Problema

* **Corta-fuegos:** descartamos todos los mecanismos menos los servicios web (SOAP y REST)
* **Ancho de banda limitado:** Descartamos SOAP
* **Limitación mensaje:** Descartamos UDP y SOAP.
* **Síncrona:** descartamos colas de mensajes
* **Asíncrona:** colas de mensajes
* **Alto nivel de funcionalidad:** Descartamos UDP y TCP
* **Baja funcionalidad:** Sockets
* **Desacoplados en el tiempo:** colas de mensajes
* **Acoplado en el tiempo**: Sun-RPC, UDP, CORBA, SOAP
* **Orientado a objetos:** JAVA RMI, CORBA
* **Clientes heterogéneos:** descartamos UDP
* **Nuevos servicios:** WS-SOAP mejor que WS-Rest
* **Múltiples lenguajes de programación:** CORBA(mejor que RPC porque es estándar), SOAP, descartamos JAVA RMI
* **Interfaz de programación**: SUN-RPC, CORBA
* **Prioridades**: colas de mensaje

Colas de mensaje:

* Cliente y servidor no necesitan estar conectados en el mismo instante de tiempo (Desacoplados en el tiempo).
* El cliente no queda bloqueado tras la petición al servidor (Asíncrono)
* Además, las colas de mensajes hacen posible que las peticiones sean procesadas de acuerdo a ciertas prioridades (tipo de cuenta de usuario) y la temporización en el procesamiento de mensajes en función de la carga del sistema
* La cola de mensajes debería de encargarse de la traducción de datos para garantizar la transparencia de acceso puesto que se trata de sistemas heterogéneos
* En este caso puede tardar mucho desde que un cliente manda su solicitud hasta que obtiene el mensaje de que su trabajo ha terminado (5 días como máximo). Esto sugiere funcionalidad asíncrona, evitando que los clientes se queden bloqueados. Además, existen mensajes con prioridad y mensajes que deben ser filtrados, algo implícito en la comunicación mediante colas de mensajes. Todas estas características sugieren el uso de colas de mensajes, único mecanismo de comunicación asíncrono. Debido a que los sistemas son homogéneos, no habrá que traducir datos, lo que no presenta problema en comunicación mediante colas de mensajes, aunque no soporte transparencia de datos. Al ser sistemas homogéneos no es necesario traducir datos.

Rest:

* Los clientes que usarán el servicio serán heterogéneos.
* Corta-fuegos
* Síncrono
* Limitación de mensaje
* Entre ellos elegimos REST ya que es más eficiente que SOAP. Podríamos usar JSON para la representación de los datos, lo que nos proporcionaría transparencia de datos con lo que no tendríamos que traducirlos aunque los clientes sean heterogéneos.
* Descartamos UDP debido a la limitación de tamaño de mensaje, y al hecho de que los clientes serán heterogéneos y este mecanismo no proporciona transparencia de datos. Descartamos colas de mensajes ya que la interacción será síncrona, y colas de mensajes tampoco proporciona transparencia de datos.
* Estos son WS-SOAP y WS-REST. De entre ellos preferiríamos WS-REST frente a WS-SOAP, ya que el ancho de banda de la empresa es limitado y la capacidad computacional de los clientes es reducida. Esto sería una desventaja para WS-SOAP pues los mensajes en XML son costosos de procesar y consumen mucho ancho de banda.

JAVA RMI:

* Síncrono
* Clientes heterogéneos
* funcionalidad del sistema es bastante complicada
* los mensajes han de ser pequeños
* técnicas orientadas a objetos
* El tráfico no será filtrados
* Fácil de implementar (mejor que Corba)
* No habría que añadir nada para traducir datos, ya que Java RMI se encargaría de ello.
* Hay que implementar funcionalidades síncronas, por lo que descartaríamos las colas de mensajes. Este mecanismo también quedaría descartado, ya que los clientes son heterogéneos, y no incorpora nada para la traducción de datos. Descartaríamos UDP y TCP ya que la funcionalidad del sistema es bastante complicada y los sockets son de muy bajo nivel. Como los mensajes han de ser pequeños, descartamos WS-soap. Hemos de usar técnicas orientadas a objetos, por lo que descartaríamos RPCs frente a JAVA RMI o CORBA. Los datos no serán filtrados, por lo que WS-rest no es tan atractivo frente a estos mecanismos. Como hemos de usar un único lenguaje de programación y el proceso de desarrollo ha de ser sencillo, elegiríamos Java RMI y descartaríamos CORBA. No habría que añadir nada para traducir datos, ya que Java RMI se encargaría de ello.

UDP:

* Síncrono
* Las interacciones serían acopladas en el tiempo (servidor y cliente conectados a la vez).
* El cliente podría quedarse o no bloqueado a la espera de la respuesta del servidor.
* No será necesario implementar ningún mecanismo adicional de traducción ya que los clientes y el servidor tienen la misma arquitectura
* La funcionalidad es sencilla (solo hay. una) y deber de garantizarse el envío eficiente de los mensajes
* Habría que implementar algún tipo de funcionalidad adicional para garantizar la transparencia de datos al tratarse de sistemas heterogéneos, pero la información intercambiada es pequeña y sencilla de codificar
* Hay que implementar funcionalidades síncronas, por lo que descartaríamos las colas de mensajes. No será necesario implementar ningún mecanismo adicional de traducción ya que los clientes y el servidor tienen la misma arquitectura. La funcionalidad es sencilla (solo hay una) y deber de garantizarse el envío eficiente de los mensajes. Esto descarta WS-soap y en general todos los mecanismos que son de alto nivel como RPC, Java RMI o CORBA pues son muy complicados para implementar algo tan sencillo. Por su eficiencia y simplicidad elegiríamos protocolos de bajo nivel como TCP o UDP, y entre ellos nos quedaríamos con UDP, pues es aún más eficiente que TCP (si se pierde algún mensaje no tiene sentido volverlo a pedir, ya que es video en tiempo real) y además UDP permite direccionamiento multi-cast que permite llegar a múltiples destinatarios poniendo un único mensaje en la red.
* No merece la pena establecer una conexión para el envío de un único mensaje de petición y respuesta, por lo que se descarta TCP. Además, en caso de pérdida de un mensaje, éste se podría reenviar sin problema puesto que las interacciones son del tipo petición/respuesta e idempotentes.

TCP:

* Síncrono
* muy bajo nivel
* Por otro lado, al ser sistemas que usan la misma arquitectura no sería necesario traducir datos, lo que para los sockets no es un problema. Un comando puede resultar en mensajes muy grandes que no quepan en un datagrama UDP. Además, importa el orden de los mensajes recibidos. Por estas razones elegiríamos TCP frente a UDP. Al ser sistemas homogéneos no es necesario traducir datos.

SOAP:

* Síncrono
* Funcionalidad compleja
* Corta-fuegos
* Sencillez para integrar servicios
* Mucho ancho de banda
* Clientes heterogeneos
* no resulta crítico procesar los mensajes en poco tiempo
* También elegimos este último mecanismo ya que podríamos usar UDDI para buscar nuevos servicios en tiempo de ejecución y mediante el fichero WSDL podríamos obtener la información para invocarlos de forma automática. Como SOAP, UDDI y WSDL son estándar, no habría problema en usar múltiples lenguajes de programación, y tendríamos transparencia de datos.
* Hay que implementar funcionalidades síncronas, por lo que descartaríamos las colas de mensajes. Este mecanismo también quedaría descartado, ya que los clientes son heterogéneos, y no incorpora nada para la traducción de datos. Descartaríamos UDP y TCP ya que la funcionalidad del sistema es complicada y un corta-fuegos podría filtrar los mensajes. Por esta misma razón descartamos RPC, CORBA y Java RMI. Propondríamos usar WS-soap ya que es estándar, soporta múltiples lenguajes de programación y sistemas heterogéneos, y pasa a través de cortafuegos. Frente a WS-Rest elegiríamos WS-soap ya que resultaría sencillo integrar nuevos servicios descubiertos en tiempo de ejecución mediante (UDDI y WSDL). Como la empresa tiene suficiente ancho de banda y no es necesario procesar rápidamente los mensajes WS-soap no estaría penalizado por usar mensajes grandes y costosos de procesar. Ya que los WS-soap usan XML para representar la información, que es independiente de la plataforma, no habrá que implementar ningún mecanismo adicional de traducción de datos.

CORBA:

* Síncrono
* Cliente heterogéneos
* Tamaño de mensaje pequeño
* Trafico no filtrado
* Múltiples arquitecturas y mensajes
* Complejidad no penaliza
* Grado bajo de acoplamiento
* Usando WSDL y UDDI podríamos registrar y localizar de forma sencilla los nuevos servicios.
* Los dos sistemas tiene que estar conectados a la vez (acoplado en el tiempo) y el cliente quedaría bloqueado a la espera de la respuesta del servidor: las colas de mensajes pierden su atractivo
* Funcionalidad complicada y traducción de datos
* Hay que implementar funcionalidades síncronas, por lo que descartaríamos las colas de mensajes. Este mecanismo también quedaría descartado, ya que los clientes son heterogéneos, y no incorpora nada para la traducción de datos. Como el tamaño de los mensajes debe ser pequeño y el tráfico no será filtrado descartaríamos web servicies basados en SOAP. Como los clientes serán implementados en múltiples arquitecturas y múltiples lenguajes de programación, y la complejidad de desarrollo no penaliza, elegiríamos CORBA. Java RMI lo descartaríamos al tener que usar múltiples lenguajes de programación. También preferiríamos CORBA frente a RPC, ya que CORBA es estándar, y RPC depende de la versión utilizada. CORBA tendría un abanico de compatibilidad mucho más grande. Descartaríamos WS-rest ya que queremos un grado bajo de acoplamiento entre cliente y servidor. Descartaríamos TCP y UDP ya que la funcionalidad a desarrollar es complicada y hay que traducir datos. CORBA ya proporciona un mecanismo propio de traducción de datos, por lo que no será necesario implementar ninguno adicional.

SUN-RPC:

* Los clientes acceden al servicio a través de una interfaz de programación definida.
* Se esperaría que las peticiones incluyan cierta lógica de proceso compleja.
* Es necesaria la traducción de datos (arquitecturas heterogéneas).
* Los dos sistemas tienen que estar conectados a la vez (acoplado en el tiempo) y el cliente quedaría bloqueado a la espera de la respuesta del servidor (Sincrono)
* Todo esto hace indicar que lo más apropiado serían los servicios web o llamadas a procedimiento remoto. Sin embargo, es crítico que los mensajes sean eficientes así como su procesamiento para poder trabajar en tiempo real. Esto hace que los servicios web basados en SOAP (XML) estén en desventaja respecto a Sun-RPC.
* Sería necesario llevar a cabo operaciones de marshalling/unmarshalling para garantizar la transparencia de acceso a datos. Sin embargo, al usar Sun-RPC, este proceso sería transparente y no haría falta añadir funcionalidad adicional al middleware.