Involucrando al humano en la toma de decisiones en sistemas auto-adaptativos

Fabián Andrés Sabogal Ocampo

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

Esta tesis se presenta para el grado de

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Vicente Pelechano Ferragud

Joan Fons i Cors

Septiembre 2016

Dedicatoria

Me gustaría dedicar esta tesis a mis padres.

Agradecimientos

Me gustaría dar agradecimientos a Miriam Gil por su ayuda y dedicación en el acompañamiento de la Tesina. Y especialmente a Catalina Ochoa por ser un apoyo en esta etapa de mi vida.

Índice general

Índice de figuras

Índice de cuadros

Algoritmos

Capítulo 1

Introducción

En un mundo cada vez más dinámico, inteligente y descentralizado, las tecnologías y los tipos de aplicaciones evolucionan hacia la creación de ecosistemas compuestos por una amplia variedad de dispositivos y servicios heterogéneos y distribuidos, de naturaleza claramente móvil y ubicua, y en constante evolución tecnológica. Ante esta situación, surge evidentemente una necesidad de desarrollar sistemas que sean capaces de adaptarse de forma continua y dinámica (en tiempo de ejecución) a nuevas condiciones del entorno, situaciones impredecibles, necesidades cambiantes de sus usuarios, nuevos dispositivos y tecnologías con los que interactuar o nuevos servicios que consumir. Esta adaptación debe gestionarse de forma autónoma (auto-adaptabilidad) ya que en primera instancia no parecería viable asumir que un «humano» intervenga activamente en materializar todas las adaptaciones en tiempo de ejecución. Sin embargo, un tema cada vez más interesante es analizar de qué manera y con qué roles un humano puede participar en estas tareas con el objetivo de enriquecerlas, ajustarlas o resolver las excepciones críticas que puedan ocurrir.

En estos casos, la adaptación se podría beneficiar de recibir información de los humanos ya sea actuando como sofisticados sensores (por ejemplo, proporcionando información del contexto), como tomadores de decisiones (por ejemplo, resolviendo algún conflicto de objetivos del sistema) o como efectores a nivel de sistema para ejecutar adaptaciones (por ejemplo cuando la automatización no es posible o como mecanismo de último recurso si no se pueden llevar a cabo las adaptaciones de forma automática).

La idea es que la participación de los usuarios sirva para mejorar la adaptabilidad del sistema, participando en la toma de decisiones, pero siempre consiguiendo un equilibrio entre el comportamiento autónomo y la participación del usuario.

Para llegar a tal nivel de interacción con el humano de manera que se convierta en una parte imprescindible dentro del proceso de toma de decisiones de los sistemas auto-adaptativos, surge la necesidad de facilitar la integración al humano dentro del bucle de las aplicaciones

existentes y las próximas en llegar. Y para ello es importante que exista un componente software capaz de comunicarse con el sistema auto-adaptativo en cuestión que será el encargado de exponer el estado del bucle de control a los usuarios (qué mostrarle, cuándo mostrárselo y cómo mostrárselo) y capturar el feedback del usuario (input del usuario) para mejorar las adaptaciones.

Dependiendo del rol que se necesite del usuario y su nivel de atención, la interacción se proporcionará mediante unos mecanismos u otros con el fin de conseguir una interacción natural sin intervenciones molestas hacia los usuarios.

Este trabajo presenta una propuesta de desarrollo que interactúa con estos sistemas auto-adaptativos, e intenta facilitar la integración de usuario dentro del bucle de adaptación, con lo cual se pretende incrementar significativamente la asertividad en la toma de decisiones de los mismos.

En el capítulo a continuación se presenta una breve descripción del contexto manejado para esta investigación incluyendo algunos trabajos relacionados. En el capítulo 3, se lleva a cabo un análisis detallado de las necesidades que se deben tener en cuenta para llegar a construir una solución exitosa. En los dos capítulos siguientes se lleva a cabo la explicación y aplicación de la propuesta presentada, y finalmente se cierra en los últimos capítulos con los resultados obtenidos y las conclusiones a resaltar.

1.1. Planteamiento del problema

Para paliar las carencias existentes de los sistemas auto-adaptativos actuales para incluir la interacción con el humano, se plantea abordar tres diferentes puntos de vista.

A nivel de **requisitos** de interacción se requiere identificar cómo y en qué situaciones se puede participar, el grado de atención que se requerirá por parte del usuario, y el grado de molestia que éste es capaz de soportar. La idea es que la participación de los usuarios sirva para mejorar la adaptabilidad y autonomía del sistema, participando en la toma de decisiones, pero siempre consiguiendo un equilibrio entre el comportamiento autónomo y la participación del usuario.

A nivel de **diseño** se pretende diseñar los sistemas para y por los usuarios, que sean transparentes y entendibles, y especificar técnicas de interacción donde el comportamiento autónomo del sistema se intercala con input humano. En este sentido se plantea cómo diseñar las interacciones para que los humanos sean capaces de entender qué está pasando en el sistema y puedan prever los comportamientos futuros, para que ante posibles fallos se les notifique a los humanos de la manera más adecuada y sean capaces de solucionarlo, y para mejorar la experiencia del usuario en general proporcionando sistemas seguros, usables y fiables.

Por último, a nivel de **ejecución**, se pretende introducir al usuario en el bucle de adaptación y poder llevar a cabo una toma de decisiones colaborativa en los casos donde se necesite. Para involucrar al usuario en el proceso de adaptación se deberá filtrar y presentar la información adecuada, así como utilizar los mecanismos de interacción más afines a la situación para conseguir una interacción fácil y efectiva y la aceptación de los sistemas por parte de los usuarios. Se deben investigar cómo las capacidades de los humanos y los sistemas se pueden combinar de manera óptima para llevar a cabo el proceso de toma de decisiones. En este sentido, se pretenden proporcionar técnicas para introducir al usuario en el bucle de adaptación y poder llevar a cabo una toma de decisiones colaborativa en los casos donde se necesite.

Para la interacción con el usuario se definen unos patrones de interacción entre el sistema y el usuario que tengan en cuenta las capacidades de atención de los humanos y el contexto en el que se encuentren para proporcionarle la información adecuada y conseguir una toma de decisiones consistente. Analizando el comportamiento de los usuarios y su feedback (explícito o implícito) ayuda a mejorar las adaptaciones del sistema y conseguir sistemas más "resilientes" al comportamiento impredecible de los humanos. De esta manera los usuarios pueden "formar parte del bucle" garantizando su confianza con el sistema.

Es así como se requiere de una entidad que, por una parte, se encargue de definir y gestionar los nuevos comportamientos que puede tener el sistema a partir de las interacciones con el humano, así como también establecer la lógica detrás de la comunicación entre el sistema y los sensores de contexto. Además, en su diseño se debe incluir la manera en que las diferentes entradas posibles definen un flujo específico para cada caso y de acuerdo a cada contexto. En cuanto a la ejecución, es necesario proveer los mecanismos y componentes necesarios para establecer una comunicación exitosa y cómoda con el humano.

Para esto, parece viable la construcción de un framework que cumpla a cabalidad con los requerimientos antes mencionados, a la vez que sirva como herramienta para cualquier campo de aplicación que utilice un sistema auto-gestionable.

1.2. Objetivos

1.2.1. Generales

- Crear un framework de software que facilite el desarrollo a los programadores de la integración al humano en la toma de decisiones para sistemas auto-adaptativos

1.2.2. Específicos

- Describir un DSL que permita al programador especificar la participación de los bucles de control de sistemas auto-adaptativos dentro del framework
- Prototipar un software que permita con el DSL facilitar la integración del humano en los bucles de control a los programadores
- Crear un caso de estudio con el cual se pueda validar la utilidad del framework desarrollado con diferentes casos

Capítulo 2

Contexto de la Tesina

Dentro del marco del desarrollo de este TFM se acudió al uso de trabajos anteriores, conceptos y tecnologías, los cuales serán presentadas a continuación.

2.1. Trabajos Relacionados

A lo largo de estos últimos años se ha investigado en el área de los sistemas sensibles al contexto y los sistemas adaptativos y se ha visto que un problema potencial de los sistemas existentes es que no involucran a los usuarios en el proceso del cambio de políticas de adaptación ni en la trazabilidad de dicha adaptación, como indicaron Salehie y Tahvildari [26]. Esto puede llevar a que los usuarios desconfíen de estos sistemas, que hagan un mal uso de ellos, o que los abandonen por completo [22]. Para paliar este problema, Bellotti y Edwards declararon que los sistemas sensibles al contexto debían ser inteligibles, es decir «ser capaces de representar a sus usuarios lo que saben, cómo lo saben, y lo que están haciendo al respecto»[2]. Diversos autores han trabajado en esta línea para obtener la mejor manera de proporcionarle al usuario explicaciones de lo que está pasando en el sistema con el objetivo de aumentar la confianza del usuario [5, 7]. Además, se ha comprobado que el tipo de explicación influye en el nivel de confianza y comprensión por parte de los usuarios [20]. Sin embargo, la mayoría de soluciones no incorporan al usuario en el bucle de control para el cambio de políticas de adaptación o procesos de trazabilidad de la adaptación [26]. Barkhuss y Dey [1] examinaron el grado de autonomía que las aplicaciones sensibles al contexto deberían tener desde el punto de vista de los usuarios. Definieron tres niveles de interactividad:

1. Personalización: aplicaciones que permiten al usuario especificar el comportamiento de la aplicación en una situación dada

2. Conciencia del contexto pasiva (aplicaciones que presentan el contexto actualizado al usuario y le permiten a este decidir cómo cambiar el comportamiento de la aplicación

3. Conciencia del contexto activa (el sistema cambia de forma autónoma el comportamiento de la aplicación en base a la información de contexto percibido)

De este estudio observaron que los usuarios prefieren sistemas sensibles al contexto sobre la personalización, pero al mismo tiempo con estos sistema experimentaron una falta de control. Por tanto señalaron la necesidad de gestionar un equilibrio adecuado entre el control del usuario y la autonomía del sistema en tiempo de ejecución.

Varios autores también han destacado la importancia de la influencia del usuario en los sistemas auto adaptativos poniendo énfasis en la necesidad de distinguir los distintos roles entre el usuario y el sistema [4, 26, 34]. Shin [30] presentó una propuesta de resolución de conflictos combinando la resolución automática con la resolución dirigida por el usuario de acuerdo a sus preferencias. Esta solución solo considera dos tipos de participación: automática y manual. Recientemente, Dorn y Taylor [6] introdujeron un framework para razonar acerca de los efectos de los cambios a nivel de software en las interacciones de los usuarios y viceversa. Este enfoque se centra en una topología de colaboración capaz de ofrecer un proceso de adaptación colaborativo para permitir adaptaciones mas sofisticadas, sin embargo no introduce al usuario explícitamente en el proceso de adaptación y no tiene en cuenta los distintos roles del usuario. Evers [8] proporcionó una solución para tener en cuenta al usuario en interfaces auto adaptativas de acuerdo a sus prácticas de interacción. Sin embargo, las soluciones existentes solo tratan con tipos de participaciones de usuario aisladas, y no consideran todo el rango de roles y tipos de participación que el usuario pueda tener. Por otra parte, Cámara, Moreno y Garlanx [3] proporcionaron una extensión del framework Rainbow para razonar acerca de la integración del usuario en la auto-adaptación. Sin embargo se centran solo en el rol de usuario como ejecutor de adaptaciones. Además pocas soluciones tienen en cuenta al usuario en fases posteriores al diseño del sistema para poder cambiar su comportamiento o evolucionarlo dinámicamente en tiempo de ejecución.

2.2. Conceptos Generales

2.2.1. Computación Autónoma

La Computación Autónoma surgió como una propuesta de IBM [10, 15] ante la necesidad de encontrar un mecanismo para los sistema de software que provea un comportamiento similar al del sistema nervioso humano, es decir, con capacidad de auto-gestionarse o repararse sin

necesidad de una intervención externa. Esto evidentemente añadiría un nuevo nivel de complejidad al sistema y sus usuarios, aunque podría integrarse a su infraestructura de tal manera que se automatice.

Entre los beneficios de los sistemas de Computación Autónoma se encuentran la simplificación de tareas del administrador humano, una mayor disponibilidad y una disminución en los costos.

Algunos ejemplos de aplicación de estos sistemas se encuentran en la gestión de bases de datos, reduciendo la complejidad y el coste, y para balanceo de cargas en Sistemas de Información.

En un sistema autónomo (auto-gestionable), el humano tiene un nuevo rol: en vez de controlar el sistema directamente, éste define las políticas generales y las reglas que guían el proceso de auto-adaptabilidad.

Un sistema de Computación Autónoma comprende cuatro características «self» [24]:

1. Self-configuration: Configuración automática de los componentes
2. Self-healing: Descubrimiento automático y corrección de fallas
3. Self-optimization: Monitoreo y control automático de los recursos para garantizar un óptimo funcionamiento con respecto a los requerimientos definidos
4. Self-protection: Identificación y protección proactiva frente a ataques arbitrarios

Hay una versión extendida de esta definición propuesta por otros autores Nami y Bertels [23] hacia lo siguiente:

1. Self-regulation: Un sistema que opera para mantener un parámetro (ej: QoS)
2. Self-learning: El sistema utiliza técnicas de aprendizaje que no requieran de control externo
3. Self-awareness: El sistema debe conocerse a sí mismo, de sus componentes internos y enlaces externos para poder controlarlos y administrarlos
4. Self-organization: La estructura del sistema guiada por modelos físicos sin presión explícita o interacción con el mundo exterior
5. Self-creation: La estructura del sistema es guiado por modelos ecológicos y sociales, sin presión ni interacción con el mundo exterior
6. Self-management: Un sistema que se auto gestiona sin intervención externa. Lo que se está administrando depende del sistema y la aplicación

7. Self-description: Un sistema se explica a sí mismo. Es capaz de ser entendido por humanos sin tener una explicación a fondo

De acuerdo a IBM, son ocho las condiciones que definen un sistema autónomo:

1. Conocerse en términos de a qué recursos tiene acceso, cuáles son sus capacidades y limitaciones y cómo y por qué está conectado a otros sistemas

2. Ser capaz de configurarse automáticamente y reconfigurarse a sí mismo dependiendo del entorno computacional cambiante

3. Ser capaz de optimizar su desempeño para garantizar la mejor eficiencia para el proceso computacional

4. Ser capaz de encontrar soluciones alternas a problemas identificados ya sea reparándose a sí mismo o delegando funciones lejos del problema

5. Detectar, identificar y protegerse en contra de varios tipos de ataques para mantener la seguridad e integridad del sistema completo

6. El sistema debe ser capaz de adaptarse a su entorno en la medida que va cambiando, interactuando con sus sistemas vecinos y estableciendo protocolos de comunicaciones

7. Basarse en estándares abiertos y no existir en un entorno propietario

8. Anticiparse a la demanda en sus recursos siendo transparente a sus usuarios

IBM también definió los cinco niveles de evolución para los sistemas autónomos a través del modelo de deployment autónomo[1]:

- El primer nivel es el básico que presenta la situación actual donde básicamente los sistemas son manejados de manera manual,

- Los niveles 2 a 4 introducen funciones administrativas automáticas incrementalmente

- El nivel 5 representa el objetivo máximo de los sistemas autónomos auto-gestionables

[1] http://www.ibm.com/press/us/en/pressrelease/464.wss

2.2.2. Sistemas Auto-adaptativos

Los sistemas de software auto-adaptativos son sistemas capaces de adaptar su comportamiento o estructura en tiempo real. Los sistemas auto-gestionables, por su parte, son instancias de sistemas de software auto-adaptativos que se adaptan entre ellos de acuerdo a objetivos administrativos.

La auto-adaptabilidad puede ser estática o dinámica. En la auto-adaptación estática, los mecanismos de adaptación son definidos explícitamente por los diseñadores para que el sistema sepa cuál de estos elegir durante su ejecución, mientras que en la auto-adaptación dinámica, deben ser producidos planes de adaptación y requerimientos de monitoreo, elegidos por el sistema en tiempo real.

Tal como se mencionó anteriormente en este documento, IBM introdujo la notación de componente autónomo como un elemento esencial para los sistemas auto-gestionables, definido con la forma de un loop MAPE (monitorizar-analizar-planear-ejecutar). Este loop es el bloque básico para soportar la auto-adaptación en cuanto al control y administración de un elemento computacional. En la primera fase de este loop, de monitorización, los monitores obtienen y procesan la información de contexto del entorno que es relevante para el proceso de adaptación. Esta información monitorizada es enviada a la segunda fase, de análisis, donde los analizadores correlacionan la información de contexto para inferir síntomas sobre el entorno de ejecución y el comportamiento del sistema. En la tercera fase, de planeación, los planners utilizan síntomas ya analizados para definir los planes de adaptación correspondientes. En la última fase del loop, de ejecución, los ejecutores implementan y ejecutan planes para adaptar el sistema actual y obtener el comportamiento deseado. La fase de monitoreo continuamente re-alimenta el loop de adaptación, reiniciando el ciclo.

Un sistema adaptativo puede ser un conjunto de loops de feedback colaborativos que garantizan el cumplimento de los objetivos del sistema. En este orden de ideas, un sistema auto-adaptativo está diseñado como una colección de colaboradores de manejadores de control de objetivos, loops de control adaptativos, y loops de control de manejadores de contexto para controlar el proceso de adaptación [33].

2.2.3. Component

Un componente de software es un elemento de un sistema que ofrece un servicio predefinidos y es capaz de comunicarse con otros componentes. Una definición mas sencilla puede ser «Un componente es un objeto escrito de acuerdo a unas especificaciones».

La capacidad de ser reutilizado es una característica importante en los componentes de software. Debe ser diseñado e implementado de tal forma que pueda ser reutilizado en muchos

programas diferentes.

La programación orientada a componentes es una rama de la ingeniería de software con énfasis en la descomposición de sistemas ya conformados en componentes funcionales con interfaces bien definidas para la comunicación entre componentes.

Se considera que el nivel de abstracción de los componentes es más alto que el de los objetos y por lo tanto no comparten un estado y se comunican intercambiando mensajes que contienen datos.

2.2.4. Software Framework

Un framework es una plataforma para el desarrollo de aplicaciones de software. La cual proporciona una base sobre la que los desarrolladores de software pueden crear programas para una plataforma específica. Dentro de las responsabilidades de un framework están:

- Hace que sea más fácil trabajar con tecnologías complejas
- Agrupa objetos discretos y componentes en algo más útil
- Obliga al equipo que implementar código de una manera que promueva la codificación coherente, menos errores, y las aplicaciones más flexibles
- Todo el mundo puede probar fácilmente y depurar el código, incluso el código que no escribe

2.3. Tecnologías involucradas

2.3.1. Python

Python es un lenguaje de programación de alto nivel, de propósito general, interpretado y dinámico. Creado por Guido van Rossum, a finales de 1980. Python es un lenguaje multiplataforma, orientado a objetos y de programación funcional.

La filosofía de Python se resume por el documento «The Zen of Python» (PEP 20)[2] como:

- Hermoso es mejor que feo
- Explícito es mejor que implícito
- Simple es mejor que complejo

[2]https://www.python.org/dev/peps/pep-0020/

- Complejo es mejor que complicado
- Plano es mejor que anidado
- Escaso es mejor que denso
- Cuenta la legibilidad
- Los casos especiales no son lo suficientemente especial como para romper las reglas
- Aunque sentido práctico supera pureza
- Los errores nunca debe pasar en silencio
- A menos que explícitamente silenciados
- Ante la ambigüedad, rechaza la tentación de adivinar
- Debería haber una - y preferiblemente sólo una - manera obvia de hacerlo
- Aunque esa manera puede no ser obvia al principio, a menos que seas holandés
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que, la * justo * ahora.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede ser una buena idea.
- Namespaces son una gran idea de fanfarria - Vamos a hacer más de esos!

Python en los últimos años ha estado constantemente en los ranking de los primeros diez lenguajes de programación mas populares según el indice TIOBE[3]. Para septiembre del 2016, se encuentra posicionado como el quito lenguaje mas popular, manteniendo la misma posición con la del año pasado.

[3] http://www.tiobe.com/tiobe-index/

2.3.2. DSL

Una lenguaje especifico de dominio (en ingles Domain Specific Language), es un lenguaje computacional especializado en un dominio particular y solucionar problemas específicos como lo mencionan F. Martin y P. Rebecca en su libro [9]. Los DSL son de dos tipos principales:

- Externo es un lenguaje que se analiza de forma independiente del lenguaje de propósito general de dominio
- Internos son una forma particular de API en un lenguaje de propósito general de dominio

Actualmente hay un montón de DSL alrededor, como por ejemplo «make», «rake», «pip» para describir construcciones de software, y mucha gente encuentra un DSL valiosos, debido a un DSL bien diseñado puede ser mucho más fácil de programar, que una librería tradicional. Esto mejora la productividad del programador, lo cual siempre genera valor. En particular, mejora la comunicación con los dominios, con lo cual se convierte en una herramienta importante para abordar uno de los problemas más difíciles en el desarrollo de software.

2.3.3. JSON

JSON (JavaScript Object Notation) en ingles, es un forma de intercambio de datos ligero y basado en un subconjunto del lenguaje de programación Javascript el cual su especificación[4] se encuentra en la versión 6. JSON usa convenciones parecidas a los lenguajes de la familia C, entre ellos se pueden incluir C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para intercambio de datos. JSON esta construido por dos estructuras una colección de llave/valor y una lista de valores ordenada.

2.3.4. JSON Schema

Esta basado en JSON el cual define una estructura JSON para la validación, documentación y control de interacción de documentos JSON. JSON Schema se basa en el concepto de XML Schema[5] pero con base a JSON. Actualmente esta especificación se encuentra en desarrollo[6].

[4] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
[5] http://www.w3.org/standards/xml/schema
[6] https://tools.ietf.org/html/draft-zyp-json-schema-04

Capítulo 3

Análisis de la participación del humano en bucles de autonomía

3.1. Introducción a la computación autónoma

La comunidad científica propone el uso de bucles de control para abordar la concepción de los sistemas autónomos y auto-adaptativos [18].

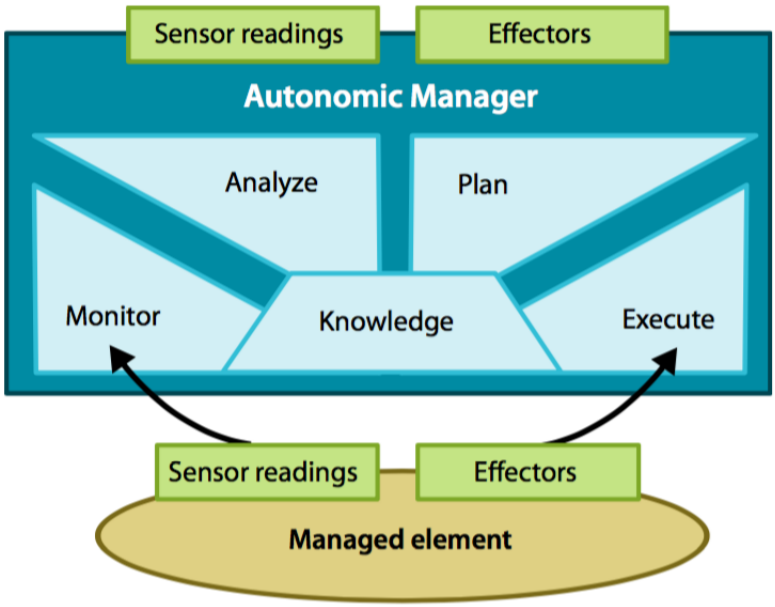

Figura 3.1: Modelo MAPE de referencia de IBM para los bucles de control [16]

Estos bucles de control son los encargados de monitorizar, analizar, decidir y actuar sobre

un sistema base para gestionar de manera continua los eventos que le ocurren al propio sistema o a su entorno, y adecuarse a estas situaciones por sí mismo. Pero abordar la computación autónoma con este enfoque exige que los bucles de control sean conscientes en todo momento de todas las situaciones y casuísticas que podrían ocurrirle a un sistema, y deberían tener una posible respuesta (configuración) para cada una de ellas (problema del mundo abierto) que garantice que el sistema seguirá funcionando en la mejor configuración posible para cada situación (maximizando requisitos operacionales/funcionales y necesidades de los usuarios). Esta estrategia tiene limitaciones claras en este sentido y puede convertirse en inabordable en macro sistemas heterogéneos y altamente dinámicos.

3.2. Involucrando al humano en sistemas autónomos

Como mecanismo para dar una respuesta efectiva a este problema, se está empezando a trabajar en la línea de la incorporación del humano en los bucles de adaptación, con el objetivo de que sea este humano (un operador, un usuario final de un sistema, etc.) el que ayude al sistema en la toma de decisiones en lo que a auto-adaptación o autonomía se refiere. Así, por ejemplo, cuando el sistema identifica de manera autónoma que se requiere realizar una adaptación del sistema, pueden aparecer problemas en la monitorización del contexto, o se identifica cierto grado de incertidumbre (por ejemplo, en las mediciones), conflictos en la satisfacción de objetivos, o algún fallo en la materialización de la adaptación (por ejemplo), el sistema puede llegar a un estado en el que el bucle de adaptación no tiene la respuesta para atender esta petición de reconfiguración, por sí mismo.

Ante estas situaciones, los sistemas se pueden beneficiar de la intervención humana (los cuales pueden tener un criterio más «humano» o pueden tener más información para decidir) mediante:

1. La recepción de información difícil de monitorizar o analizar autónomamente (por ejemplo indicando si hay una situación anómala)

2. La incorporación de decisiones humanas en el proceso de toma de decisiones (por ejemplo para resolver un conflicto de «objetivos»)

3. Utilizando a los humanos como efectores del sistema para ejecutar adaptaciones (por ejemplo en casos en los que una completa automatización no es posible)

Uno de los principales problemas en este trabajo es caracterizar correctamente el tipo de participación del humano que se requiere con un fin doble:

1. Primero, permitir que el sistema sea capaz de seguir operando de la manera más autónoma posible, pidiendo ayuda al humano sólo en aquellas situaciones que realmente sea necesario

2. Que la interrupción o molestia de éste humano sea la mínima posible (pero la necesaria) para evitar el desarrollo de sistemas intrusivos y molestos

Por otro lado, los humanos interactúan en su día a día con estos servicios o sistemas autónomos (por ejemplo servicios ofrecidos por viviendas y edificios inteligentes, los coches semi-autónomos, los servicios de recomendación de música, la asistencia médica personalizada, los servicios ofrecidos por las ciudades, cada vez más inteligentes como el de gestión de tráfico). Pero la interacción con estos sistemas todavía está muy limitada al no haberse concebido y diseñado para interactuar con los humanos en los procesos de auto-adaptación. Es por ello que los sistemas autónomos no suelen ofrecer suficiente información o «explicaciones» al usuario final que ayude a entender en qué situación se encuentran, porqué se comportan cómo lo hacen, provocando la percepción de 'caja negra mágica' de cara a los humanos. Estas situaciones problemáticas surgen porque en la mayoría de ocasiones el diseño de estos sistemas se ha llevado a cabo pensando en conseguir la mayor autonomía posible sin preocuparse de cómo se produce la comunicación/interacción con el usuario. Un claro ejemplo de esto aparece en las diversas implementaciones de coches autónomos que requieren del conductor, «Ahora, tomas tú el volante», cuando se encuentran situaciones de conducción compleja o emergencias. Este problema se conoce como el problema del «handoff», y los diseñadores de coches autónomos aún no han encontrado una manera de cómo hacer que un conductor que puede estar distraído mirando el móvil porque no está conduciendo retome el control del coche en la fracción de segundo que se requiere en caso de emergencia [21, 29].

Por tanto, el papel del usuario en el contexto de los sistemas auto-adaptativos no debe ser el de un mero consumidor de servicios, sino que tiene que constituir el pilar fundamental en todo el proceso de diseño y adaptación del sistema si se quiere conseguir una amplia aceptación de este tipo de sistemas. Aunque las soluciones completamente autónomas han resultado satisfactorias en muchos dominios de aplicación, la capacidad de estos sistemas para proporcionar servicios de confianza en presencia de cambios, ya sea en su entorno (por ejemplo, la disponibilidad de recursos) o en el propio sistema (por ejemplo, aparición de fallos), se ve afectada cada vez más por su creciente complejidad, así como por la naturaleza dinámica e impredecible de los entornos en los que, por lo general, tienen que operar [3]. Por lo tanto, es inevitable que en algún momento el usuario necesite entender qué está pasando o sea necesaria su ayuda, y en este punto una interacción óptima y efectiva con el usuario resulta clave para el éxito del sistema. Por lo que un comportamiento completamente autónomo no es siempre lo más adecuado [4, 12].

Para conseguir la interacción óptima deseada, el diseño de sistemas autónomos debería tener en cuenta los aspectos de interacción con el usuario desde el principio para dar soporte a [25]:

1. Modelos de gestión de la atención del usuario para evitar interrumpir y abrumar a los usuarios

2. Inspeccionabilidad y control del sistema para el post-análisis de fallos, la comprensión, y su recuperación

3. Mecanismos de reporte del comportamiento del sistema que permitan a los usuarios entender lo que el sistema está haciendo con cierta medida de confianza

A pesar de que tener un alto grado de autonomía es una condición necesaria para el crecimiento de los sistemas auto-adaptativos, es importante establecer un equilibrio entre el grado de autonomía y la intervención humana para evitar llegar a resultados no deseados o una mala experiencia de usuario. Como consecuencia, la intervención humana en los sistemas auto-adaptativos puede ser discutida desde dos puntos de vista:

1. El grado de automatización de los mecanismos de adaptación (relacionado con el nivel de autonomía conseguido)

2. La calidad de interacción del sistema con los humanos

Por tanto deberíamos conseguir un grado de autonomía ajustable en tiempo de ejecución basada en el hecho de que la autonomía no siempre puede ser vista como algo estático sino que debe ser preferiblemente dinámico, adaptándose a factores internos y externos tales como la capacidad, experiencia o estado de atención de los humanos, aparición de fallos o conflictos en el sistema, condiciones del entorno, etc. La autonomía debe diseñarse para que se adapte dinámicamente a los humanos por medio de personalización explícita (manualmente por los usuarios) o implícita (aprendiendo del comportamiento de los usuarios). Además, los sistemas autónomos se deben diseñar y concebir pensando en tres retos de interacción esenciales [32]:

1. La transparencia, para hacer que las decisiones de los sistemas y su comportamiento sea entendible por los usuarios finales

2. El control, para que los humanos sean capaz de participar cuando se requiera

3. La experiencia de usuario, para conseguir sistemas altamente personalizables que satisfagan las expectativas de los usuarios

3.3. Framework para la inclusión del humano en bucles de adaptación

Para iniciar a detallar la solución propuesta, se revisara el comportamiento a travez de flujos que se deben tener en cuenta para una participación de un usuario dentro de un bucle de adaptabilidad, a partir de cada uno de los ellos se incluirán entidades las cuales se proponen para manejar las interacciones internas. A su ves se describirá en general su comportamiento en una forma general, para llegar luego a posteriori dar un detalle mas técnico.

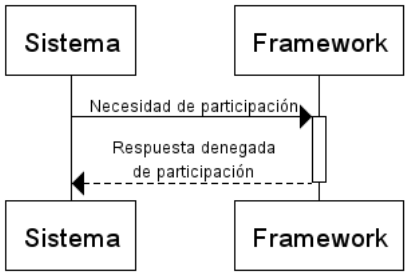

Figura 3.2: Proceso denegado de participación

En la figura 3.2 se muestra como debe reaccionar el framework ante una solicitud de participación la cual puede no estar definida o no cumplir con las condiciones de contexto especificadas. Este tipo de respuestas dará una retroalimentación inmediata al sistema para que pueda tomar acciones, debido a que no es posible que el usuario tome participación del proceso. Es importante notar que en primera instancia enmarcaremos como proceso global al componente framework el cual encapsulara los flujos internos.

A continuación se revisará el flujo en caso de que una solicitud de participación sea exitosa, al igual que la figura 3.2 se encapsulará el flujo a través de el componente framework.

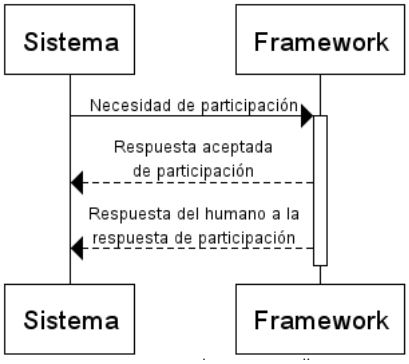

Figura 3.3: Proceso satisfactorio de participación

Al igual que la figura 3.2, la figura 3.3 responde inmediatamente dando una retroalimentación al proceso de solicitud de participación (en este caso una respuesta satisfactoria) dando por terminada la solicitud al sistema. En un segunda instancia retornará la respuesta de la interacción con el ser humano. Dicha interacción se realiza en un segundo plano por lo cual se trata de un proceso en paralelo, el cual llegará en un futuro cercano al sistema. En general, toda solicitud generada por cualquier sistema genera una retroalimentación con la cual puede tomar una decisión inmediata. Por esta razón, a continuación se desglosará las solicitudes internas después de ejecutar la solicitudes de participación anteriormente expuestas.

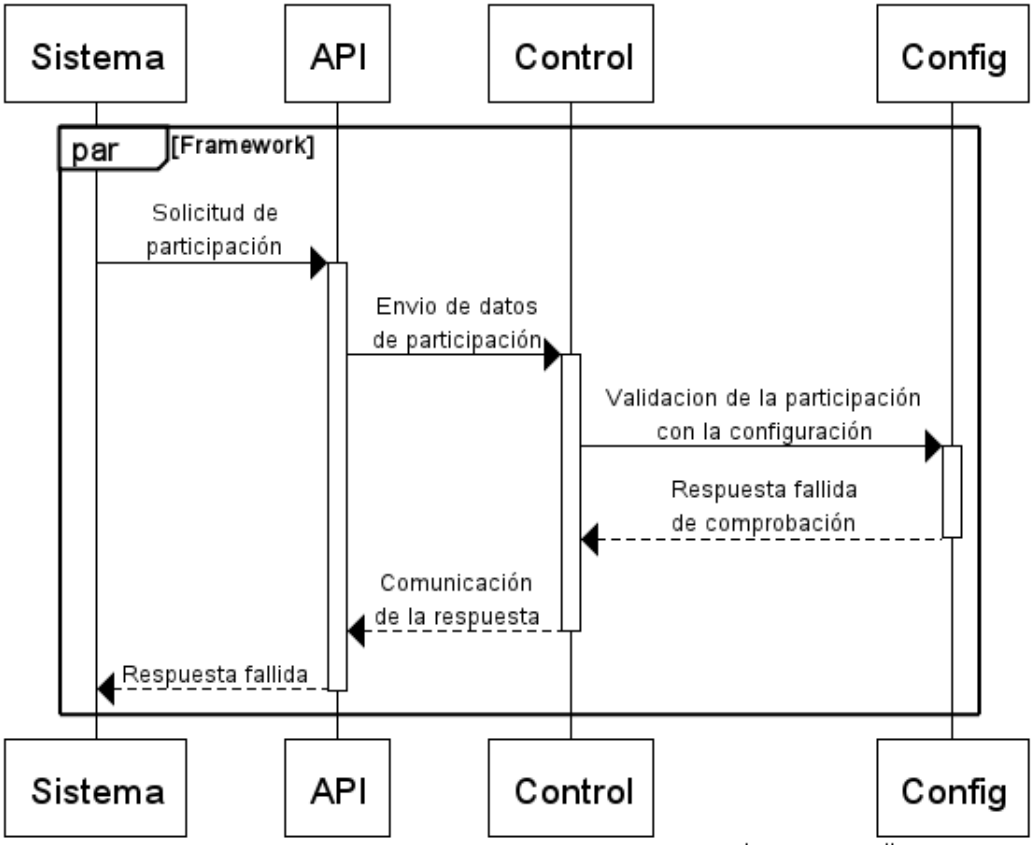

Figura 3.4: Participación fallida por configuración

Basado en el flujo de la figura 3.2, la figura 3.4 describe una participación fallida del sistema debido a que los datos de participación no coinciden con los que se ha descrito en el DSL el cual maneja la configuración. En este caso, dentro del framework se introduce tres nuevos componentes:

- «API» encargado se ser el punto de comunicación con el sistema,
- «Control» encargado de gestionar la participación solicitada y

- «Config» encargado de validar los datos de entrada con la configuración descrita por el programador.

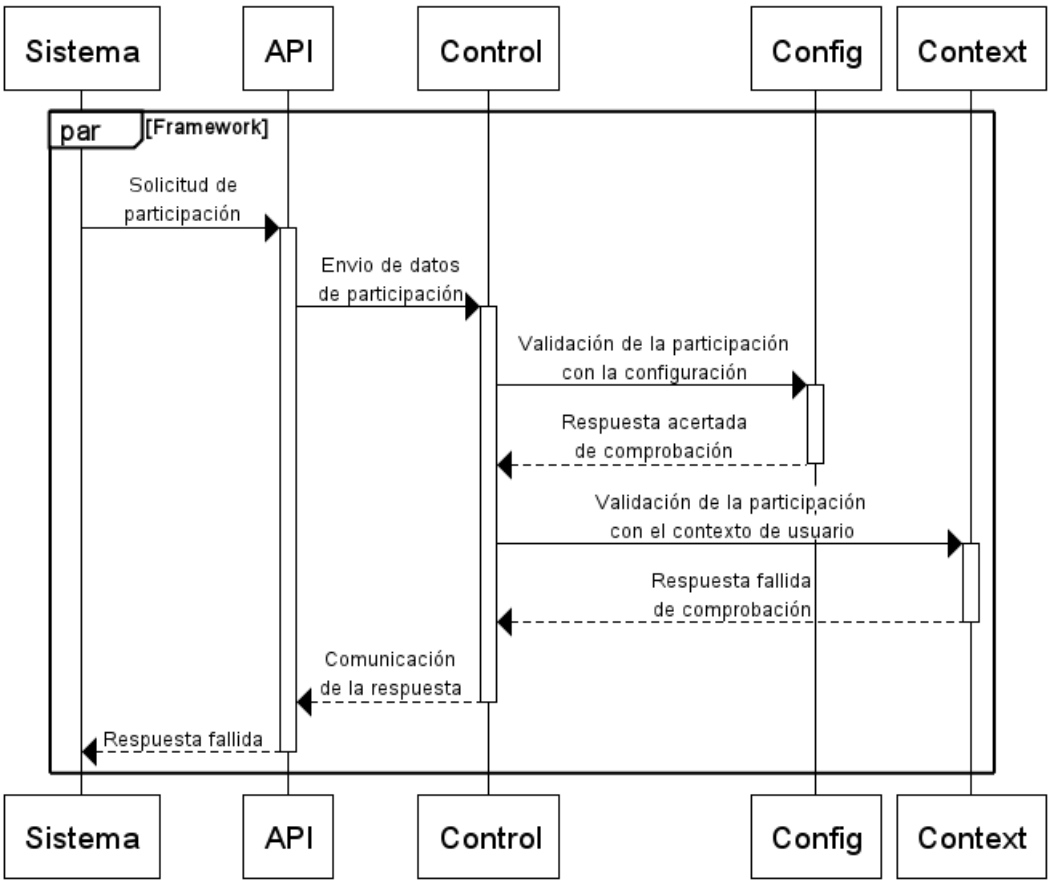

Figura 3.5: Participación fallida por contexto

El flujo de la figura 3.5 extiende a las figuras 3.2 y 3.4, y agrega el componente «Context», el cual se encarga de la validación de contexto con el usuario. Esto permite revisar si la participación solicitada por el sistema encaja con el contexto en el que se encuentra el usuario. En este caso, la validación de la participación con la configuración es emparejada con la configuración actual, pero el contexto del usuario en ese momento no es suficiente para que el framework pueda ejecutar una participación solicitada.

A continuación se explorará el caso en que la participación que solicitó el sistema es aprobada tanto por «Config» como por «Context»:

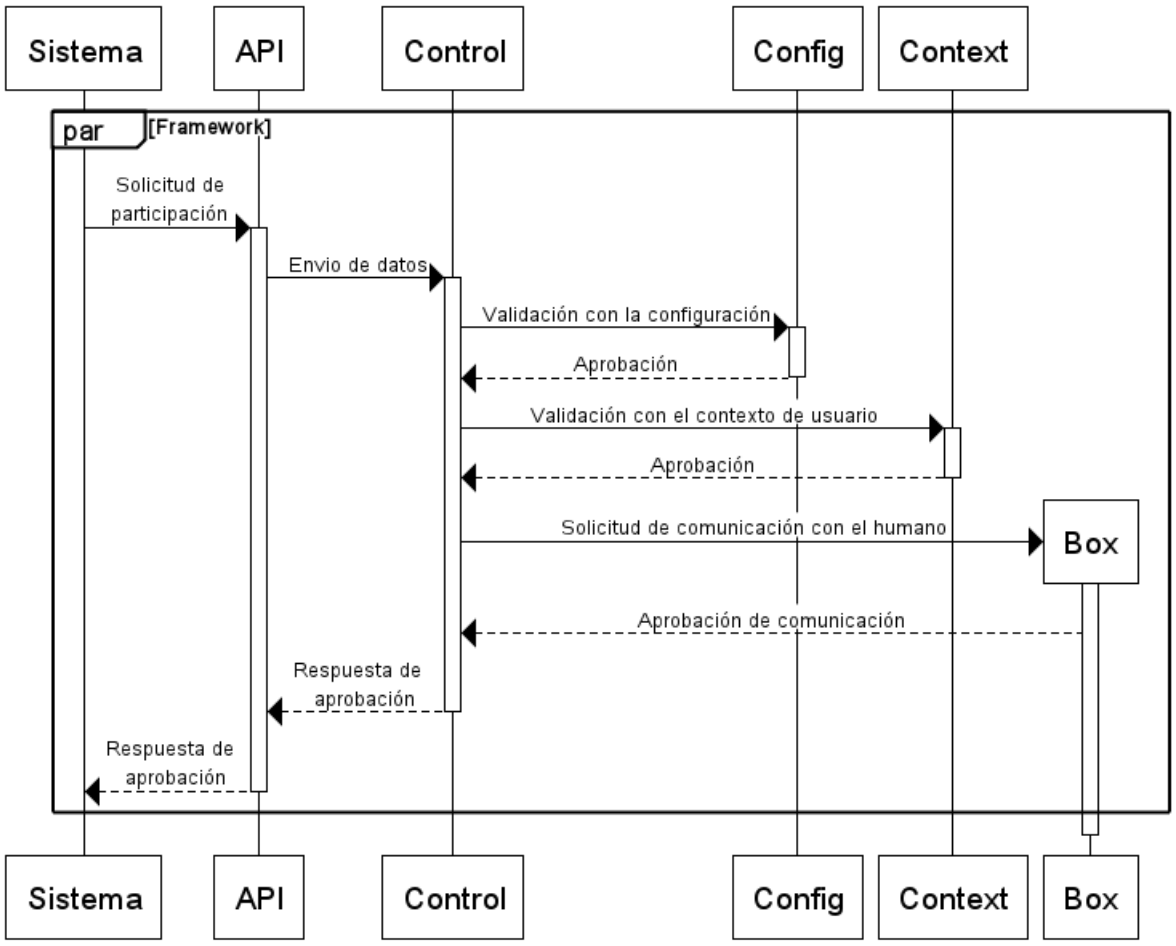

Figura 3.6: Participación satisfactoria sin feedback

En la figura 3.6 el modelo agrega al componente «Box» el cual se encarga de comunicarse con el usuario para comenzar la interacción. En este flujo, «Box» recibe la solicitud y notifica que la comunicación hacia el «Control» ha sido iniciada, y a su vez este notifica al sistema que se inicio un proceso de interacción. En este punto, la comunicación iniciada por el sistema es terminada, pero en segunda instancia queda a espera de la respuesta del usuario, debido a que es un flujo un poco mas complejo. A continuación se detallará más a fondo el proceso interno de «Box» luego después de recibir la solicitud por parte de «Control» de iniciar la interacción.

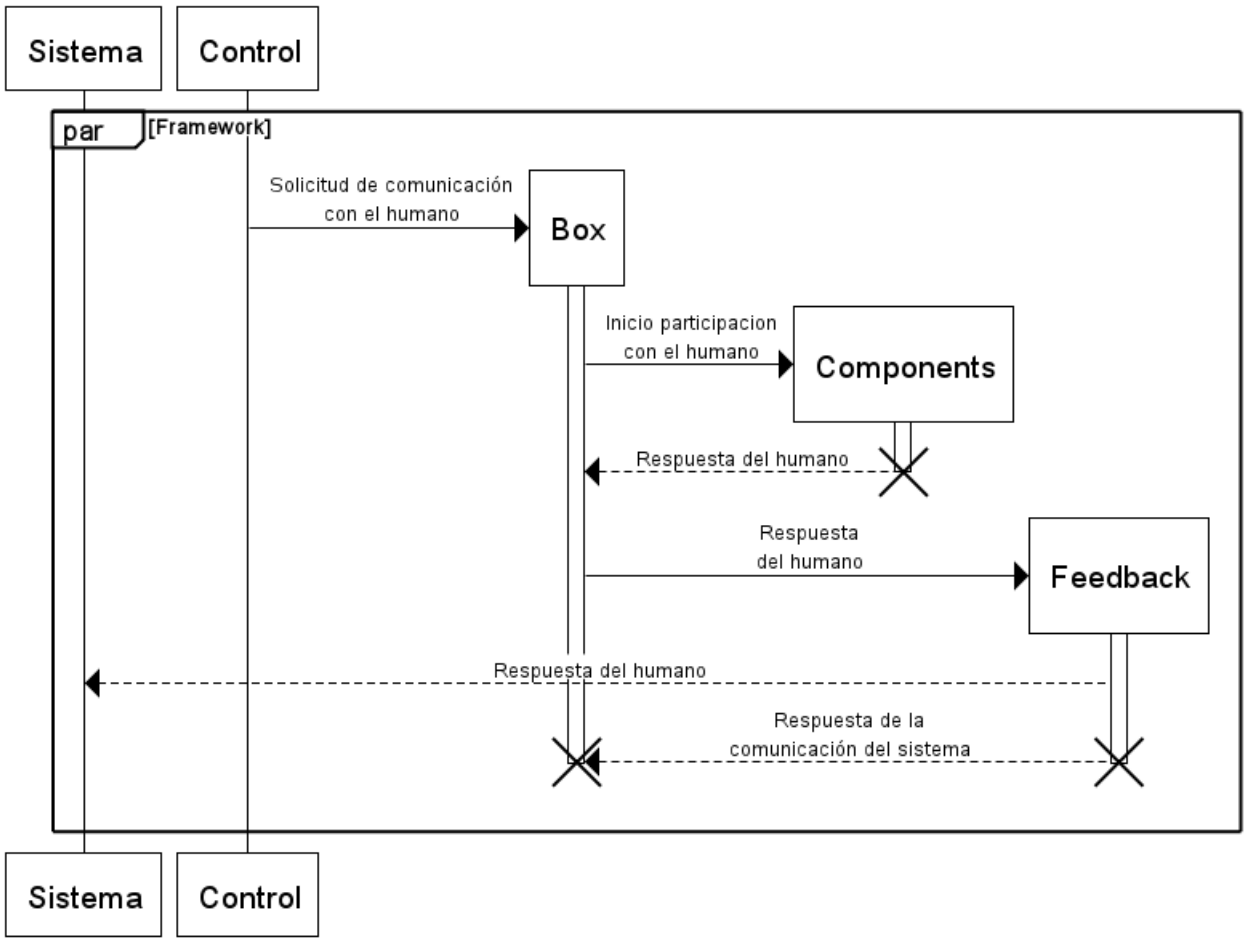

Figura 3.7: Interacción de la participación y retroalimentación al sistema

Como se observa en la figura 3.7 se muestra el flujo dentro de «Box», la cual en una primera instancia como lo muestra la figura 3.6 retorna una respuesta al sistema notificando una iniciación de la interacción. En la figura 3.7 se obvia el flujo expuesto y se muestra el flujo que se realiza en segundo plano y el cual se centra con la comunicación con el usuario. Esta comunicación se realiza a través de «Components» encargado de la interacción directa con el usuario. Al ser finalizada la interacción, una nueva interacción «Feedback» es convocada con la responsabilidad de dar a conocer al sistema la respuesta del humano.

Lamentablemente, no podemos predecir cuanto tiempo le tomara a el usuario actuar frente a una participación, puede que la respuesta sea muy tarde o nunca halla una, y desafortunadamente puede que sea demasiado tarde para que el sistema procese la información. Dado este caso se propone que el sistema al iniciar la solicitud de participación al framework envíe el tiempo máximo el cual el sistema puede esperar una por retroalimentación, esto con el fin de prevenir envió de mensajes los cuales no sean de utilidad para el sistema y para proteger al

mismo framework de tener una participación en ejecución con el usuario indefinidamente.

Capítulo 4

Diseño del Framework de Adaptación

En el capítulo anterior se ha propuesto un framework conceptual para abordar la inclusión del humano en en bucles de adaptación. En este capítulo se realizará el diseño de este framework, identificando los componentes principales y sus dependencias, así como una propuesta arquitectónica. Esto con el fin de dar solución a la problemática planteada en la sección 1.1, y crear un framework de desarrollo que permita a los desarrolladores integrar las aplicaciones actualmente existentes, así como las próximas por crearse.

4.1. Diagrama de componentes

A partir de los flujos de comportamiento abordados anteriormente la figura 4.1 muestra cada uno de los componentes interactúando entre si.

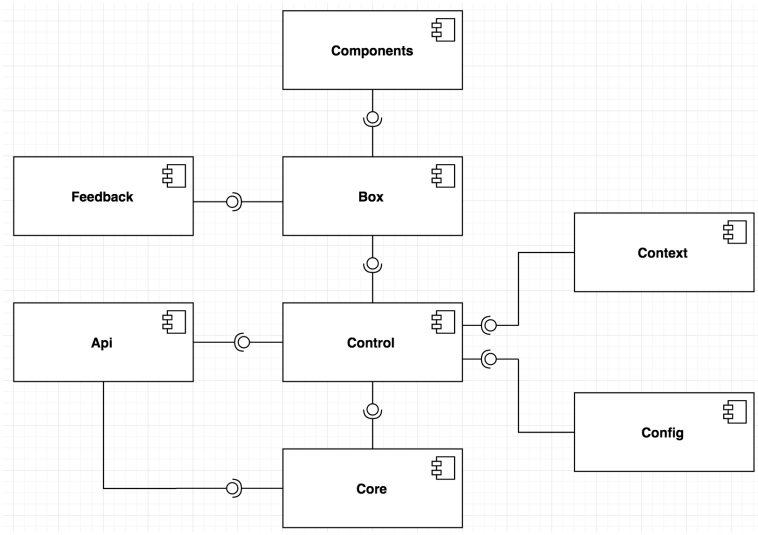

Figura 4.1: Diagrama de componentes

4.2. Arquitectura del framework

A partir del flujo de comportamiento tratado anteriormente y los componentes extraídos, se connota la aparición de distintos módulos de software los cuales toman un rol fundamental dentro del framework. Por lo tanto, se plantea la siguiente arquitectura para dar solución al problema :

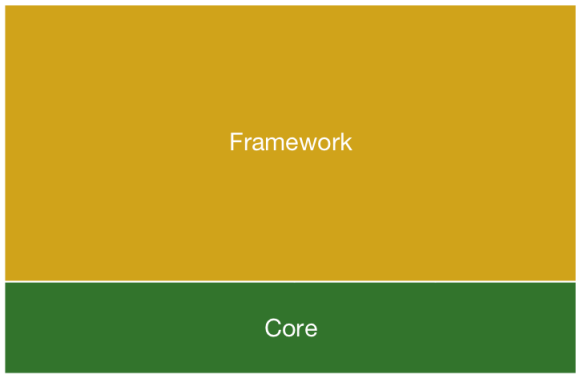

Figura 4.2: Solución propuesta

De forma global se plantea dos componentes principales:

- «Core» como componente principal encargado de sostener en ejecución cada uno de los componentes del framework
- «Framework» como un conjunto de herramientas de para los diseñadores de sistemas que integren al usuario dentro del bucle de adaptación.

A continuación se explorara «Framework» en la figura 4.3 de manera mas especifica.

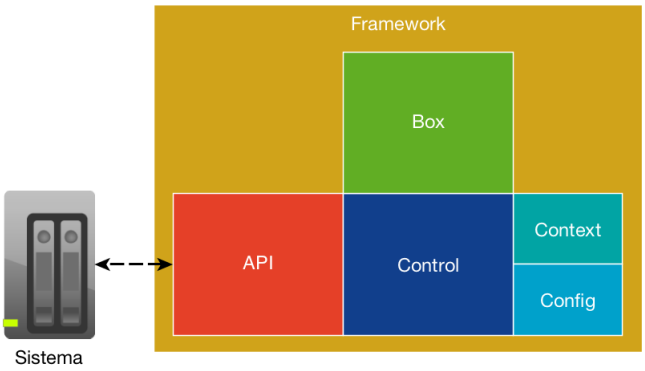

Figura 4.3: Framework

Cada uno de los componentes de la figura 4.3 se comunican con sus vecinos directos, los cuales a su vez siguen el flujo planteado al inicio del capítulo, y en caso de una respuesta satisfactoria se inicia un proceso alterno dentro del componente «Box» el cual se describe en figura 4.4.

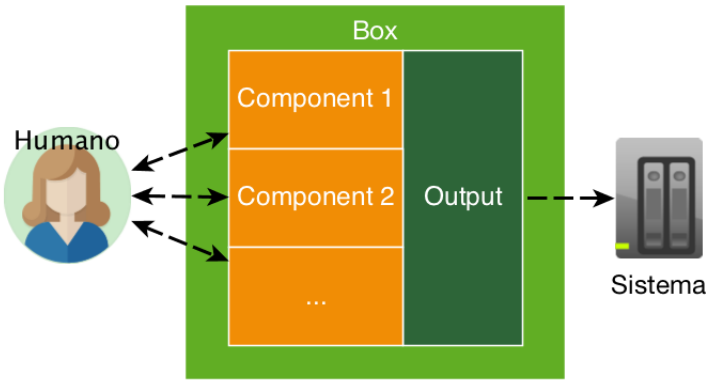

Figura 4.4: Interacción humano - sistema

A partir del concepto de solución propuesto anteriormente, se abordará cada uno de los componentes planteados a continuación.

4.2.1. Core

Este componente nace ante la necesidad de comunicarse con el sistema operativo y la comunicación de cada uno de los componentes de «Framework», por lo cual es el encargado de crear y mantener los mensajes que conectan los procesos del framework. Como se muestra en la figura 4.2, el «Core» funciona de base para todo el framework, razón por la cual es el componente mas importante, puesto a que al ser inicializado, inicializa los demás componentes, delega los mensajes y permite mantener disponible al framework para que los sistemas externos se comuniquen con el framework.

4.2.2. API

Es la interfaz de comunicación con el sistema auto-adaptativo y es la encargada de:

- Recibir solicitudes de participación los sistemas
- Validar en primera instancia la información recibida
- Enviar la solicitud a «Control» para su evaluación

- Dar una respuesta al sistema de si la participación es posible o no

La «API» es un componente el cual es inicializado por el «Core» y el cual se comunica directamente con «Control» a través de mensajes para el intercambio de la solicitud de participación del sistema, y a su vez «Control» envía la respuesta del framework «API» dependiendo si la participación es aceptada o no.

4.2.3. Control

«Control» es el modulo principal del framework, y es el encargado de manejar la comunicación entre la «API» y los módulos de validación, ya sea la configuración «Config», o el contexto del usuario «Context». Dependiendo de la respuesta de los primeros dos, rechaza la solicitud de sistema o la aprueba. Si es aprobada la participación, se comunica con «Box» para iniciar la comunicación con el humano.

4.2.4. Config

Describe el comportamiento el cual realizara el framework, por lo tanto en la inicialización del framework el modulo «Config» validada la configuración actual respecto al DSL especificado para el desarrollo sobre el framework. Esta configuración esta descrita por el programador que usa el framework.

4.2.5. Context

Es un modulo que se encuentra ejecutando en segundo plano y guarda las variables de contexto del humano en tiempo real, por medio del cual «Control» puede acceder a estas variables de forma permanente para validar una solicitud de participación solicitada por parte del sistema. El programador accede a este modulo para que pueda revisar el contexto del usuario y pueda darle los valores a las variables de contexto que el especificó en el archivo de configuración.

4.2.6. Box

Se encarga de la interacción con el humano, actúa como un elemento aislador del sistema con el resto del framework e interacciones actualmente en ejecución, en primera instancia recibe la solicitud de «Control» de ejecutar una o varias participaciones especificas, la cuales son ejecutadas por «Box» recibiendo la respuesta del humano o no. Para resolver uno de los principales problemas el cual es que el humano no realice ninguna retroalimentación al sistema,

la «Api» recibirá un tiempo máximo en que una solicitud de participación puede ser ejecutada en «Box» y con ella dar finalizado el proceso de interacción del humano, garantizando así una respuesta de la participación. De esta forma «Box» garantizara una respuesta para que el sistema pueda tomar una decisión, ya sea que el humano retro alimente o no, «Box» enviara esta información al componente «Feedback» el cual se explicara a continuación.

4.2.7. Component

La figura 4.4 muestra como «Box» contiene la interacción con el usuario y la comunicación con el sistema, y usa «Component» para encapsular cada uno de las aplicaciones que se ejecutan para interactuar con el usuario. «Component» es un modulo el cual le permite a los desarrolladores describir el comportamiento de como se comunican con el usuario, por lo cual es un modulo programable y extensible del framework.

4.2.8. Feedback

Es el encargado de darle la respuesta del humano al sistema, por lo que se ejecuta asincrónicamente después del inicio de la primera solicitud de participación, con el fin de dar el resultado de la interacción del humano cuando esta ha sido aceptada por el framework.

4.3. Diseño de clases

Dado el análisis anterior, aparecen varios módulos los cuales se abordaran mas en detalle en su composición:

4.3.1. Core

Como se menciona en la sección 4.2.1 y se detalla en la figura 4.5,«Core» implementa los modelos contiguos que aparecen en la figuras 4.2 y 4.3. Y como se menciona en la parte arquitectónica se implementan tres métodos básicos «start», «stop» y «restart» para interactuar con el sistema operativo.

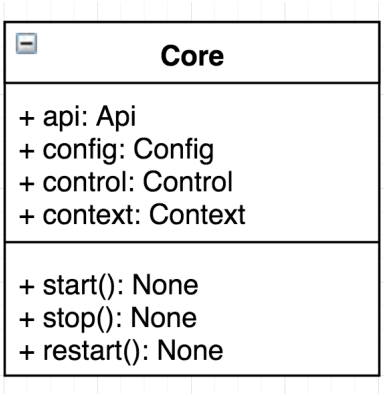

Figura 4.5: Diagrama de clase «Core»

4.3.2. API

Toda solicitud inicial de solicitud de participación un sistema auto-adaptativo, comienza por la clase que aparece en la figura 4.6, hay que tener en cuenta que «API» debe tener una disponibilidad permanente para atender las solicitudes, por lo que la clase declara una variable «running» para detectar si se esta atendiendo a peticiones del sistema y los métodos de clase «start» y «stop» para manejar la disponibilidad. El método «event» se utiliza para recibir las solicitudes de el sistema.

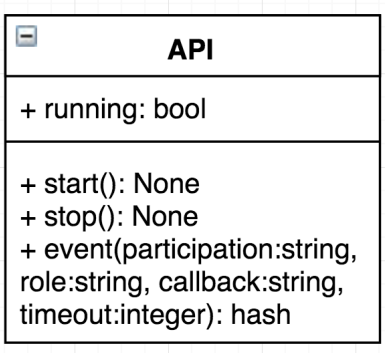

Figura 4.6: Diagrama de clase «Api»

4.3.3. Control

El modulo encargado de la comunicación con los modelos internos, en la figura 4.7 se crean los atributos que guardan el acceso al contexto y la configuración. Este modulo se comporta igual a los anteriores por lo cual es necesario tener los métodos de «start» y «stop» para revisar el estado del objeto, relacionado a esto aparece el atributo «running» para guardar el

estado de la ejecución. Por otro lado implementa el método «trigger» el cual es el encargado de revisar la participación solicitada por el sistema con la configuración y el contexto. Este ultimo método es el encargado de responder a la «API» y en caso de que la validación y configuración pasen ejecutara la participación.

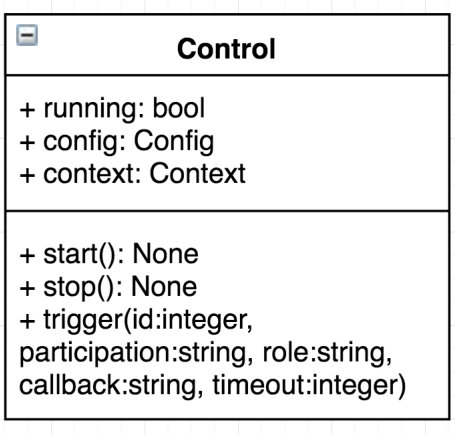

Figura 4.7: Diagrama de clase «Control»

4.3.4. Config

Como muestra la figura 4.8, el modulo «Config» almacena la descripción dada por el programador, y usa los attributos «apps», «contexts», «participations» y «attention_levels». Esta clase implementa solo un método «validateConfig» el cual se encarga de validar la configuración proporcionada.

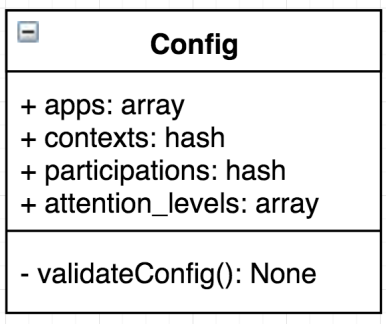

Figura 4.8: Diagrama de clase «Config»

4.3.5. Box

Toda solicitud de participación inicia cuando los datos son enviados a la «API», estos datos son transferidos a «Control» y luego este al realizar las validaciones envía los datos a «Box» de «participation», «role», «actions», «callback» y «timeout» los cuales son almacenados como se muestra en la figura 4.9. A partir de que la solicitud de participación es aceptada se ejecutaran los siguientes métodos:

- «run» inicia el proceso del objeto, y valida si el atributo «timeout» llega a su fin
- «launchProcess» inicia la comunicación con el humano
- «timeoutProcess» termina los procesos, cuando el atributo «timeout» llega a su fin
- «processResponse» cuando finaliza la interacción con el usuario es el encargado de iniciar la comunicación de respuesta.

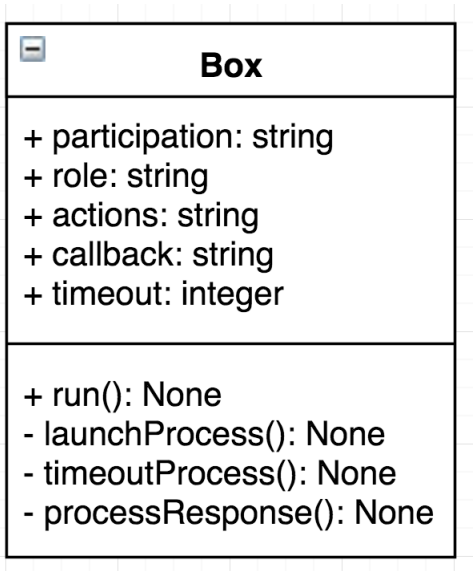

Figura 4.9: Diagrama de clase «Box»

4.3.6. Component

La figura 4.10 muestra el diagrama de clase de «Component», el cual encapsula la ejecución con el usuario. Esta clase tiene los atributos de «participation» y «role» que son enviados por el sistema son almacenados en atributos locales.

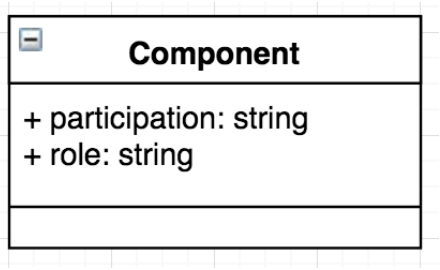

Figura 4.10: Diagrama de clase «Component»

4.3.7. Feedback

Al final del proceso del framework se envía la respuesta del usuario, la figura 4.11 que aparece a continuación muestra el diagrama de clase del modulo «Feedback», que al igual que «Box» implementa el método «run» para inicializar el proceso de envió de respuesta al sistema, el cual tiene como atributos:

- «data» respuesta del usuario ante la participación
- «retry» numero de veces de intento de comunicación
- «timeout» tiempo de expiración de la comunicación
- «callback» url donde se debe comunicar la respuesta del usuario

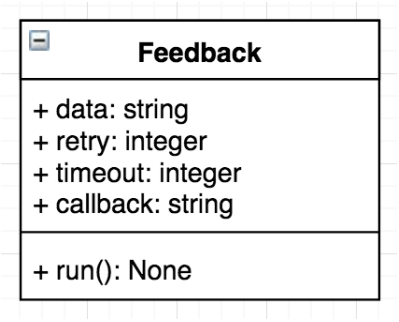

Figura 4.11: Diagrama de clase «Feedback»

Capítulo 5

Implementación del Framework

5.1. Implementación de la solución

A través de los capítulos anteriores se introdujo los conceptos de comportamiento, diseño y arquitectura del sistema. En este capítulo se aborda sobre decisiones en términos de implementación para desarrollar la arquitectura planteada y los tipos de mensajes entre módulos.

5.1.1. Lenguaje de programación

Para el desarrollo del prototipo se eligió Python 3 como lenguaje de programación debido a las siguientes razones:

- Sintaxis sencilla y legible
- Multi-plataforma
- Baja curva de aprendizaje
- Conocimientos previos y buena opción para rápido prototipado.

5.1.2. DSL y validación de la configuración

Como se menciona en la sub sección 4.2.4respecto a la introducción de un DSL, esta validación se propone hacerse a través de un Schema, tal como se muestra a continuación:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "minProperties": 1,
  "patternProperties": {
    "^[a-zA-Z][a-zA-Z0-9_]*$": {
      "type": "array",
      "items": {
        "type": "object",
        "required": ["contexts", "attention_levels"],
        "minProperties": 2,
        "maxProperties": 2,
        "properties": {
          "contexts": {
            "type": "object",
            "minProperties": 1,
            "patternProperties": {
              "^[a-zA-Z][a-zA-Z0-9_]*$": {
                "oneOf": [{
                    "type": "string"
                  },
                  {
                    "type": "integer"
                  }
                ]
              }
            }
          },
          "attention_levels": {
            "type": "object",
            "minProperties": 1,
            "patternProperties": {
              "^[a-zA-Z][a-zA-Z0-9_]*$": {
                "type": "object",
                "minProperties": 1,
                "patternProperties": {
                  "^[a-zA-Z][a-zA-Z0-9_]*$": {
                    "type": "array",
                    "minItems": 1,
                    "uniqueItems": true,
                    "items": {
                      "type": "string"
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Figura 5.1: Schema de validación de configuración

La figura 5.1 se basa en [27, 28], y recibe como entrada la configuración de los eventos que serán notificados por el sistema, así como el nivel de atención, tipo de participación y contexto del usuario para cada uno de los eventos.

5.1.3. Procesos

Para independizar la ejecución de cada modulo, evitar problemas de «bloque mutuo»[1] y «condición de carrera»[2] se utilizan procesos de sistema operativo.

5.1.4. Comunicación entre procesos

Para asegurarse de la comunicación entre procesos, teniendo en cuenta lo expuesto en la sección 5.1.3, se decidió usar tuberías[3] de doble sentido. El protocolo de mensaje es simple, se hace uso de un mensaje de envío de información y uno de respuesta a esta información.

Cada uno de los módulos que aparecen contiguos en la figura 4.3 utilizan tuberías independientes para prevenir problemas de lectura y escritura sobre ellas, y como aparece en 4.2.1 es «Core» el encargado de administrar las tuberías que son utilizados por cada uno de los módulos, razón por la cual esta responsabilidad no es delegada a los módulos.

5.2. Aplicaciones sobre el framework

Para las aplicaciones que integren el framework, uno de los retos es iniciar a integrar a sus usuarios dentro del bucle de sistemas auto-adaptativos, Es así como este apartado aborda los componentes mínimos necesarios a configurar y muestra una estructura básica de organización de código que ayuda a crear rápidamente una aplicación que integra al humano.

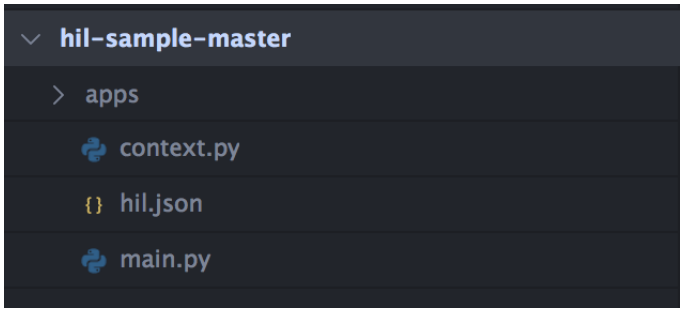

Figura 5.2: Estructura básica de archivos de una aplicación

Para empezar, como se muestra en la figura 5.2, se propone la estructura básica de archivos y directorios a continuación.

En primera instancia «hil-sample-master» representa el nombre del proyecto, dentro de cual se descomponen los siguiente:

[1] https://es.wikipedia.org/wiki/Bloqueo_mutuo
[2] https://es.wikipedia.org/wiki/Condición_de_carrera
[3] https://es.wikipedia.org/wiki/Tuber%C3%ADa_(informática)

- «Apps» Directorio que contiene los componentes que el usuario puede ejecutar

- «Context.py» Archivo que contiene todo el proceso de censado del contexto del usuario

- «Hil.json» Archivo de configuración principal del proyecto, en donde se describe el comportamiento del sistema auto-adaptativo con el framework

- «Main.py» Archivo principal del nuevo proyecto de integración del usuario en el que se integraran todos los componentes anteriormente descritos

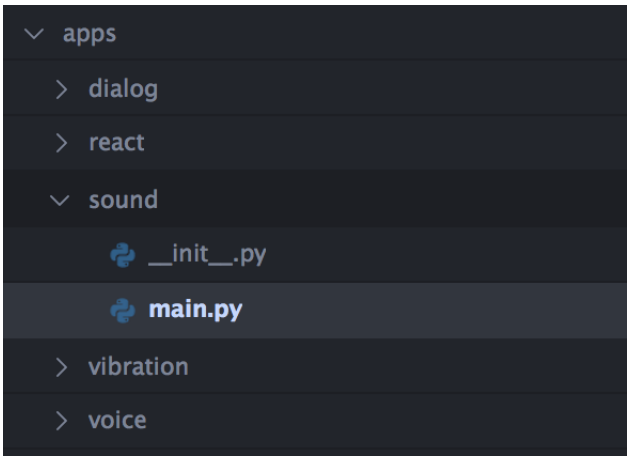

Figura 5.3: Estructura de archivos de los componentes

En la figura 5.3 se muestra más a fondo cómo organizar los componentes por carpetas, donde cada uno contiene un archivo «main.py», el cual contiene la lógica principal de la participación del usuario con el componente.

Esta estructura se propone como una organización básica de proyectos para ayudar a los programadores a tener un desarrollo y despliegue de una manera rápida.

5.2.1. Main.py

```
1  # -*- coding: utf-8 -*-
2
3  from hil import Core
4
5  from context import AppContext
6  from apps.vibration.main import Vibration
7  from apps.voice.main import Voice
8  from apps.react.main import React
9  from apps.sound.main import Sound
10 from apps.dialog.main import Dialog
11
12 class Main:
13   def __init__(self):
14     super(Main, self).__init__()
15     self.components = [Vibration, Voice, React, Sound, Dialog]
16     self.context = AppContext
17
18   def run(self):
19     self.core = Core(self.components, self.context)
20     self.core.start()
21
22 if __name__ == "__main__":
23     main = Main()
24     main.run()
25
```

Figura 5.4: Modulo principal del sistema de ejemplo

La figura 5.4 muestra un de archivo principal de un proyecto ejemplo que esta programado con el framework para comunicarse con el usuario. A continuación se explicara el contenido mas detalladamente. La anterior imagen muestra una clase básica de Python 3 con la diferencia de las lineas:

- Línea 15: Crea una lista con clases las cuales son importadas en las lineas 6, 7, 8, 9 y 10
- Línea 16: Guarda la clases de contexto que se importa en la linea 5
- Línea 19: Instancia el «Core» del framework y pasa como parámetros los componentes y el contexto utilizados
- Línea 20: Inicializa el framework para comenzar a escuchar las peticiones del sistema

En general, los puntos anteriormente descritos se encargan de presentar la configuración que recibe el framework pero en realidad la responsabilidad de la lógica recae en los módulos que se exponen en las secciones a continuación.

5.2.2. Hil.json

Es importante apreciar que uno de los requisitos del framework es que la aplicación que se programe, utilice un archivo de configuración llamado hil con formato json. La razón de esto

es que este archivo contiene la descripción del comportamiento con el sistema externo. En la figura 5.5 se muestra una configuración de ejemplo que será detallada a continuación:

```
{
  "Sound": [
    {
      "contexts": {
        "m_var1": "1",
        "m_var2": "2",
        "m_var3": 3
      },
      "attention_levels": {
        "hight": {
          "role_1": ["Bell"],
          "role_2": ["Dialog"]
        },
        "medium": {
          "role_1": ["Bell"]
        }
      }
    }
  ]
}
```

Figura 5.5: Comportamiento del sistema auto-adaptativo con el framework

El archivo principalmente está compuesto principalmente por un diccionario, donde:

- La **llave** es el nombre del escenario planteado por el sistema (en el caso anterior solo se plantea un escenario llamado «Sound»)
- El **valor** es una lista de casos del escenario (en este caso, de igual manera, sólo se plantea uno). Cada caso está compuesto por dos objetos **diccionario**:
  - El primero es un diccionario con llave «context», y contiene las variables de contexto de usuario
    - La llave es el nombre de la variable de contexto
    - El valor es el que debe tener el contexto para poder ejecutarse en ese escenario.
  - El segundo es un diccionario con llave «attention_levels», y contiene un diccionario con diferentes niveles de atención:
    - La llave es el nivel de atención

- El valor es un diccionario de los roles, y se componen de esta forma:
  - La llave es el nombre del rol el que se va a participar
  - El valor es un listado de componentes a ejecutar

5.2.3. Context.py

Cada proyecto debe implementar un contexto. El framework provee una implementación que puede ser utilizado por cualquier proyecto para su comunicación y para permitir guardar el contexto actual del usuario del sistema.

```
1  # -*- coding: utf-8 -*-
2
3  from hil import Context
4
5  class AppContext(Context):
6    def run(self):
7      self.attention_level = "feedback"
8      self.hands = "steering_wheel"
9      self.user_state = "alert"
10     self.user_capacity = "driver"
11     self.eyes_orientation = "front"
12
```

Figura 5.6: Modulo de contexto de ejemplo

En la figura 5.6 cabe apreciar dos cosas de alta importancia:

- La línea 5 utiliza el módulo «Context» que es importado del framework en la línea 3, y es utilizado como herencia de la clase implementada por la aplicación

- La línea 6, implementa un método de clase llamado «run» (que está por defecto) y este debe ser utilizado por todo contexto que haga uso del framework. Dentro de este método se pueden asignar las variables de contexto de usuario

Es, por tanto, decisión del programador que utilice el framework implementar la lógica de observación de las variables de contexto de usuario. Por defecto, el framework crea las variables de contexto especificadas en el archivo de configuración. Como se muestra en la figura 5.5 en las lineas 5, 6 y 7, las llaves especificadas son aquellas que el framework permite que sean asignadas.

5.2.4. Component.py

Al igual que la observación del contexto, toda aplicación programada con el framework, para que pueda implementar un componente, debe utilizar el módulo definido por el framework.

```
1  # -*- coding: utf-8 -*-
2
3  from hil import Component
4
5  class Sound(Component):
6    def run(self, id, participation, role):
7      print(chr(7))
8      return 'Sound'
9
```

Figura 5.7: Componente ejemplo programado con el framework

La figura 5.7 implementa la herencia del módulo «Component» del framework importado en la línea 3, y utilizado como herencia en la línea 5. Cada nuevo componente implementa el método «run», el cual recibe: el id del framework, la participación con la cual se solicitó, y el rol del usuario. Se deja a decisión del programador la lógica interna de interacción con el usuario. El único requisito del sistema es que al final del proceso, se retorne la respuesta del usuario en el método «run», tal como aparece en la imagen anteriormente mencionada, en la que se retorna «Sound» como respuesta.

5.2.5. Ejecución

Para comenzar a interactuar con cualquier aplicación programada con el framework, es necesario únicamente ejecutar el archivo principal en el terminal con el comando que muestra el algoritmo 5.1.

Algoritmo 5.1 Ejecución del framework

```
$ python main.py
```

Para realizar pruebas necesarias para comprobar el funcionamiento del framework, es posible ejecutar el siguiente comando a través en el terminal.

Algoritmo 5.2 Solicitud ejemplo realizada con el framework

```
$ curl -i -X POST --header "Accept:application/json" http://localhost:8000 -d
'participation=drive-strong-
rain&role=sensor&callback=http://0.0.0.0:3000&timeout=100'
```

El algoritmo 5.2 hace uso del software curl para enviar una solicitud a la aplicación programada. Para especificar los diferentes tipos de casos solo es necesario cambiar valores de los parámetros como: participation, role, callback y timeout.

Capítulo 6

Caso de Estudio

Para demostrar la utilidad de el prototipo de framework el cual se describe en este documento, se analizó el ejemplo planteado en la sección 4.2 de [13], y se usó como caso de estudió del coche autónomo.

6.1. Comprobación de la participación humana

Se enumerara los pasos con los cuales se decidirá si hay una participación del framework con el sistema auto-adaptativo del coche autónomo:

1. Debe ocurrir la necesidad de participación por parte de el sistema auto-adaptativo
2. El sistema autónomo informa al framework del conflicto y el rol necesario de participación del usuario
3. El framework comprueba como tiene que participar el humano:
   a) Mira la situación del usuario (contexto) para decidir el nivel de atención requerido (parte pro activa) y dar feedback al usuario
   b) Dependiendo del rol y el contexto, decide los mecanismos de interacción con el usuario adecuados a la situación (para pasar el control al humano)
4. Por último la entrada del usuario se devuelve al sistema autónoma

6.2. Niveles de atención y tipos de participación

A continuación, se revisara los niveles de atención que puede tener un usuario frente a caso de estudio planteado, y los tipos de participación que tendrá el usuario dependiendo de

su nivel de atención.

Alertar al usuario (retroalimentación)	**Interactuar con el usuario (acción)**
Proactivo / Primer plano Alertas por medio de sonidos y habla.	**Reactivo / Primer plano** Conducción mediante el volante, diálogos usando comandos de voz
Proactivo / Ligeramente notable Aviso usando vibración, sonidos cortos y pantalla.	**Reactivo / Ligeramente notable** Diálogos de aceptar o rechazar mediante pantalla visual y botones
Proactivo / Fondo Indicadores de estado en panel visual	**Reactivo / Fondo** Presionar frenos, acelerador, mover volante

Cuadro 6.1: Nivel de atención y tipos de participación

En la 6.1 se muestra agrupado por columnas las diferentes tipos de participación se plantean por cada nivel de atención, es importante notar que el tipo de participación esta directamente relacionado con el nivel de atención, por lo cuales se agrupan por el concepto de tipo de acción la cual es posible realizar entre sistema y el usuario.

6.3. Escenarios propuestos

Se plantea los siguientes escenarios en los cuales el sistema auto-adaptativo necesite incluir al usuario dentro de su bucle de adaptación y con esto poder evaluar el framework. En cada escenario se denotara los posibles roles el cual el usuario puede tener en el momento, el contexto el cual puede estar sucediendo alrededor de el y en caso de que halla un emparejamiento por una solicitud de un sistema, se mostraran las transiciones de participación de debe tener.

6.3.1. No es posible conducir autónomamente con lluvia fuerte

El usuario en este caso debe tomar el control del vehículo y conducirlo.

6.3.1.1. Escenario

Rol del usuario	**Contexto**	**Transiciones en tipos de participación**
Sensor, tomador de decisiones, actuador	1. Manos = Volante 2. Estado Usuario = Atento 3. Capacidad Usuario = Conductor	1. Proactivo / Ligeramente notable 2. Reactive / Primer plano
	1. Manos = Volante 2. Estado Usuario = Distraído 3. Capacidad Usuario = Conductor	1. Proactivo / Primer plano 2. Reactivo / Primer plano

Cuadro 6.2: Problema de conducción sobre lluvia

6.3.1.2. Configuración en el framework

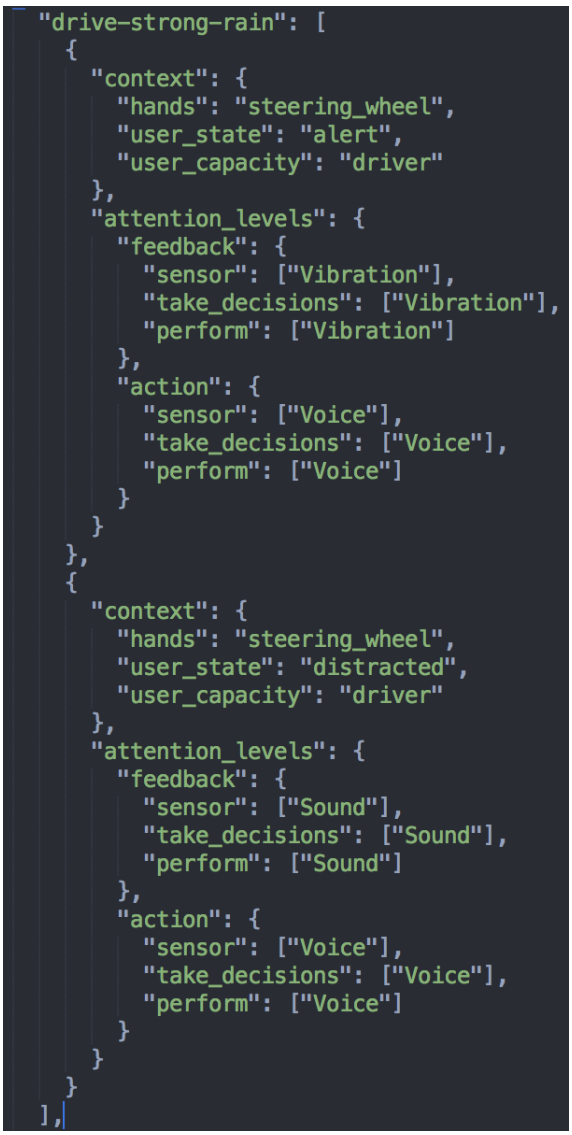

Figura 6.1: Configuración en el framework de el escenario «No es posible conducir autónomamente con lluvia fuerte»

6.3.2. Incerteza en cruzar una intersección

El coche le indica al usuario cuando que va a pasar sobre una intersección el problema de saber si puede cruzar o no, el usuario puede presionar el acelerador para indicar una oportunidad para cruzar.

6.3.2.1. Escenario

Rol del usuario	Contexto	Transiciones en tipos de participación
Tomador de decisiones	1. Capacidad Usuario = Conductor	1. Proactivo / Ligeramente notable 2. Reactivo / Fondo

Cuadro 6.3: Problema de cruce de una intersección

6.3.2.2. Configuración en el framework

```
"crossing-an-intersection": [
 {
    "context": {
      "user_capacity": "driver"
    },
    "attention_levels": {
      "feedback": {
        "take_decisions": ["Vibration"]
      },
      "action": {
        "take_decisions": ["Voice"]
      }
    }
  }
],
```

Figura 6.2: Configuración en el framework de el escenario «Incerteza en cruzar una intersección»

6.3.3. Incerteza en reconocer una señal de trafico temporal (de limite de velocidad)

Sistema alerta al usuario ya que no puede reconoce la señal, y el usuario indica el significado de la señal (mediante freno o acelerador).

6.3.3.1. Escenario

Rol del usuario	Contexto	Transiciones en tipos de participación
Sensor	1. Estado Usuario = Atento	1. Proactivo / Primer plano 2. Reactivo / Fondo

Cuadro 6.4: Problema al reconocer una señal de trafico

6.3.3.2. Configuración en el framework

Figura 6.3: Configuración en el framework de el escenario «Incerteza en reconocer una señal de trafico temporal»

6.3.4. Dificultad en reconocer señales de un policía

El sistema tiene problemas para reconocer que significa la señal de un policía

6.3.4.1. Escenario

Rol del usuario	Contexto	Transiciones en tipos de participación
Sensor	1. Mirada Usuario = Hacia Adelante	1. Proactivo / Ligeramente notable 2. Reactivo / Ligeramente notable
	1. Mirada Usuario= No Hacia Adelante	1. Proactivo / Primer plano 2. Reactivo / Ligeramente notable

Cuadro 6.5: Problema de reconocer las señales de un policía

6.3.4.2. Configuración en el framework

Figura 6.4: Configuración en el framework de el escenario «Dificultad en reconocer señales de un policía»

6.3.5. Dificultad en gestionar semáforos rotos

El coche le indica al usuario que no puede decidir sobre cuando pasar, y el usuario puede indicar una oportunidad para pasar.

6.3.5.1. Escenario

Rol del usuario	Contexto	Transiciones en tipos de participación
Tomador de decisiones	1. Capacidad Usuario = Conductor 2. Estado Usuario = Atento	1. Proactivo / Ligeramente notable 2. Reactivo / Ligeramente notable

Cuadro 6.6: Problema tratando de gestionar semáforos rotos

6.3.5.2. Configuración en el framework

Figura 6.5: Configuración en el framework de el escenario «Dificultad en gestionar semáforos rotos»

Capítulo 7

Resultados

En resumen, en el presente documento se plantean dos resultados tangibles: el primero es un prototipo de framework para integrar al humano dentro de la toma de desiciones, y el segundo un caso de estudio que sirve como guía para permitir la integración de software ya existente o próximo a desarrollarse.

7.1. Framework

Para este prototipo se decidió asignarle el nombre de «HIL», que en inglés quiere decir «Human In the Loop». El código fuente de este framework se puede encontrar en https://github.com/fvioz/hil.

7.2. Caso de estudio

Como caso de estudio y guía de utilización se creó un proyecto alterno el cual aprovecha todas las ventajas del framework. Dicho proyecto se puede encontrar en https://github.com/fvioz/hil-sample.

Capítulo 8

Conclusiones

Después de realizar este proyecto se concluye lo siguiente:

- La integración del usuario en el proceso de un bucle de adaptación provee al sistema auto-adaptativo más información del contexto, y permite que éste tenga mayor eficacia a la hora de tomar de decisiones
- Desafortunadamente, no sucede que en todos los casos el usuario produce una retro-alimentación lo suficientemente oportuna con la cual el sistema pueda incluirla en la toma de una decisión. Por consiguiente, se identifica la necesidad de que el sistema auto-adaptativo informe al framework el tiempo máximo de espera a una solicitud
- Como resultado se presenta un framework que incluye al humano en la toma de decisiones en sistemas auto-adaptativos, teniendo en cuenta un tiempo máximo de espera para su respuesta.

Referencias

[1] L. Barkhuus and A. Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. 2003.

[2] V. Bellotti and K. Edwards. Intelligibility and accountability: Human considerations in context-aware systems,. *Human-Computer Interaction 16*, pages 193,212, 2009.

[3] Javier Cámara, Gabriel A. Moreno, and David Garlanx. Reasoning about human participation in self-adaptive systems. In IEEE Press, editor, *In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '15).*, pages 146,156, 2015.

[4] B.H.C. Cheng. Software engineering for self-adaptive systems: A research roadmap. *Springer LNCS, pp*, pages 1–26, 2009.

[5] K. Cheverst. Exploring issues of user model transparency and proactive behavior in an office environment control system. *UMUAI 05,*, 15(3-4),:235,273, 2005.

[6] C. Dorn and R. N. Taylor. Coupling software architecture and human architecture for collaboration-aware system adaptation. in d. notkin, b. h. c. cheng, and k. pohl, editors, 35th international conference on software engineering, icse '13, san francisco, ca, usa, may 18-26, 2013. *IEEE / ACM*, pages 53–62, 2013.

[7] M. Dzindolet. The role of trust in automation reliance. *International Journal of Human-Computer Studies 58*, pages 697,718, 2003.

[8] C. Evers, R. Kniewel, K. Geihs, and L. Schmidt. The user in the loop: Enabling user participation for self-adaptive applications. *Future Generation Computer Systems*, 34: 110,123, 2014.

[9] Martin Fowler and Rebecca Parsons. *Domain-Specific Languages*. Addison-Wesley Signature Series, 2010.

[10] A.G. Ganek and T.A. Corbi. The dawning of the autonomic computing era. *IBM System Journal*, 42(1):5–19, 2003.

[11] D. Garlan. A 10-year perspective on software engineering self-adaptive systems. SEAMS, 2013.

[12] W. Gibbs. Considerate computing. *Scientific American*, 292(54, 61), 2005.

[13] Miriam Gil, Vicente Pelechano, Joan Fon, and Manoli Albert. Designing the human in the loop of self-adaptive systems. UCAmI, 2016.

[14] Miriam Gil, Vicente Pelechano, Joan Fons, and Manoli Albert. Involucrando al humano en el bucle de control de sistemas auto-adaptativos. XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD), 2016.

[15] P. Horn. Autonomic computing: Ibm's perspective on the state of information technology. *Technical report, IBM Corporation*, October 2001.

[16] IBM. An architectural blueprint for autonomic computing. URL http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf.

[17] IBM. Practical guide to the e to the ibm autonomic computing toolkit, April 2004. URL https://www.redbooks.ibm.com/redbooks/pdfs/sg246635.pdf.

[18] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36:41–50, 2003.

[19] S. Lightstone. Seven software engineering principles for autonomic computing development. *Innovations in Systems and Software Engineering 3*, pages 71,74, 2007.

[20] B.Y. Lim and A.K. Dey. Assessing demand for intelligibility in context-aware applications. *Ubicomp 09*, pages 195,204, 2009.

[21] John Markoff, 01 2016. URL http://www.nytimes.com/2016/01/18/technology/driverless-cars-limits-include-human-nature.html.

[22] B.M. Muir. Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems, ergonomics.

[23] M.R. Nami and K. Bertels. A survey of autonomic computing systems. pages 26–30, 2007.

[24] Stefan Poslad. Autonomous systems and artificial life. *Wiley. pp.*, ISBN 978-0-470-03560-3:317–341, 2009.

[25] D.M. Russell, P. Maglio, R. Dordick, and C Neti. Dealing with ghosts: Managing the user experience of autonomic computing. *IBM Sys*, 42:177–188, 2003.

[26] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *Transactions on Autonomous and Adaptive Systems 4*, pages 1,42, 2009.

[27] JSON Schema. Core definitions and terminology, . URL http://json-schema.org/latest/json-schema-core.html.

[28] JSON Schema. Interactive and non interactive validation, . URL http://json-schema.org/latest/json-schema-validation.html.

[29] Siraj Ahmed Shaikh and Padmanabhan Krishnan. A framework for analysing driver interactions with semi-autonomous vehicles. *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, 105(85,99).

[30] C. Shin, A.K. Dey, and W. Woo. Mixed-initiative conflict resolution for context-aware applications. *ACM Request Permissions, New York, New York, USA*, UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing:262, 2008.

[31] M. Söllner, A. Hoffmann, H. Hoffmann, and J.M. Leimeister. Towards a theory of explanation and prediction for the formation of trust in it artifacts. *10th Annual Workshop on HCI Research in MIS, Shanghai, China*, pages 1–6, 2011.

[32] S. Stumpf, M. Burnett, V. Pipek, and W.K. Wong. End-user interactions with intelligent and autonomous systems. in: J. a. konstan, e. h. chi k. höök (eds.), chi '12 extended abstracts on human factors in computing systems. *ACM. ISBN 978-1-4503-1016-1*, pages 2755–2758, 2012.

[33] N. Villegas, H.A. Müller, and G. Tamura. On designing self-adaptive software systems. *Revista ST*, 9(18):29–51, 2011.

[34] Xiaohui Zhang. A user's perspective of design for context-awareness. volume In Proceedings of the 13th international conference on Ubiquitous computing, pages 531,534. UbiComp '11, 2011.