Kaomi

Cargador remoto de ensamblados como Servicio WCF

Documentación

V1

# Ámbito del sistema

Visión general del sistema Kaomi.

Kaomi (hawaiano para ‘Esqueleto’) consiste en un servicio WCF alojado en una máquina remota capaz de recibir, cargar e instanciar ensamblados DLL a través de la red, además de gestionar procesos implementados en dichos ensamblados.

Conceptualmente, resuelve el problema de desplegar versiones sucesivas de una librería en máquinas fuera de control físico, con la burocracia y retrasos que eso puede provocar.

# Descripción general

Aspectos técnicos y otras suposiciones sobre el sistema.

## Perspectiva de producto

El sistema Kaomi se comunicará con los usuarios para recibir instrucciones, además de con servicios de alojamiento en la nube, para descargar los ensamblados correspondientes a su máquina local.

## Características generales

Las principales características del sistema Kaomi son las siguientes:

* Cargar en la memoria local ensamblados DLL obtenidos de una URI remota o del sistema de archivos.
* Reemplazar los ensamblados cargados, si el usuario indica que hay una versión más nueva disponible.
* Instanciar componentes de esos ensamblados, manteniéndolos controlados incluso para procesos continuos en segundo plano.
* Detener o reiniciar esos componentes si el usuario lo indica.
* Enviar comandos del usuario a determinados procesos en segundo plano.
* Recibir y gestionar los resultados de procesos en segundo plano, ya sea en base a una petición del usuario o a un listener determinado.
* Administrar su propio estado y configuración, pudiendo incluso detenerse y reiniciarse a sí mismo para actualizarse.

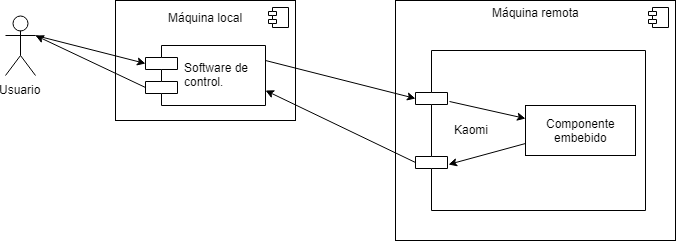
## Suposiciones y dependencias

En condiciones ideales, Kaomi solo debería ser desplegado una vez en cada máquina remota. A partir de ese momento, todas las iteraciones que se produzcan durante el desarrollo de los componentes embebidos se inyectarán de forma transparente para la máquina en la que esté desplegado.

Particularmente, se presuponen las siguientes condiciones operativas:

* Kaomi estará siempre en ejecución en la máquina remota (lo ideal sería ejecutarlo automáticamente al iniciar el sistema), o podrá iniciarse a petición mediante un mensaje de red remoto (lo cual seguramente supone tener un proceso escuchando en un puerto).
* Kaomi podrá recibir instrucciones remotas, y conectarse a servidores remotos para descargar ensamblados a su máquina local.
* Kaomi se ejecutará con permisos de administrador.

## Entorno operacional



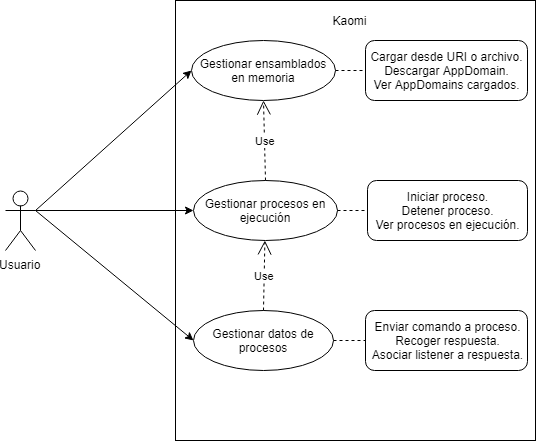
El usuario estará físicamente en una máquina separada, y se comunicará con Kaomi a través de mensajes de texto serializables transmitidos por Internet. Estos mensajes podrían estar cifrados con DotNet.Misc.Security para dar mayor seguridad al sistema.

Como respuesta a los mensajes del usuario, Kaomi podría comunicarse con otros servicios (por ejemplo, OneDrive) para descargar nuevos ensamblados, o bien transmitir los mensajes a alguno de sus componentes embebidos, que será el encargado de interpretarlos.

Los componentes embebidos recibirán los mensajes dirigidos a ellos, y modificarán su comportamiento en consecuencia. En algunos casos, estos componentes pueden producir una respuesta que deberá ser devuelta al usuario, preferentemente de forma asíncrona.

La recepción de respuestas podría basarse en nuevas peticiones del usuario, o bien en un patrón Observer, de forma que las respuestas sean almacenadas en una base de datos en la nube sin intervención humana.

De forma más específica, el usuario podrá enviar los siguientes tipos de mensajes a Kaomi:



* Descargar un ensamblado desde URI [**Kaomi::DownloadAssembly::<uri>**]: Indica a Kaomi que debe descargar un ensamblado desde una URI y almacenarlo en su máquina local.
  + Esta llamada devuelve un mensaje informando del resultado.
* Cargar en memoria un ensamblado local [**Kaomi::LoadAssembly::<path>**]: Indica a Kaomi que cargue en memoria un ensamblado ya almacenado en su máquina local.
  + Nota: Este proceso presupone que cualquier dependencia del ensamblado a cargar ya está presente en el mismo directorio.
  + Esta llamada devuelve una ID que permite hacer referencia al AppDomain que se ha creado para el ensamblado.
* Ver los AppDomains cargados [**Kaomi::ListDomains**]: Devuelve una lista de todos los dominios actualmente en memoria, junto con sus ID.
* Descargar un AppDomain de la memoria [**Kaomi::UnloadDomain::<id>**]: Indica a Kaomi que descargue el AppDomain indicado de la memoria, deteniendo todos sus procesos en ejecución.
* Iniciar un nuevo proceso [**Kaomi::Run::<appID>::<type>::<attr>**]: Instancia un componente del AppDomain **appID** que sea de tipo **type** y que esté anotado con el atributo **attr**.
  + Esta llamada devuelve una ID que permite hacer referencia al proceso que se acaba de instanciar.
* Ver procesos en ejecución [**Kaomi::ListProcess::<appID>**]: Devuelve una lista de todos los procesos en ejecución para un AppDomain determinado.
* Detener un proceso [**Kaomi::Stop::<appID>::<prID>**]: Detiene un proceso indicando su ID y la ID de su AppDomain.
* Ver ajustes de proceso [**Kaomi::GetConfig::<appID>::<prID>**]: Devuelve el contenido del fichero de configuración asociado a un proceso concreto.
* Cambiar ajustes de proceso [**Kaomi::SetConfig::<appID>::<prID>::<data>**]: Envía el contenido de un fichero de configuración para sobrescribir el actual en un proceso concreto.
* Enviar orden a proceso [**<appID>::<prID>::<data>**]: Envía una instrucción a un proceso concreto. Las instrucciones pueden tener un formato propio, y serán ignoradas por Kaomi.
  + Esta llamada devuelve un mensaje no definido, que puede ser un informe de estado, o bien la respuesta del proceso.

# Detalles técnicos

Información sobre detalles de implementación.

Los componentes diseñados para ejecutarse desde Kaomi deben tener una estructura determinada:

* Cada componente tendrá un archivo de configuración propio, donde podrá indicar parámetros de entrada, modos de funcionamiento o cualquier otro dato necesario para funcionar de forma autónoma. No se garantiza que ese fichero sea inmutable, pues podrá ser reemplazado por el usuario en cualquier momento.
* Los componentes deberán instanciarse mediante un constructor sin parámetros. Cualquier dato de entrada necesario deberá figurar en el archivo de configuración asociado.
* Se debe diseñar un mecanismo para que los componentes notifiquen a Kaomi que su ejecución ha terminado, con el fin de que sean retirados de la lista de procesos en ejecución.
* Deben comprobar, a intervalos regulares, si han recibido alguna instrucción del usuario. Esas instrucciones llegarán a través de un sistema estándar de Kaomi, y cada proceso solo recibirá instrucciones dirigidas a él.
* Si deben emitir datos de salida, lo harán a través de un sistema estándar de Kaomi, que será independiente del método de recogida (petición expresa o patrón Observer).
* Cada tipo de componente irá acompañado de un proyecto adaptador que defina las interfaces apropiadas. Esas interfaces serán compartidas por el componente, Kaomi y el usuario.

Kaomi también permitirá asociar un listener a las respuestas de cada proceso, usando una interfaz estándar. Esos listeners estarán definidos en ensamblados (que se cargarán normalmente). **Esto no está del todo claro, hay que comprobar si es útil y factible**. Otra alternativa es empaquetar el listener con el propio componente. De todas formas, sus parámetros se pueden configurar de forma remota.

Además de gestionar componentes embebidos, Kaomi también tendrá un módulo de autoprotección, que se encargará de controlar su propio estado. Sus funcionalidades a grandes rasgos son:

* Asegurarse de que Kaomi se ejecuta al inicio del sistema, pudiendo para ello manipular tareas programadas, iniciar o detener procesos, o incluso controlar servicios en segundo plano.
* Evitar llamar la atención innecesariamente. Para conseguirlo, monitorizará todos los componentes embebidos, y los detendrá si empiezan a consumir recursos de forma descontrolada.
* Mantener la infraestructura de comunicaciones, manejando puertos u otros permisos si fuera necesario.

Como Kaomi de por sí se ejecuta con permisos de administrador, no debería haber problema en conseguir esto. No obstante, se trata de funciones avanzadas, y se implementarán seguramente en versiones posteriores.

Algunas de las funcionalidades anteriores pueden requerir usos creativos de reflexión, tipos dinámicos y compilaciones en tiempo de ejecución. No es necesario detallar esos pormenores en la documentación final.

## Interfaces de componente

Los componentes podrían implementar una interfaz que defina métodos template para cada fase del ciclo de vida, por ejemplo:

* OnInitialize: Se ejecuta una vez al crear el proceso. Útil para cargar configuraciones y establecer variables.
* OnIteration: Se ejecuta una vez por cada iteración. El contenido de este método en sí debe terminar (sin bucles infinitos), y hacer las acciones correspondientes a cada iteración.
* OnUserCommand: Se ejecuta al terminar una iteración si se ha recibido una instrucción del usuario para este proceso. Se puede responder en este mismo método o en la siguiente llamada a OnIteration.
* OnFinalize: Se ejecuta una vez al detener el proceso. Útil para liberar recursos.

Durante la inicialización, cada proceso puede configurar, entre otras cosas, el intervalo de tiempo entre cada iteración (milisegundos, siendo 0 un proceso puntual con una sola iteración).

Esta arquitectura implica que los procesos no se ejecutan a sí mismos, sino que Kaomi instancia un ‘motor de ejecución’ que va llamando a los métodos correspondientes.