0	Ofrecer información referente a la empresa cuyos datos están almacenados en la base de datos. Esta información debe poder obtenerse independientemente de la plataforma o lenguaje.	RF 1.0
RF 2.1	Permitir realizar una comprobación de usuarios válidos y obtención de permisos de forma remota.	RF2.0
RF 2.2	Dar el soporte para obtener y añadir CRMS	RF 2.0
RF 4.0	La descripción de propiedades de los activos debe ser compatible con IEC CDD.	
RF 4.1	Identificación de los activos, utilizando los criterios de la documentación I4.0	
RF 4.2	Serialización en formato JSON o XML de activos identificados y descritos.	RF 4.0,RF 4.1
RF 4.3	Descripción y representación de activo siguiendo una semántica común.	RF 4.0, RF 4.
RF 3.0	Aplicación cliente capaz de ser utilizado en cualquier plataforma y poder obtener información remota.	RF 4.0
RF 3.1	Ventana de configuración, donde especificar la dirección o direcciones para obtener un o más servicios.	RF 3.0

Tabla 2.1: Requisitos funcionales

2.3.2. Requisitos no funcionales

Requisito	Descripción	Dependencias
RNF 1.1	Disponibilidad de la app cliente, debe ser multiplataforma	
RNF 1.2	La estructura de directorios, tendrá que seguir las buenas prácticas documentadas en la web oficial Web API ASP.NET	
RNF 1.3	Tiempo de respuesta aceptables para un uso móvil y visualizado de máquinas	
RNF 1.4	Facilitar y priorizar la visualización de información.	
RNF 1.5	Permitir escalabilidad futura.	
RNF 1.6	Uso del lenguaje de programación para la parte del backend, que ya esté implementado en el servidor.	

Tabla 2.2: Requisitos no funcionales

CAPÍTULO 3
Análisis del problema

En el siguiente dibujo o esquema se muestra, un ejemplo del sistema de funcionamiento que se ha venido implementado a lo largo de los últimos años. Concretamente, es un dibujo generalizado que representa la obtención de datos de un activo y como luego son sintetizados y mostrados por pantalla.

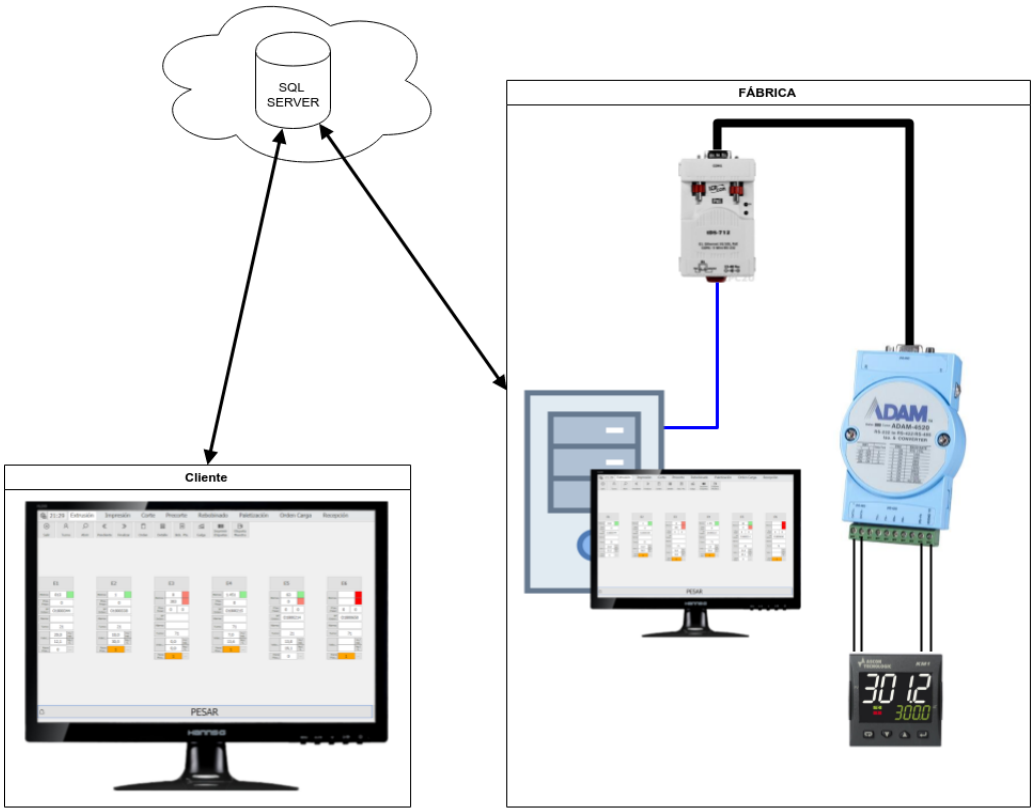


Figura 3.1: Representación sistema actual

En la figura 3.1, un ADAM-4520[1] envía los datos recopilados por KM1[2] y los envía, a través de, un puerto serie RS-485 conectado a un PC. Este mediante un servicio en marcha, procesa la información, extrayendo e insertándolo los datos en un SQL Server situado en un servidor. Una vez almacenados, se visualizan haciendo uso de un programa

[1] https://www.advantech.com/products/gf-5u7m/adam-4520/mod_8dcee4b7-fbde-4c5d-9752-ed2f2fbd00bf

[2] https://www.ascontecnologic.com/es/automatizacion-industrial/controladores-industriales/serie-kube/km1

software en el PC del cliente. Este programa suele conectarse directo al SQL Server para realizar las consultas apropiadas para las vistas (views)[3].

El sistema comentado, alberga una serie de inconvenientes, si se desea mejorar la disponibilidad de información o la facilidad de asimilar y entender los datos generados. A continuación, se explican algunos de estos problemas.

- La información en este ejemplo, está muy centralizada y el procesamiento de datos, recae en un solo equipo, debido a disponer de una única conexión para obtener el flujo de datos generados por el sensor.
- El mismo equipo, es el encargado de sintetizar la información, para realizar las vista en el SQL Server.
- El programa software cliente, está limitado a las vistas generadas y ir realizando conexiones constantes a la base de datos.
- Uso de interfaces poco intuitivas para todo tipo de usuario, se debe tener una formación previa.
- El acceso a la información está restringida al PC.
- No hay manera de deducir la causa o gestionar algún posible problema que pueda surgir.
- No se están obteniendo resultados en tiempo real.
- No se siguen el uso de una nomenclatura y representación del activo y datos de manera formal y estandarizada, por tanto, no puede hacer uso de protocolos como OPC UA mediante su meta-modelo.
- En ningún momento los activos y sus propiedad están representados mediante submodelos ni identificados, por tanto, desde fuera no hay manera de interpretar el sistema montado, sino es por el conocimiento humano.

3.1 Identificación y análisis de soluciones posibles

Se planteó una solución inicial para cumplir los requisitos de funcionalidad, utilizar Xamarin como framework de C# para el desarrollo del software cliente y aportar la capacidad adaptarse a varias plataformas con un acceso a datos basado en consultas directas a la base de datos del cliente. La ventaja es la familiaridad con C# pero en contra, dependemos de un framework de pago y más pesado que la segunda alternativa. Esta alternativa es utilizar el sdk de código abierto Ionic 3 como programa cliente utilizando todas las ventajas que Angular y TypeScript nos proporcionan. Angular y en consecuencia Ionic, es más ligero, alberga mucha documentación y ejemplos, y puede utilizarse código en javaScript, por consiguiente, se puede aprovechar para enriquecer de nuevas funciones la capa de presentación. Por último, como alternativas para la representación estandarizada y uso de OPC-UA, en consecuencia, ofrecer los cimientos para futuros desarrollos encaminados hacia la elaboración de un sistema o componente I.40, son planteadas dos alternativas.

[3] más información https://docs.microsoft.com/es-es/sql/relational-databases/views/views?view=sql-server-2017

Por un lado, el uso de Phyton OPC-UA, como extensión de phyton para la implementación de OPC-UA y por el otro, utilizar la extensión de Nodejs node-opcua para el mismo objetivo. De ambas se ha decidido utilizar la extensión Nodejs, debido a la gran semejanza y compatibilidad con JavaScritp y por tanto, con Ionic. De este modo, en avances futuros se puede utilizar esta misma extensión para elaborar un cliente OPC-UA en nuestro software cliente Ionic.

Por último, cabe señalar que el estándar OPC UA ofrece un modelo de datos o de información específicos y alojar meta-información. Por tanto también permite cumplir con el requerimiento de la representación y uso de una semántica común.

3.2 Solución propuesta

Para poder eliminar los anteriores inconvenientes y mejorar la escalabilidad del sistema, se realiza un nuevo servicio independiente que ayude a la interoperabilidad y una nueva herramienta software de administración, además de cambiar la metodología de obtención y lectura de datos.

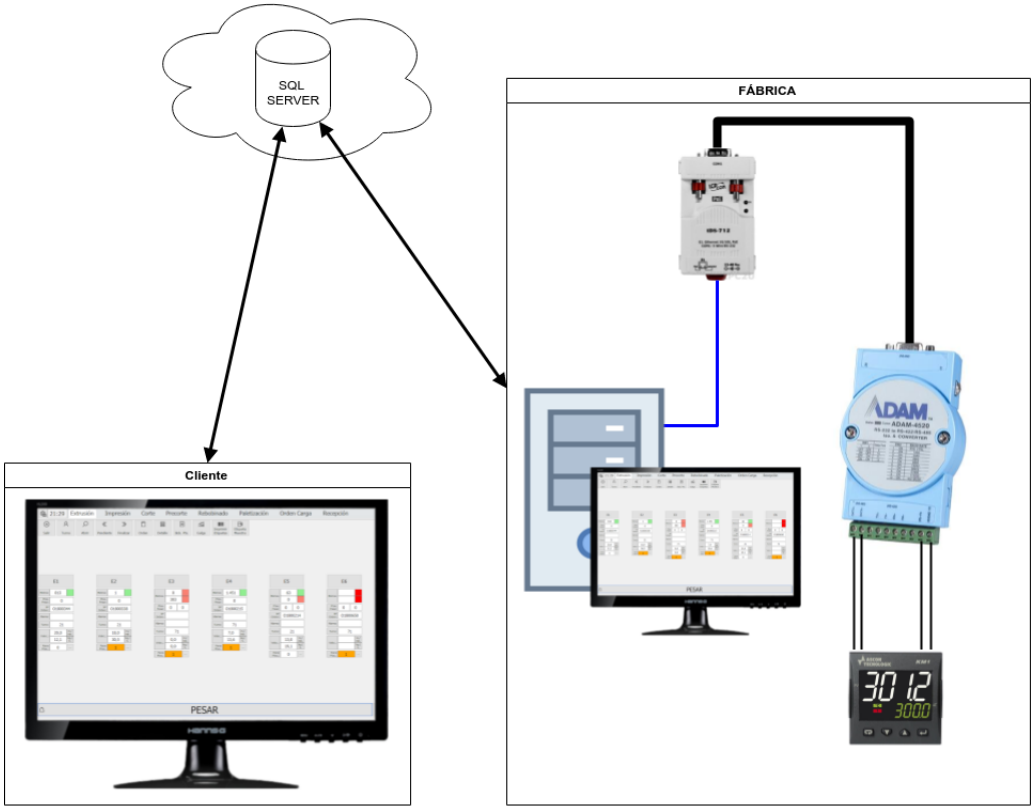


Figura 3.2: Esquema del sistema propuesto

En este concepto, la obtención de datos del sensor y almacenamiento se mantiene idéntico a la figura 3.1. Pero una vez los datos son insertados en la base de datos en el servidor, son tratados y sintetizados mediante un nuevo framework Web API mediante servicios REST. Gracias a REST permitimos a que sistemas externos utilicen la lógica de negocio implementada, fomentando su reutilización y aumentando la exposición de recursos. Además, se incorpora una página web que hace uso de herramientas ofrecidas

por una librería de componentes DevExpress [4], ayudando a crear una representación visual de una manera interactiva y personalizada. Por último, la información ofrecida por la Web API es consumida por varios dispositivos diferentes gracias al uso de una aplicación cliente multiplataforma.

En un sistema I4.0[5] el principal punto de entrada a los datos y servicio es mediante el AAS. Por tanto, es el miembro más importante para una arquitectura de servicio. Por esta razón, para aportar ciertas características del ámbito I4.0 al proyecto original, se desea crear un nuevo servicio web dentro del entorno establecido, donde ya conviven ciertos servicios(Web API y Web Page). Para ello, se empieza realizando varios ejemplos de creación de un AAS, ya que es el miembro más importante para la arquitectura de servicios y principal punto de entrada de datos y servicios en un sistema I4.0. La finalidad de estos ejemplos es ofrecer los primeros pasos para la creación de un sistema I4.0 o la elaboración de un AAS que puede formar parte de uno existente.

Durante el proceso de implementación se ha tenido en cuenta todos los estándares, protocolos y buenas prácticas que en la serie de documentos de la plataforma I4.0 establecen. Actualmente existe una especificación UNE 0061[6] sobre sistemas de gestión para la digitalización y criterio de requisitos para Industria 4.0 pero la documentación de esta norma es de pago.

3.3 Plan de trabajo

Al finalizar el seguimientos de las tareas se tiene una estimación más exacta del tiempo utilizado para el desarrollo del backend, frontend y documentación:

Número de tareas	Etiqueta	Descripción	Tiempo (Horas)
9	Documentación	Tareas derivada de la recopilación de información y documentación.	60
14	Backend	Tareas para llevar a cabo, la parte backend del proyecto.	200
6	Frontend	Tareas aplicación cliente	120
Total			**380**

[4]Lista de componentes en https://www.devexpress.com/#ui
[5]Sistemas de producción auto-gestionables implementado en Industria 4.0
[6]https://www.une.org/encuentra-tu-norma/busca-tu-norma/proyecto?c=P0050866

CAPÍTULO 4
Diseño de la solución

4.1 Estructura

Este proyecto está formado por 3 subproyectos, Web Pages junto a una Web API, aplicación Ionic 3 y el AAS creado. Todo ello para cumplir con los criterios de diseño establecidos.

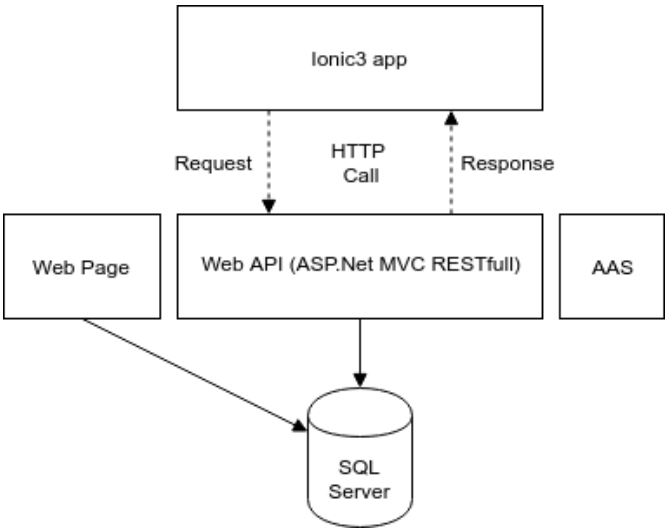


Figura 4.1: Capas con los Servicios implicados

Para proporcionar escalabilidad e independencia en algunos servicio se opta por el desarrollo mediante una estructura SOA utilizando servicios web para conexiones. Actualmente se utilizan los microservicios para el desarrollo de la arquitectura SOA, estos se pueden interconectar entre sí, para lograr un objetivo final. SOA no deja de ser una arquitectura de aplicación en la cual todas sus funciones son servicios independientes y pueden tener un orden de llamada para conseguir un objetivo deseado. Se han establecido una serie de criterios de diseño para afrontar los requisito propuestos.

Criterios	Descripción	Requisito
CRD 1.0	Herramienta para la creación y visualizado de tableros de datos es DashBoard componente de DevExpress. Accesible para todas las plataformas y almacenadas en XML.	RF1.0
CRD 1.1	Todo el contenido se debe hostear en un servidor bajo el servicio IIS.	RF1.0, RF 1.1, RF1.2, RF1.3
CRD1.2	La web de administración debe estar creado bajo ASP.NET WebForms. Para ser compatible con la herramienta DashBoard y ofrecer una Dashboard Viewer's.	RF 1.1, RF 1.2
CRD 1.3	Creación de un servicio RESTFul bajo en concepto Web API con la tecnología ASP.NET. Proporcionando la información en diferentes formatos y manejado mediante métodos HTTP.	RF 2.0, RF 2.1
CRD 1.4	Comprobación de usuarios válidos se realizará bajo el soporte de la Web API y los permisos de cada usuario están determinados por la empresa y obtenidos a partir de una tabla en la base de datos.	RF 2.1
CRD 1.5	Consultas con la implicación de la relacion de muchas entidades de distintas tablas realizar vistas SQL	RNF 1.4
CRD 2.0	Cada propiedad de un activo identificada con una ID CDD o identificador único local, cunado no exista en las librerías IEC CDD. Esta estructuración y serialización del activo debe ser aceptado bajo el modelo que el glosario Industria 4.0 considera válido.	RF 4.0, RF4.1, RF 4.2
CRD 2.1	La representación estandarizada y serializada se debe proporcionar, sin variar nada de la información referente a las máquinas. Se obtendrá los json generados bajo otro servicio independiente, con el fin de no modificar el ya impuesto con anterioridad.	RF 4.2
CRD 3.0	La aplicación cliente será desarrollada, bajo la tecnología web, utilizando el framework Ionic y sus componentes.	RF 3.0, RF 3.1
CRD 3.1	La aplicación Ionic 3 al iniciar realizará una prueba de conexión con el servidor. En caso de fallo, se le notificará en la propia aplicación. En el apartado de ajustes contendrá todos los parámetros necesarios, para realizar una conexión con éxito.	RF 3.1,RF 3.2, RF 3.6
CRD 3.2	La página principal contendrá información referente a la empresa cliente y la navegación se llevará acabo con un menú deslizable, con todas las interacciones que el usuario tiene permitido realizar en base a los permisos.	RF 3.3, RF 3.4, RF 3.5

4.2 Diseño de alto nivel

4.2.1. Lado backend Web Page y servicio Web API

Para cumplir con los criterios de diseño: CRD 1.0, CRD 1.1 y CRD 1.2, se ha seguido el modelo de programación ASP.NET Web Forms. Optamos por este modelo ofrecido por ASP.NET debido a su nivel de abstracción con un modelo de programación basado en eventos y controles. En consecuencia, se reduce la cantidad de código que se genera y se aporta mayor simplicidad mediante la programación declarativa. Además, proporciona compatibilidad completa con la herramienta DevExpress y el uso de sus componentes. La figura después de la implementación seria el siguiente:

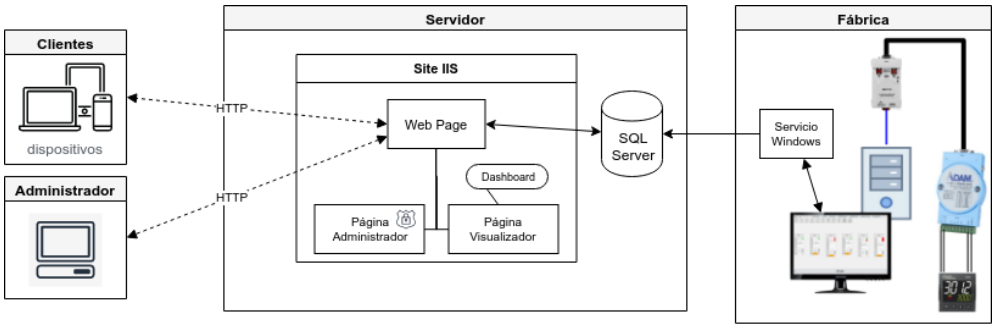


Figura 4.2: Implementación del sistema inicial deseado

El núcleo central está formado por la base de datos y el servicio que la maneja, mientras la página web alojada en el mismo servidor, mantiene una conexión directa con la base de datos. De este modo, se permite crear Dashboards y Listado (DataGridView) a través de una conexión remota permitiendo utilizar los datos contenidos en la base de datos del núcleo central.

A continuación, para cubrir los criterios de diseño CRD 1.3 y CRD 1.4 se genera una Web API ASP.NET, aprovechando la total compatibilidad con el servicio IIS[1] disponible. Por consiguiente, se crea una Web API sobre el framework .NET. Debido a su gran compatibilidad los datos generados ahora pueden ser consumidos por una amplia gama de clientes, incluyendo navegadores, dispositivos móviles y aplicaciones de escritorio tradicional. Ahora la Figura 4.2 anterior evoluciona:

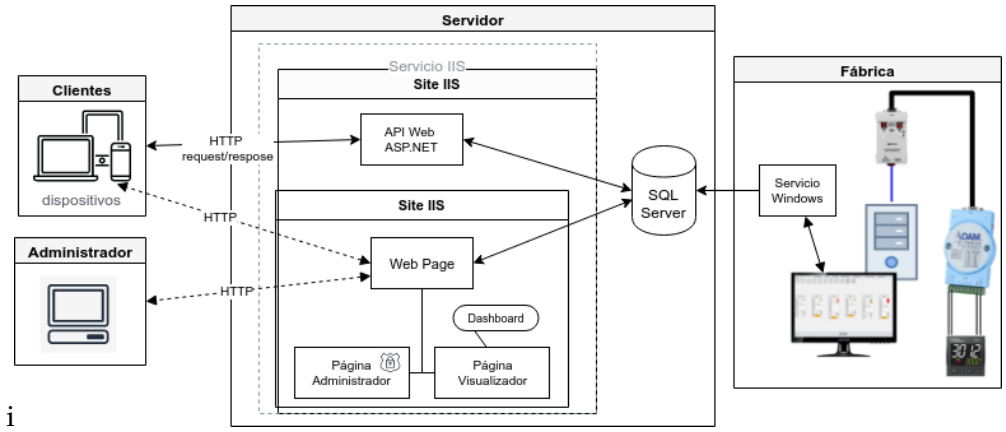


Figura 4.3: Implementación del sistema final deseado 2

[1] *Internet Information Services* https://docs.microsoft.com/en-us/IIS-Administration/

Con la Figura 4.3, se consigue ofrecer gráficas dinámicas e interactivas con DevExpress en todas las plataformas y datos de forma jerárquica y versátil, favoreciendo la creación de aplicaciones donde gran parte de la lógica consiste en emplear servicio y, debido a ello, aporta mayor tolerancia a fallos y escalabilidad al sistema.

En último lugar para cubrir el criterio CRD 1.5 se crean Vistas SQL para agilizar consultas pesadas en la base de datos SQLServer.

4.2.2. Lado cliente aplicación Ionic

Para el lado del cliente, siguiendo los criterios de diseño CRD 3.0, 3.1, y 3.02, se utiliza como lenguaje de desarrollo Ionic 3, para poder crear una aplicación multiplataforma y *responsive*.

Esta aplicación está corriendo bajo Cordova, por tanto, se desarrolla como si fueran páginas web, al utilizar WebViews para las vistas. Ionic utiliza Cordova para poder acceder a herramientas nativas del sistema en el que está corriendo. Angular utiliza la arquitectura Model-View-Controller (MVC), utilizada en el desarrollo de aplicaciones web y por tanto Ionic mantendrá esa misma arquitectura que consiste en:

- Model: La estructura de datos que administra y recibe información del controlador.
- View: Representación de la información
- Controlador: Responde a las e interactúa con el modelo (Model).

La construcción de una aplicación Ionic se crea mediante componentes, es como un bloque básico de construcción. Estos componentes siempre tiene una plantilla y está formado por 3 partes:

1. La estructura del componente o template (HTML).
2. La lógica del componente (TypeScript).
3. Los estilos del componente (CSS).

La información es ofrecida por la Web API remotamente, para ser luego recogida mediante servicios la aplicación Ionic. Un servicio, es un objeto que mantiene el estado y que es igual para todos los componentes de la aplicación. Estos objetos son “inyectados“ en los componentes y son proveídos en cualquier módulo de la aplicación.

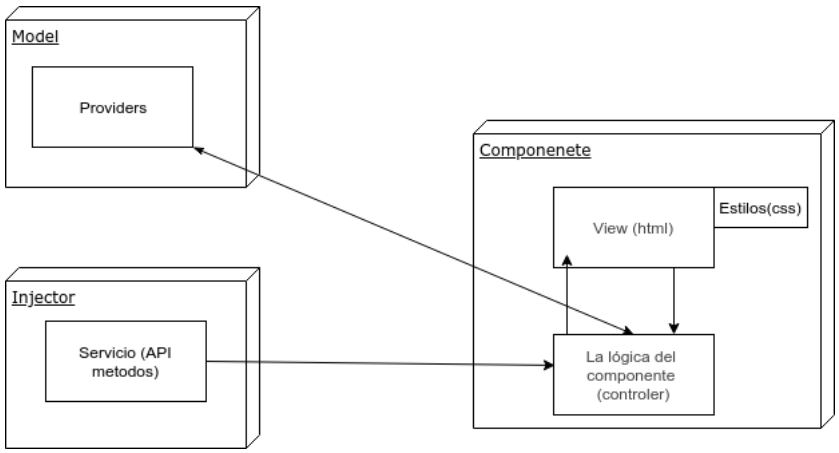


Figura 4.4: Componente y servicio Ionic

4.2.3. Ejemplos AAS

Para abordar los criterios de diseño CRD 2.0 y CRD 2.1 se ha elegido NodeJs como lenguaje y trabajado con las extensiones node-opcua[2] y node-opcua-coreaas[3]. Gracias a ello, se consigue diseñar e implementar un ejemplo de un *Asset Adminitarion Shell* funcional.

4.3 Diseño detallado

4.3.1. Lado Backend Web Page y servicio Web API

Mediante la guía de buenas prácticas de desarrollo documentadas por Microsoft[4], la estructura del subproyecto Web API sigue el patrón repositorio y tiene el diseño de la siguiente figura:

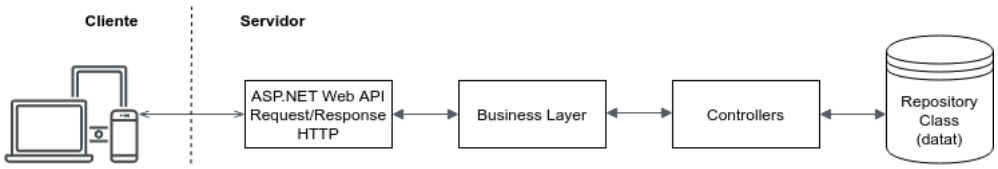


Figura 4.5: Patrón seguido en el servicio en ASP.NET

Web API hace de interconexión para las solicitudes, recibe peticiones de las aplicaciones clientes y un método de acción localizado en un controlador es invocado. Al mismo tiempo, el método de acción inicia la capa de negocio para obtener o actualizar la información, lo que a su vez invocará a la clase Repository para devolver la información de la base de datos. En base a este patrón de diseño, la solución es dividida en subproyectos, separando la capa de negocio de la capa de acceso a datos y de las vistas. La solución con una estructura correcta siguiendo las buenas prácticas es el siguiente:

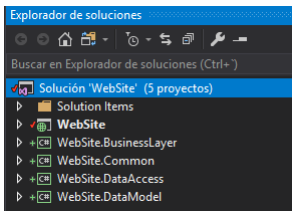


Imagen 4.6: Estructura solución principal

La solución o proyecto WebSite se encuentra dividida en varios subproyectos, representado cada uno de ellos una fase de la arquitectura Web API. Primero, el proyecto WebSite depende de unos subproyectos y representa la capa "*user-interface*"de todo el proyecto Back-End.

[2] https://github.com/node-opcua/node-opcua
[3] https://www.npmjs.com/package/node-opcua-coreaas?activeTab=readme
[4] https://docs.microsoft.com/es-es/azure/architecture/best-practices/api-design

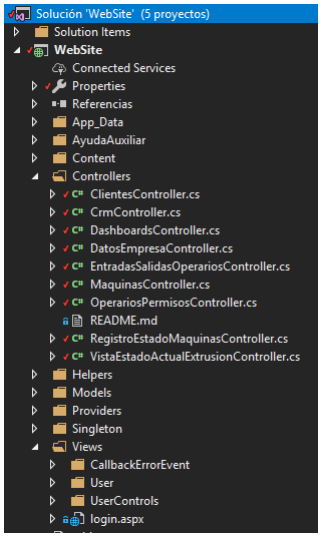


Imagen 4.7: Proyecto Website

Este proyecto tienen una serie de directorios, de los que cabe destacar dos, Controllers y Views. La carpeta Controllers está formada por clases públicas de tipo , que pueden ser accedidas con peticiones HTTP. En cuanto a la carpeta Views, almacena toda la serie de vistas que la Web Page va a manejar. A continuación los subproyectos representan todas las demás capas de la estructura del servicio API.

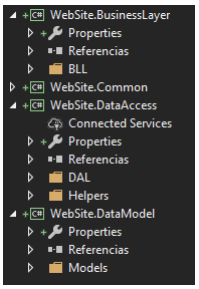


Imagen 4.8: Subproyectos Website

4.3.2. Lado cliente aplicación Ionic

Las aplicaciones Ionic, siguen una organización de directorios semejante a Angular en el proyecto. Los requerimientos mínimos para empezar con el desarrollo son tener instalados Node.js y el gestor de paquetes npm. Node.js para interactuar con el ecosistema Ionic y npm para la administración de dependencias para bibliotecas y marcos de JavaScript instalados en nuestra máquina, al instalar Node.js instala npm por defecto.

Con npm instalado, se debe instalar Ionic framework con el comando "*npm install -g ionic*". Estos nos creará una estructura de directorios donde se profundizará a continuación.

Imagen 4.9: Estructura general de carpetas

En el directorio "*src*" se encuentra el código de Ionic e inicia con "ionic server", este código es convertido a una versión de JavaScript que el navegador pueda entender. Eso significa desarollar en un nivel superior utilizando Componente Dashboard, pero compilar en JavaScript para el navegador.

El punto de entrada principal en la aplicación es "*./src/index.html*", encargado de configurar los scripts, css y arrancar la aplicación. Una vez compilada, su arranque empieza a buscar en el html la etiqueta "*<ion-app>*" comenzando por el directorio "*src/app/*" con el siguiente contenido, ordenado por orden de inicio:

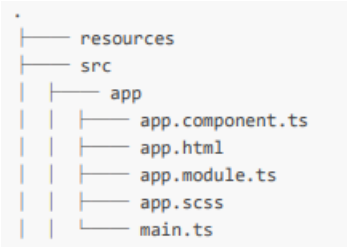


Imagen 4.10: Contenido carpeta src

El fichero "*src/app/app.module.ts*" es el punto de entrada, especificando el módulo raíz de Ionic. El modulo raíz es *src/app/app.component.ts* siendo el primer componente que se carga al arrancar. Por tanto, la carpeta app inicia, carga todos los componentes necesarios y inicia, estableciendo un mecanismo de navegación. A partir de este momento, Ionic carga la página asignada como raíz (root) y se instaura la navegación de la aplicación. En la estructura del proyecto Ionic, es importante puntualizar tres directorios: pages, shareds y providers.

La carpeta Pages alberga componentes, vistas y funciones de las páginas, estas son fundamentales para la navegación de la aplicación e interfaz.

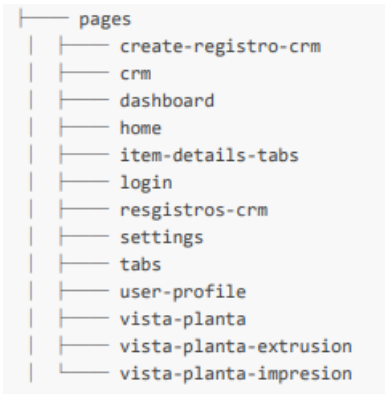


Imagen 4.11: Contenido carpeta pages

La carpeta Shared contiene las clases o componentes que van a ser compartidos en todo el proyecto.

Imagen 4.12: Contenido carpeta shared

La carpeta Providers contiene los proveedores de servicio y de datos. Los providers son los encargados de manipular los datos, accedidos desde una fuente de datos como un servicio REST API, base de datos o almacenamiento local.

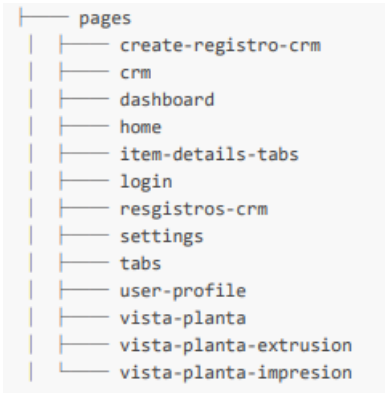


Imagen 4.13: Contenido carpeta providers

4.3.3. Ejemplos AAS

Asset Administration Shell para lograr utilizar el AAS como un servicio añadido en REST, este debe abordar en cierta medida, mínimo 3 capas de RAMI 4.0:

1. Capa de Información: Lograr disponibilidad de datos, para el alcance de SOA, entra en juego el modelo de información, servicio de acceso a datos o método de mapeo.

2. Capa de Comunicación: Proporcionar un mecanismo de transporte de datos lo más transparente posible y bajo protocolos seguro. Aquí entra en acción el estándar abierto OPC UA.

3. Capa funcional: Garantizar una interoperabilidad, en consecuencia, las implementaciones creadas deben cumplir con ciertos criterios específicos.

Como resultado se proporciona un modelo común(uso de semántica común) entre el meta-modelo de una tecnología middleware (OPC UA) y los modelos de dominio/aplicación, Ionic y Web API en nuestro caso. En el directorio raíz del proyecto están ubicado los ejemplos sample1.js y sample2.js y dentro de la carpeta CoreAAS el ejemplo sample3.js. Se estructura de este modo para separar el ejemplo que maneja la extensión "*node-opcua*" y el ejemplo que utiliza implementación "*node-opcua-coreaas*".

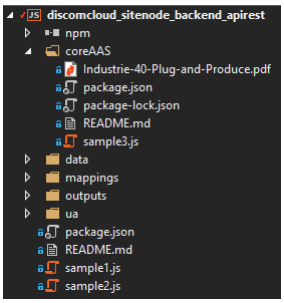


Imagen 4.14: Estructura principal proyecto AAS

Además, como se observa en la Imagen 4.14 existe una carpeta data donde se alberga esquemas AAS y IEC61360 para el tipo de dato o propiedad.

4.3.4. Tecnologías utilizadas

Para el desarrollo del proyecto, se manejan de diversos lenguajes, frameworks, librerías y extensiones, se muestran a continuación 3 tablas haciendo referencia a cada subproyecto.

Nombre	Tipo o Modalidad	Descripción
C#	Lenguaje	Tipado fuerte, orientado a objetos, desarrollado y estandarizado por Microsoft.
ASP.NET	Framework	Crear aplicaciones y servicios web con .Net y C#
DevExpress	Librería	Almacén de componentes de UI para el desarrollo en todas las plataformas de .NET, C# o JavaScript.
ASP.NET Web Forms	Framework	Utilizado para el desarrollo de páginas Web utilizando C# o .NER.

Tabla 4.1: Tecnologías aplicación Web Page y API

Nombre	Tipo o Modalidad	Descripción
TypeScript	Entorno de ejecución	De código abierto, basado en JavaScript, agregando una escritura estática opcional a esta.
Angular	Framework	De código abierto utilizado para crear aplicaciones web, utilizando TypeScript como lenguaje de control.
Ionic	SDK	SDK de código abierto para desarrollar Progressive Web Apps (PWA) y apps nativas con JavaScript, CSS y HTML5
DevExpress	Librería	Almacén de componentes de UI para desarrollar en .NET, C# o JavaScript.

Tabla 4.2: Tecnologías aplicación cliente

Nombre	Tipo o Modalidad	Descripción
TypeScript	Entorno de ejecución	De código abierto, basado en JavaScript, agregando una escritura estática opcional a esta.
Angular	Framework	De código abierto utilizado para crear aplicaciones web, utilizando TypeScript como lenguaje de control.
Ionic	SDK	SDK de código abierto para desarrollar Progressive Web Apps (PWA) y apps nativas con JavaScript, CSS y HTML5
DevExpress	Librería	Almacén de componentes de UI para desarrollar en .NET, C# o JavaScript.

Tabla 4.3: Tecnologías ejemplos Asset Administration Shell

CAPÍTULO 5
Desarrollo de la solución propuesta

5.0.1. Web Page

El desarrollo backend, empieza con la diseño y creación de una Web Page, haciendo de punto de contacto principal entre los Dashboards, permisos de usuarios y permitir realizar pruebas de conexión. Para empezar se debe tener instalados los paquetes DevExpress y Microsoft ASP.NET Forms y utilizar Visual Studio 2017[1] junto al SDK .NET Core para l desarrollar. Los requerimientos para el funcionamiento mínimo de la Web Page localmente, es tener creado el fichero de configuración DatosLocales.xml. Dicho fichero se encuentra en Apéndice B.1

La Web Page cuando inicia realiza una conexión a la base de datos con el propósito de determinar si los datos de configuración asignados son correctos. Una vez la conexión con la base datos SQL Server ha sido exitosa, se procede al login y al funcionamiento normal de la web. A continuación un ejemplo del lanzamiento y las clases y funciones implicadas:

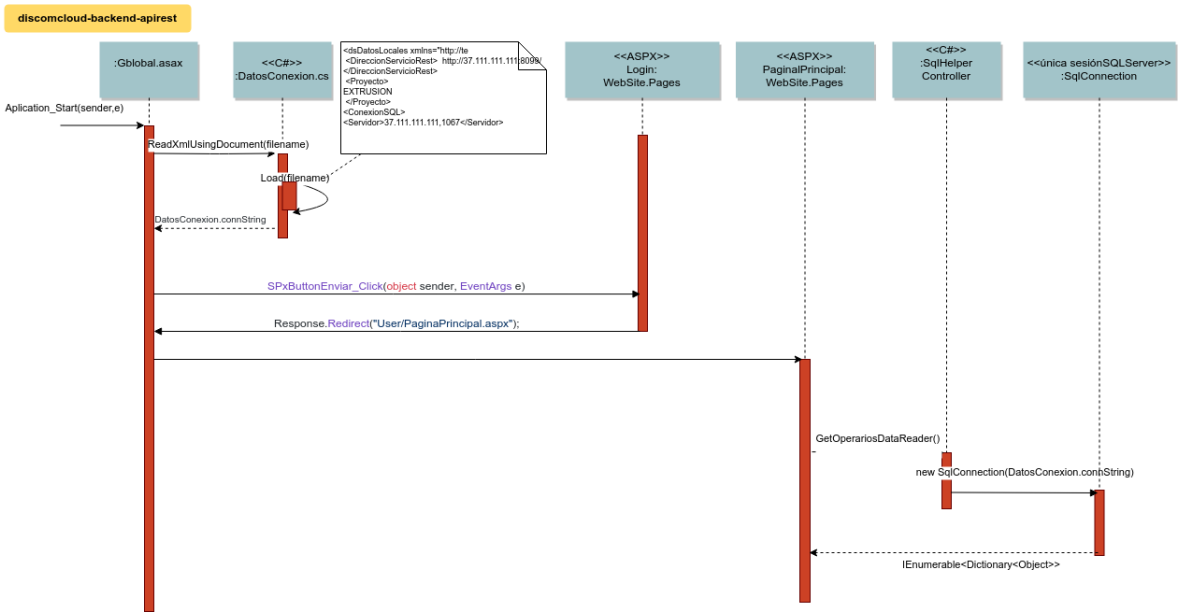


Figura 5.1: Contenido carpeta shared

[1] https://visualstudio.microsoft.com/es/

5.0.2. Servicio Web API

Ahora se procede con los criterios de diseño CRD 1.3 y CRD 1.4. Donde al igual que la Web Page, el servicio Web API está ubicado en el servidor IIS. Este servicio es un elemento central del sistema a realizar, esta vez hace de punto de contacto entre las interfaces del usuario (móvil o web) y el SQL Server del servicio IIS. Se sigue la arquitectura Modelo Vista Controlado (MVC), donde cada modelo corresponde a una tabla de la base de datos, por tanto, se representan las tablas que se van a utilizar en objetos en C#, estas clases serán manejadas por los controladores mediante acciones como "Get()":

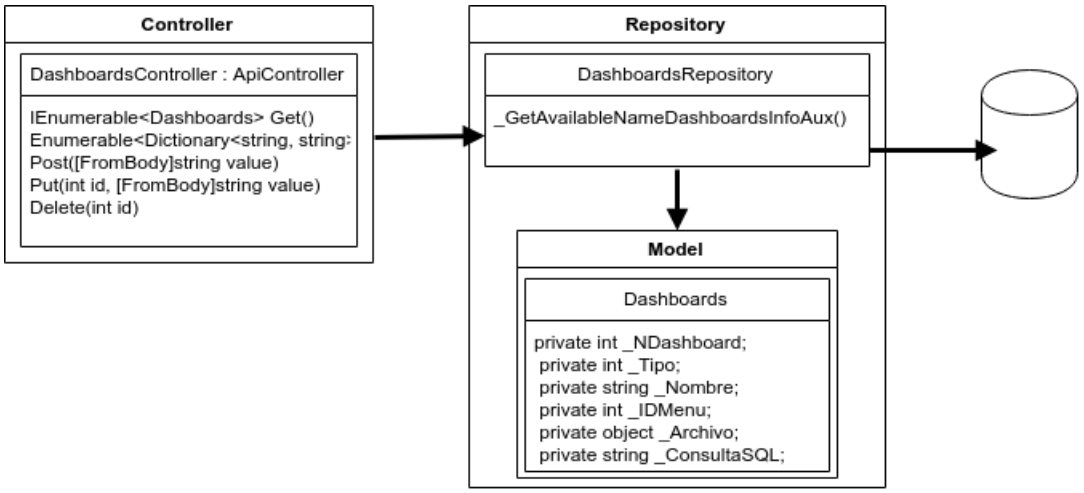


Figura 5.2: Obtención de datos por el controlador

Se utiliza JSON como medio para la comunicación de datos entre la API y las interfaces, aportando simplicidad y favoreciendo una mayor integración con otros sistemas.

Por último, el enrutamiento APS.NET realiza el proceso de recibir una solicitud HTTP y asignarla a una acción del controlador (controller), este es un objeto que gestiona solicitudes HTTP. Código de una clase controlador en apéndice A. La plantilla de enrutamiento predeterminado de Web API es "api/controller/id" y su configuración se define en el archivo WebApiConfig.cs:

```
1    RouteTable.Routes.MapHttpRoute(
2        name: DefaultApi,
3        routeTemplate: api/{controller}/{id},
4        defaults: new { id = RouteParameter.Optional }
5    );
```

Código 5.1: Objeto de gestión de solicitudes Routing

Tanto la Web Page ASP.NET Forms como el servicio Web API están desarrolladas en la misma solución en Visual Studio. De todas sus funcionalidades y requerimientos, se detalla el proceso de obtención del objeto "PermisosAplicación" para un determinado usuario. Los datos de este objeto son utilizados por la aplicación cliente Ionic 3, para generar el menú personalizado y mostrar información permitida, como "*Dashboards*".

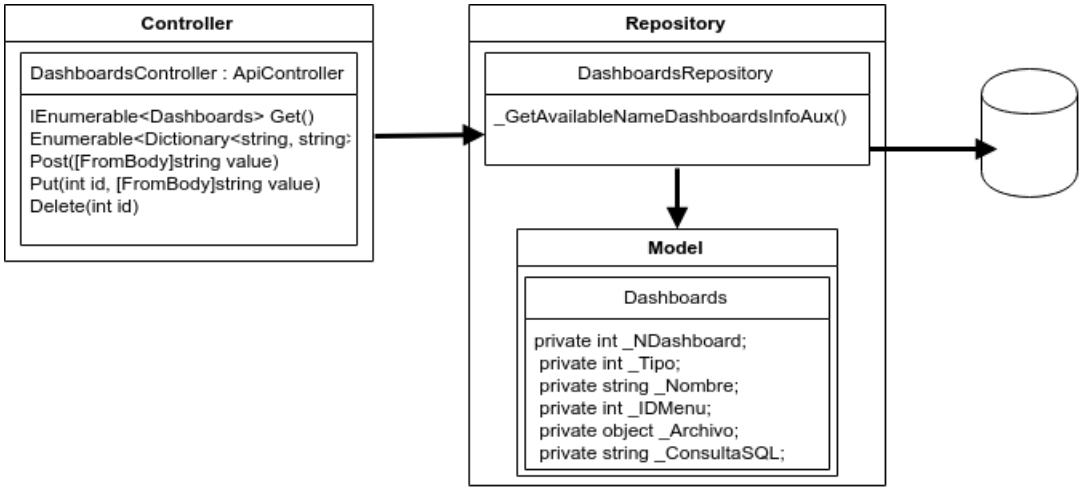


Figura 5.3: Permisos aplicación para un determinado usuario

En ultimo lugar, algunas consultas no son realizadas sobre tablas sino sobre vistas creadas en la base de datos. Este es el caso de las operación de obtener:

1. Estado actual de extrusión de máquinas.

2. El cortado de extrusión.

Estas vista se encuentran especificadas por la Imagen B.2 y Imagen B.1 en apéndice B

5.0.3. Aplicación cliente Ionic

Para el desarrollo de los criterios de diseño CRD 3.0, CRD 3.1 y CRD 3.2, se obtienen los datos a través del servicio Web API, con ayuda de servicios y objetos de tipo *Objeto Observable*. A continuación se ejemplifica la obtención datos del servicio Web API y carga de la vista:

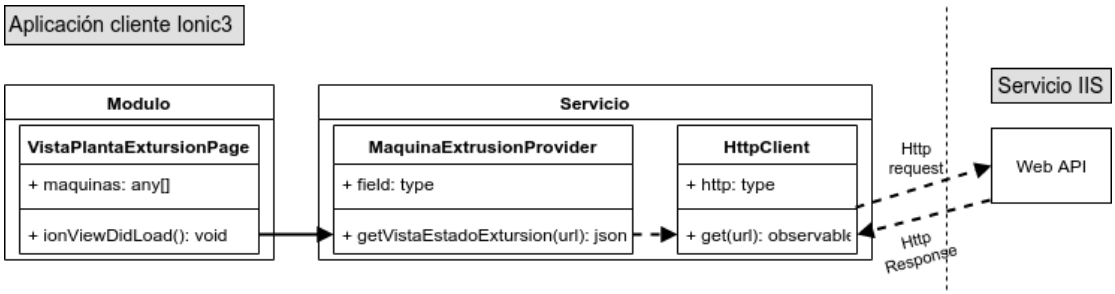


Figura 5.4: Obtener datos Web API desde Ionic3

Por último, se describe mediante un diagrama de flujo, el login y generación del menú:

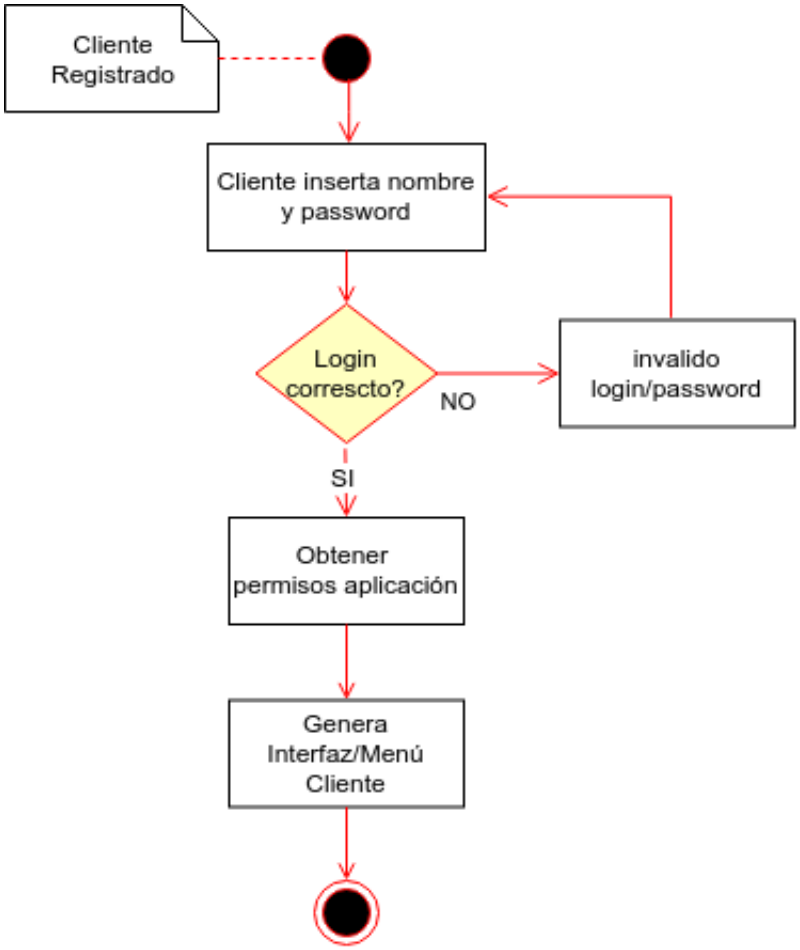


Figura 5.5: Diagrama de flujo login y generación del menú

Destacar también el proceso de obtención de los *"Dashboards"* que el usuario tiene disponible o permitidos y el proceso de visualizar estos componentes:

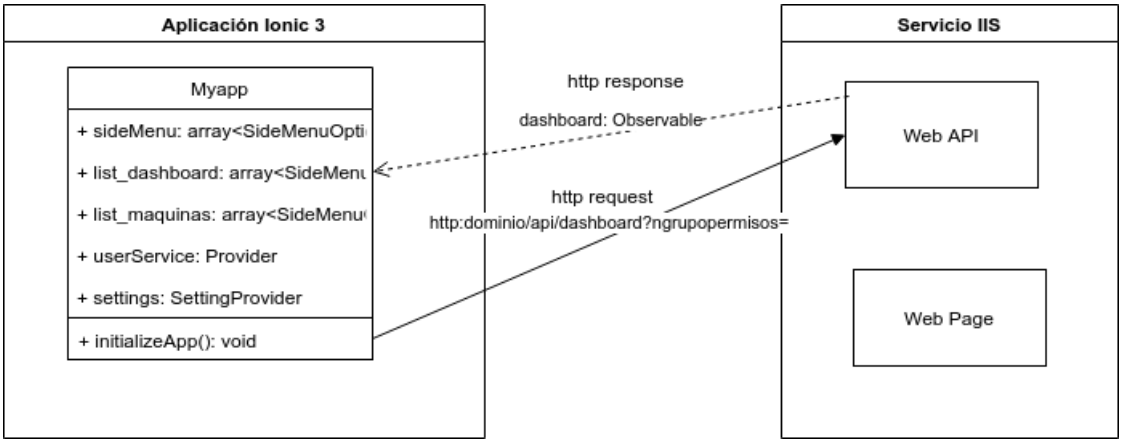


Figura 5.6: Obtención dashboards

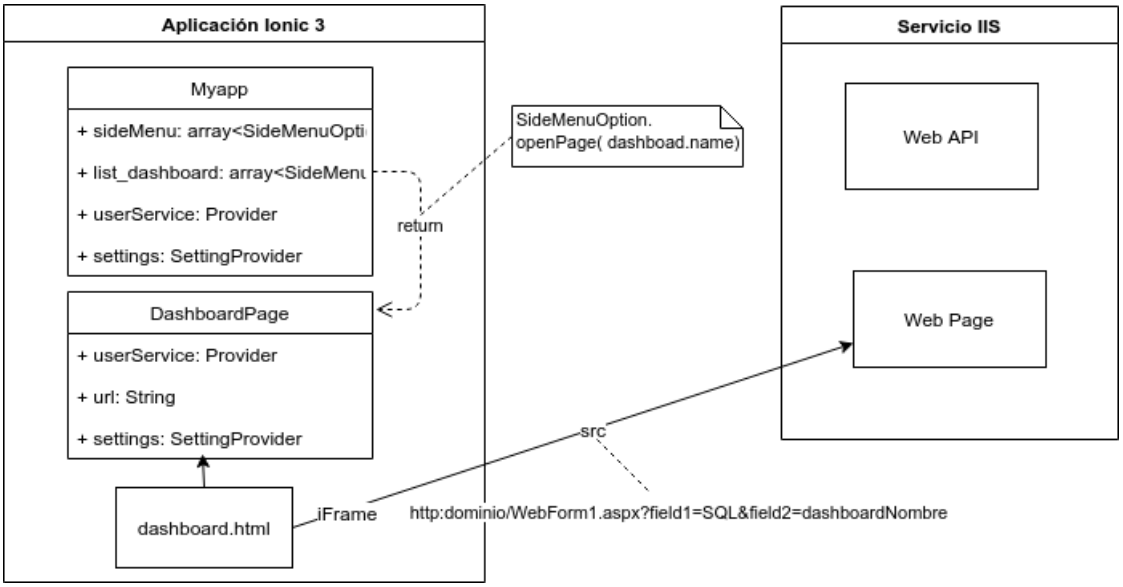


Figura 5.7: Visualizado dashboards

El visualizado de un "*Dashboard*" como en la Figura 5.6 se debe realizar mediante la carga de la página web ofrecida por el servicio Web Page, debido a que necesitan ser cargados bajo una aplicación bajo C# o ASP.NET. Son las únicas plataforma donde la librería *DevExpress* permite la carga de este componente.

5.0.4. AAS Ejemplos

Primer ejemplo

Se aborda la capa de información por medio de la definición de objetos con una sintaxis correcta que contengan la información deseada y mapearlos correctamente. Para validar sintácticamente, la plataforma I4.0 dispone de dos esquemas:

- AAS.xsd: Para la estructura y definición del objeto AAS.
- IEC61360.xsd: Para la definición del tipo de dato o propiedad.

Se alcanzan dos puntos:

1. Conseguir parsear un AAS en formato XML en un objeto json serializable (*unmarshalling*).
2. Lograr mapear de un objeto AAS a XML en base al XML Schema AAS.xsd.

Se utiliza la librería Jsonix[2] para ofrecer funcionalidades de parseo [3]y señalización. Para realizar las conversiones correctamente es necesario antes mapear[4] el esquema AAS.xsd:

```
1 //java -jar jsonix-schema-compiler-full.jar -logLevel TRACE -d mappings -p AAS AAS.xsd
2 var AAS = require('./mappings/AAS').AAS;
```

Código 5.2: Generar objeto a partir de AAS.xsd

[2] https://www.npmjs.com/package/@boundlessgeo/jsonix
[3] Análisis del texto y determinar si cumple o no reglas o patrones.
[4] Generar un modelo de clases a partir de un fichero XSD

A continuación dos fragmentos de código que representan el proceso de unmarshalling y marshalling:

```
var unmarshaller2 = context2.createUnmarshaller();
unmarshaller2.unmarshalFile('data/XML_Example_for_Administration_Shell.xml',
    function (unmarshalled) {
        object = unmarshalled;
        fs.writeFile(appDir + '/outputs/objectNodejs.json', JSON.stringify(unmarshalled
            , null, 2),
            (err) => { if (err) console.error(err) });
        if (unmarshalled.value.assets) {
            if (unmarshalled.value.assets.asset.length > 0) {
                asset = unmarshalled.value.assets.asset[0];
                property = unmarshalled.value.submodels.submodel[0].submodelElements.
                    submodelElement[0].property;
                property2 = unmarshalled.value.submodels.submodel[0].submodelElements.
                    submodelElement[1].property;
            }
        }
    });
```

Código 5.3: Convertir un XML a un objeto

```
// Marshaller para serializar un objeto en NodeJs a XML
    var marshaller = context2.createMarshaller();
    marshaller.marshalString(assetObject);
```

Código 5.4: Generar fichero XML a partir de un objeto

Por último, se aporta accesibilidad a la información con un servicio REST.

```
var express = require('express')
var server = app.listen(process.env.PORT || 8081, function () {
    var port = server.address().port;
    console.log(App now running on port, port);
    console.log(http://localhost: + port + /api/sample1);
});
app.get(/api/sample1/, function (req, res, next) {
    res.contentType('application/xml');
    res.send(marshaller.marshalString(assetObject).toString());
    next();
});
```

Código 5.5: Crear y iniciar servicio REST

Segundo ejemplo

El segundo ejemplo se implementa parte de la capa de información para la disponibilidad de datos y ofrece un protocolo de comunicación por medio de un servidor OPC UA junto a un cliente OPC UA con ayuda de la extensión "*node-opcua*". Más acerca de OPC UA en apéndice K.

OPC UA es el protocolo bajo el estándar IEC 62541[5], que ahora forma parte del estándar global de interoperabilidad para Industria 4.0. En la creación del servidor OPCUAServer en NodeJS se utiliza la extensión NodeOPCUA y para el cliente OPCUA se emplea UaEx-

[5]OPC Unified Architecture: Device Interface ttps://webstore.iec.ch/publication/21987

pert[6] que dispone de una interfaz gráfica.

Este segundo ejemplo, se crea un objeto que representa un controlador de temperatura KM1[7], su variable y una pequeña simulación de su actividad. Este objeto es representado de forma estándar gracias al espacio de direcciones (*adressSpace*). Como se ha visto en el documento, KM1 es utilizado para la medición de temperaturas de las máquinas del sector plástico. A continuación un ejemplo del diseño:

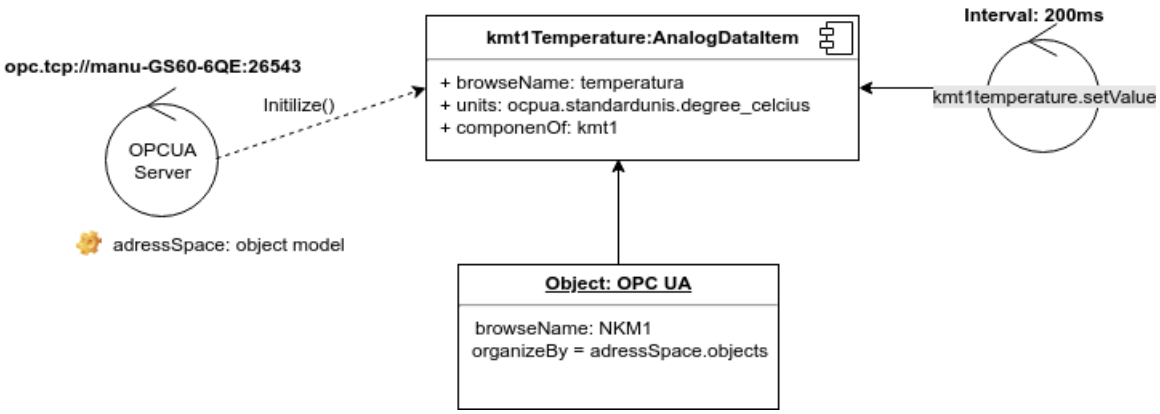


Figura 5.8: Ejemplo de inicialización OPCUA Server

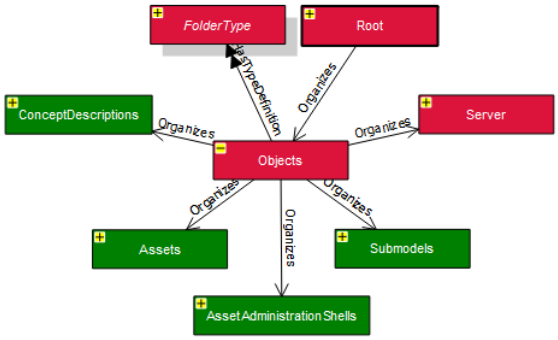
Los tipos de datos utilizados son los especificados por *opc foundation*[8] en el fichero *OpcUaTypes.bsd*, incluido ya en la extensión. El código referente a este ejemplo se localiza en apéndice C.1.

Tercer ejemplo

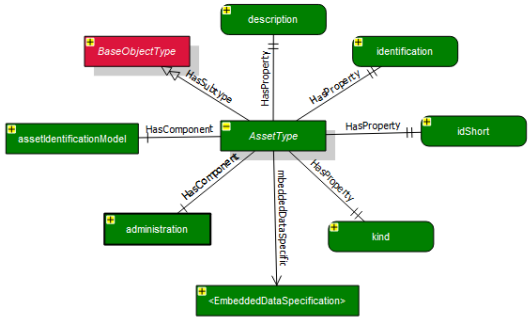
En este último ejemplo, son cubiertas las 3 capas mínimas requeridas (información, comunicación y funcional) para crear un AAS funcional. Para llevarlo acabo se aprovecha la extensión *node-opcua-coreaas*, una implementación de *node-opcua* de un meta-modelo AAS proporcionado por la plataforma de Industria 4.0 y basándose en referencias del documento "*Details of the Asset Administration Shell.pdf*"[5]. Exactamente el mismo meta-modelo expuesto en este trabajo en la figura A.6.

Implementa CoreAAS, de ahí la nueva extensión *coreaas* a al nombre *node-opcua*. Esta implementación del meta-modelo AAS en OPC UA define un modelo de información abierto que describe las partes principales del meta-modelo AAS. Este puede visualizase utilizando un software gráfico como UAModeler de Unified Automation[9]. A continuación dos figuras que representan objetos y tipos fundamentales para el meta-modelo AAS.

[6] https://www.unified-automation.com/products/development-tools/uaexpert.html
[7] https://www.ascontecnologic.com/es/automatizacion-industrial/controladores-industriales/serie-kube/km1
[8] https://opcfoundation.org/
[9] https://www.unified-automation.com/products/development-tools/uamodeler.html



(a) Objeto CoreAAS.



(b) Objeto AssetType

Figura 5.9: Representación gráfica del modelo AAS con UaModeler

En concreto, se realiza un AAS centralizado con un activo único KM1. Este activo realmente es un activo compuesto formado por el controlador KM1 con un sensor de temperatura compatible KTY81.

Se debe tener en cuenta que un AAS representa exactamente un activo, con un identificador único. En este ejemplo hay un controlador y un de sensor conectado, existe la opción de asignar por separado un AAS para el controlador y AAS para el sensor. Por simplificar, se crea un AAS con identificador único para representar el activo compuesto, solución igualmente válida.

El AAS se crea con el activo tiene junto a dos propiedades (especificadas en *Submodel-Property*), la temperatura (idShort="*TEMPERATURE*""), un valor de medición (categoria=Variable) y temperatura máxima permitida (idShort="*TEMPMAX*") con un parámetro fijo. El activo es identificado con id "KM1". El AAS debe favores la interpretación, por consiguiente, se añade en su diccionario (clase *ConceptDictonary*) nuevas definiciones semánticas sobre los elementos contenidos en submodelos. En este caso las definiciones, son extraídas de IEC 6130 y referenciadas por la clase "*ConcepDescription*". Se crean dos objetos de esta clase, una con idShort "@T" con IRDI[10] *0112/2////61360_4#AA-E685#001* con el nombre predefinido "*temperature*" y otra con un idShort "Tstress(max)" con IRDI*0112/2////61360_4#AAF277#002* junto al nombre predefinido "*stress temperature max*". Estas nomenclaturas y referencias son fruto del resultado bajo IEC 61360 en IEC CDD, el resultado en la Figura A.11 y Figura A.10 en apéndice J.

[10]Identificación global http://wiki.eclass.de/wiki/IRDI

A continuación la figura del modelo resultante:

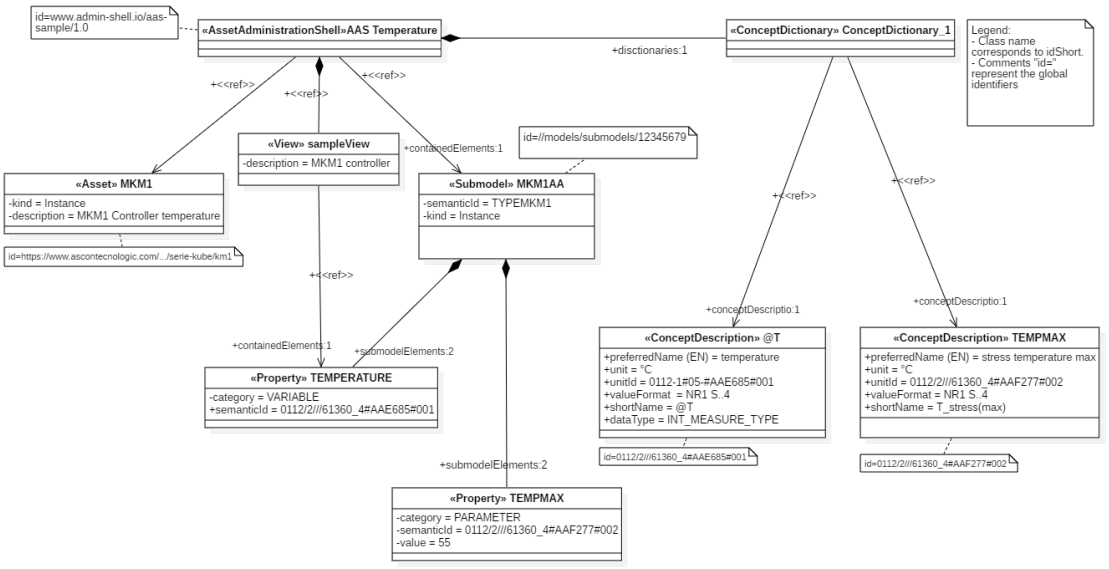


Figura 5.10: meta-modelo del AAS implementado

Las "ids" de la figura 5.10 se elaboran siguiendo los patrones recomendados por Industria 4.0, indicado en el apéndice H. Implementar este modelo junto a OPC UA tiene un funcionamiento similar al ejemplo 2 con la diferencia que ahora el componente o tipo de objeto que se maneja es un AAS y sus propiedades.

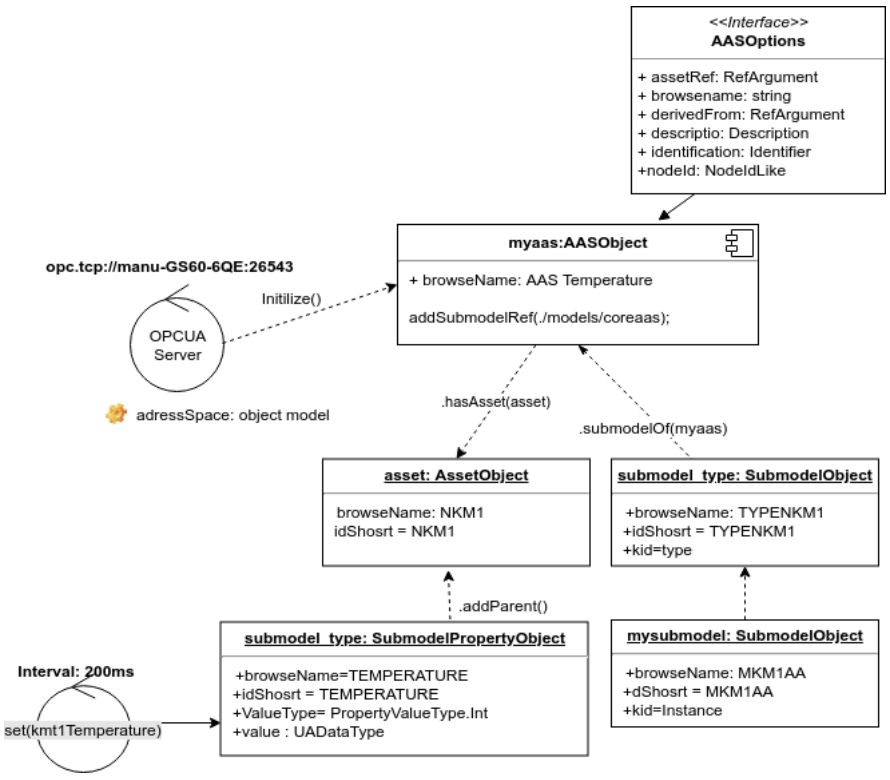


Figura 5.11: Ejemplo de inicialización OPCUA Server con AAS Object

El fragmento código referente a este ejemplo en apéndice C.2

CAPÍTULO 6
Implementación

6.0.1. Web Page y Web API

Para el funcionamiento remoto se despliega la Web Page junto a la Web API en un servicio IIS con .NET Framework 4.0 y como medio de configuración un fichero xml denominado DatosLocales.xml. El contenido de este fichero se especifica en apéndice B.1 En la siguiente imagen un ejemplo de subida del proyecto al con IIS:

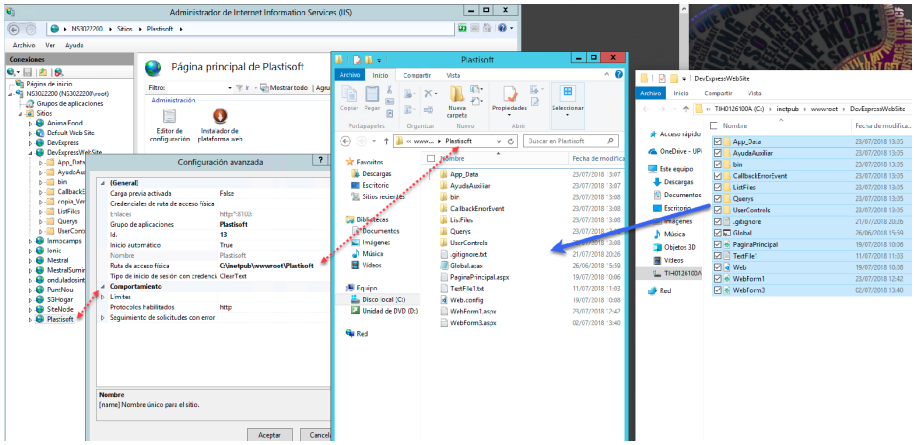


Imagen 6.1: Subida del proyecto al servicio IIS

La plataforma Web Page cuando es lanzada en el servidor IIS, solo puede acceder un usuario administrador. Este usuario visualiza todos los datos o parámetros necesarios para determinar el correcto funcionamiento de la base de datos remota y su información. El administrador, por tanto, es el único capaz de acceder mediante un login:

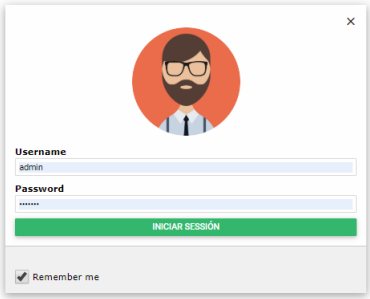


Imagen 6.2: Login Administrador

Una vez inicia sesión el administrador, se permiten realizar una serie de funcionalidades:

- Conexión con la base de datos, a partir de los datos de configuración.
- Listado, visualización y interactividad con *Dashboards* o Listados (*DataGridView*).
- Resumen de los permisos de todos los operarios que actúan en el sistema como clientes.
- Listado de los datos locales que puede visualizar el servicio.
- Creación en tiempo real, de *Dashboards* o tablas dinámicas a partir de la conexión con la base de datos o otra fuente de datos.

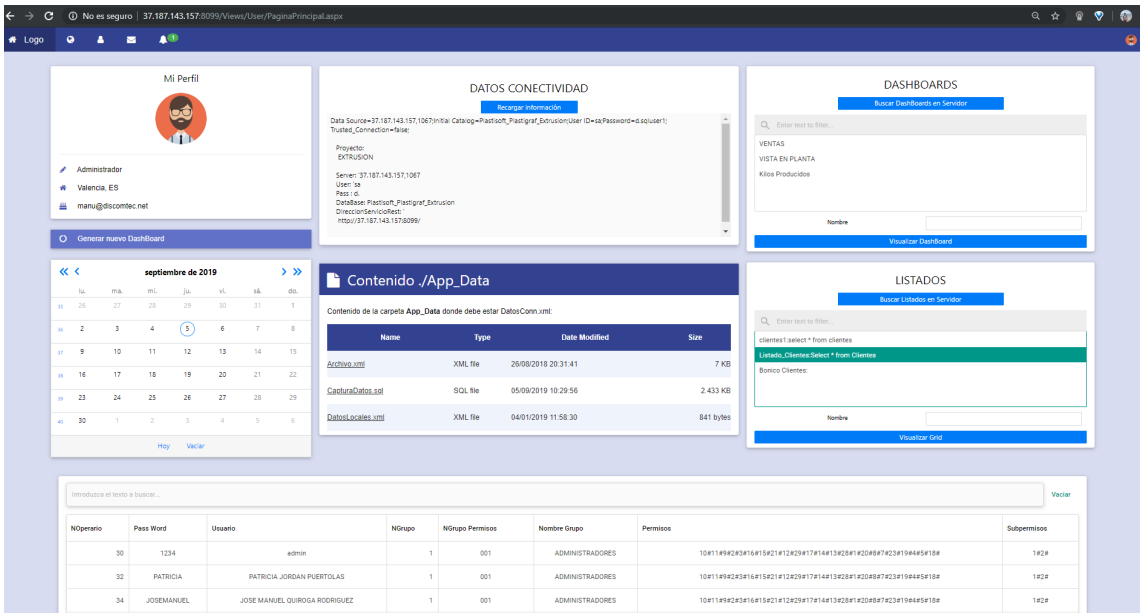


Imagen 6.3: Página Principal Administrador

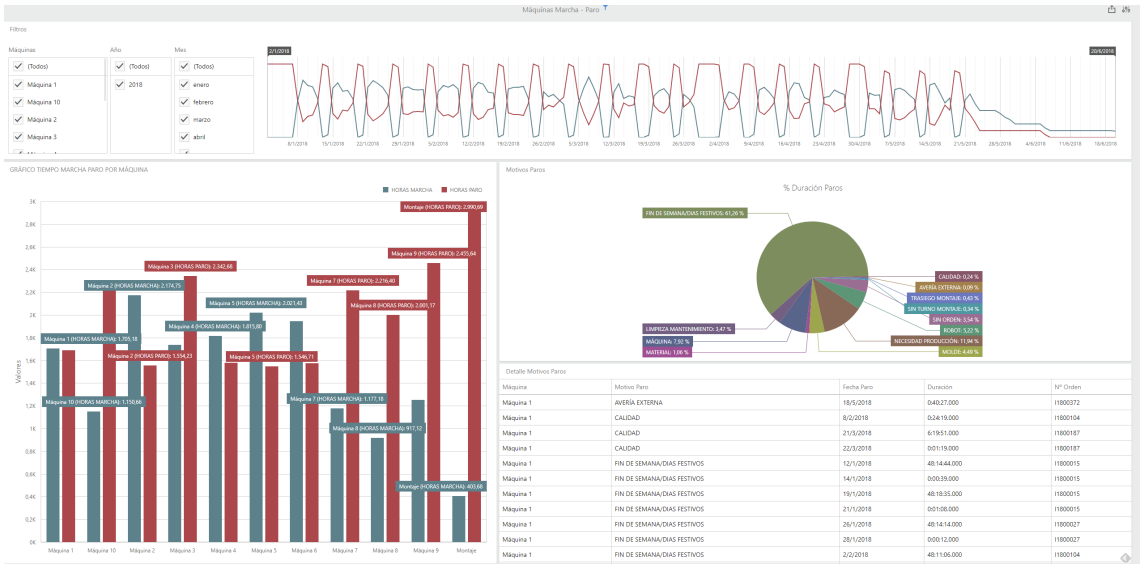


Imagen 6.4: Vista de un Dashboard creado

6.0.2. Aplicación cliente

Primero, un diagrama de casos de uso, para visualizar las interacciones más relevantes que al actor puede realizar:

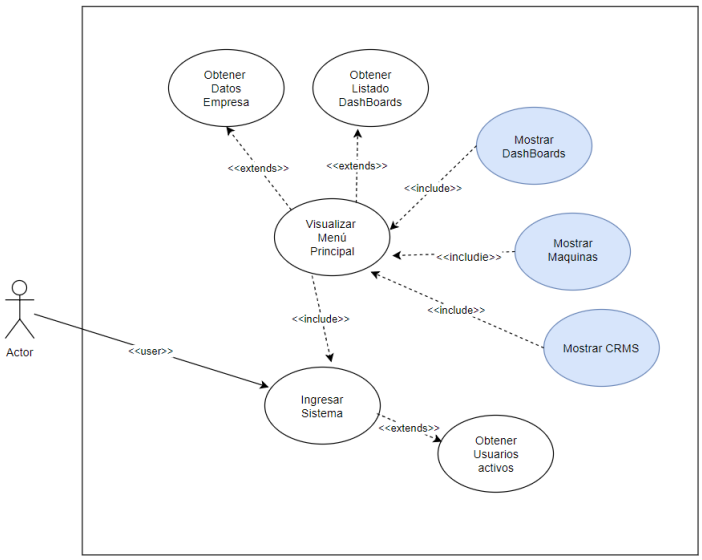


Imagen 6.5: Diagrama caso de usos

A continuación una serie de imágenes de las funcionalidad más relevantes de la aplicación Ionic iniciada:

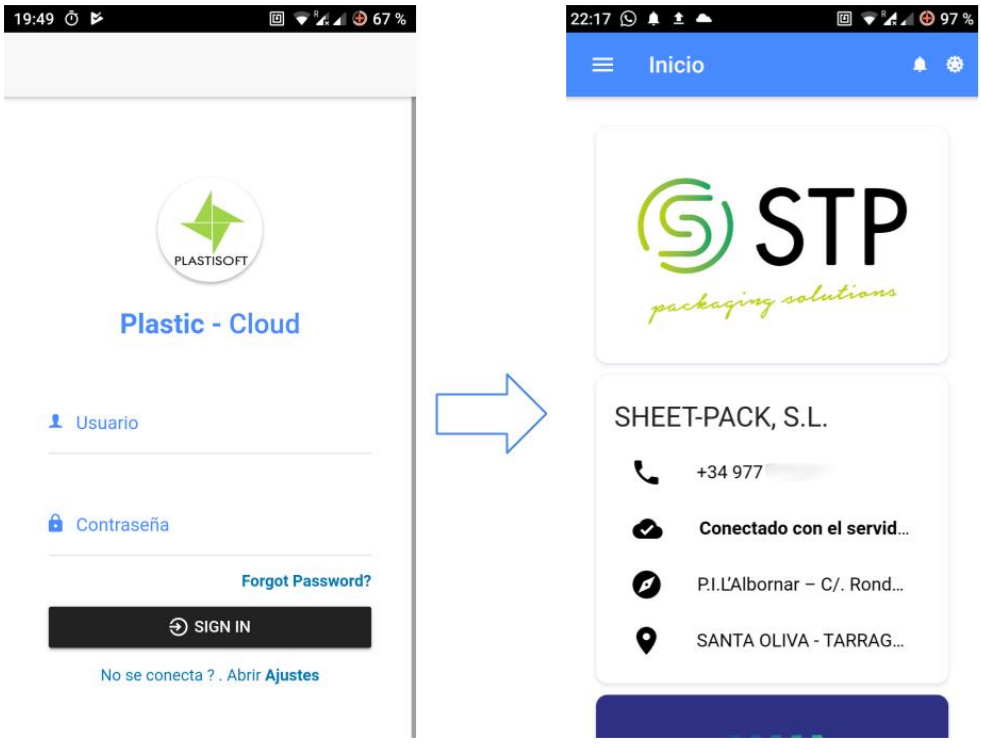


Imagen 6.6: Interfaz Login y página principal

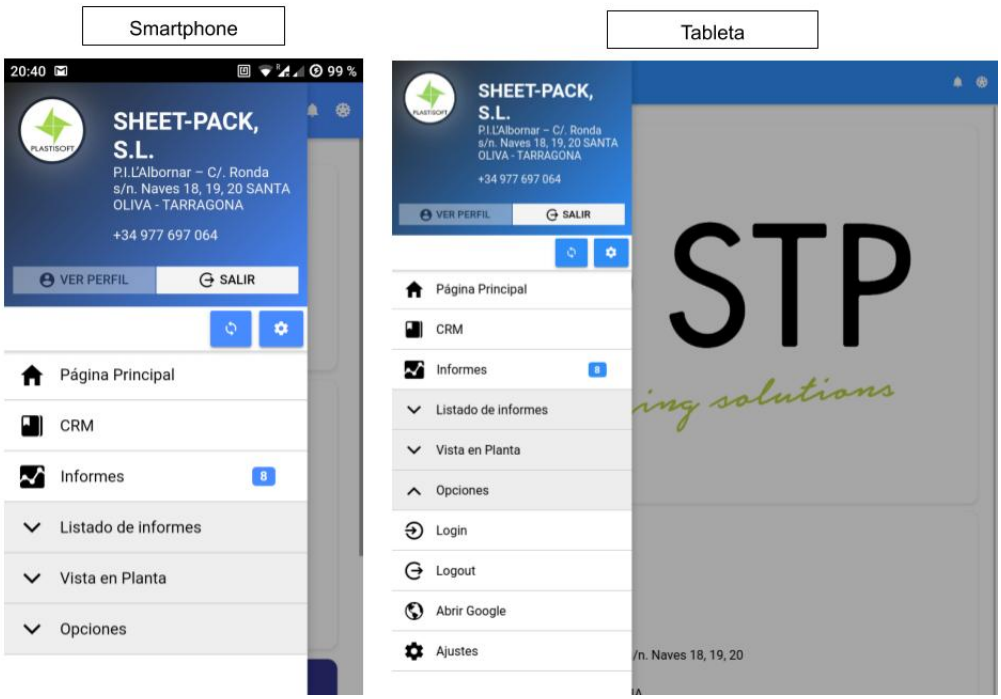


Imagen 6.7: Visualización en diferentes formatos

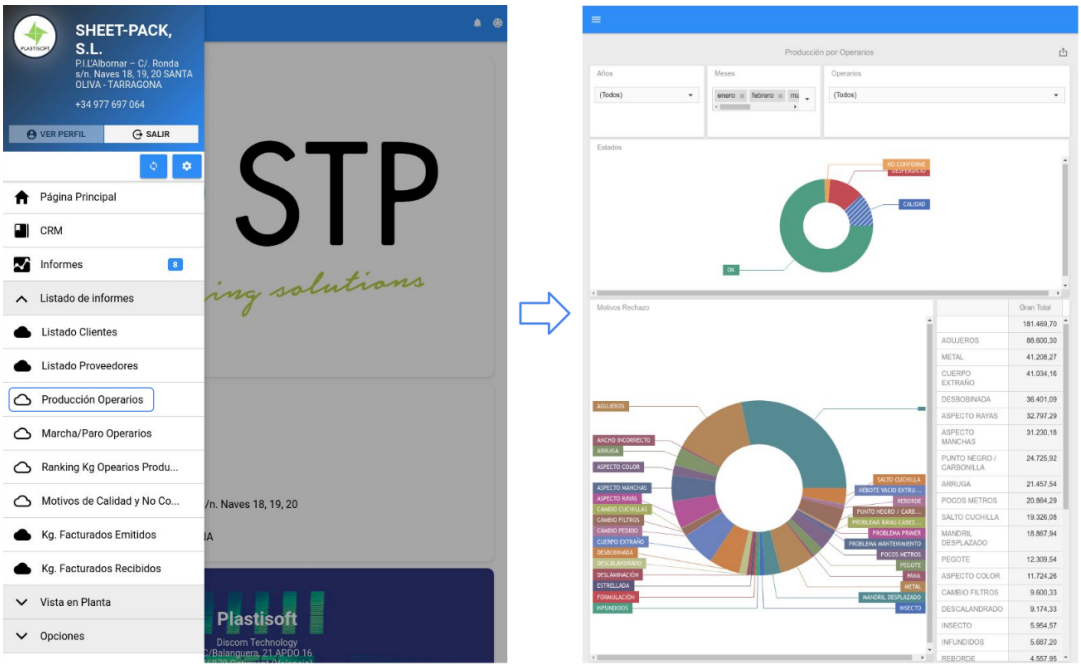
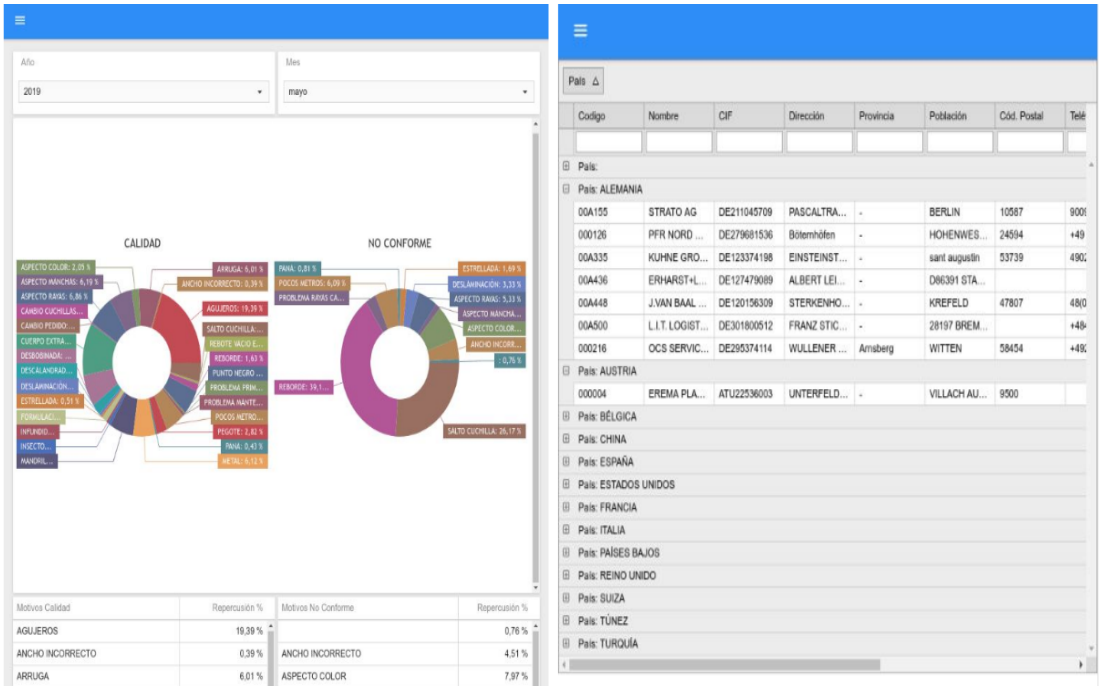


Imagen 6.8: Visualización en diferentes formatos



(a) Componente Dashboard. **(b)** Componente Listado (Data Grid).

Imagen 6.9: Dos componentes DevExpress diferentes en la aplicación cliente

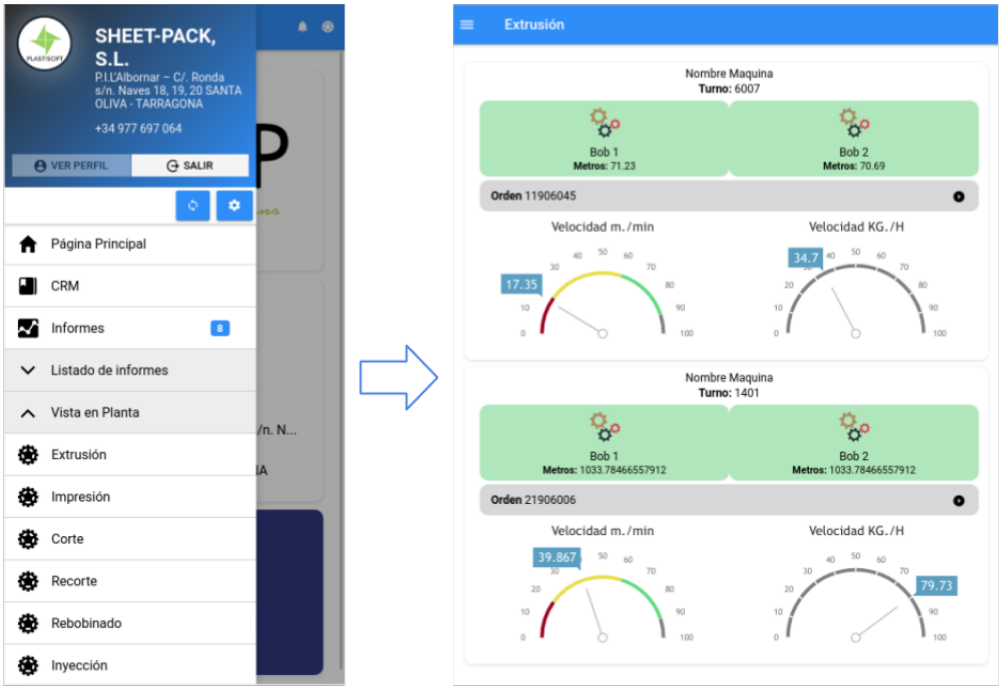


Imagen 6.10: Vista en planta Extrusión

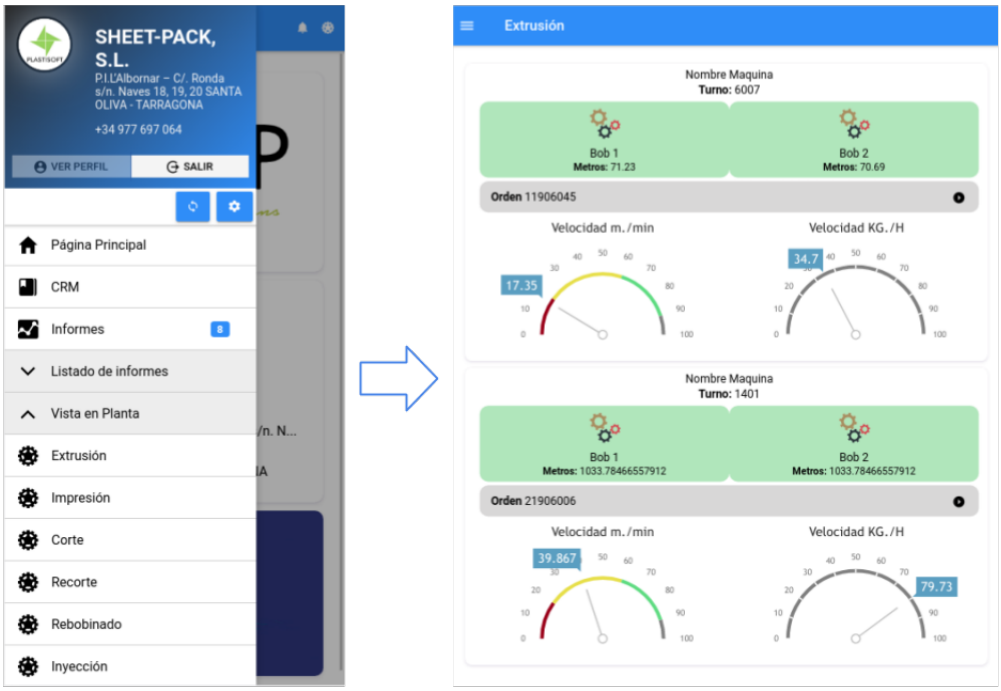


Imagen 6.11: Vista en planta Extrusión

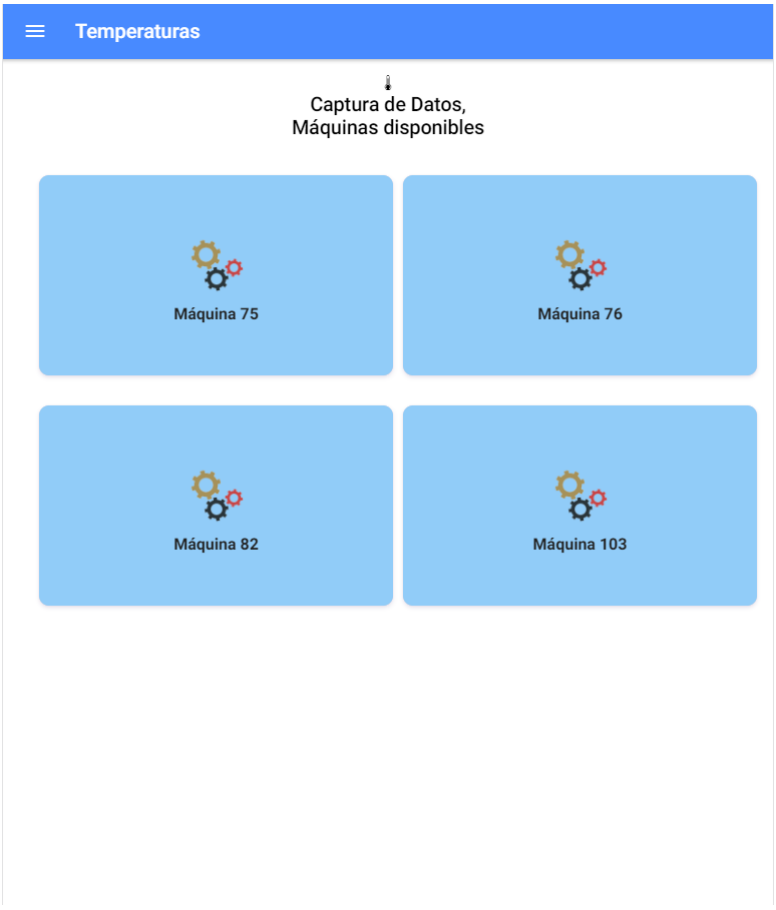


Imagen 6.12: Vista de máquinas en marcha

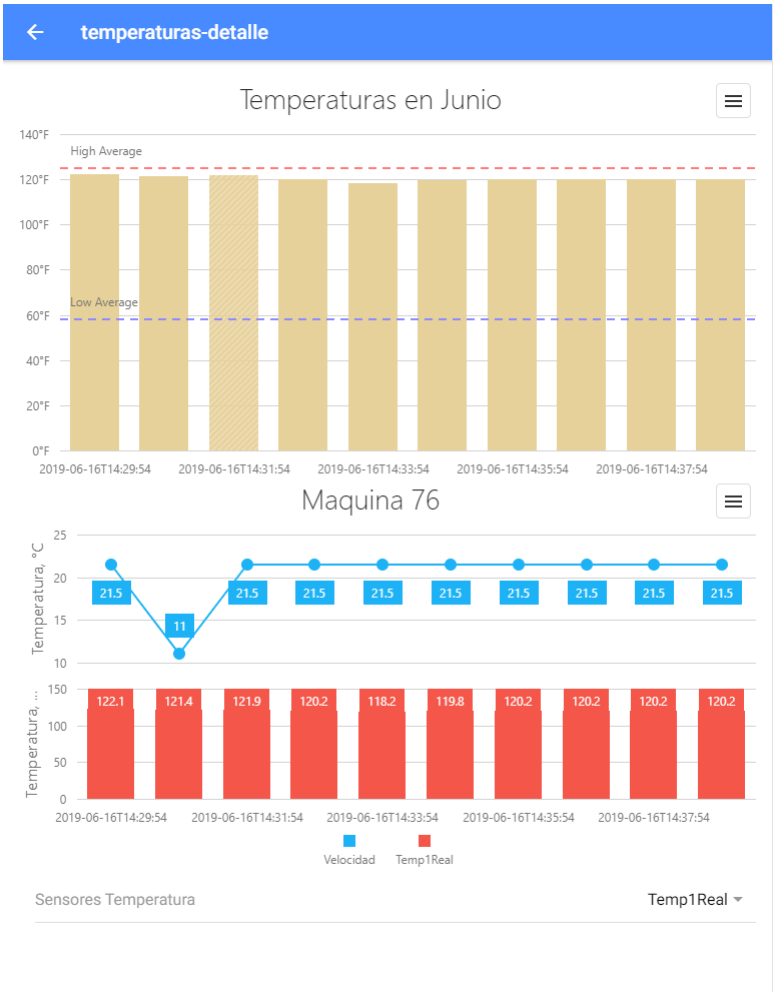


Imagen 6.13: Detalle temperaturas de máquinas

6.0.3. Ejemplos AAS

Para la implementación AAS solo es necesario tener instalado NodeJs e importar las extensiones necesaria mediante el comando *"npm install nombre-extension"*. Para el uso del ejemplo 1 se rutean las dos funciones *unmarshalling* y *marshalling* sobre las siguiente direcciones:

Dirección	Operación
`http://hostname:8081/api/sample1/marshal`	*marshal*
`http://hostname:8081/api/sample1/unmarshal`	*unmarshal*

Tabla 6.1: Direcciones del servicio REST

```
- <aas:aasenv>
  - <aas:assets>
    - <aas:asset>
        <aas:idShort>MKM1</aas:idShort>
      - <aas:description>
        - <aas:langString lang="ES">
            Controlador con Regulador con temporizador independiente KM1
          </aas:langString>
          <aas:langString lang="EN">CONTROLLER AND MINI-PROGRAMMER KM1</aas:langString>
        </aas:description>
        <aas:identification idType="URI">http://red.upv.com/KMT1HCRRRRDS-S</aas:identification>
        <aas:kind>Instance</aas:kind>
      </aas:asset>
    - <aas:asset>
        <aas:idShort>KMT1HC</aas:idShort>
      - <aas:description>
          <aas:langString lang="ES">Controlador con Sensor de temperatura de silicio</aas:langString>
          <aas:langString lang="EN">Controller with Silicon temperature sensor</aas:langString>
        </aas:description>
      - <aas:identification idType="URI">
          https://www.ascontecnologic.com/es/automatizacion-industrial/controladores-industriales/serie-kube/km1
        </aas:identification>
        <aas:kind>Instance</aas:kind>
      </aas:asset>
    </aas:assets>
</aas:aasenv>
```

(a) Resultado de *marshal*.

AAS

Variable	Valor	Tipo
ID	http://pk.festo.com/3s7plfdrs35	URI
kind	Instance	String
description	Festo Controller	AAS.LangStringsT
property	rotationSpeed	AAS.PropertyT
property	NMAX	AAS.PropertyT

(b) Resultado de *unmarshal*.

Imagen 6.14: Dos componentes DevExpress diferentes

Para la implementación del ejemplo 2 es primordial utilizar la extensión *node-opcua* e implementarlo utilizando como cliente OPC UA, el programa con interfaz gráfica. A continuación una serie de imágenes sobre las partes y funcionalidades más importantes:

Imagen 6.15: Vista History Data en UaExpert

En la imagen 6.15 se puede apreciar un histórico sobre el nodo con nombre temperatura, este histórico, el cual se va actualizando en tiempo real. Existe en "*AdressSpace*", un objeto

creado KM1 con 1 nodo temperatura (nodoId ns=1;i=1001). Toda la información acerca de un nodo en OPC UA en apéndice K.2 y tabla A.4 Para finalizar este programa tiene la posibilidad de mostrar el número de llamadas escrituras o lecturas realizadas durante un determinado número de ciclos. Para el ejemplo solo existe una estadística de lecturas sobre el nodo consultado.

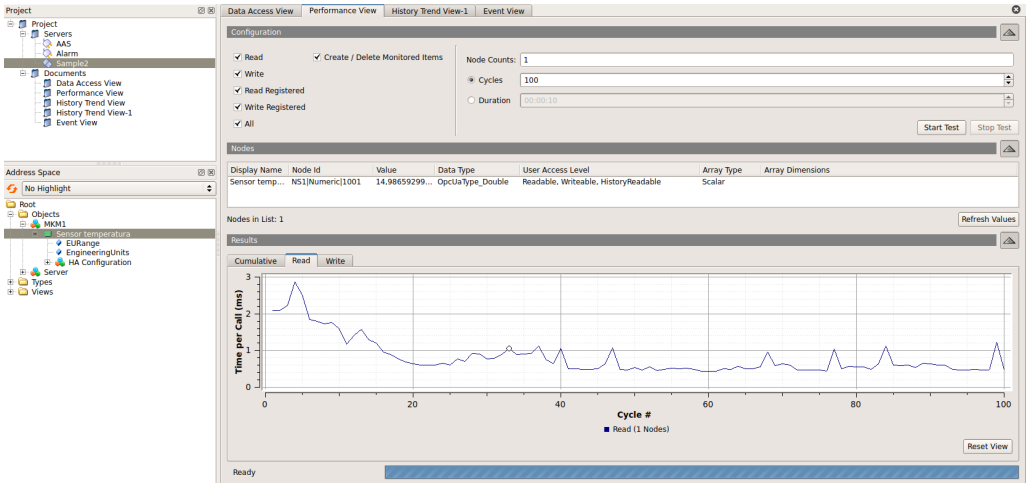


Imagen 6.16: Vista Performance View en UaExpert

Se termina la implementación con el tercer y último ejemplo, donde el AAS creado es consultado con el programa OPC UA cliente anterior. De esta manera se puede visualizar toda la estructura de nodos y referencias creadas en *AdressSpace*. Consiguiendo por parte de un cliente determinar las características y propiedades del activo que contiene AAS, sus referencias a submodelos entre otros. Como resultado exitoso se obtiene la siguiente imagen:

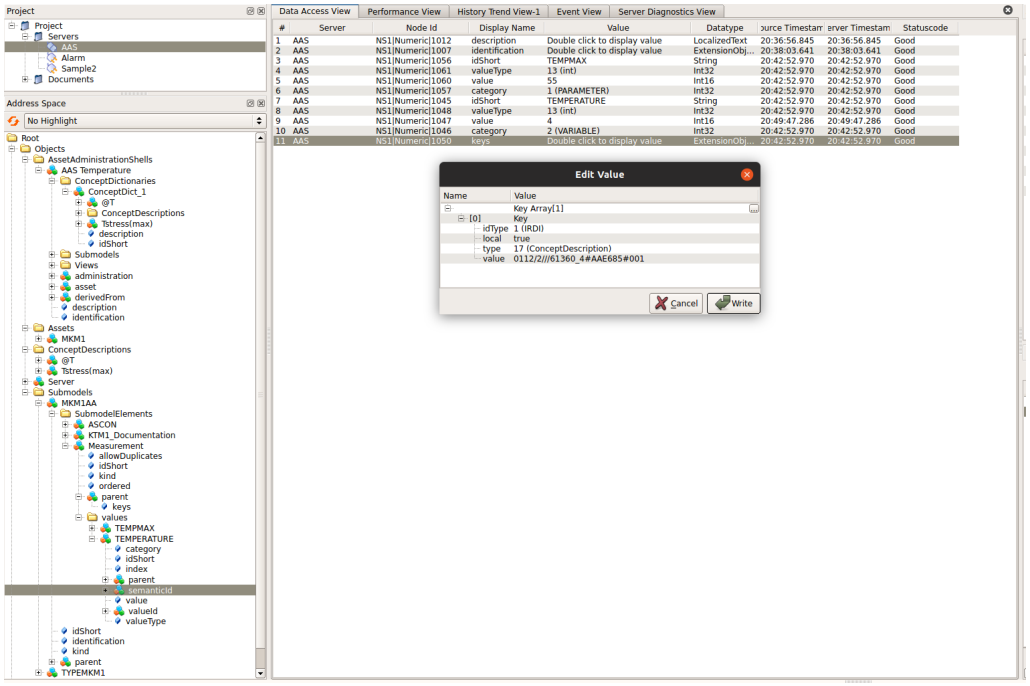


Imagen 6.17: Vista Data View, datos relevantes de un AAS

Para finalizar si dos imágenes donde se observa la estadística de llamadas y escrituras por ciclo de la variable TEMPERATURA y parámetro TEMPMAX.

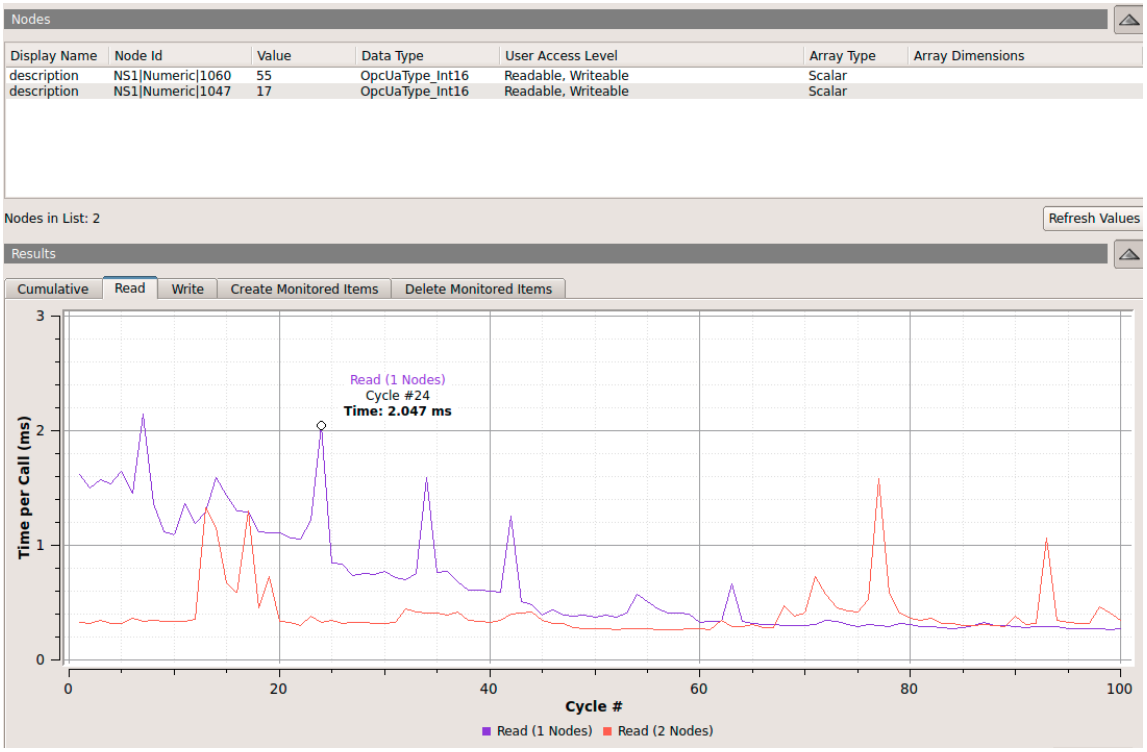


Imagen 6.18: Vista Performance View, llamadas por ciclo

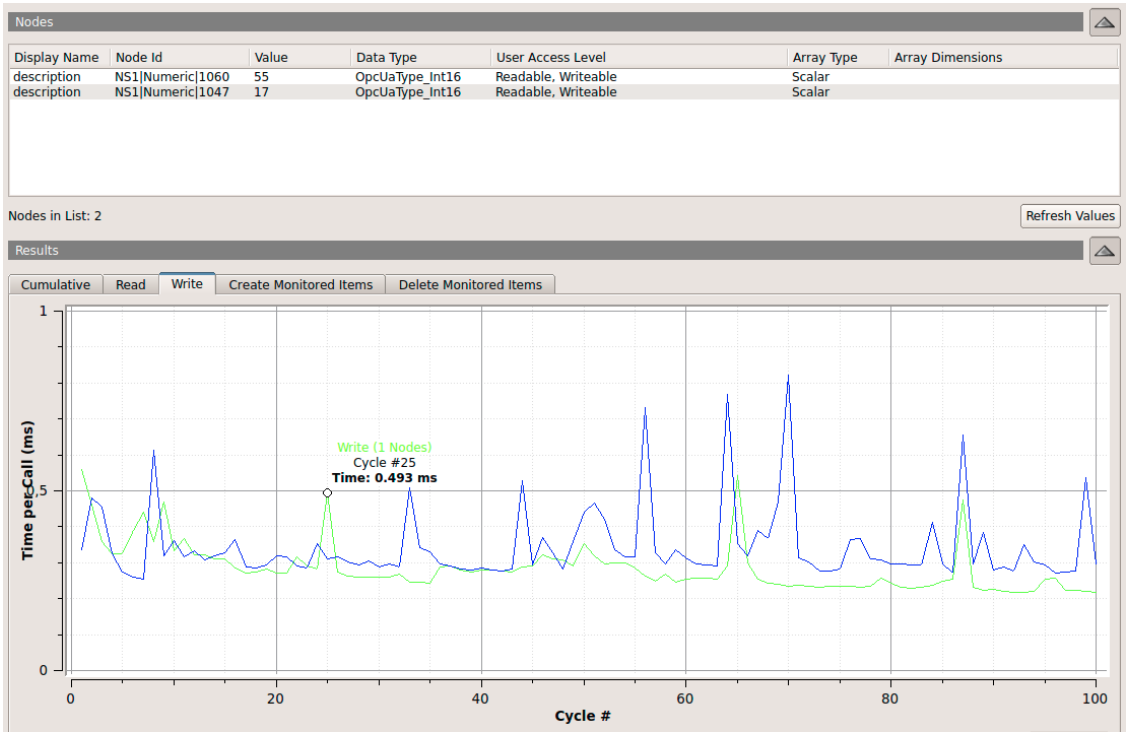


Imagen 6.19: Vista Performance View, escrituras por ciclo

Las 3 implementaciones en un futuro pueden formar parte de una arquitectura orientada al servicio SOA, formada por la interacción de microservicios. A continuación un esquema general:

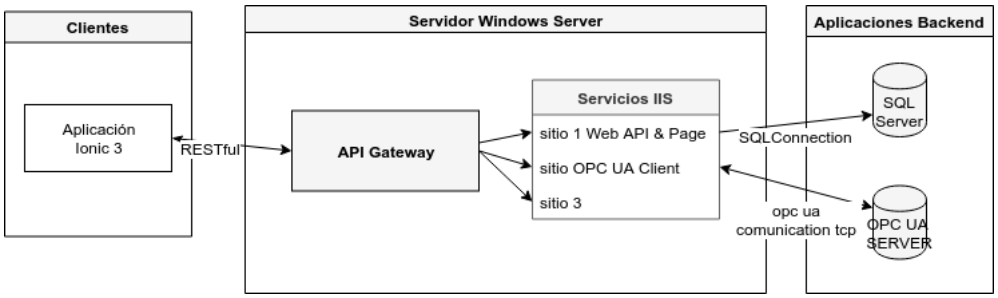


Figura 6.20: Arquitectura general deseada

CAPÍTULO 7
Conclusiones

Las tecnologías de la información están en constante evolución, abarcando diferentes ramas y creciendo a pasos agigantados. El análisis y representación de datos son una constante necesidad. En este contexto, las tecnologías web proporcionan mucha flexibilidad a la hora de enfrentarse a problemas como la dependencia de datos externos y ayudan a construir un sistema multiplataforma, escalable y tolerante a fallos. Además favorecen una constante disponibilidad y interactividad de datos.

Durante el proceso de desarrollo destacan el uso de una gran cantidad de lenguajes y tecnologías, desde SQL, ASP.NET Wep API etc.. del lado backend hasta la utilización del framework Ionic y el lenguaje Angular con TypeScript por parte de la aplicación cliente. Todo ello, hace percatarse del gran dominio y conocimiento que se necesita para el desarrollo de un sistema full stack. Respecto a los objetivos iniciales, se han cumplido los siguientes:

- Servicio capaz de ofrecer mediante servicios REST.
- Entorno software capaz de mostrar de manera adecuada y ordenada la información ofrecida.
- Software capaz de modificar el aspecto o información, para distintos tipos de usuarios o roles.
- Sistema visualizado de información de manera detallada y personalizada.
- Ofrecer un software para cualquier plataforma.

Al empezar el proyecto se tenía constancia de las ventajas ofrecidas por las tecnologías o servicios web, pero ha sido durante la implementación de un AAS de la Industria 4.0 donde realmente se ha apreciado lo relevante no sólo de adaptarse a nuevo tiempos, sino de seguir estándares actuales, utilizar el modelado, mantener una semántica común y permitir ofrecer una futura interoperabilidad, sobretodo bajo un entorno industrial.

La mayor dificultad viene de recopilar toda la información referente a Industria 4.0 para poder aportar toda la información más relevante de una forma estructuralmente ordenada y de fácil comprensión. Además también ha sido una tarea laboriosa el poder implementar adecuadamente un AAS debido a que Industria 4.0 es un concepto que está en constante cambio y donde entran en conflicto diferentes estándares. Existen normalizaciones como la UNE que ayudan al desarrollo correcto, pero es de pago. Por tanto, se ha debido de desarrollar bajo extensiones o implementaciones,que cumplan con los requerimiento y normas especificadas en la documentación.

La estandarización global de muchos elementos que forman parte de algún proceso de producción industria, se consigue un mejor sistema donde, no solo la comunicación está controlada, si no también los involucrados están identificados, alcanzando metas como: La automatización de sistemas de producción, Activos auto-gestionables etc.. .

En conclusión, el proyecto ha aportado un paso más para alcanzar las metas de Industria 4.0. No se han podido conseguir todos los objetivos en este contexto, pero mediante investigación, consultas y pruebas se obtiene una documentación necesaria para encarar y avanzar en el proyecto hacia nuevas metas.

7.1 Relación del trabajo desarrollado con los estudios cursados

Todos los mecanismos, metodologías, lenguajes o estructuras utilizadas se pueden relacionar directamente las asignaturas cursadas a lo largo del año. A continuación se nombran aquellas que han tenido más relevancia a la hora de afrontar el proyecto:

7.1.1. Diseño y Aplicación de Sistemas distribuidos

Asignatura de cuarto curso de la rama de “Ingeniería de computadores” , se obtuvieron conocimientos respecto a tecnologías web REST o modelo cliente servidor, contribuyendo de gran ayuda para el desarrollo del proyecto

7.1.2. Redes

Aquí se hace referencia tanto a la asignatura de segundo curso “Redes de computador” como la asignatura de cuarto “Tecnología y administración de redes de computador”, esenciales para tener el conocimiento en protocolos de comunicación implicados en el proyecto

7.1.3. Gestión de proyectos

Asignatura de tercero, se aprendieron metodologías de planificación y manejo de herramientas apropiadas de mucha relevancia para empezar abordar el proyecto de una manera controlada

7.1.4. Interfaces persona computador

Los conocimientos adquiridos en esta asignatura de segundo ayudaron a desarrollar interfaces clientes funcionales y hacer uso de patrones de diseño en el trabajo.

7.1.5. Base de datos y sistema de la información

Esta asignatura fue fundamental la hora de implementar consultas elaboradas SQL, manejarse con vista y realizar las optimización oportunas en la base de datos.

7.1.6. Ingeniería del Software

Asignatura de tercero la cual aportó conocimientos relevantes para el proceso de desarrollo del proyecto, como el uso de buenas prácticas, control de versiones o lenguaje de modelado.

CAPÍTULO 8
Trabajos futuros

El proyecto empezó con la idea original de poder adaptar un sistema industrial y crear nuevos servicios o herramientas que impliquen un visualizado e integración de información en tiempo real, con el uso de nuevas tecnologías y solucionando problemas de escalabilidad, disponibilidad y tolerancia a fallos. Como resultado, nace un sistema de monitorización de máquinas adaptado al sistema y entorno definidos, añadiendo posibles mejoras que han ido apareciendo durante el desarrollo, como la automatización, estandarización o interoperabilidad, dónde Industria 4.0 tiene mucho que aportar al respecto. Por ello, se ha visto la necesidad de aportar una serie de ideas y conceptos sobre la Industria 4.0 y su posible contribución mediante la implementación de requerimientos mínimos en un AAS.

Debido a la envergadura del proyecto y la cantidad de factores en juego para implementar un sistema I4.0 totalmente funcional, no ha sido posible poder crear un AAS completamente funcional y se dejan sin cubrir otras capas de RAMI 4.0 como es la seguridad o administración. Hubiera sido interesante en el AAS poder haber creado más funciones, vistas o alarmas, además de crear submodelos específico para el activo que se representa. Como resultado, se pueden añadir ampliaciones del trabajo realizado a partir de los ejemplos creados. Por ejemplo, gracias al uso del lenguaje Nodejs, un AAS puede ser implementado en una RaspberryPI realizando un monitoreo de temperaturas, también en el propio software cliente Ionic, existe la posibilidad de crear un cliente OPC-UA.

Para terminar, como mejora futura, se pretende mejorar la seguridad del sistema Backend y aumentar su escalabilidad, por tanto se debe actualizar el framework .Net del servidor IIS, para poder aportar nuevas versiones de ASP.NET, crear nuevos servicios para funciones concretas con lenguajes más versátiles como Node.js, utilización de tokens, entre otros muchos cambios.

Bibliografía

[1] Department of Computer Integrated Design, Technische Universität Darmstadt. Integrated Data Model and Structure for the Asset Administration Shell in Industrie 4.0. Consulta a https://www.sciencedirect.com/science/article/pii/S2212827117300495.

[2] A systematic approach to OPC UA information model design. Institute for Production Engineering and Laser Technology Consulta a https://publik.tuwien.ac.at/files/publik_252957.pdf.

[3] RESTful Industrial Communication with OPC UA. Artículo obtenido en IEEE Transactions on Industrial Informatic Consulta a https://www.researchgate.net/publication/294723351.

[4] The Secure implementation of OPC UA for operators, integrators and manufacturers. Consulta a https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/secure-implementation-of-opc.html.

[5] Details of the Assets in Administration Shell Consulta a https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/2018-details-of-the-asset-administration-shell.html.

[6] Reference Architectural Model Industrie 4.0 (RAMI4.0)-An Introduction Consulta a https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.htm.

[7] Relationships between I4.0 Components-Composite Components andSmart Production Consulta a https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publiktion/hm-2018-relationship.html.

[8] Making Industrie 4.0 components interoperable with the Administration Shell Consulta a https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/2019-administration-shell-flyer.html.

[9] Examples of the Asset Administration Shell for Industrie 4.0 Components Consulta a https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2017/April/Asset_Administration_Shell/ZVEI_WP_Verwaltungschale_Englisch_Download_03.04.17.pdf.

[10] OPC UA based Field Device Integration, Daniel Grossmann ; Klaus Bender ; Benjamin Danzer Consulta a https://ieeexplore.ieee.org/document/4654789.

[11] OPC Unified Architecture Specification, Overview and Concepts Consulta a http://www.diit.unict.it/users/scava/dispense/II/OPCUA.pdf.

[12] Examples of the Asset Administration Shell for Industrie 4.0 Components Consulta a https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2017/April/Asset_Administration_Shell/ZVEI_WP_Verwaltungschale_Englisch_Download_03.04.17.pdf.

[13] Usage View of Asset Administration Shell Consulta a https://www.researchgate.net/publication/331959412_Usage_View_of_Asset_Administration_Shell.

APÉNDICE A
Documentación Industria 4.0

Dentro de este apartado del anexo, se incluye toda la información que se ha ido recopilando a lo largo del desarrollo del proyecto ordenadamente. Esta información extiende los conceptos de Industria 4.0 en apartados que afectan a nuestro proyecto como es la normalización y representación de activos, los datos que se manejan o la comunicación realizada entre activos tanto internamente como hacia el exterior.

A Contexto

Industrie 4.0 nace de la idea de realizar una representación virtual de un proceso de producción que puede ser comprendida por otra en un lugar distinto, mediante el uso normas, estándares y metodologías comunes que I4.0 acepta como válidos. A continuación se procede a profundizar, poco a poco, sobre las posibles metodologías y normativas a seguir válidas dentro del ámbito de I 4.0, centrándose en la representación de activos para una correcta identificación, incluyendo medidas y buenas prácticas a tener en cuenta para poder añadir una futura interoperabilidad. Este ámbito de Industria 4.0 se rige y aplica bajo un Modelo de Arquitectura de Referencia Industria 4.0 (RAMI 4.0).

Este modelo incluye la estandarización de funciones y recubrimiento de gran parte de los activos. Estos activos o "assets" son cubiertos y referenciados con ayuda de una capa de software, llamado "Administration Shell". El encargado de conectar I4.0 a los objetos físicos, almacenan todos los datos e información sobre el activo, proporciona una interfaz estandarizada de comunicación de red y proporciona la posibilidad de integrar activos pasivos.

Todo activo, puede ser representado virtualmente, por tanto, también contiene la llamada meta-información. Junto a ello, una serie de campos obligatorios, como vínculos con activos por medio de identificación y seguridad. Este mismo activo forma parte de uno o varios componentes, bajo el Admin Shell. Proporcionando un el enlace, que permite, un acceso externo a su representación virtual y funcionalidades técnicas del mismo.

Este componente puede ser vinculado a una arquitectura de servicios (SOA) o implementarse en un *Administration Shell* en un repositorio. Este componente, es denominado, componente 4.0, unión de uno a más activos, que siguen unos esquemas de representación de sus datos estandarizados (CDD), con un Administration Shell, cada uno de estos componentes a su vez se comunican siguiendo protocolos de comunicación (OPC UA).

B RAMI 4.0

Es un modelo de arquitectura de referencia para Industria 4.0 presentado en la feria Hannover Messe 2015, Su uso está en constante cambio y se considera un reto, debido en gran parte, a su inusual representación tridimensional.

1. El eje vertical existen seis capas apiladas entre sí y son utilizadas para representar diferentes perspectivas, tales como mapas de datos, activos, descripción funciones, características de la comunicación y sistemas de negocio. Estas capas son asset, Integration, Comunication, Information, Functional y Bussines.

2. El eje horizontal incluye un ciclo de vida (tipo) y vida útil (instancia) de los productos, junto a sistemas de producción con el valor que contiene cada etapa de la producción (cadena de valor).

3. El tercer eje(z), representa la jerarquía funcional de los datos como la asignación de funciones y responsabilidades.

Estos niveles de jerarquía viene de la norma IEC 62264 que RAMI 4.0 amplia añadiendo el nivel de "Producto"[6]. Por tanto, RAMI 4.0 tiene una importante combinación de cadena de valor y ciclo de vida con una estructura jerárquica con la que definir componentes I4.0. En la siguiente imagen muestra una idea global de la forma que se obtiene al combinar los ejes:

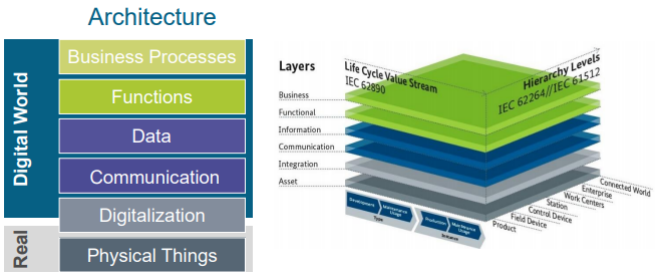


Imagen A.1: Arquitectura RAMI 4.0[5]

El modelo RAM I4.0 en su capa de activos, cuenta con un ciclo de vida generalizado, que se deriva de la norma IEC 62890. La idea básica es distinguir de todos los activos en Industrie 4.0 entre los posibles tipos y ejemplos o instancias. Esto hace que sea posible implementar una distinción tipo/instancia para todos los elementos tales como el tipo de material/ejemplo del material, tipo de producto/instancia del producto, tipo de máquina/instancia de la máquina y más. Por consiguiente, en la capa de activo se postula cada característica, función o fuente de datos (sensor).

C Activos

Los activos representan un valor para una empresa, pueden ser parte del mundo real (físico o virtual) y pueden estar formados por elementos tales como fábricas, sistemas de producción, equipos, máquinas, componentes, productos elaborados y materias primas, procesos de negocio y las órdenes, activos inmateriales (procedimientos, software, documentos, planes, normas), servicios y personal humano. En este proyecto, por ejemplo, se deben de comprender elementos orientados a máquinas del sector plástico, como las temperaturas. Todos los activos están representados en la capa de activos según el eje de

capa RAMI 4.0 y por ende será donde se enfoca el proyecto.

La información que puede obtenerse de un activo debe estar disponible siempre que sea deseable, esto es un requisito necesario en el sistema de I.40, sin embargo, esta información puede variar drásticamente dependiendo del nivel de abstracción donde se precise, pero como mínimo para obtener una información válida del activo, necesitas una representación virtual y descripción adecuada del mismo, más adelante se profundiza en la representación y descripción de un activo. Estos se relacionan formando un meta-modelo compuesto por las clases tipos y instancias de los activos y sus relaciones. Este meta-modelo permite ofrecer una virtualización global de ciertas características de un máquina.

Puede surgir otras relaciones entre activos derivada de la retroalimentación entre ellos, a lo largo del ciclo de vida. Normalmente estas relaciones no son planificadas al inicio y se crean durante el proceso de desarrollo o implementación,

D Asset Administration Shell

Por simplificar el concepto, se puede considerar el *Administration Shell* como la capa de abstracción de los activos que contiene, como si de un etiquetado y métodos de uso de un producto se tratara. La información almacenada como documentación, índices y diccionarios como es compartida globalmente y se encuentra localizada en la denominada carcasa shell.

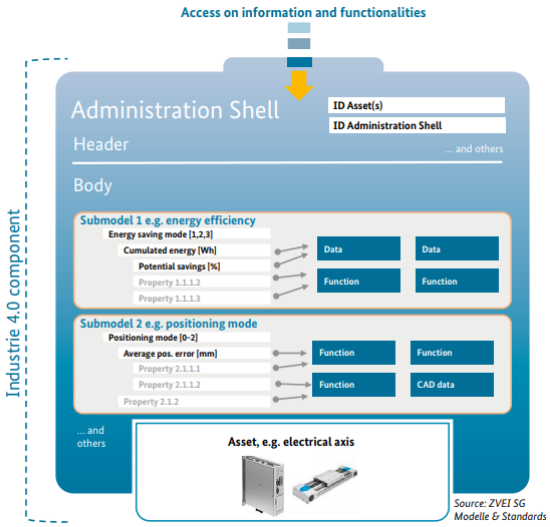


Figura A.2: Ejemplo de Componente 4.0 y Administration Shell[5]

El nombre AAS se utiliza constantemente como sinónimo de la cáscara de la administración y su estructura está dividida en dos partes. Una cabecera DF[1] y un Cuerpo DF, esto son concepto del marco *Digital Factory* (referencia IEC TS 62832). La cabecera cuenta con una lista de propiedades I4.0 compatible, como la identificación del activo físico y de la AAS. Esto designa la identidad única del componente I4.0. Esta lista es definida como un conjunto definido de meta-información que proporciona información acerca de las propiedades funcionales y no funcionales del componente I4.0.

[1]Digital Factory

El cuerpo DF contiene un componente manejador, responsable de la gestión y el acceso a los servicios en submodelos individuales. Cada submodelo consta de propiedades jerárquicamente bien organizados vinculados a diversos datos o funciones siguiendo un formato de datos normalizado, tal como eCl@ss o IEC IEC61360. Por último un AAS, consta de una interfaz de comunicación orientada a servicios, permitiendo el acceso externo a los datos y funciones. AAS debe garantizar la adquisición continua de datos en tiempo de ejecución generados por los activos físicos.

E Componente 4.0

Una de las características esenciales de los sistemas I40 es que los activos se representan como componentes I40 y entran directamente en contacto entre sí para ejecutar tareas. Para este propósito se requieren patrones especiales de interacción. El concepto fundamental es que los componentes I40 intercambian mensajes que son manejados por un administrador de interacción.

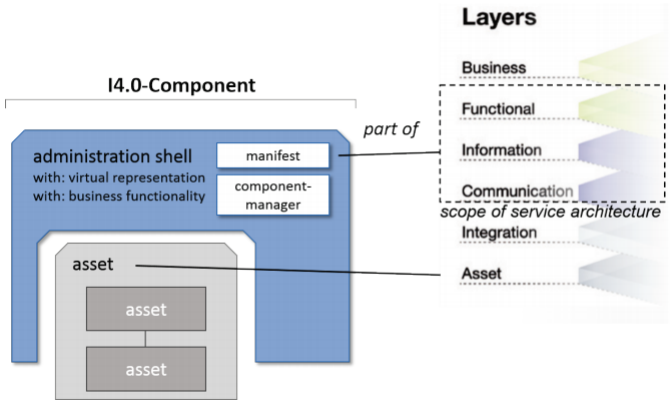


Imagen A.3: Capas RAMI 4.0 del componente 4.0[5]

Este componente es definido por la normalización I4.0 cómo "*component 4.0*", representando una entidad física o digital, pudiendo ser accesible desde el exterior. Por tanto, se considera a un componente I4.0 la combinación del propio activo y *Administration Shell*.

F AAS en un componente 4.0

El AAS puede dividirse en cinco segmentos[1]:

- Interfaz externa
- Autenticación y seguridad
- Gestión de datos
- Funcionalidad y administración
- Interfaz interna

La interfaz externa administra el flujo de datos con los "componentes I4.0" locales y otras redes interconectadas. Por lo tanto, la comunicación tiene que ser altamente estandarizada para asegurar una interoperabilidad. Para ello su representación interna de sus elementos como los activos, también debe ser la adecuada (identificación, parametrización

etc..). El I4.0 componente, debe tener un correcta identificación tanto del propio AAS, como todos sus elementos(activos o assets), una diferenciación de sus elementos (tipo o instancia) y una correcta implementación y diferenciación de todas las relaciones existentes en el modelo AAS.

Para que los componentes I4.0 puedan interoperar, se requiere de un identificador único para cada propiedad, submodelo y AAS, lo que asegura que la información mediante submodelos quede localizada y luego se realice una comunicación entre componentes 4.0 mediante una semántica común. Los submodelos juegan un papel fundamental en el núcleo de AAS en la compresión mediante la información que representa, de todas las propiedades o elementos de un AAS.

Existen submodelos genéricos que son submodelos estandarizados que se utilizan como base para una amplia variedad de activos. Se caracterizan por hacer uso del estándar IEC 61360 para las referencias a propiedades de un activo. Por tanto para un AAS, es esencial que contenga una descripción de las características de la clase de activo que contenga, cuyas propiedades son referenciadas bajo un submodelo con referencias al estándar.

En resumen, es necesario disponer de una semántica común para lograr una interoperabilidad entre AAS, para ello se utiliza IEC 62632 como modelo de estructuración y IEC 61360 o eCl@ss para especificar varios conceptos fundamentales como diccionarios, clases de elementos (propiedades), tipos de elementos de datos y lista de valores. Por ejemplo “max valor de temperatura permitida“, esa propiedad debe estar descrita por alguna referencia de IEC 61360. La definición original se puede hacer sin seguir dichas normativas y ser interpretadas por los humanos para la implementación del comportamiento del componente o incluso de la información de un activo. Sin embargo, uno de los propósitos de la industria 4.0, es conseguir tener datos legibles por máquinas y pudiendo interoperar sin ningún esfuerzo humano.

En este proyecto se busca la máxima normalización posible de los activos, que en un futuro, puede estar asociados a un Administration Shell y formar un componente I4.0. Para ello, estos activos deben seguir una definición de un meta-modelo que indica que atributos son obligatorios y cuales son opcionales para todo AAS. Se intenta representar adecuadamente las direcciones, identificadores y la semántica, que no dejan de ser tipos esenciales de metadatos de cualquier entidad en el AAS.

G Requisitos de un AAS

Basándose en los apartados anteriores, se especifican una serie de requisitos importante con respecto al Asset Administration Shell. Estos requerimientos son independientes de los requisitos relacionados con el contenido del AAS.

Identificación: Cada AAS, activo y submodelos de instancias y tipo asociados deben tener una identificación como las propiedades asignadas provenientes de repositorios externos como eCl@ss o IEC CDD.

1. Representación: El AAS debe contener una descripción mínima, pero suficiente y fiable de la información referente al activo que alberga.

2. Comunicación: Cada AAS debe ser capaz de establecer una comunicación con otros AAS, activa o pasivamente.

3. Etapa de ciclo de vida: Cada AAS pueden y deben operar en una etapa del ciclo de vida del activo durante un cierto periodo.

4. Función: El AAS debe proporcionar funcionalidades técnicas relativas a la función del activo en una tarea de producción.

5. Interoperabilidad: Debe cumplirse para permitir una compresión semántica común de la información que se intercambia.

Otros requerimientos que se deben considerar en un AAS son:

- El poder acceder a la información en el Shell de administración mediante una arquitectura orientada a servicios (API)
- El Shell de administración puede incluir referencias a otros Shell de administración o información I4.0
- Posibilidad de añadir propiedades adicionales como por ejemplo, algún parámetro específico del fabricante.

H Identificación de un AAS

El Shell de Administración, necesita de un identificador único, así como cada uno de los activos que lo describe, ya que estos se deben diferenciar de otros componentes I4.0. Estos identificadores únicos son globales y deben de seguir el esquema permitido dentro de los estándares ISO 29002-5, ISO IEC 6523 e ISO IEC 11179-6.

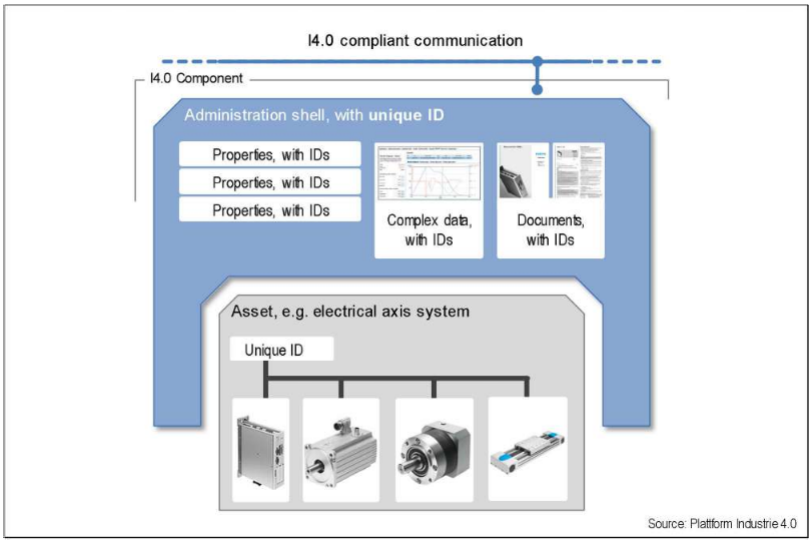


Figura A.4: Componente I4.0 y sus elementos[5]

Estos elementos de la Figura A.4 carecen de un identificador único y pueden tener una identificación URI/URL para acceder a dicha información. Dentro de un AAS pueden existir identificadores o URLs locales donde son únicos dentro de un mismo AAS y su creación no está normalizada. Sin embargo existen una serie de buenas prácticas y propuestas en la documentación oficial de Industria 4.0. Para el alcance del proyecto se crea una correcta representación bajos los estándares de Industria 4.0, de un *Asset Administration Shell*.

En las siguientes tablas se especifica la creación de identificadores dependiendo del ámbito o proposito de su creación:

Ámbito del ID	Estándares utilizados	Ejemplo
Global	ISO 29005, ISO IEC 6523 e ISO IEC 11179-6 Espacio de nombres: RFC 5141 y RFC 4151	urn:tag:iso.org,2019:ISO-29002-5:0173-1#02-BAA036#010 Propiedad eCl@ss de "mínima temperatura de operación"de un motor
Local	Espacio de nombre no registrados:RFC 4151	urn:tag:local.domain:myidentifier:3456

Tabla A.1: Ejemplo de URN e IDs basados en URLs del Shell de administración

Para la interacción entre componentes I4.0, la plataforma 4.0 sugiere el siguiente esquema para la creación de URIs:

Elemento	Descripción	Sintaxis
Submodelol	El identificador, con el que se representa un activo en el Shell.	RFC 3986 (URI)
Propiedad/Elemento-ID	Propiedad específica de un activo.	RFC 3986 (URI)
número de instancia	Numeración individual de la instancia de un tipo de activo.	RFC 3986 (URI)

Tabla A.2: Propuesta de la estructura URI

Con este esquema, se elaboran URN y URL válidos, ambos URI[2], en caso concreto de identificas un AAS es preferible la URL para permitir una mejor funcionalidad en servicios REST porque a su vez, puede vincularse a los propios identificativos.

Identificador	Descripción	Clase de propiedad	Ejemplos
ID propiedad o parámetro	Identificación de los tipos de parámetros y de estado.	Específico del dominio	`http://www.hostname.es/SG2/aas/1/1/modelMK1/maxtemp`
ID propiedad o parámetro	Identificación de instancia de la propiedad, y el estado del parámetro.	Específico del dominio	`http://www.hostname.es/SG2/aas/1/1/modelMK1/maxtemp#0002`

Tabla A.3: Ejemplo de URLs basados en identificadores del Administration Shell

[2]Indetificador Uniforme de Recursos

I Modelo de información y meta-modelo de AAS

Ahora surge la pregunta de cómo se armoniza toda la información respecto al AAS, en muchos implementaciones los atributos (assetId y kind), así como el identificador global están presente para todos los AAS. Sin embargo, sin un estándar, no está claro su semántica ni si son obligatorios o específicos para un activo (tipo o instancia). Para ello, la plataforma Industria 4.0 define un meta-modelo para especificar atributos obligatorios y opcionales para todos los AAS. A partir de aplicar lenguajes de modelado, siendo el favorito el UML, se consigue modelar el AAS y lograr una comprensión global. A continuación se muestra el meta-modelo referente a un activo.

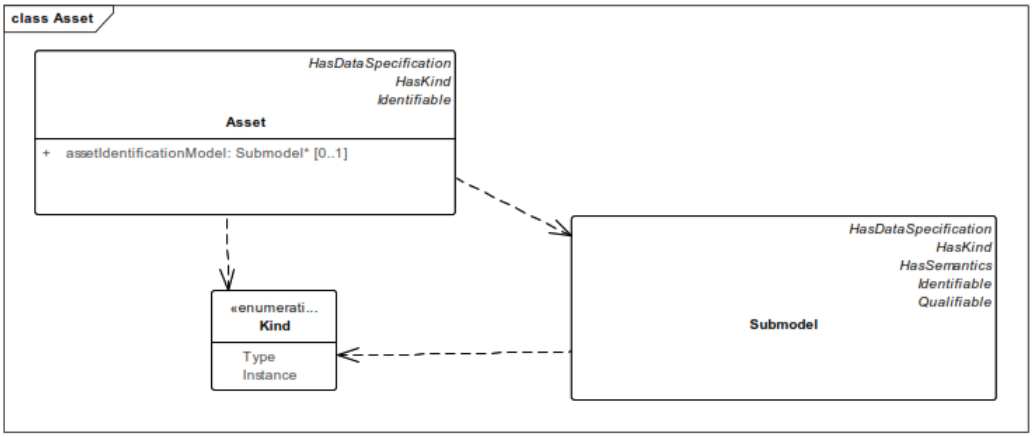


Figura A.5: Meta-modelo de un clase activo[5]

Un AAS contiene una descripción mínima, pero suficiente y fiable de información modificable del activo durante el ciclo de vida, por consiguiente, es importante disponer de una definición genérica o consensuada mediante submodelos genéricos o específicos. La tendencia actual es utilizar submodelos genéricos[**?**] y en caso de alguna propiedad del activo que no esté normalizada se especifica en el submodelo que describe dicha propiedad. De este modo, se definen conceptos básicos para el modelo de información, que se pueden asignar a meta-modelos específicos de "tecnología adyacentes" como por ejemplo OPC UA Meta Model.

En conclusión, es fundamental tener un meta-modelo y los tipos de submodelos bien definidos (*submodel asset, submodel element types, submodel element attributes* etc..) y al combinar los datos del modelo con los métodos de comunicación compatibles con Industria 4.0 se contribuye un AAS con los requisitos mínimos expuestos en apéndice G. El meta-modelo AAS que la plataforma 4.0 propone:

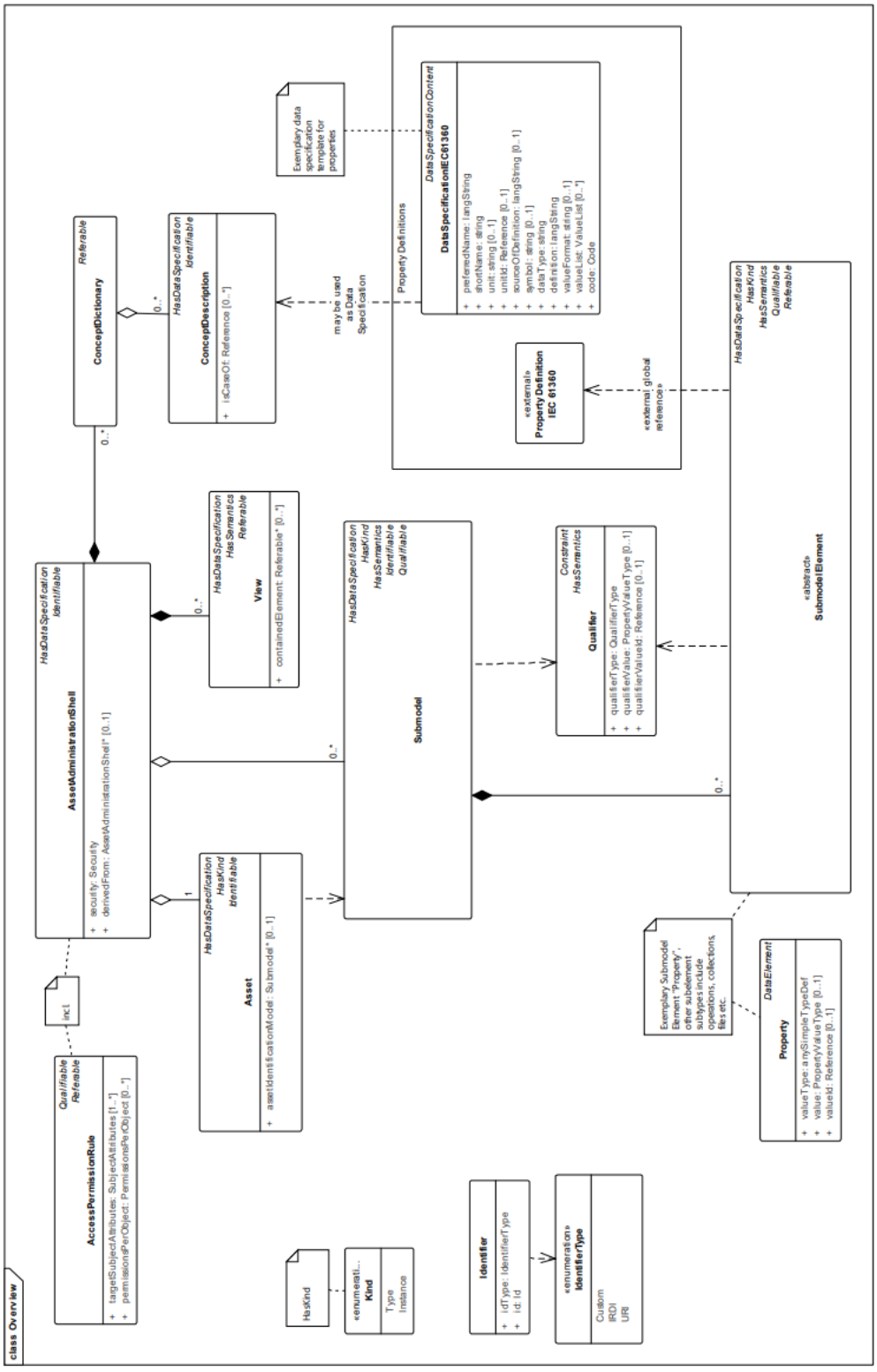


Figura A.6: Meta-modelo de un Asset Administration Shell[5]

J Representación y descripción de un activo

No hay un único lenguaje "estándar" I4.0, en la plataforma I4.0 se describe el término lenguaje 4.0, se usa como un término genérico sobre todos los aspectos que engloba la comprensión mutua entre componentes de Industria 4.0. El objetivo es generalizar para conseguir una semántica común y en consecuencia, obtener cierta interoperabilidad. Para lograr esta meta, se utilizan estándares eCl@ss o IEC 61360 para especificar varios conceptos. En ocasiones esto no es posible, porque no existen definiciones para todo, en ese caso, se debe especificar, por ejemplo, haciendo referencia a datos del fabricante. En esta sección se indaga en el vocabulario del lenguaje que implican términos como anotaciones de elementos de datos, descripciones, sintaxis, las características del activo y sus propiedades.

Si se quiere describir un activo se puede hacer a través de sus características y relaciones. En el caso de un sensor de temperatura, tiene un parámetro de medición, un fabricante, un valor máximo etc...

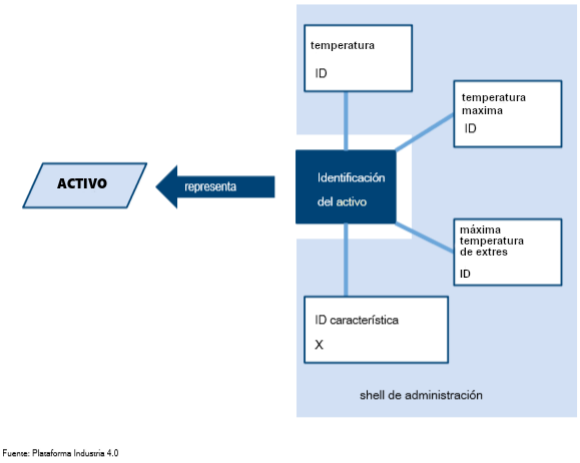


Figura A.7: Representación de un activo[9]

Estas propiedades deben ser legibles por máquinas y por tanto, requiere de anotaciones con contenidos y formatos claros. Estas propiedades disponibles en formato digital se denominan características. Deben estar descritas de manera estandarizada y para ello se deben describir bajo un atributo (metadato):

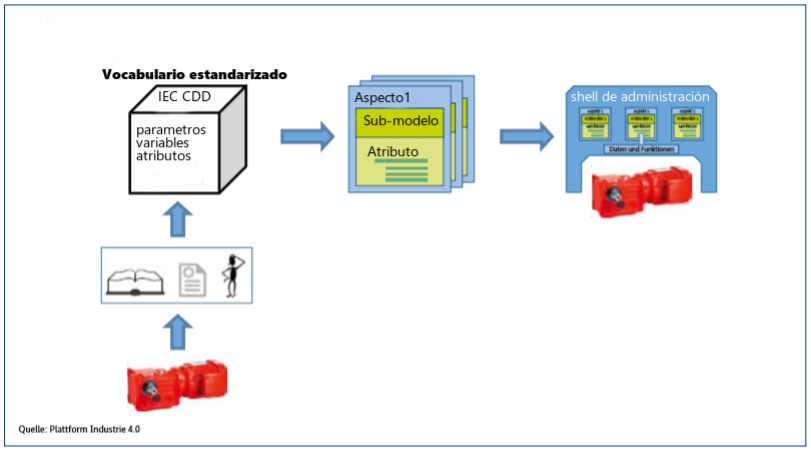
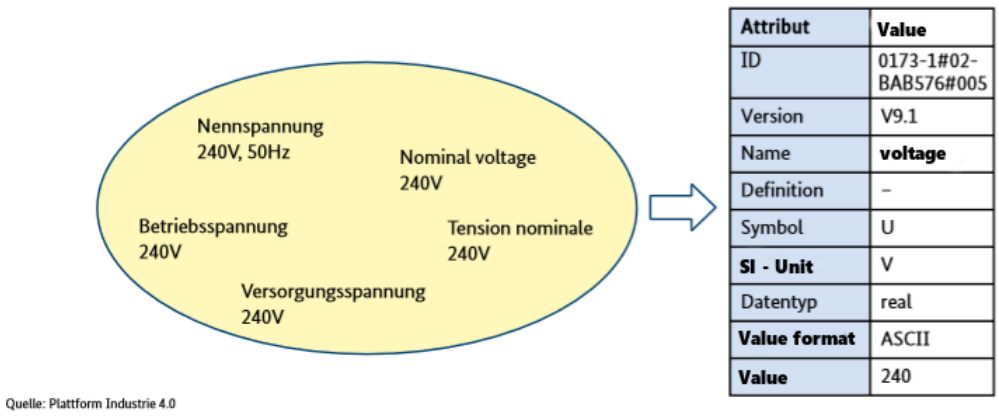


Figura A.8: Ejemplificación de traducción al vocabulario estandarizado[9]

Los atributos se definen en la norma IEC 61360 o eCl@ss, donde el tipo de producto es la clase y las características están representadas por propiedades. Estas propiedades o atributos, es importante que estén estandarizadas, se pueden encontrar en repositorios como IEC CDD[3] bajo IEC 61360, donde existen clases y definiciones de metadatos con identificación única global bajo la norma ISO 29002-5. En un desarrollo para Indus-



Attribut	Value
ID	0173-1#02-BAB576#005
Version	V9.1
Name	**voltage**
Definition	–
Symbol	U
SI - Unit	V
Datentyp	real
Value format	ASCII
Value	240

Figura A.9: Propiedades a un atributo estandard

tria 4.0, para caracterizar una propiedad de un activo siguiendo la norma ISO 29002, se realiza por medio de una búsqueda en el diccionario IEC CDD[4]. Si es encontrada, en su descripción existirá una identificación global referente a la propiedad. En caso de no encontrarse, existe la opción de definir URI/URL que utiliza la propia empresa o URIs específicos del propio fabricante. Por ejemplo, en caso de representar un sensor de temperatura, en consecuencia, se indaga en el diccionario IEC CDD, donde existe la referencia "Electric/electronic components (IEC 61360-4)"[5] con la definición de la clase, que representa el tipo.

[3] IEC "*Common Data Dictionary*"
[4] https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet
[5] https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TU0/0112-2---61360_4%23AAA110

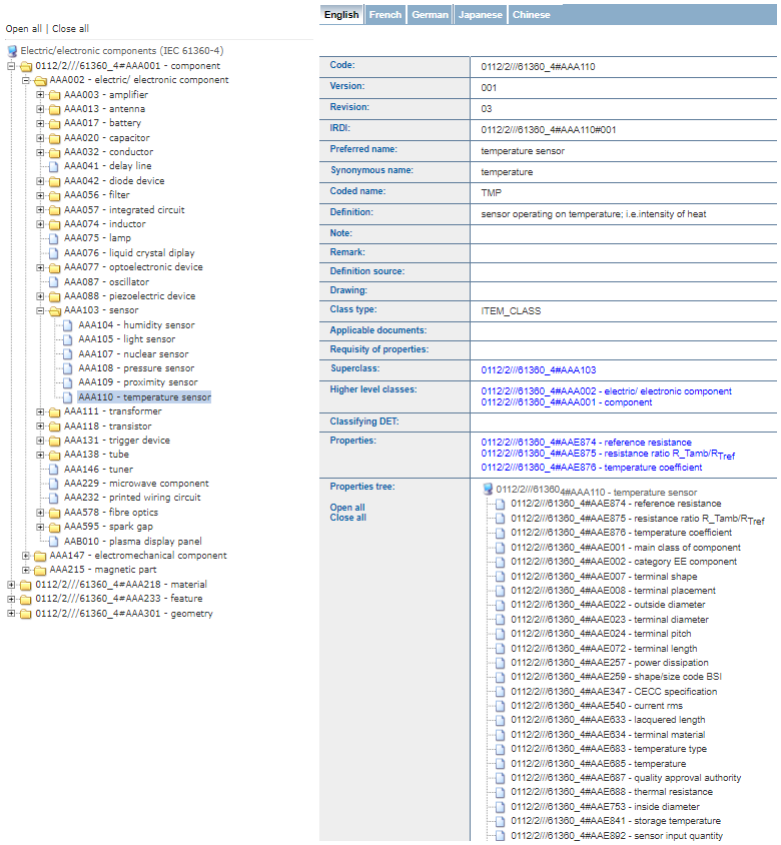


Imagen A.10: Clase sensor de temperatura IEC CDD

Code:	0112/2///61360_4#AAE685
Version:	001
Revision:	05
IRDI:	0112/2///61360_4#AAE685#001
Preferred name:	temperature
Synonymous name:	
Symbol:	@T
Synonymous symbol:	
Short name:	@T
Definition:	temperature of a component, or its environment, as a variable
Note:	
Remark:	This data element to be used in combination with AAE683-005.
Primary unit:	°C
Alternative units:	
Level:	
Data type:	INT_MEASURE_TYPE
Format:	NR1 S..4
Definition source:	
Value source:	
Property data element type:	CONDITION_P_DET
Drawing:	
Formula:	
Value list code:	
Value list:	
DET class:	H02
Applicable classes:	0112/2///61360_4#AAA001 - component
Definition class:	0112/2///61360_4#AAA001
Code for unit:	0112/2///62720#UAA033 - degree Celsius
Codes for alternative units:	
Code for unit list:	
Status level:	Standard
Published in:	IEC 61360-4
Published by:	IEC
Proposal date:	2015-08-17
Version initiation date:	1996-08-01
Version release date:	2011-08-31
Revision release date:	2015-08-17
Obsolete date:	
Responsible Committee:	SC3D
Conditions:	-
Change request ID:	
⊞ Version history:	001-05 (2015-08-17 15:40:34)

Imagen A.11: Propiedad temperatura del sensor de temperatura[6]

En la imagen A.10, se observa un árbol de propiedades, disponibles para la clase "*temperature sensor*". Entre ellos, si seleccionamos la propiedad "*temperature*", apareciendo como resultado la información de la imagen A.11. La descripción contiene una ID global (IRDI) y debe aparecer al realizar un sub-modelo y una clase. Además para validar el tipo de dato (*Data Type*) existe un esquema iec6130.xsd con las definiciones.

En resumen, es importante seguir lo más aproximado posible el modelo IEC CDD para cumplir con una de las intenciones de Industria 4.0, proporcionar un significado proveniente de los valores de datos, a partir de referencias a sus definiciones en diccionarios, con el fin de lograr la representación de activo de forma normalizada, mediante una semántica común.

K Comunicación

Debe ser posible compartir información entre componentes 4.0, incluso entre AAS dentro de uno mismo. OPC UA es elegido por Industria 4.0 como protocolo de comunicación favoreciendo la arquitectura orientada a servicios como SOA.

El protocolo está normalizado internacionalmente como IEC 62541 y es aceptado para ser una parte del modelo de arquitectura de referencia para Industria 4.0 (DIN SPEC 91345). Algunas de sus principales ventajas son el análisis eficiente de los mensajes, incorporación mecanismos de seguridad de autenticación y confidencialidad y interopabilidad, debido a que, proporciona facilidades para mapear meta-modelo utilizado (AAS o activo) a un meta-modelo OPC UA[7], este incorpora un modelo de información basado en objetos para el acceso simplificado de los datos.
El meta-modelo de información describe una serie de bases y reglas encima de los que crear los modelos que se almacenaran y podrían ser examinados dentro de un servidor OPC UA.
El servicio OPC UA en sí, son interfaces con un punto de acceso para la gestión y visualización de los modelos almacenados en el servidor, permitiendo a los clientes acceder al modelo de información almacenado (función base de UA*Unified Architecture*) En la capa de transporte, se serializa/deserializar los dato y trasmiten a través de la red. OPC UA define diferentes mecanismos, un protocolo binario (`opc.tcp://server`) y otro para el servicio web (`http://servers`).

El binario ofrece mejor rendimiento y menos sobrecarga, requiriendo menos recursos, ofreciendo mayor interoperabilidad y utilizando un puerto tcp, elegible aleatoriamente para facilitar la comunicación por túneles o activación por firewall. En cambio, el servicio web (SOAP) se administra mejor en entornos como Java o .Net, utilizando los puertos HTTP(s) estándar.

Appendix K.1. Especificaciones

El estándar OPC UA organiza en conjuntos según la tipología y relevancia, siendo las especificaciones 4 y 3 de la Figura A.12 las más importantes del estándar.

[6] `https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TU0/0112-2---61360_4%23AAA110`
[7] `https://opcfoundation.org/developer-tools/specifications-unified-architecture/isa-95-common-object-model`

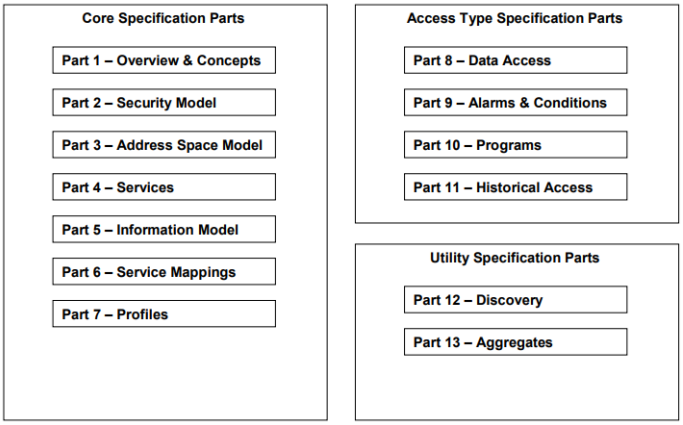


Figura A.12: Especificaciones OPC UA[11]

Appendix K.2. Moldeo de información OPC UA

El estándar OPC UA aloja la meta-información de los datos depositados en el servidor. Un cliente OPC UA puede acceder a la información para interpretar los datos manejados por activos industriales, obtenido una información semántica de los datos. Favoreciendo una interpretación común de los modelos de información, la fundación OPC proporciona modelos para la definición de tipos de dispositivos[8] y su información en OPC UA[10].

En el modelo de información, los nodos son el elemento básico, sus atributos en la Tabla A.4. Cuando un cliente establece conexión con un servidor OPC UA, busca en el espacio de nombres (lista de NodeIDs) el apropiado y se conecta acede a los servicios que el servidor dispone, interactuando con los atributos y funciones que el nodo (tipo NodeObject) contiene. Tipos de nodos especificados en la Figura y

[8] https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-100-device-information-model/

Atributo	Tipo de dato	mDefinición
NodeId	*NodeId*	mIdentifición única de un nodo en un servidor OPC UA.
NodeClass	*NodeClass*	mIdentifición de clase del nodo.
BrowseName	*QualifiedName*	mIdentificación del nodo en la navegadción del servidor OPC UA.
DisplayName	*LocalizedText*	mAlmacena el nombre del nodo que interfaz cliente debe utlizar.
Description	*LocalizedText*	mDescripción del nodo
WriteMask	*UInt32*	mIndica que atributos pueden ser modicables por el cliente OPC UA

Tabla A.4: Atributos de un NodeObject

A conituación se hace uso OPC UA Web Client[9] para conectar con unos nodos disponibles actualmente:

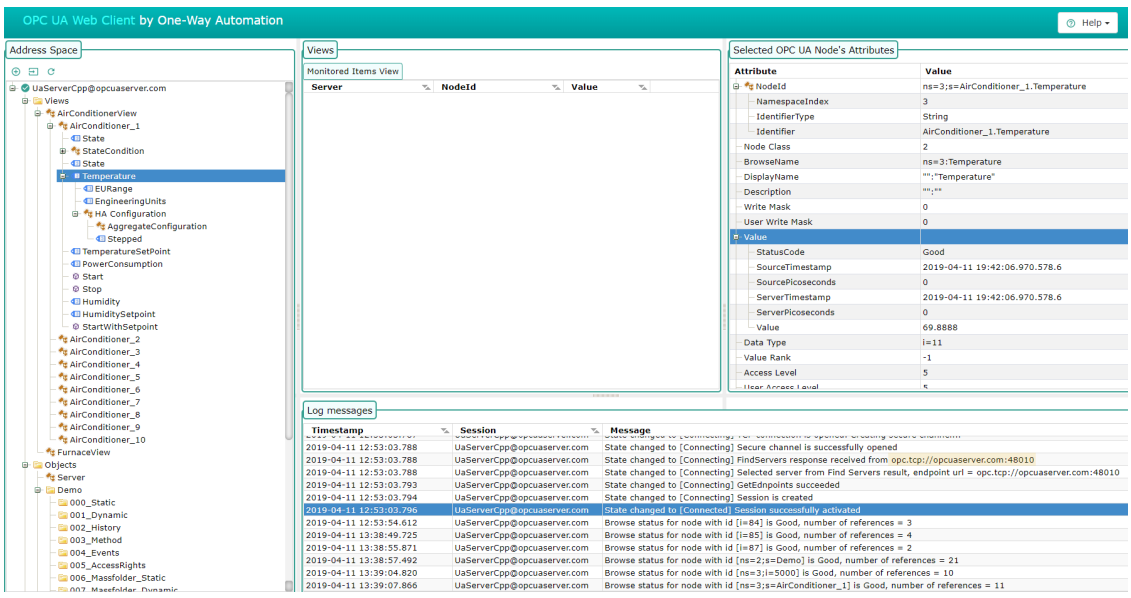


Figura A.13: Acceso a un servidor OPC UA con OPC UA Web Client

Además se puede añadir una monitorización de algún nodo como por ejemplo la temperatura.

[9]OPC UA Web Client https://uaclient.com/

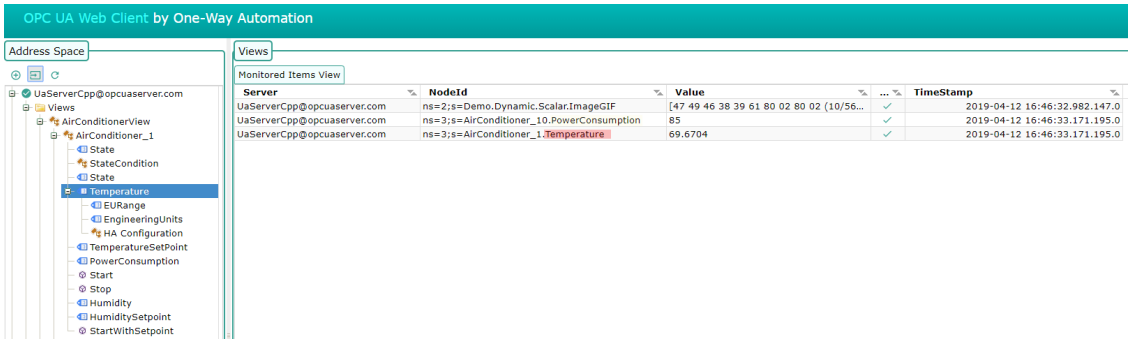


Figura A.14: Monitorización de un Nodo

Todos estos datos son mostrados de forma segura y en tiempo real, gracias al protocolo OPC UA. Para tener una buena representación de nuestro modelo es importante tener un buen mapeado de nuestro objeto o modelo en el meta-modelo de OPC UA[10]. Este protocolo deja a disposición algunos modelos genéricos para favorecer el uso de una semántica común. Entre estos se encuentra el modelo genérico del tipo de dato:

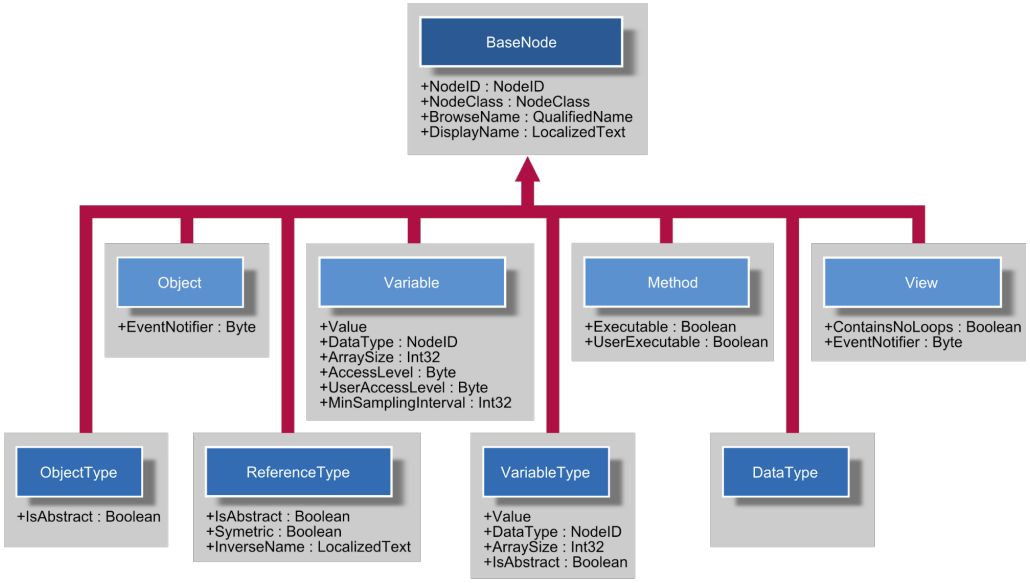


Figura A.15: Modelo Data Type[11]

Appendix K.3. OPC UA Industria 4.0

Existen nodos predefinidos, pero OPC UA contempla también la posibilidad de crear nuevos tipos de clases de nodos. La base para definir el tipo de dato es *DataTypes*. La fundación OPC proporciona una serie de esquemas[12] donde se representa el modelo específico, por ejemplo, el modelo anterior de tipos de datos[13].

Muy importante esta característica para la Industria 4.0, por que, se pueden definir los modelos creados bajo el paradigma de Inudstria 4.0 en nuevas clases para OPC UA, permitiendo un compatibilidad entre diferentes modelos y disponer de una estructura AAS bajo un servido OPC UA.

[10] Information Models https://opcfoundation.org/developer-tools/specifications-opc-ua-information-models
[11] http://www.ascolab.com/en/technology-unified-architecture/meta-model.html
[12] Esquemas OPC UA https://opcfoundation.org/UA/schemas/1.04/
[13] Esquema tipo de dato OPC UA https://opcfoundation.org/UA/schemas/1.04/Opc.Ua.Types.bsd

L Mappings

Debería ser posible compartir información compatible con I4.0 entre diferentes sistemas en toda el área cubierta por el modelo RAMI4.0. En la siguiente tabla se describen los diferentes formatos que puede albergar un AAS:

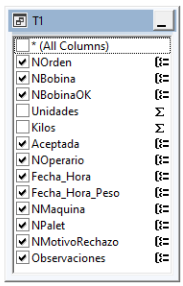
Formato de datos	**Propósito / motivación**
OPC UA	Acceso a la información administrativa y el intercambio de datos con otras operaciones de producción.Acceso para sistemas de fábrica de nivel superior a esta información.
AutomationML(UML)	Intercambio de información entre tipos e instancias de activos.
XML,JSON	Serialización de información para la comunicación.
RDF	Mapeo de esta información para permitir el uso completo de las ventajas de las tecnologías semánticas.

Tabla A.5: Formato de datos

Para permitir la importación y exportación, el el meta-modelo de un *Asset Administration Shell* necesita ser serializado. Uno de estos formatos de serialización es XML, que junto a JSON, proporcionan la posibilidad de validar sintácticamente nuestras definiciones (assets,descripciones,propiedades..) con ayuda de esquemas (formato XSD).

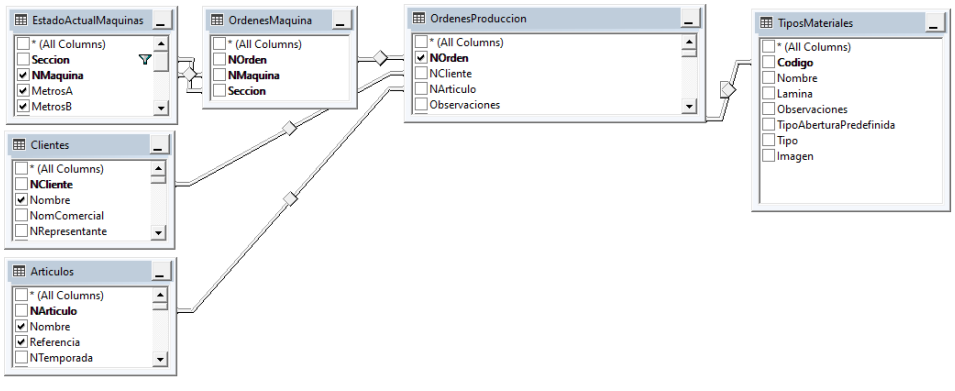
APÉNDICE B
Imágenes Desarollo

A Vistas SQL




```
SELECT      NOrden, NBobina, NBobinaOK, SUM(Unidades) AS Unidades, SUM(Kilos) AS Kilos, Aceptada, NOperario, Fecha_Hora, Fecha_Hora_Peso, NMaquina, NPalet, NMotivoRechazo, Observaciones
FROM        (SELECT      NOrden, NBobina, NBobinaOK, Unidades, Kilos, Aceptada, NOperario, Fecha_Hora, Fecha_Hora_Peso, NMaquina, NPalet, NMotivoRechazo, CAST(Observaciones AS varchar(1000)) AS Observaciones
             FROM        dbo.BobinasCorte
             UNION
             SELECT      NOrden, NBobina, NBobinaOK, Unidades, Kilos, Aceptada, NOperario, Fecha_Hora, Fecha_Hora_Peso, NMaquina, NPalet, NMotivoRechazo, CAST(Observaciones AS varchar(1000)) AS Observaciones
             FROM        dbo.BobinasPrecorte) AS T1
GROUP BY NOrden, NBobina, NBobinaOK, Aceptada, NOperario, Fecha_Hora, Fecha_Hora_Peso, NMaquina, NPalet, NMotivoRechazo, Observaciones
```

Imagen B.1: Vista Cortado Extrusión




```
SELECT   dbo.EstadoActualMaquinas.NMaquina, dbo.EstadoActualMaquinas.MetrosA, dbo.EstadoActualMaquinas.MetrosB, dbo.EstadoActualMaquinas.MarchaA, dbo.EstadoActualMaquinas.MarchaB,
         dbo.EstadoActualMaquinas.Velocidad,
         dbo.OrdenesProduccion.AnchoExtrusion * dbo.OrdenesProduccion.GalgaTubo * dbo.EstadoActualMaquinas.MetrosA * dbo.OrdenesProduccion.Densidad / CASE Lamina WHEN 1 THEN 2000000.0 ELSE 1000000.0 END AS KgA,
         dbo.OrdenesProduccion.AnchoExtrusion * dbo.OrdenesProduccion.GalgaTubo * dbo.EstadoActualMaquinas.MetrosB * dbo.OrdenesProduccion.Densidad / CASE Lamina WHEN 1 THEN 2000000.0 ELSE 1000000.0 END AS KgB,
         dbo.OrdenesProduccion.AnchoExtrusion * dbo.OrdenesProduccion.GalgaTubo * dbo.EstadoActualMaquinas.MetrosA * dbo.OrdenesProduccion.Densidad / CASE Lamina WHEN 1 THEN 2000000.0 ELSE 1000000.0 END
          + dbo.OrdenesProduccion.AnchoExtrusion
         * dbo.OrdenesProduccion.GalgaTubo * dbo.EstadoActualMaquinas.MetrosB * dbo.OrdenesProduccion.Densidad / CASE Lamina WHEN 1 THEN 2000000.0 ELSE 1000000.0 END AS KgTotal, dbo.Articulos.Referencia,
         dbo.Articulos.Nombre AS Articulo, dbo.Clientes.Nombre AS Cliente, dbo.OrdenesProduccion.NOrden
FROM     dbo.EstadoActualMaquinas INNER JOIN
         dbo.OrdenesMaquina ON dbo.EstadoActualMaquinas.NMaquina = dbo.OrdenesMaquina.NMaquina AND dbo.EstadoActualMaquinas.Seccion = dbo.OrdenesMaquina.Seccion INNER JOIN
         dbo.OrdenesProduccion ON dbo.OrdenesMaquina.NOrden = dbo.OrdenesProduccion.NOrden INNER JOIN
         dbo.Articulos ON dbo.OrdenesProduccion.NArticulo = dbo.Articulos.NArticulo INNER JOIN
         dbo.Clientes ON dbo.OrdenesProduccion.NCliente = dbo.Clientes.NCliente LEFT OUTER JOIN
         dbo.TiposMateriales ON dbo.OrdenesProduccion.TipoMaterialExtrusion = dbo.TiposMateriales.Codigo
WHERE    (dbo.EstadoActualMaquinas.Seccion = 1)
```

Imagen B.2: Vista Estado Actual Extrusión

APÉNDICE C
Fragmentos de código

A Fragmento clase Controller

```
namespace WebSite.Controllers
{
    public class MaquinasController : ApiController
    {

        [HttpGet]
        public IEnumerable<Maquinas> Get()
        {
            return MaquinasService._GetMaquinas();
        }



        // GET api/<controller>/5
        public string Get(int id)
        {
            return "value";
        }

        // POST api/<controller>
        public void Post([FromBody]string value)
        {
        }

        // PUT api/<controller>/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE api/<controller>/5
        public void Delete(int id)
        {
        }

    }
}
```

Código C.1: Clase ejemplo tipo Controlador

B Ficheros XML/JSON

Appendix B.1. Ficheros Web API y Web Page

```xml
<?xml version="1.0" standalone="yes"?>
<dsDatosLocales xmlns="http://tempuri.org/dsDatosLocales.xsd">
    <DireccionServicioRest>http://37.111.111.111:8099/</DireccionServicioRest>
    <Proyecto>EXTRUSION</Proyecto>
    <ConexionSQL>
        <Servidor>37.111.111.111,1067</Servidor>
        <Base>Plastisoft_Plastigraf_Extrusion</Base>
        <Usuario>anonimo</Usuario>
        <Contrasea>d.password</Contrasea>
        <AutentificacionWindows>false</AutentificacionWindows>
    </ConexionSQL>
    <Empresa>
        <Empresa>01</Empresa>
    </Empresa>
    <ConexionSQLRemota>
        <Servidor>37.111.111.111,1067</Servidor>
        <Base>Plastisoft_Plastigraf_Extrusion</Base>
        <Usuario>anonimo</Usuario>
        <Contrasea>d.password</Contrasea>
        <AutentificacionWindows>false</AutentificacionWindows>
    </ConexionSQLRemota>
</dsDatosLocales>
```

Código C.2: Fichero DatosLocales.xml

C Ejemplos AAS

Appendix C.1. Ejemplo 2

```
const opcua = require("node-opcua");
// Crea el objeto AAS, su variable y una pequea simulacion de su actividad
function construct_address_space(server) {
    const addressSpace = server.engine.addressSpace;
    const namespace = addressSpace.getOwnNamespace();
    //OBJECT
    const kmt1 = namespace.addObject({
        browseName: "MKM1",
        organizedBy: addressSpace.rootFolder.objects
    });
    //VARIBALE
    const kmt1Temperature = namespace.addAnalogDataItem({
        browseName: "temperature1",
        engineeringUnitsRange: {low: -55,high: 50.0},
        engineeringUnits: opcua.standardUnits.degree_celsius,
        componentOf: kmt1
    });
    addressSpace.installHistoricalDataNode(kmt1Temperature);
    let t = 10;
    setInterval(function () { // simulacion de cambio de temperatura
        let value = (Math.sin(t / 50) * 1.90 + Math.random() * 2.80) * 6.0 + 5.0;
        kmt1Temperature.setValueFromSource({ dataType: "Double", value: value });
        t = t + 1;
    }, 200);
};
(async () => {
    const opcua = require("node-opcua");
    try {
        const server = new opcua.OPCUAServer({
            port: 26544,
            resourcePath: "/ua/resource", //se agregar al nombre del recurso en el
                endpoint
            nodeset_filename: [opcua.nodesets.standard_nodeset_file ]
        });

        await server.initialize();

        construct_address_space(server);

        await server.start();
        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
        console.log(" the primary server endpoint url is ", endpointUrl);
    } catch (err) {
        console.log("Error = ", error);
    }
})();
```

Código C.3: Crear objeto y servidor OPC UA

Appendix C.2. Ejemplo 3

```
var opcua = require("node-opcua");
require("node-opcua-coreaas")(opcua);
var xmlFiles = [opcua.standard_nodeset_file,opcua.coreaas.nodeset_file];
var server = new opcua.OPCUAServer({nodeset_filename: xmlFiles,});

    // Se instancian los elementos de la clase opcua.coreaas
```

```
7
8       // Crear un objeto AAS representa ISTR-MKM1-3ENG10 Communication Controller
9       const myaas = addressSpace.addAssetAdministrationShell({
10          browseName: "AAS Temperature",
11          description: [new opcua.LocalizedText({
12              locale: "en", text: "MKM1 CONTROLLER AND MINI-PROGRAMMER" }),
13              new opcua.LocalizedText({ locale: "es", text: "CONTROLADOR MKM1" })],
14          identification: new Identifier({
15              id: "www.admin-shell.io/aas-sample/1.0",
16              idType: IdentifierType.URI
17          }),
18          derivedFromRef: [new Key({
19              idType: KeyType.IRDI,
20              local: false,
21              type: KeyElements.AssetAdministrationShell,
22              value: "AAA#1234-454#123456789"
23          })],
24          assetRef: [new Key({
25              idType: KeyType.URI,
26              local: true,
27              type: KeyElements.Asset,
28              value: "https://www.ascontecnologic.com/es/automatizacion-industrial/
                    controladores-industriales/serie-kube/km1"
29          })]
30      }).addSubmodelRef([new Key({
31              idType: KeyType.URI,
32              local: true,
33              type: KeyElements.Submodel,
34              value: "./models/coreaas"
35          })]);
36      /**
37       * Aadir un Asset
38       */
39      let asset = addressSpace.addAsset({
40          browseName: "MKM1",
41          idShort: "MKM1",
42          identification: new Identifier({
43              id: "https://www.ascontecnologic.com/es/automatizacion-industrial/
                    controladores-industriales/serie-kube/km1",
44              idType: IdentifierType.URI
45          }),
46          kind: Kind.Instance,
47          description: new opcua.LocalizedText({ locale: "en", text: "MKM1 controller and
                 programmer temperature" }),
48          assetIdentificationModelRef: [new Key({
49              idType: KeyType.URI,
50              local: false,
51              type: KeyElements.SubmodelElement,
52              value: "//submodels/identification_KTY81"
53          })]
54      });
55      myaas.hasAsset(asset);
56      /*** Aadir un Submodel Tipo */
57      const submodel_type = addressSpace.addSubmodel({
58          browseName: "TYPEMKM1",
59          kind: Kind.Type,
60          idShort: "TYPEMKM1",
61          identification: new Identifier({
62              id: "//models/submodels/TYPEMKM1",
63              idType: IdentifierType.URI
64          })
65      })
66  /*** Aadir un Submodel Instacnia */
67      const mysubmodel = addressSpace.addSubmodel({
```

```
        browseName: "MKM1AA",
        kind: Kind.Instance,
        idShort: "MKM1AA",
        identification: new Identifier({
            id: "//models/submodels/MKM1AA",
            idType: IdentifierType.URI
        })
    }).submodelOf(myaas)
        .hasSubmodelSemantic(submodel_type)
        .addParent([new Key({
            idType: KeyType.URI,
            local: true,
            type: KeyElements.AssetAdministrationShell,
            value: "www.admin-shell.io/aas-sample/1.0"
        })]);
    /*** Aadir SubmodelElementCollection*/
    let collection = addressSpace.addSubmodelElementCollection({
        idShort: "Measurement",
        submodelElementOf: mysubmodel,
        ordered: true,
        kind: Kind.Instance
    });
    /*** Aadir SubmodelProperty*/
    const temperature = addressSpace.addSubmodelProperty({
        browseName: "TEMPERATURE",
        idShort: "TEMPERATURE",
        category: PropertyCategory.VARIABLE,
        valueType: PropertyValueType.Int,
        value: {
            dataType: "Int16",
            value: {
                get: () => {
                    return new opcua.Variant({ dataType: opcua.DataType.Int16, value:
                        kmt1Temperature });// random value
                }
            }
        }
    }).addSemanticId([new Key({
            idType: KeyType.IRDI,
            local: true,
            type: KeyElements.ConceptDescription,
            value: "0112/2///61360_4#AAE685#001"
        })])
        .addParent(new Key({idType: KeyType.idShort,
            local: true,
            type: KeyElements.SubmodelElementCollection,
            value: "Measurement"
        }))
        .addValueId([new Key({
            idType: KeyType.URI,
            local: true,
            type: KeyElements.GlobalReference,
            value: "//value/of/temperature"
        })]);

    const tempmax = addressSpace.addSubmodelProperty({
        browseName: "TEMPMAX",
        idShort: "TEMPMAX",
        semanticId: [new Key({
            idType: KeyType.IRDI,
            local: true,
            type: KeyElements.ConceptDescription,
            value: "0112/2///61360_4#AAF277#002"
        })],
```

```
131         category: PropertyCategory.PARAMETER,
132         valueType: PropertyValueType.Int,
133         value: {
134             dataType: "Int16",
135             value: {
136                 get: () => {
137                     return new opcua.Variant({ dataType: opcua.DataType.Int16, value: 55
                            });
138                 }
139             }
140         }
141     });
142     collection.addElements([tempmax, temperature]);
143     /*** Aadir File*/
144     addressSpace.addAASFile({
145         idShort: "KTM1_Documentation",
146         kind: Kind.Instance,
147         description: [new opcua.LocalizedText({ locale: "en", text: "Documentation for
                the KTM1 Controller." }),
148             new opcua.LocalizedText({ locale: "it", text: "Documentazione per il
                Controller KTM1" })],
149         submodelElementOf: mysubmodel,
150         value: "./aas/files/KTM1/documentation",
151         mimeType: "application/pdf"
152     });
153     /** * Aadir Concept Dictionary*/
154     const conceptDictionary = addressSpace.addConceptDictionary({
155         browseName: "ConceptDict_1",
156         idShort: "ConceptDictionary_1",
157         conceptDictionaryOf: myaas,
158         description: [new opcua.LocalizedText({ locale: "en", text: "Dicitonary for the
                temperature controller ." }),
159         new opcua.LocalizedText({ locale: "it", text: "Diccionario para el controlador
                de temperatura" })]
160     }).addConceptDescriptionRef([
161             new Key({
162                 idType: KeyType.IRDI,
163                 local: true,
164                 type: KeyElements.ConceptDescription,
165                 value: "0112/2///61360_4#AAE685#001"
166             })
167         ]).addConceptDescriptionRef([
168             new Key({
169                 idType: KeyType.IRDI,
170                 local: true,
171                 type: KeyElements.ConceptDescription,
172                 value: "0112/2///61360_4#AAF277#002"
173             })
174         ]);
175     /** Aadir ConceptDescription con EmbeddedDataSpecification*/
176     const embedded_1 = addressSpace.addEmbeddedDataSpecification({
177         browseName: "EmbeddedDS_1",
178         hasDataSpecification: [new Key({
179             idType: KeyType.URI,
180             local: false,
181             type: KeyElements.GlobalReference,
182             value: "www.admin-shell.io/DataSpecificationTemplates/
                    DataSpecificationIEC61360"
183         })],
184     }).addDataSpecificationIEC61360({ // Tipo definido por el estandard iec61360
185        identifier: "0112/2///61360_4\#AAE685",
186         preferredName: "temperature",
187         definition: "temperature of a component, or its environment, as a variable",
188         dataType: "INT_MEASURE_TYPE",
```

```
189         unit: "C",
190         shortName: "T",
191         valueFormat: "NR1 S..4",
192         unitId: [new Key({
193             idType: KeyType.IRDI,
194             local: false,
195             type: KeyElements.GlobalReference,
196             value: '0112/2///62720\#UAA033\#001' // Preferred name: degree Celsius
197         })],
198     });
199 }
200     /**
201      * Iniciar OPC UA Server
202      */
203     server.start(function () {
204         var endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
205         console.log(" the primary server endpoint url is ", endpointUrl);
206     });
207     server.initialize(post_initialize);
```

Código C.4: Crear Objeto AAS para OPC UA

APÉNDICE D
Referencias

1. **ASP.NET Web Forms**
   Consulta a https://dotnet.microsoft.com/apps/aspnet/web-forms.
2. **C Sharp**
   Consulta a https://docs.microsoft.com/en-us/dotnet/csharp/index.
3. **NodeJS**
   Consulta a https://nodejs.org/es/.
4. **Plataform Industrie 4.0**
   Consulta a https://www.plattform-i40.de/PI40/Navigation/EN/Home/home.html.
5. **NodeJS**
   Consulta a https://nodejs.org/es/.
6. **NPM**
   Consulta a https://www.npmjs.com/.
7. **RaspberryPI**
   Consulta a https://dotnet.microsoft.com/apps/xamarin.
8. **Xamarin**
   Consulta a https://www.une.org/
9. **UNE**
   Consulta a https://www.une.org/