ASÍ FUNCIONA SIRIKIT

Julio César Fernández Guías

Siri es el asistente de voz de Apple, que hasta ahora solo podía interactuar con apps del sistema. Y es una de las características que, desde que fue lanzado, los desarrolladores han pedido poder integrar en sus apps. Pues bien, ha pasado mucho tiempo pero al final el mismo año en que otros como Google o Microsoft abren sus asistentes a apps de terceros, Apple hace lo propio. **Vamos a repasar cómo se funciona esta integración**.

Siri, solo para algunas cosas

Eso es lo primero que hemos de entender: SiriKit no puede usarse para cualquier cosa. Al igual que la multitarea en iOS, que tiene solo unas determinadas funciones permitidas por el bien de la estabilidad y autonomía energética, **SiriKit solo puede ser integrado en determinados casos, o lo que Apple llamada: dominios**. Además, lo primero a tener presente es que **Siri requiere permiso para ser activado en nuestra app, al igual que pasa cuando queremos usar el micrófono por primera vez o enviar notificaciones**. En dicho mensaje, hemos de registrar los dominios pertinentes para que el usuario sepa qué podrá hacer Siri con nuestra app.

Los servicios que podemos incluir son: **envío de mensajes en apps de mensajería, establecer llamadas de voz sobre IP, pagos, ejercicios, búsqueda de fotos o encargar un recorrido** (es decir, pedir un Uber, un taxi…).

La forma de interactuar con Siri es muy diversa. Podemos hacerlo por pasos o podemos hacerlo todo dentro de un contexto, y por lo tanto, **hay que programar la forma en dichos contextos serán interpretados por Siri para saber qué acción ha de realizar** a través de la correspondiente extensión que hagamos para nuestra app.

En una app de mensajería, podríamos decir: "envía un mensaje de Whatsapp" y Siri nos preguntaría "a quién". Seguido a esto contestaríamos "A mi mujer". La respuesta obvia es aquella en la que Siri pide lo que falta para completar la acción: el mensaje. "¿Qué mensaje por Whatsapp quieres enviar a tu mujer?". Y entonces le diríamos el contenido del mismo. Pero también tenemos otras formas de invocar la misma orden, sin seguir tantos pasos (y todas tienen que funcionar). Podemos decir "Envía a mi mujer un mensaje de Whatsapp" y estaríamos realizando los dos primeros pasos anteriores en uno solo y Siri nos preguntará por el mensaje. O podemos hacerlo todo a la vez diciendo: "Whatsapp a mi mujer, voy de camino" y estaríamos dando las instrucciones en modo abreviado. Tal vez en modo más gramatical: "Envía un mensaje por Whatsapp a mi mujer diciéndole que voy de camino". No solo eso: podríamos querer ser totalmente amables con Siri y decirle: "Oye Siri, perdona, hazme el favor de envíarle un mensaje por Whatsapp a mi mujer en el que le digas que ya voy de camino". **Todas estas opciones han de ser interpretadas para llegar a la misma acción**.

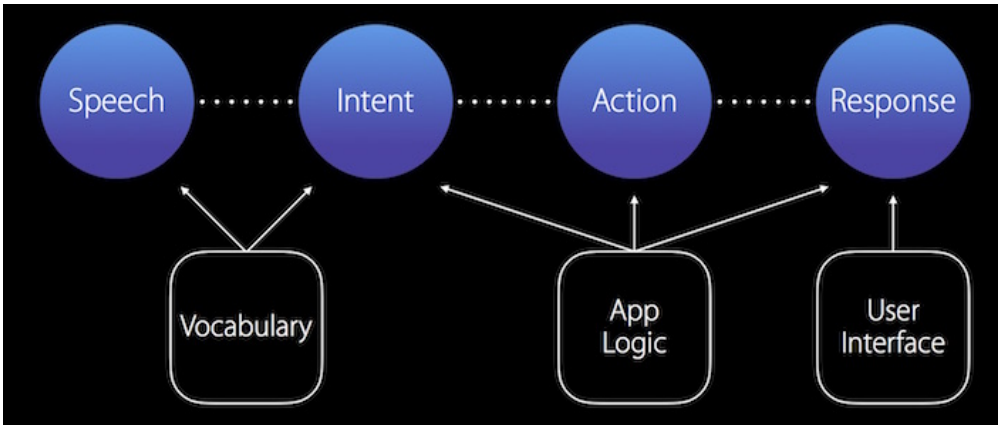
De igual forma, la experiencia con Siri no es la misma en función del contexto en que invoquemos a esta. Si somos invidentes, Siri nos dará mucha más información, si lo hacemos dejando pulsado el botón Home nos dará menos *feedback* de voz y más visual porque sabe que estamos mirando la pantalla. Y si lo usamos en Apple Car o diciendo "Oye Siri" para activarlo, sabrá que no miramos (o no debemos) y nos hablará más. **Todo esto, obviamente hay que tenerlo presente a la hora de desarrollar la experiencia con Siri**.

Al final, llegamos al gran problema de abrir Siri a terceros porque hemos de tener presente la calidad, la consistencia de la experiencia así como una forma fácil en que las apps adopten a Siri y que al final, el

usuario, **no pueda percibir distinción alguna entre el uso de una funcionalidad integrada en el sistema o una a través de una app de terceros**. Todo tiene que ser fluido, como dirían los americanos, una experiencia *seamless* (integrada).

Los pasos esenciales

Todo experiencia de SiriKit se compone de cuatro pasos: el habla, que es traducida a texto para conocer las instrucciones, la intención, una respuesta que se genera desde el habla y que define qué hemos de realizar, la acción, que supone aquello que se envía a través de una extensión a la app que tiene que generar esta y la respuesta, aquella que la app devuelve y que Siri nos trasladará como *feedback* de vuelta a nuestra acción.



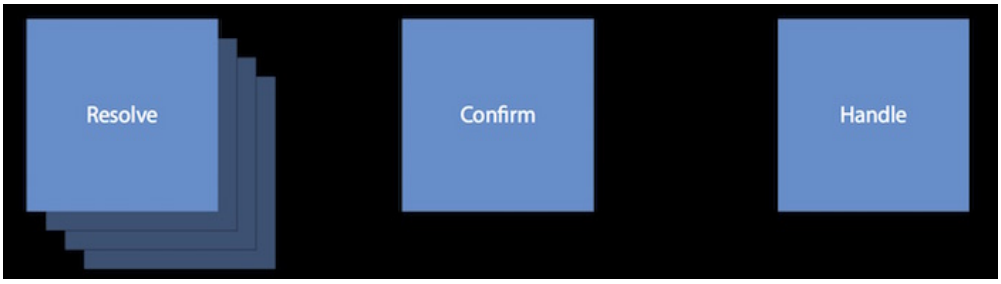
Cuando decimos: “Envía un mensaje de Whatsapp a mi mujer”, la compresión de vocabulario que se genera desde la parte de habla, encuentra la preposición “a” y entiende que aquello que se encuentra tras esta es el destinatario de la acción que quiere realizar, en este caso “mi mujer”. Buscará en su base de datos de contactos la información relativa y la almacenará en el contexto apropiado para definir la intención que se enviará como acción. Tras esto, Siri comprueba que le falta información y la pide: “¿Qué mensaje quieres a enviar a…?” y ahí meterá el nombre real de tu mujer. “Voy de camino” le diríamos. Ahora el sistema ya conoce los contenidos de la intención: **el dominio es “mensajes”, la intención en sí es “enviar un mensaje”, el destinatario es “tu mujer” y el contenido “voy de camino”**. Con estos datos, es donde la lógica de la app entra en funcionamiento a través de recoger la intención y los datos necesarios para realizar la acción. La lógica de la app hace en este caso de controlador de toda la operación.

Una vez realizada la acción, **Siri devuelve una respuesta a través de la extensión de la app, que a su vez devuelve una interfaz de usuario que puede ser personalizada**. Con esto, quedaría completada la operación y el mensaje estaría enviado. El resto de opciones, son combinaciones o permutaciones del vocabulario interpretado que daría lugar a la intención, la acción, su respuesta y posibles variaciones de varias respuestas que puede haber entre la app y Siri hasta tener todos los parámetros necesarios válidos.

Integración técnica

Lo primero que hay que entender es el concepto del dominio, aquel que establece el tipo de operación principal y que en esta primera versión está limitado a unos concretos: **mensajería, llamadas de voz sobre IP, petición de recorridos (taxi, Uber…), pagos, ejercicios o búsqueda de fotografías**. El siguiente es el concepto de las intenciones, o grupos de acciones conocidos que se pueden realizar dentro de cada uno de esos dominios: enviar, realizar, pedir, buscar…

Los pasos esenciales del proceso son 3: la resolución, que puede contar con muchos pasos hasta que sucede, la confirmación y por otro lado la gestión de la comunicación.

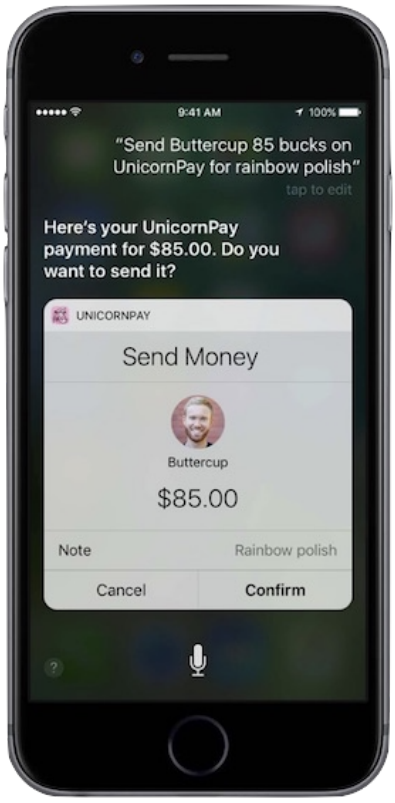


Cualquier petición que realicemos, será procesada por Siri para obtener todos aquellos parámetros que necesite. Imaginemos la petición: "Oye Siri, envía a Antonio 10 euros con Twip por el cine de ayer". **El análisis en conjunto de las palabras "envía" e indicar una cantidad de dinero activa el dominio "pagos"**. La intención será: "enviar un pago" (es como si fuera el nombre de la función que va a ejecutarse). Los parámetros: la app es "Twip", el destinatario del pago "Antonio" (lo buscará en la agenda y si no lo reconoce Siri por su cuenta pedirá más detalles para tener más claro este parámetro), la cantidad del pago es 10 y la divisa "EUR". Además, tiene en cuenta una nota: "por el cine de ayer". Con ese contenido, **ya sabe que acción tomar y podrá llamar a la lógica de nuestra app, a la extensión que hayamos declarado en ella que recibe estos parámetros y que tiene como función de entrada "enviar un pago"**.

En todo este proceso hay algo muy importante: `NSUserActivity`. **La forma en que podemos declarar datos que ayuden a Siri a entender mejor los contextos y lo que va a usarse para proporcionar una mejor experiencia**. Al igual que haríamos con *Handoff* donde declaramos una actividad que puede continuar en otro dispositivo, el sistema usa este mismo sistema para crear la comunicación entre Siri y nuestra app así como, si declaramos datos de búsqueda para la app, Siri podrá buscar datos necesarios para la petición en los propios datos de la app y no del sistema. Por ejemplo, **si nuestra app tiene una agenda de contactos que difiera de la del sistema, Siri podrá buscar en estos datos**. Esto pasa, por ejemplo, si pedimos que en Skype que inicie una nueva llamada a un contacto de la agenda de Skype que no está en la agenda del sistema (lo cual es muy habitual).

Una vez la acción es enviada a nuestra app, esta tiene que resolver cada uno de los parámetros de forma individual con una función `success(with:)`. **Vamos a suponer una orden "llama por Skype a Julia Torres"**. En dicho contexto pueden pasar varias cosas que hagan que no se procese alguno de los parámetros y nuestra app requiera más información del usuario (y de Siri), tal vez con los siguientes casos de uso:

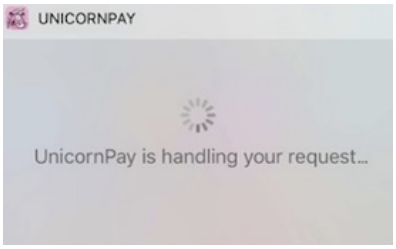
- El parámetro que hemos enviado como contacto “Julia Torres” no existe en los contactos de Skype (ni en los del sistema). En dicho caso **Siri tendrá que dar una respuesta negativa a nuestra petición indicando que dicho contacto no ha sido encontrado**. Podríamos pedir más información con `needsMoreDetails(for:)` indicando el parámetro que no sabemos resolver o llamar a `unsupportedWithReason` para indicar que este no nos sirve y que se vuelva a repetir la petición completa. En función de la opción, Siri contextualizará una petición acorde a él.
- También puede pasar que haya un contacto que sea similar, y podríamos tener que enviar una petición `confirmationRequired(with:)` donde **con dicha función estamos proponiendo a la no resolución correcta del parámetro destinatario, una alternativa que Siri propondrá al usuario**. En caso de confirmarse, ya tendríamos resuelto el parámetro del destinatario y haríamos la acción
- Podríamos tener más de una posible solución para la petición (muchas coincidencias de “Julia Torres”), en cuyo caso se usaría otra función, `disambiguation(with:)` **donde enviaríamos un *array* de posibles selecciones que Siri nos ofrecería para resolver el conflicto**.
- Si decimos “llamar por Skype”, está claro que nos falta un parámetro, el contacto, por lo que a través de `needsValue` **pedimos un nuevo valor que nos es obligatorio para completar la petición**.
- Si el usuario nos da demasiado información y no la necesitamos o no podemos hacer nada con ella, usamos `notRequired`, para informar a Siri que tenemos información de más en nuestra petición.



Cuando todos los parámetros sean confirmados, se procede a la gestión de los mismos y **a la realización de aquello que es el objeto de la intención que ha generado la acción**. Tras esto, en muchos casos la respuesta es una confirmación. Existen algunos casos donde, según el dominio, la confirmación es obligatoria como en los pagos. Vamos a volver al caso de este.

Hemos dicho que se hiciera un pago y por lo tanto, Siri ha enviado la intención, esta ha sido procesada y ahora la app tendrá que enviar una confirmación de lo que va a realizar (que puede ir acompañada de una interfaz propia creada por nuestra app) y pedir un estado de chequeo de la operación: **“Este es su pago en Twip para Antonio Sanchez por 10€. ¿Lo envío?”**. Y esto tendrá un botón de cancelar o de confirmar (que además puede ser contestado por voz igualmente). La respuesta afirmativa, hará que la acción se confirme a la app y, por lo tanto, sea realizada. En ese momento es cuando nuestra app tendrá que realizar la lógica interna para dar por bueno el pago: si el servicio está disponible, si el usuario ha iniciado sesión en el sistema o si el saldo del mismo es suficiente para realizar la operación. En caso que todo sea positiva, **se devuelve un estado** `success` **a Siri con la operación realizada**.

La respuesta



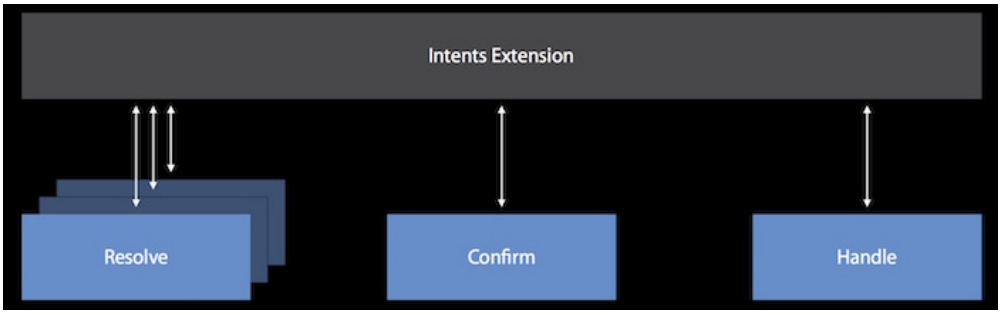
Pero ahí no acaba la cosa. Porque cuando nuestra app ya ha sido capaz de realizar la operación, **tenemos la obligación de proporcionar a Siri el mayor número de información posible sobre el resultado de la misma**. Y puede ser, que dicha operación no se haya realizado. Imaginemos algo que requiere un proceso de red. En dicho caso, Siri tiene una UI específica de estado en espera, que hará entender al usuario que su petición se está procesando. Mientras, Siri seguirá esperando una confirmación por parte de nuestra app confirmando que la operación ha terminado con éxito (o no).

Si pasados unos segundos la respuesta no ha llegado, tendríamos que enviar un estado `inProgress` en vez de `success`, **donde Siri indicaría que la operación se está realizando y que no necesita nada más, pero que está en cola de proceso, en espera, etc.** y que tendremos que ir al contexto de la app para verificar cuándo dicha operación se ha realizado.

Extensiones

Para hacer todo este proceso posible y que Siri funcione, Apple ha creado dos nuevos tipos de extensiones para el sistema: las extensión de intención (*intent extension*) y la extensión de la UI de la intención (*intent UI extension*). La primera, es la responsable de gestionar todo el proceso que hemos comentado hasta ahora: **es el controlador o la vía de comunicación entre app y Siri para gestionar todo el proceso**. Y es nuestra app quien ha de crearlo.

La extensión de intenciones es la base de SiriKit y permite varias instancias a la vez, actúa en segundo plano cuando Siri está activo e implementa los métodos de resolución, respuesta y manejo de todo el proceso.



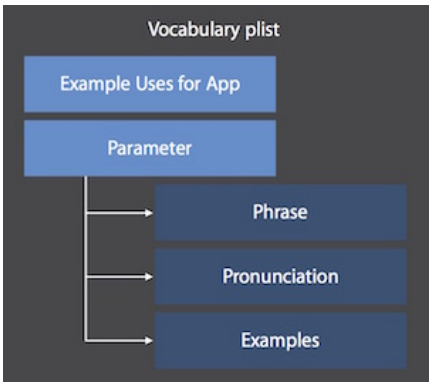
SiriKit también tiene en cuenta la seguridad y hay determinadas operaciones que, por defecto, no pueden ser realizadas con el dispositivo bloqueado, como los pagos o encargar un transporte. En dichos casos, **podemos solicitar al usuario una autenticación local (por ejemplo con Touch ID) para que la operación se realice a través de nuestra petición** (como cuando borramos un email desde la pantalla de bloqueo y se nos pide la huella o el código para confirmar la operación). Igualmente SiriKit soportará Apple Pay para ser usado en las extensiones de intención.

En el caso de las extensiones de UI de intenciones, **podemos dar una experiencia de uso personalizada proporcionando un** `UIViewController` **específico que creemos en cada una de las respuestas que partan de nuestra app hacia Siri**. Usarlo es opcional, pero obviamente es recomendable para tener una mejor experiencia de cara al usuario. Estas extensiones nos permitirán mostrar información adicional de nuestra operación o una actualización del estado de nuestra petición y se mostrará dentro del propio contexto de Siri.

Podemos incluso cambiar los elementos de interfaz de Siri en algunos casos para mejorar la experiencia. **Lo que sí hemos de tener en cuenta es que estas vistas no son interactivas**, aunque sí podemos usar animaciones u otro tipo de contenido dinámico. Estos dos tipos de extensiones son las que tendremos que usar para implementar Siri en nuestra app.

Vocabulario

Un elemento imprescindible de las apps para una buena integración de Siri es el vocabulario, y nosotros como desarrolladores podremos proporcionar formas específicas de comunicarse con nuestra app: **nuestro propio vocabulario de app donde informaremos de aquellas expresiones, frases o palabras específicas que ayudan a la experiencia de la app**. Este vocabulario se indica en la app en una lista de propiedades que, obviamente, puede ir localizada para dar una versión especifica para cada idioma.



En esta lista de propiedades se pueden especificar, en texto, casos reales de uso de nuestra app. Por ejemplo: “Whatsapea a un amigo”. **Esta sería una forma que podríamos indicar como ejemplo, creando nuestra propia palabra o expresión que identifique nuestra app con una forma verbal basada en el nombre de la misma**. Por lo tanto, “whatsapea” se convierte en una acción. Pero también **habría que indicar a Siri no solo la grafía si no la fonética “güasapea” para que sepa cómo será dicho por el usuario**. Y luego también habrá que darle más ejemplos para que acote mejor los casos.

También podemos proporcionar un conjunto de palabras que formen parte de un vocabulario específico del usuario, **una terminología que solo el usuario concreto de la app usará y que se proporciona a través de un tipo** `OrderedSet`. Este vocabulario específico del usuario podría ser cualquier tipo de combinación de datos: el nombre de un álbum de fotos que ha creado en una app donde van a buscarse fotografías, nombres de ejercicios, el nombre de un grupo de Whatsapp al que queremos enviar un mensaje… En definitiva, palabras clave de cualquier tipo que le permiten buscar información o incluso una agenda de contactos propia de la app (que no sea la del sistema, que siempre va incluida en Siri).

Estos datos, obviamente, **han de ser mantenidos y si el usuario borra cualquiera de estos datos, han de ser borrados del diccionario para que Siri** no pueda seguir entendiendo estos contextos. Algo tan

simple como si nos salimos de un grupo de Whatsapp, Siri deje de entender este nombre de grupo como posible destinatario de un mensaje.

Conclusiones

Hemos de entender que Siri es parte integral del sistema. Tendremos, no obstante, que incluir el *framework* `Intents` para incluir SiriKit en nuestra app, teniendo en cuenta que al hacerlo hay una serie de *frameworks* que también vendrán incluidos y que nos darán funcionalidad extra, como los mapas, los contactos o el nuevo `CallKit` que permite integrar llamadas de voz sobre IP de apps de terceros dentro de la propia app del teléfono. De esta forma, **podremos decirle a Siri que haga un Skype y este se realizará desde la app del teléfono, no teniendo que entrar nunca al contexto del propio SKype**. De hecho, si usamos SiriKit, también tendremos la integración con mapas incluida por lo que tenemos muchas más opciones de interacción.

Lo importante que hemos de recordar es que para que SiriKit funcione, **el usuario ha de decir el nombre de nuestra app y esta será reconocida a partir del *bundle display name* (el nombre que se muestra bajo el icono)**. Puede ser usado en cualquier parte de la petición a Siri y este nombre se entenderá dentro del contexto sin problema. Siri admite usar el nombre de nuestra app como verbo de la frase para dejar clara la acción que queremos iniciar.

Como podemos ver, las posibilidades que se presentan son muchas y, aunque parezca complejo, solo hay que seguir las indicaciones dadas en cada uno de los pasos para crear las acciones pertinentes. **Tal vez el reto más importante sea la forma en que se comunica Siri con nuestra app si queremos darle más posibilidades y formas específicas de comunicarse con ella** (lo que requiere localizar a cada uno de los idiomas que soporta Siri). Pero la funcionalidad de Siri, en rasgos generales, es multi-idioma y **lo que hagamos a nivel de funcionalidad, servirá en cualquier idioma sin problema porque al final el propio Siri es el que asocia los verbos o expresiones al nombre de las acciones concretas**, que siempre son los mismos. Da igual que digamos “Envía un Whatsapp” en inglés, español o alemán… Siri siempre enviará la acción apropiada.

Más que nunca, **la mejor forma de integrar Siri en nuestra app es probar, experimentar y afinar la experiencia**. Así que, adelante y ya nos contaréis vuestra experiencia. Hasta la próxima y *Good Apple Coding*.

Documentación oficial de SiriKit (en inglés):

Guía de programación de SiriKit | Acceder
Referencia del *framework* Intents | Acceder
Referencia del *framework* Intents UI | Acceder

Pepe Mérida
¡Pasada de artículo Julio! ¡Qué ganas tengo de tener tiempo libre y ponerme a trastear con Swift y todos sus entresijos!