Universidad Rey Juan Carlos

Ingeniería del Software

Diseño y Arquitectura del Software

Evaluación de Atributos de Calidad

Stefano Tomasini Hoefner - Redactor de escenarios
Shuheng Ye - Redactor de escenarios
Carlos Solsona Álvarez - Cuestionador
Jonathan Xavier Medina Salas - Cliente

s.tomasini.2019@alumnos.urjc.es

20 de diciembre de 2023

Índice

Índice de figuras

Índice de cuadros

0. Asignación de roles

Los estudiantes tomaron los siguientes roles:

Estudiante	**Rol**
Stefano Tomasini Hoefner	Redactor de escenarios
Shuheng Ye	Redactor de escenarios
Carlos Solsona Álvarez	Cuestionador
Jonathan Xavier Medina Salas	Cliente

Tabla 1: Asignación de roles

1. Presentación del método ATAM

El equipo tuvo una primera reunión con el cliente con el objetivo de explicar y justificar el uso del método ATAM (*Architecture Tradeoff Analysis Method*). Se destacó que es útil para evaluar cuantitativamente el grado de satisfacción de los deseos de calidad que se consideran más prioritarios por parte del cliente para la arquitectura diseñada.

También se le explicó que ATAM facilita la identificación temprana de posibles problemas arquitectónicos y que su estructura modular permite abordar de manera eficiente diversas preocupaciones, desde la escalabilidad hasta la seguridad.

Tras esta explicación, el cliente transmitió la duda de cómo concretamente se conseguirían estas mejoras en el software al seguir la metodología que se le recomendó. El equipo ATAM le hizo saber al cliente que esto se hace a través del estudio de atributos de calidad. El cliente preguntó qué es un atributo de calidad exactamente, a lo que se le respondió que un atributo de calidad sirve para manejar de una forma técnica, entre el equipo ATAM, un deseo de negocio del cliente. Los atributos de calidad en los que el equipo se centra se corresponden con aspectos concretos cuantificables pertinentes a los objetivos de negocio pedidos.

El equipo expuso que, además, la metodología ATAM tiene la utilidad de poder detectar preventivamente conflictos entre varios atributos de calidad que se consideraron prioritarios a maximizar. En este momento el cliente no entendió muy bien cómo pueden entrar en conflicto dos atributos que son distintos.

Se le explicó que en el proceso ATAM, este conflicto se denomina *trade-off* y se le planteó un escenario de la vida real: un cliente quiere que le fabriquen un neumático con mucho agarre y que a su vez le permita al vehículo mantener grandes velocidades. Sin embargo, el vendedor le aclara que el aumento de uno de los dos valores es en detrimento del otro, por lo que el cliente en este caso tiene que decidir cuál aspecto es más importante para él, y hasta qué punto está dispuesto a sacrificar un aspecto para mejorar el otro, teniendo en cuenta las recomendaciones específicas que le de el equipo ATAM al respecto.

A continuación, se le hizo saber al cliente que ATAM es especialmente útil para descubrir los puntos críticos (*sensitivity points*) del sistema que requieren un cuidado especial. El cliente pidió una explicación más extensa de lo que comprende un punto crítico. Se le clarificó que estos puntos son aquellos componentes tales que el sistema tiene una alta o extrema dependencia de ellos, y también, los que son un cuello de botella del sistema. Presentan un alto riesgo para la continuidad operativa del negocio y, por lo tanto, son un importante elemento de estudio. Así que, se le advirtió al cliente que cabe encajar componentes que sean fiables dentro de esos sitios críticos (o de alto rendimiento, en el caso de ser cuellos de botella).

El equipo de ATAM procede a explicar la siguiente fase, la fase de "*brainstorming*". Se le aclara al cliente que en esta fase, el equipo ATAM genera ideas y perspectivas sobre posibles escenarios

que evalúan una parte concreta del sistema. Posteriormente, el equipo entra en la fase de priorizar, seleccionar, y descartar escenarios. En esta, se revisan los escenarios diseñados para decidir cuáles serán tenidos en cuenta para mejorar la arquitectura, y cuáles serán descartados debido a: ser irrelevantes, no estar cuantificados, o no ser comparables.

Al ilustrar el proceso de evaluación, describimos cómo se lleva a cabo una serie de reuniones entre todas las partes. Subrayamos que este enfoque colaborativo fomenta una comprensión holística de la arquitectura propuesta y garantiza que todas las perspectivas relevantes se tengan en cuenta: tanto los deseos del cliente, como lo que los arquitectos software consideran que es viable en la realidad. Intentamos transmitir la idea al cliente de que el seguimiento de esta metodología contribuirá significativamente al éxito de su negocio y a la sostenibilidad a largo plazo del sistema software construido.

2. Definición de las prioridades de negocio con el cliente

El equipo ATAM le comunicó al cliente que meditara sobre sus objetivos de negocio; que reflexione sobre cuáles son sus mayores prioridades para el éxito de su negocio. Tras dicho periodo de reflexión, se dio lugar la reunión entre el equipo ATAM y el cliente para discutir sobre los deseos que le gustaría que el sistema software final potencie o satisfaga.

Para empezar, el cliente nos pidió que el sistema contribuya en el factor de rentabilidad, ya que quería maximizar el beneficio de la factoría frente al costo. En respuesta a esto, el equipo ATAM le explicó que su pedido de que el sistema "favorezca la rentabilidad" es muy ambiguo y que no se puede transformar en un atributo de calidad cuantificable por el equipo. Si no es cuantificable, el equipo no puede realizar sus tareas de definir, comparar, y estudiar los escenarios correspondientes de una forma objetiva y precisa.

Una vez aclarado lo anterior, el cliente exigió un alto grado de **seguridad** en el software para la transmisión de los datos de los sensores, ya que el negocio donde se encuentra tiene mucha competencia y espionaje, y quiere estar seguro sobre la exclusividad de los datos que fluyen por la factoría. Ante este pedido específico, el equipo ATAM le informó que se aceptó priorizar su deseo ya que se corresponde con un atributo de calidad cuantificable: *Security*.

Acto seguido, el cliente exigió una **disponibilidad** casi total para el software ya que la fábrica tiene un funcionamiento diario ininterrumpido. Por esto mismo, una caída de un componente crítico del sistema dejaría inutilizable a la factoría y provocaría una gran pérdida de productividad y de ingresos. El equipo ATAM aceptó tratar esas preocupaciones específicas del cliente, y se dedujo que el atributo de calidad *Availability* es el que corresponde a ser maximizado, el cual es un sub-atributo de *Reliability*[1].

Aparte, pidió un rendimiento excelso que no sufra retrasos ni pausas en el funcionamiento del sistema. El equipo ATAM le respondió que es necesario que especifique más concretamente qué parte del sistema requiere de un alto rendimiento para poder enfocar los esfuerzos de mejora del sistema. El cliente aclaró que no se puede permitir cuellos de botella en el procesamiento de datos del software de la factoría, ya que podría llegar a afectar al número de unidades producidas diariamente. El equipo aceptó finalmente este deseo y lo descompuso en dos atributos de calidad como objetivos de optimización: *Performance* y *Reliability*.

3. Divulgación de la arquitectura al cliente

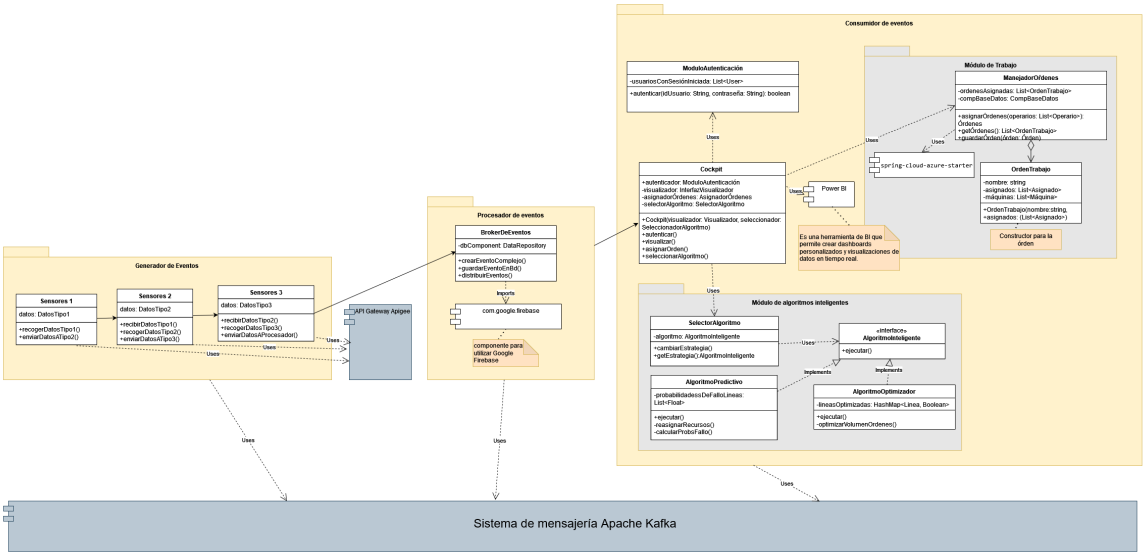

Figura 1: Diagrama de clases y paquetes UML de la arquitectura

El equipo ATAM presentó la arquitectura resultante del sistema al cliente, que sigue el modelo *Event-Driven*. El cliente preguntó qué significa que el sistema sea *Event-Driven*. Se le respondió que *Event-Driven* significa que se define una estructura arquitectónica que esté orientada a tratar un tráfico de eventos, en nuestro caso proveniente de unos sensores. Se le explicó que hay tres paquetes principales que definen a una arquitectura de este estilo: uno de dónde salen los eventos, otro dónde se procesan los eventos, y un último dónde se consumen los eventos (se hace uso de su información).

El primero es el paquete **generador de eventos**, que encapsula a todos los sitios de origen en donde se pueden producir eventos. Estos eventos son capturados por unas familias de sensores y son consiguientemente pasados al siguiente paquete. Para facilitar la gestión de eventos provenientes de diversos sensores se utiliza una API Gateway.

El segundo es el **procesador de eventos**, que se encarga de gestionar los datos provenientes del generador de eventos. Puede crear eventos complejos resultantes del procesado de eventos simples (una combinación de eventos simples específicos producidos al mismo tiempo pueden implicar un evento complejo), para luego ser transmitidos. En este paquete también se encuentra la base de datos NoSQL, que sirve para poder persistir data de los eventos recibidos, si es que nace la necesidad.

Por último, el **consumidor de eventos** es el paquete en donde se extrae y se usa información de los eventos generados. En nuestro diseño incluye diferentes módulos como: un componente de visualización, un módulo de trabajo y otro de algoritmos inteligentes. Todos ellos están conectados al *cockpit*, que es el punto de entrada al sistema para los operarios de la factoría.

Además, para la implementación de este tipo de arquitectura, hace falta instalar un sistema de mensajería que funcione como base de la comunicación transversal entre los paquetes. Así que, se decidió hacer uso del componente de mensajería Apache Kafka.

En relación a los objetivos de calidad expresados por el cliente se procedió a identificar las partes del sistema en las que podían situarse sus deseos de calidad.

El primer pedido del cliente expresaba un deseo de seguridad: evitar que agentes ajenos a la factoría intercepten y lean los datos de los sensores. El equipo ATAM le contestó que esto podría conseguirse mediante la utilización de encriptado. En este caso, la parte fundamental de la arquitectura para lograr este objetivo sería el generador de eventos, que encapsula a los sensores.

Para cumplir el requisito de disponibilidad total, se debe maximizar la disponibilidad de los puntos críticos del sistema, es decir, aquellos puntos con más impacto en el funcionamiento del sistema y que si fallan tienen graves consecuencias para el rendimiento o operatividad de la factoría. El equipo identificó que los puntos críticos del sistema eran las bases de datos, y el flujo de eventos entre paquetes (no debería ser interrumpido).

Por último, para satisfacer el requisito de rendimiento, el cliente especificó que no se deberían producir cuellos de botella en el procesamiento de los datos. El punto del sistema que se encarga de procesar los datos es el paquete del procesador de eventos, por lo que tanto el broker de eventos como la base de datos NoSQL deben ser capaces de funcionar a máximo rendimiento ante el flujo de datos recibido del generador de eventos.

4. Extracción de *QA*s y búsqueda de aproximaciones arquitectónicas

El equipo ATAM analizó los deseos calidad del cliente y extrajo los atributos de calidad (*QA* en inglés) correspondientes. Posteriormente, se discutieron diferentes aproximaciones o posibles soluciones para cumplir este *QA*.

Para lograr el primer requisito de seguridad se extrajo el atributo de calidad *Security*. Anteriormente se identificó como punto clave el generador de eventos. Se propuso incluir métodos de encriptado al transmitir los datos y desencriptado al recibirlos, de manera que, si fueran interceptados al propagarse por las redes, no sean reconocibles y, por lo tanto, no se vean comprometidos.

Respecto al segundo requisito, que pedía una disponibilidad casi total, se extrajo el atributo de calidad *Availability*. Como puntos clave se determinaron los puntos críticos del sistema: el sistema de mensajería y las bases de datos. Para contrarrestar posibles caídas del sistema, se diseñó una clase que favorezca la recuperación rápida y segura del estado previo a la caída, con el fin de conseguir un mayor grado de *Reliability* en general, aunque de una forma no cuantificada. De igual manera, es necesario comprobar qué bases de datos y qué sistemas de mensajería están más tiempo disponibles. Se propuso considerar los sistemas o bases de datos que estuvieran basados en la nube. Estos suelen ofrecer sus servicios en *clusters*, lo que quiere decir que tienen su información almacenada en diferentes puntos, y normalmente con redundancia, por lo que ofrecen un alto grado de disponibilidad.

Por último, el cliente expresó su deseo de que no se produjeran cuellos de botella en el procesamiento de los datos. El equipo ATAM descompuso este deseo en dos atributos de calidad: *Performance* y *Reliability*.

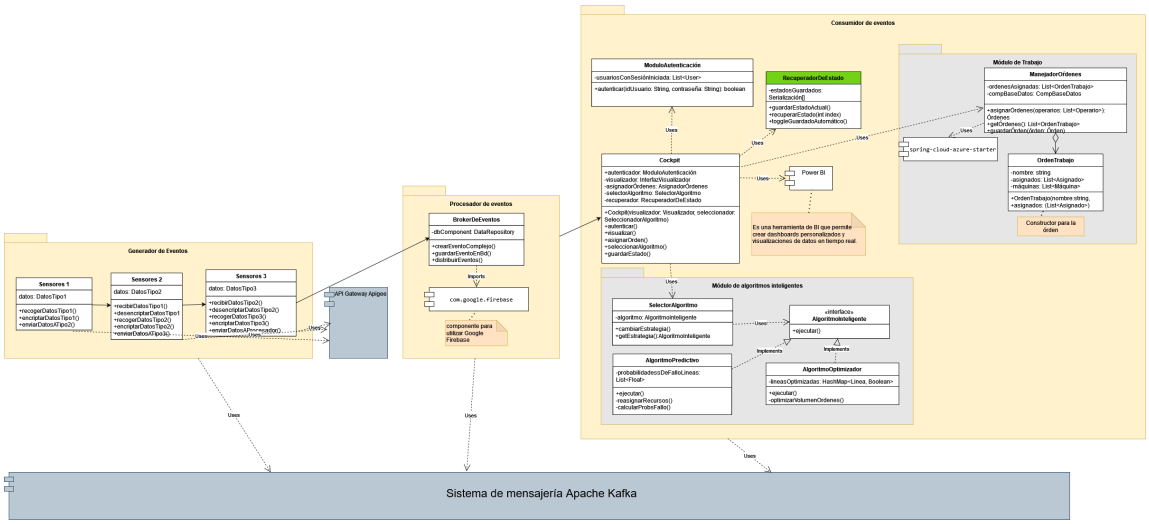

Figura 2: Diagrama de clases y paquetes UML de las aproximaciones arquitectónicas

Como primera aproximación arquitectónica a los deseos de negocio del cliente, se añadió una clase *RecuperadorDeEstado*, que se encarga de guardar el estado actual de la factoría periódicamente. Esto permite fiablemente realizar recuperaciones rápidas de un estado previo funcional del sistema en caso de caídas o fallos de la factoría. Tiene un atributo booleano *guardadoAutomático*, que determina si está activado el autoguardado periódico del estado si es verdadero. El atributo *estadosGuardados* guarda un array de *Serialización* (*Serialización* siendo una abstracción ambigua para el formato de serialización adecuado que se decida cuando se programe el sistema). Es un array porque se debería poder guardar múltiples estados previos distintos de la factoría, no solo uno, en caso de que haya que descartar un estado previo a favor de otro aún más antiguo por alguna razón.

5. Elaboración del *Utility Tree*

Posteriormente, se empezó la elaboración del *Utility Tree*, que es la herramienta eje del proceso ATAM. Para ello, se diseñó cuidadosamente cada escenario tomando como base datos numéricos reales extraídos de fuentes fiables. Los escenarios directamente comparables cuantitativamente se agruparon para poder ser estudiados en conjunto y elegir aquellos que resulten más aptos para la arquitectura específica y aquellos deseos del cliente capturados que fueron más priorizados por el equipo ATAM tras la discusión previa con el cliente.

QA	Sub-QA	Partes de la arquitectura afectada	Escenarios	Prioridad; Dificultad
Reliability	**Availability**	Base de datos SQL del módulo de trabajo	Teniendo instalada la base de datos Azure SQL, si esta se deja en funcionamiento 365 días, debido a su disponibilidad del 99,99 %, entonces habrían en promedio 52 minutos de *downtime* por año.[2]	M, M
			Si se instalase una base de datos MySQL, tomando como infraestructura para esta la que ofrece Aurora de Amazon Web Services, y se dejase en funcionamiento 365 días; ya que tiene una disponibilidad del 99,99 %, entonces en ese transcurso de tiempo habrían en promedio unos 52 minutos en los que está caída la base de datos.[3]	H, L
			El servicio RDS de AWS asegura contractualmente una disponibilidad mensual del 99,95 % para cualquier base de datos de tipo SQL que haga uso de su infraestructura, por lo que, si se dejase en funcionamiento una base de datos SQL por 365 días, habría en promedio 262 minutos de *downtime* por año.[4]	L, M
		API Gateway	La API Gateway de Apigee ofrece una disponibilidad del 99 %, lo que se traduce en un tiempo de inoperancia anual limitado a 5256 minutos.[5]	L, L
			La Amazon API Gateway tiene un 99,95 % de disponibilidad, por lo que ante 365 días de operación, de media el servicio estaría caído 265 minutos anualmente.[6]	H, M
			La API Gateway Azure ofrece un 99 % de disponibilidad, por lo que ante el transcurso de 365 días, habrían en promedio 3,65 días por los cuales estaría caída la API, o 5256 minutos.[7]	L, M
		Base de datos NoSQL del procesador de eventos	Si se instalase la base de datos NoSQL MongoDB para persistir eventos, y transcurriesen 365 días, ya que tiene una disponibilidad del 99,995 %, entonces habrían de media tan solo unos 0,01825 días de *downtime* a través del año; 26 minutos.[8]	H, L
			Si se instalase la base de datos NoSQL Google Firebase para almacenar eventos, y transcurriese un t=365d, debido a su disponibilidad del 99,95 %, el tiempo de *downtime* anual, t_d sería aproximadamente igual a 0,1825 días, o, 263 minutos.[9]	M, L
			Con la disponibilidad del 99,99 % que posee CouchBase, si se dejase operar el sistema por 365 días, habrían en promedio unos 52 minutos de *downtime* anuales.[10]	M, M

Continúa en la siguiente página

Tabla 2 – *Continuación de la anterior página*

QA	Sub-QA	Partes de la arquitectura afectada	Escenarios	Prioridad; Dificultad
			ApsaraDB for HBase ofrece un 99.9 % de disponibilidad, lo que significa que el servicio podría estar inoperativo por una duración de hasta 518 minutos en el transcurso de un año.[11]	M, H
			Amazon KeySpace for Apache Cassandra tiene una disponibilidad que varía entre el 99.99 % y el 99.0 %, por lo que su tiempo de *downtime* está entre 51.84 minutos y 5184 minutos por año.[12]	L, H
Performance	**Throughput**	Base de datos NoSQL del procesador de eventos	Si se disponen 32 nodos (servidores físicos de la B.D.), ante el estímulo recibido de 1M de transacciones a procesar, Cassandra, con su *throughput* de 326k operaciones por segundo, tardaría 2,74 segundos en computar todas las transacciones.[13]	H, H
			Si se tienen 32 nodos, ante el estímulo de 1M de transacciones a realizar, HBase, teniendo en cuenta el *throughput* que posee de 192k operaciones por segundo, tardaría 5,18 segundos en procesar todas.[13]	H, M
			Poseyendo 32 nodos de procesamiento, si se recibiese el estímulo de 1M de transacciones a procesar, con el *throughput* que tiene MongoDB de 135k operaciones/s, se tardarían unos 7,4 segundos en realizar el procesado entero.[13]	L, L
			Con 32 nodos, ante el estímulo de 1M de transacciones a realizar, CouchBase, con su *throughput* de 136k operaciones por segundo, tardaría aproximadamente 7,35 segundos en computar todas las transacciones ordenadas.[13]	M, M
	-	Generador de eventos	El algoritmo de encriptado simétrico AES, mediante el uso de una clave de cifrado de 128 bits, tarda 763 milisegundos en encriptar y 37 milisegundos en desencriptar un bloque de memoria de 1024 KB.[14]	H, M
			El algoritmo de encriptado simétrico AES, mediante el uso de una clave de cifrado de 198 bits, tarda 1086 milisegundos en encriptar y 124 milisegundos en desencriptar un bloque de memoria de 1024 KB.[14]	M, M
			El algoritmo de encriptado simétrico AES, mediante el uso de una clave de cifrado de 256 bits, tarda 1419 milisegundos en cifrar y 300 milisegundos en desencriptar un bloque de memoria de 1024 KB.[14]	L, M
Security	-	Generador de eventos	La supercomputadora The Summit tardaría unos 2^{56} años en romper AES-128 por fuerza bruta.[15]	M, M
			La supercomputadora The Summit tardaría unos 2^{126} años en romper AES-198 por fuerza bruta.[15]	M, M
			La supercomputadora The Summit tardaría unos 2^{184} años en romper AES-256 por fuerza bruta.[15]	H, M

Tabla 2: *Utility Tree*

Como se puede apreciar, se realizaron algunas clasificaciones de escenarios bajo sub-atributos de calidad, como por ejemplo *Availability* bajo *Reliability*[1].

6. Evaluación, priorización, y descarte o selección de escenarios

Una vez elaborado el Utility Tree, el equipo ATAM se reunió para evaluar la arquitectura bajo la infraestructura planteada en cada escenario.

Primero, se empezaron a evaluar los escenarios pertenecientes al atributo de calidad *Availability*, previamente extraído de los deseos del cliente. Estos escenarios afectan a diferentes partes del sistema. Se empezó valorando primero los escenarios que afectan a la base de datos SQL del módulo de trabajo. Se tomaron en consideración tres posibles escenarios, en los que el equipo decidió asignar una mayor prioridad a la instalación de una base de datos MySQL ofrecida por el servicio Aurora de Amazon Web Services debido a su mayor disponibilidad (99,99 %) frente a las alternativas de Azure y RDS. Las tres opciones tienen un gran grado de madurez y documentación en línea, pero debido a la familiaridad del equipo de desarrollo con Aurora, se le asignó un menor nivel de dificultad de implantación que sus alternativas.

Para la API Gateway, también se valoraron tres escenarios: Apigee, Amazon y Azure. Al diseñar la arquitectura original, el equipo eligió Apigee como la API Gateway, pero al ser ahora comparada con otras alternativas, se observó que ofrecía un menor grado de disponibilidad que la API Gateway de Amazon (99,95 %), a la que se le asignó un alto grado de prioridad debido a esta mejoría, junto a un bajo grado de dificultad de implementación debido a la popularidad de Amazon Web Services, ya que esta popularidad atrae un muy alto grado de documentación técnica disponible en Internet, además de que es más fácil encontrar personal especializado para Amazon Web Services. Al ser la API Gateway Apigee parte de la implementación de la arquitectura original, se le dio un grado bajo de dificultad de implantación debido a que no habría que realizar cambios. Aunque, conjuntamente con esta baja dificultad, se le dió baja de prioridad como escenario por su levemente peor disponibilidad. La API Gateway de Azure ofrecía un nivel de disponibilidad similar a la Apigee, por lo que también se le dió prioridad baja, junto a una dificultad mediana de implementación debido a la necesidad de realizar el cambio del servicio.

El último punto de la arquitectura en el que se valoró este atributo de calidad fue la base de datos NoSQL del procesador de eventos, del cuál se presentaron cinco escenarios diferentes. La base de datos ya existente en la arquitectura era la B.D. NoSQL Firebase de Google, que tiene una muy baja dificultad de implantación como escenario al ser la opción original, y también por ser muy fácil de conectarla en Java. Por su parte, Couchbase ofrece una disponibilidad mayor que esta opción, aunque con el menoscabo de que su dificultad de implementación es mayor que la que tiene Firebase. "ApsaraDB for HBase", por otro lado, tenía una disponibilidad menor que las dos anteriores, con una dificultad moderada de implementación, al ser un software más desconocido para el equipo. Una opción interesante fue "Amazon KeySpace for Apache Cassandra": ofrece su disponibilidad como un rango entre 99 % y 99,99 %, lo que hizo que se considerara un riesgo la medida algo imprecisa, ya que se trataba de una parte de la arquitectura muy sensible, por lo que se le asignó una baja prioridad, junto con un alto grado de dificultad de implantación debido al desconocimiento técnico por parte del equipo de desarrolladores. Por último, el escenario que fue designado como más prioritario para el equipo fue el que consideraba la utilización de MongoDB, ya que esta base de datos ofrecía una disponibilidad del 99,995 %, unido a que también se considera generalmente fácil de implementar una base de datos MongoDB.

A continuación, se evaluaron los escenarios correspondientes al atributo de calidad *Performance*. Más concretamente, a su sub-atributo *Throughput*. Al igual que en el anterior atributo de calidad estudiado, los escenarios también afectan a diferentes partes del sistema. En primer lugar, se valoraron los escenarios que afectan a la base de datos NoSQL situada en el paquete procesador de eventos. Todos estos escenarios se evaluaron únicamente bajo el caso de que se dispongan de 32 nodos de procesamiento (servidores físicos de la base de datos), y ante el estímulo de un millón de transacciones recibidas a procesar. Cassandra fue la opción que más transacciones realizaba por segundo, por lo que se le asignó la máxima prioridad, seguida de HBase, con una prioridad media. Por último, CouchBase y MongoDB obtuvieron los peores resultados, a lo que se les otorgó la más baja prioridad a sus escenarios.

En este punto, el cuestionador de escenarios detectó un *trade-off*, ocasionado por un conflicto entre escenarios que afectan a la misma parte del sistema, pero que estudian y miden diferentes atributos de calidad. Se descubrió que, para los escenarios de las bases de datos NoSQL, cuanta mayor era la disponibilidad de la base de datos, menor era su rendimiento entre los escenarios que estudiaban el *Throughput*. Como parte del proceso ATAM, el equipo presentó al cliente el conflicto: si se eligiese la base de datos NoSQL con mayor disponibilidad, la factoría se quedaría con la que tiene el peor rendimiento entre los escenarios de *Throughput*, y viceversa. Así que, el equipo le pidió al cliente una reunión para discutir cuál atributo de calidad es el más importante a priorizar dentro del funcionamiento del negocio, tanto a corto como a largo plazo. El cliente consideró más importante la disponibilidad de la base de datos sobre el rendimiento, ya que estimó que su negocio podría permitirse una bajada en el *Throughput* de esta base de datos, pero no podía permitirse nunca que la base de datos NoSQL se vuelva inoperativa más seguidamente, ya que, la factoría debe estar en funcionamiento las veinticuatro horas del día, toda la semana. Debe ser menos probable que se caiga la base de datos que persiste los eventos dentro de un momento critico.

Entonces, el equipo ATAM seleccionó MongoDB como el escenario finalmente aprobado al ser la opción óptima para el atributo *Availability*, además de contar con el menor grado de dificultad de implementación al haber una gran cantidad de documentación disponible en Internet para implementar MongoDB y también realizar troubleshooting. Las otras bases de datos, al ser menos populares y tener comunidades más pequeñas, disponen de un levemente menor grado de documentación disponible para solucionar problemas técnicos ultra-específicos que pudiesen aparecer.

Siguiendo con el atributo de calidad *Performance*, también se plantearon escenarios sobre el generador de eventos, en este caso para evaluar el tiempo de encriptado del algoritmo de cifrado simétrico AES de un bloque de memoria de 1024 KB. En este caso, cada escenario estudia la performance bajo cada longitud de la clave de cifrado. Como podía esperarse, la longitud de clave de 128 bits tomó la máxima prioridad al ser la que menos tardó en encriptar y desencriptar el bloque de memoria, seguida de la longitud 198 bits y, por último, 256 bits. Decidimos priorizar la rapidez sobre la seguridad en este caso porque consideramos que hay un *sensitivity point* que sería minimizado por una mejor performance, ya que un reducido flujo de datos ocasionado por un encriptado pesado podría generar un cuello de botella y resultar en una mayor latencia para la llegada de los eventos, que se traduciría en una pérdida de la relevancia de estos al llegar tardíamente, y en una pérdida en el tiempo de reacción disponible para los operarios ante eventos críticos.

El último atributo de calidad que se evaluó fue *Security*. Este atributo únicamente afectaba al algoritmo de encriptación del generador de eventos y, al igual que en el caso anterior, cada escenario pone a prueba el nº de bits abarcados por la clave de encriptación, ante un estímulo específico. En este caso particular, se cuantifica el tiempo que tardaría la supercomputadora *The Summit* en romper el algoritmo de cifrado teniendo en consideración el tamaño de clave. La longitud de clave de 256 bits tomó la máxima prioridad ante los otros escenarios al ser la que más robustez ofrecía, seguida de las de 198 y 128 bits, que también obtuvieron buenos resultados, solo que peores comparativamente. Este es un resultado "obvio" y esperado al tener una menor longitud de clave, pero habíamos estimado que podría ser útil poder apreciar "en qué magnitud" una opción es peor que otra.

El cuestionador de escenarios detectó un trade-off entre los atributos *Security* y *Performance* respecto al encriptamiento: cuantos más bits se dedique para la clave, más seguridad, pero más se tarda en realizar el encriptado. Se presentó el conflicto entre los atributos del cliente y se le exigió un repaso de sus prioridades de negocio. El cliente consideró esta vez que la seguridad que ofrecía la clave de 128 bits era suficiente, bajo la justificación de que, en el tiempo que tardaría un agente externo en romper el encriptado de un paquete de datos mediante fuerza bruta, ese paquete de datos ya habría perdido relevancia. Además, como el tiempo de descifrado mediante fuerza bruta se encuentra en una magnitud extremadamente alta de tiempo para cualquiera de todos los escenarios estudiados, tiene mucho más sentido dedicar los recursos en proteger el algoritmo/procedimiento concreto que utilicemos para generar pseudo-aleatoriamente las claves de encriptación. Así que, debido a su irrelevancia en el estudio técnico, el cuestionador de escenarios descartó estos tres

escenarios que intentaron cuantificar el atributo *Security*. Aparte, el equipo ATAM estimó que era importante el optimizar la rapidez de los algoritmos de encriptado en este punto concreto para no ocasionar un cuello de botella con un encriptado demasiado pesado. El flujo rápido de los eventos se consideró previamente como un *sensitivity point*, ya que si un evento crítico de la factoría llega con varias docenas de segundo de retraso debido al encriptado pesado, eso le daría menos tiempo a un operario para reaccionar rápidamente y prevenir una situación catastrófica. Al final, se aceptaron todos los escenarios menos los tres que referenciaban a la supercomputadora *The Summit*.

7. Presentación de la arquitectura resultante

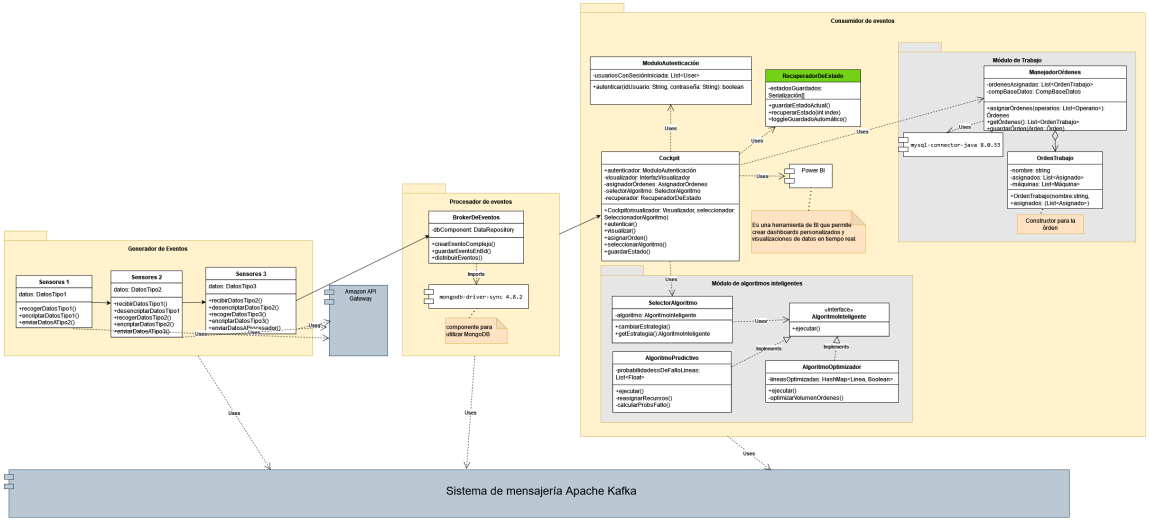

Figura 3: Diagrama de clases y paquetes UML de la arquitectura final

Se puede apreciar que hubo pocos cambios estructurales, ya que no se identificó ninguna razón específica que justifique romper con la arquitectura por eventos, ni tampoco razones para eliminar módulos anteriores, o crear módulos de tamaño significativamente grande. Nuestros escenarios, aunque no todos, se centraron más en evaluar varios servicios posibles para distintos componentes.

Entre las modificaciones realizadas, se añadió la clase *RecuperadorDeEstado*, con la justificación de que es necesaria alguna clase que se encargue de guardar ciertos estados específicos en el tiempo de la factoría. Esta asegura una recuperación fiable en caso de que se produzca alguna caída de servicio. También, puede albergar múltiples estados serializados a la vez, lo cual también es útil para realizar pruebas de estrés sobre una configuración específica antes de realizar cambios definitivos.

Asimismo, se implementaron métodos de encriptación para las clases que encapsulan a las familias de sensores, para así intentar cumplir con el pedido por parte del cliente de incrementar la seguridad de información ante posibles *eavesdroppers*. Al final, se escogió la encriptación hecha con una clave de 128 bits, ya que, como se explicó antes, este nivel de robustez es suficiente para salvaguardar la información de los eventos el suficiente tiempo hasta que estos pierdan relevancia. Si se eligiese una clave más grande, haría falta un procesado más intensivo computacionalmente y aumentaría la latencia de los eventos.

Aunque no se realizaron muchas modificaciones estructurales, sí se llevaron a cabos muchos reemplazos de componentes. Por ejemplo, se reemplazó la API gateway de Apigee por la Amazon API Gateway debido a que esta última garantiza un mayor porcentaje de disponibilidad anual. También, se instaló el controlador para la base de datos MySQL, en sustitución del controlador de Spring para Azure SQL. Esto se hizo para poder hacer uso de la base de datos MySQL que vive en la infraestructura que provee Amazon.

En cuanto a la base de datos NoSQL encargada de persistir eventos, se reemplazó el controlador la base de datos Firebase de Google por el controlador para utilizar MongoDB. Esto es, como se explicó antes, porque optamos por adoptar la que provee la más alta disponibilidad.

8. Conclusiones

Esta práctica tuvo la gran utilidad didáctica de poder practicar realizar estudios numéricos de *Quality Attributes*. Determinando de una forma cuantificada, precisa y objetiva, si el impacto de un cierto cambio en una arquitectura de software es perjudicial para un determinado atributo de calidad, o beneficioso. Además, se aprende sobre los *trade-offs*, por los cuales hay que darse cuenta de que cuando se está potenciando a una cierta medición, esto puede ser en detrimento de otra. Así que cabe estudiar hasta qué punto específico es ventajoso mejorar al QA, y hasta qué punto es aceptable empeorar el otro. O, reflexionar si es tal vez mejor empujar el *trade-off* hacia el sentido contrario.

También, se enseña a discutir previamente con el cliente sobre sus deseos de negocio, ya que es irresponsable hacer modificaciones arquitectónicas que no tengan en cuenta como funciona en sí el negocio del cliente. Se colabora con él para conseguir una arquitectura ideal, sostenible tanto a corto como a largo plazo. Se aprende a trabajar colaborativamente para intentar conseguir la máxima efectividad del equipo ATAM revisando y cuestionando continuamente, por parte de los cuestionadores de escenarios, lo que producen los redactores de escenarios. Aunque probablemente hayan presentes en esta memoria varios fallos relacionados con conceptos técnicos específicos de la informática, hay que tener en consideración que somos un grupo de estudiantes carentes de experiencia laboral real.

9. Referencias

[1] ISO 25000. *ISO/IEC 25010*. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=3 (visitado 19-12-2023).

[2] Microsoft. *Azure SQL Database Well-Architected Framework*. 14 de nov. de 2023. URL: https://learn.microsoft.com/en-us/azure/well-architected/service-guides/azure-sql-database-well-architected-framework (visitado 19-12-2023).

[3] Amazon Web Services. *Amazon Aurora*. URL: https://aws.amazon.com/rds/aurora/ (visitado 19-12-2023).

[4] Amazon Web Services. *High Availability (Multi-AZ) for Amazon RDS*. URL: https://aws.amazon.com/rds/ha/ (visitado 19-12-2023).

[5] Google Cloud. *Apigee Service Level Agreement*. 28 de sep. de 2022. URL: https://cloud.google.com/apigee/sla (visitado 19-12-2023).

[6] Amazon Web Services. *Amazon API Gateway Service Level*. 5 de mayo de 2022. URL: https://aws.amazon.com/api-gateway/sla/ (visitado 19-12-2023).

[7] Microsoft Azure. *Application Gateway Service Level Agreement*. 2019. URL: https://www.azure.cn/en-us/support/sla/application-gateway/ (visitado 19-12-2023).

[8] MongoDB. *Atlas Reliability*. URL: https://www.mongodb.com/cloud/atlas/reliability (visitado 19-12-2023).

[9] Google Firebase. *Terms - Service Level Agreement*. 9 de abr. de 2020. URL: https://firebase.google.com/terms/service-level-agreement (visitado 19-12-2023).

[10] Couchbase. *Couchbase Cloud SLA*. 29 de jun. de 2020. URL: https://www.couchbase.com/CloudSLA06292020/ (visitado 19-12-2023).

[11] Alibaba Cloud. *HBase Benefits*. 29 de jun. de 2023. URL: https://www.alibabacloud.com/help/en/hbase/product-overview/benefits-1 (visitado 19-12-2023).

[12] Amazon Web Services. *Amazon Keyspaces Service Level Agreement*. 23 de abr. de 2020. URL: https://d1.awsstatic.com/legal/amazon-keyspaces-sla/Amazon%20Keyspaces%20Service%20Level%20Agreement%202020-04-23%20Spanish.pdf (visitado 19-12-2023).

[13] DataStax. *Benchmarks: Cassandra vs MongoDB vs HBase*. 6 de mar. de 2019. URL: https://web.archive.org/web/20190306041414/https://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-hbase (visitado 20-12-2023).

[14] Suchitra.R Toa Bi Irie Guy-Cedric. *A Comparative Study on AES 128 BIT AND AES 256 BIT, figure 3*. 2018. URL: https://www.isroset.org/pub_paper/IJSRCSE/5-IJSRCSE-01186.pdf (visitado 20-12-2023).

[15] kelalaka. *Has AES-128 been fully broken?* 31 de dic. de 2023. URL: https://crypto.stackexchange.com/a/76746 (visitado 20-12-2023).