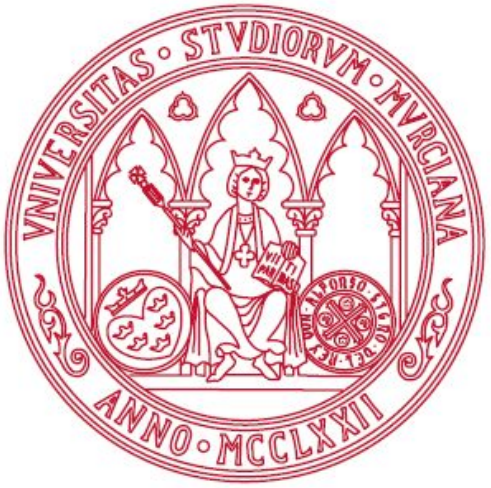


Universidad de Murcia

Grado en Ingeniería Informática

Curso 2020/2021

Programación para las comunicaciones

Profesor: Humberto Martínez Barberá

Práctica III

Jaime Ortiz Aragón - 49196689B - jaime.ortiza@um.es

- **ESCENARIO**

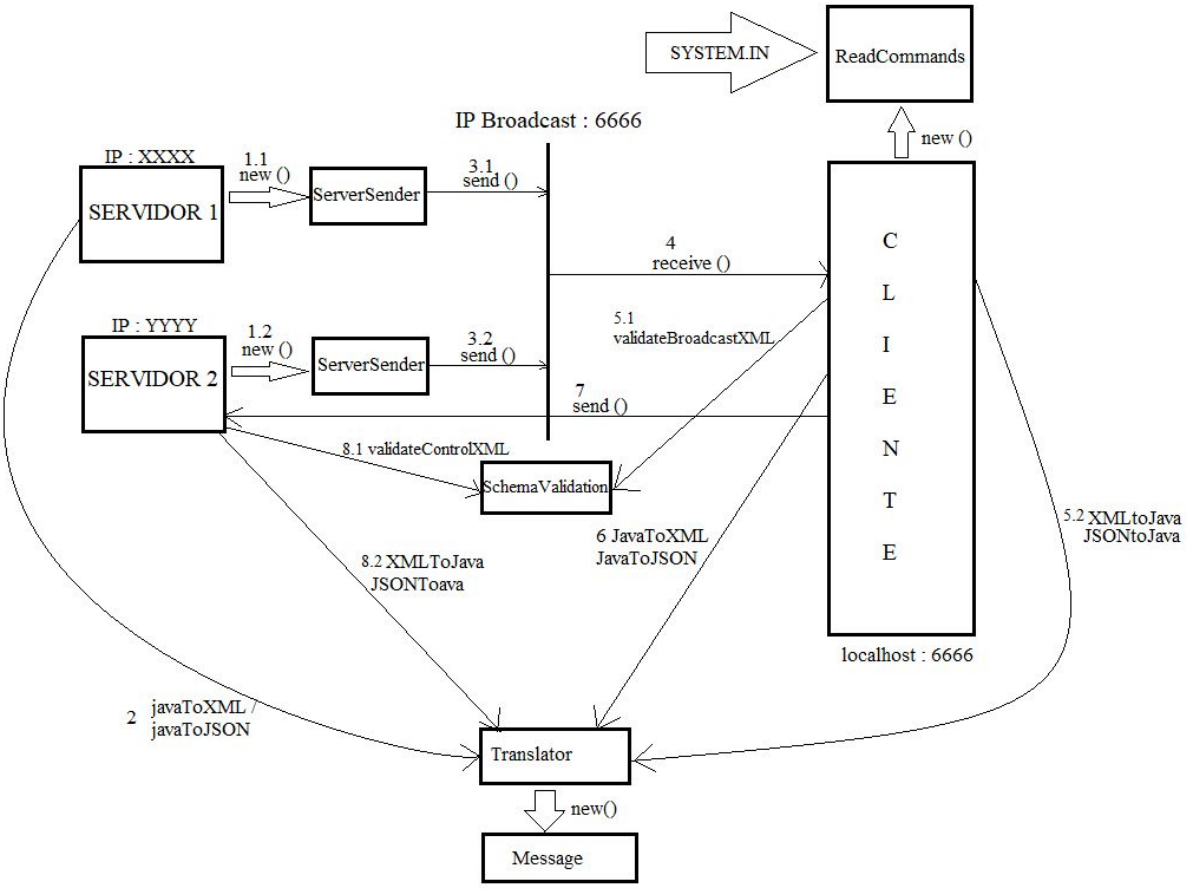
El escenario que se presenta es el mismo que se describió en la práctica 2. El cambio primordial es, que en vez de mandar los mensajes en texto plano, se envían con los formatos JSON/XML.

- **FUNCIONAMIENTO GENERAL**

Al ejecutar la aplicación, el sistema pedirá al usuario introducir en qué formato quiere que se empiecen a tratar los mensajes. Empezarán a salir por pantalla los mensajes broadcast enviados por los servidores, como en la práctica 2, pero delante tendrán información sobre la codificación.

En cualquier momento el usuario, como cliente podrá cambiar esta codificación mediante el comando *ID FORMATO*, en el cual el ID es el servidorId y el FORMATO es XML o JSON. También se puede cambiar el formato de los mensajes del cliente mediante el comando *client FORMATO*. Para cambiar la frecuencia de envío o detener el envío de datos de un servidor es igual que en la práctica 2 (*ID stop ò ID X*, siendo X el valor de la frecuencia en Hz).

Para que se cierre correctamente el *Writter* que nos permite escribir en los ficheros hay que mandar el comando *stop* al servidor o cliente correspondiente.



- **SERIALIZACIÓN (Pasos 2, 6)**

La serialización es el proceso de convertir los mensajes Java a XML o JSON. Esto lo he realizado a mano, ya que como desarrollador de la aplicación, sé el formato que van a llevar los mensajes. He creado la interfaz *Transator*, implementada por *TranslatorControl* y *TranslatorBroadcast*, que se encarga tanto de serializar como de deserializar por medio de operaciones. Para serializar los mensajes, el método recibirá un objeto mensaje, que puede ser tanto *MessageBroadcast* como *MessageControl*, ya que ambas implementan la interfaz *Message*. Para el caso de JSON, hacemos uso de la librería *Gson*, que nos convertirá el mensaje directamente en el String que deseamos en formato JSON. Para los mensajes de broadcast en XML, ponemos de raíz *server* para saber que es un mensaje del servidor y añadimos tanto el id como los sensores como elementos, esta estructura es clave para posteriormente saber cómo deserializar. Los mensajes de control es más sencillo, ya que sólo se envía el la operación a la que se ve sometida el servidor, por lo que sólo tenemos un elemento *action*.

- **VALIDACIÓN con Schema (Pasos 5.1, 8.1).**

Una vez serializado, los mensajes se envían por los sockets al igual que en la práctica 2, pero al recibirlo, el paquete contendrá el mensaje en el formato en el que lo haya escrito el emisor, por lo que para tratarlo hay que pasarlo a Java. Para conocer el formato del mensaje, comprobamos si contiene "<", ya que es un carácter que aparece seguro en los mensajes XML. Esto no es lo más correcto, pero para una aplicación controlada y de esta escala es bastante eficiente y nos permite centrarnos en los conceptos reales que se explican en el tema. De este modo, se enviará el mensaje a un objeto validador de la clase *SchemaValidation* que validará tanto los mensajes de broadcast como los de control frente a un xsd. Para el mensaje de control, simplemente comprobamos que aparezca el elemento *client* y que dentro del nodo aparezca una operación *action*. Para el mensaje de broadcast es algo más complejo porque en mi caso, como se explicó en la práctica 2, no sólo pueden aparecer 3 sensores, sino que como se dijo en clase, pueden aparecer un número aleatorio, por lo que me costó un poco validar cuando no todos los sensores aparecían. Resumiendo, los elementos en este caso pueden ser los sensores, pero pueden aparecer o no, por lo que he hecho uso del número de ocurrencias para la implementación. He elegido Schema, porque al ser un escenario controlado y con pocos elementos me resultaba más sencillo.

- **DESERIALIZACIÓN con DOM (Pasos 5.2, 8.2).**

Para realizar la deserialización, he hecho uso de DOM, ya que sobre todo para los mensajes de broadcast, al tener estructura de árbol, me permitía sacar cuántos nodos hay y así saber a priori cuántos sensores va a haber que tratar para un mensaje dado. Primero recupero el elemento *server* y de ahí miro los nodos hijos. Pongo /2 porque siempre aparecen doblados, es decir el elemento en sí y otro text:####, y como sólo queremos los relativos a los sensores, vamos almacenando en un mapa el sensor con el valor asociado a cada uno de los elementos. Después en función de cuántos elementos hay construímos el mensaje de broadcast. Para deserializar el mensaje de control simplemente hay que recuperar el valor asociado al elemento con etiqueta *action* y construir el mensaje de control.