UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Experiencia Nº 1 - RMI

Sistemas Distribuidos

Patricio Enrique Toledo Chamorro

Profesor: Mario Inostroza P.

Ayudantes: Cesar Silva

Emilio Tapia

Santiago – Chile

2012

TABLA DE CONTENIDOS

CAPÍTULO 1. INTRODUCCIÓN

1.1 MOTIVACIÓN

La tecnología y la situación actual, han generado una gran cantidad de dificultades y oportunidades para el desarrollo de nuevos “sistemas de gestión de recursos” o sistemas operativos distribuidos. Hoy en día se ha masificado mucho el modelo cliente-servidor.

Como se menciona en [2], una aplicación cliente-servidor es un modelo o esquema de trabajo, en donde existen distintas aplicaciones del cliente, las cuales requieren servicios provenientes de procesos en el servidor. En otras palabras, en una aplicación cliente-servidor existe un cliente el cual se está ejecutando remotamente (en un *host* o computador para efectos de la experiencia) y requiere de los servicios que otorga el servidor, como por ejemplo: el almacenamiento, registro y/o modificación de datos.

1.2 OBJETIVOS Y ALCANCES DE LA EXPERIENCIA

A continuación se presentan los objetivos generales, específicos que se tienen en la experiencia. También se determinan los alcances para el desarrollo.

1.2.1 Objetivos generales

La experiencia N° 1 de sistemas distribuidos tiene por objetivos generales desarrollar una aplicación cliente-servidor utilizando el lenguaje de programación Java. Se deberá también utilizar la *invocación de métodos remotos* (**RMI**) proporcionado por Java. Será necesario que el sistema funcione con 2 servidores y al menos 1 cliente, para poder computarizar el proceso de elecciones en le "República de los Sistemas Distribuidos"

1.2.2 Objetivos específicos

Cómo objetivos específicos, será el desarrollador deberá generar una GUI que sea apta y de rápido aprendizaje por parte de los usuarios. También es parte de los objetivos específicos y muy importante por lo demás, la tolerancia a fallas del sistema, como lo es cuando uno de los servidores se cáe (Se encuentra OFFLINE).

1.2.3 Alcances

Si bien, en primera instancia los objetivos generales determinan los alcances de la experiencia, es importante destacar que para el desarrollador será de especial cuidado lograr incorporar funcionalidades de tolerancia a fallas, validación, log de acciones en los servidores.

1.3 ORGANIZACIÓN DEL DOCUMENTO

Para la mejor comprensión del desarrollo de la solución a al problema planteado en la presente experiencia, a continuación se detalla el contenido del informe, su división y aspectos generales.

Inicialmente, luego de la introducción, el capítulo dos comprenderá la descripción y solución al problema, detallando tanto aspectos genéricos como técnicos involucrados en el desarrollo.

El capítulo tres involucrará el estudio y análisis de rendimiento, debiendo ser probado en distintas máquinas y configuraciones de red.

Las conclusiones se abarcarán en el capítulo cuatro, determinando si realmente se cumplieron los objetivos aquí planteados y estudiar las causales en caso del no logro. En el capítulo cinco se presentarán las referencias bibliográficas utilizadas tanto a lo largo de la comprensión del problema como en el desarrollo de la solución. Finalmente, el capítulo seis tendrá el anexo manual de usuario, con el objetivo de clarificar los requerimientos del sistema y el uso de la solución por desarrollar.

CAPÍTULO 2. DESCRIPCIÓN Y SOLUCIÓN AL PROBLEMA

2.1 DESCRIPCIÓN DEL PROBLEMA

Como se mencionó anteriormente, la problemática a resolver está involucrada con el concepto de aplicación cliente-servidor y estará ligada a la gestión de un sistema que permita realizar de manera moderna y eficiente, el proceso de elecciones de la "República de los Sistemas Distribuidos".

Por un lado, se deben generar 2 servidores, uno para el registro electoral y el otro para los partidos políticos. El Servidor de Registros Electorales (en adelante **RE**) posee las funcionalidades de iniciar, detenerse, registrar un usuario, validarlo y permitir la carga masiva de datos a través de un archivo. El Servidor de Partidos Políticos (en adelante **PP**) posee las funcionalidades de iniciar, si y solo si, RE ya está iniciado, detenerse, registrar usuario en RE y en PP a través de PP, inscribir en un Partido Político, a un usuario existente en RE, des inscribir a un usuario del partido político al cual estaba inscrito.

Por otro lado, se debe generar una GUI de cliente, el cual llama a estos procedimientos de forma remota, es decir, siguiendo el modelo cliente-servidor, existente en Java RMI.

Se requiere también que ambas aplicaciones posean GUI.

Otro requisito de la solución a desarrollar es la generación de dos ***logs*** para los servidores, los cuales registren las acciones y resultados llevados a cabo por la interaccion de los servidores con el cliente y entre si (actuando también como cliente el PP)

Además de los elementos recién planteados, se pueden encontrar otros implícitos tales como:

- El servidor deberá permitir iniciar sus servicios o detenerlos. Obviamente, cuando los servicios estén paralizados, el cliente no podrá utilizar las funciones del servidor.

- En general, se deberán presentar mensajes claros acerca de las acciones que el cliente realiza y el resultado de tales acciones sobre el servidor.

2.2 DESCRIPCIÓN DE LA SOLUCCIÓN

A continuación se describirán los pasos generales a seguir para lograr la solución. Se considera que no es necesario ahondar en aspectos demasiado técnicos ya que estos pueden ser revisados en el código fuente de las aplicaciones.

2.2.1 Servidor

Lo primero que se deberá realizar, será crear las vistas necesarias para el servidor. Como el servidor no presenta muchas funcionalidades (interacción gráfica con cliente) en sí (dado que las ocupa el cliente), basta con sólo una vista llamada “vista servidor”.

- Vista servidor: tendrá el botón para iniciar sus servicios y otro para detenerlos. Para que el usuario se pueda conectar, es necesario que se conozca la dirección IP del *host* y el puerto a utilizar. Estos datos se cambian como parámetros en el código mismo.

Además de la parte gráfica, será necesario crear las funcionalidades del servidor. Estas serán, primero por parte de RE:

- Iniciar: Se abre el puerto en el servidor, para quedar "*escuchando*" esperando a la petición del cliente.
- Detener: Se deja de escuchar en el puerto mencionado.
- Registrar usuario (con sus validaciones): Se realiza mediante las funciones de validación inicialmente, para comprobar la validez de los datos que (el cliente ingresa por gui) serán ingresados. Estas funciones son o bien directa o indirectamente (desde PP a RE): - PP-preg_estaRegRE(), validare() las cuales validan además de la integridad de la información, que no exista registro de antemano en el servidor, esto lo hacen comparando con un archivo de texto "*servRE.txt*" con el cual está el rut,nombre,comuna y fecha de ingreso. Luego se realiza la escritura del nuevo registro con registra_newuser_RE(rut,nombre,comuna).

Ambos servidores poseen la capacidad de generar registro de las actividades (fallos, ingresos de usuario,cambios,etc…), esto lo hacen mediante una clase **Log** la cual posee 2 métodos:

- getDate(): Obtiene fecha actual , para usarlo en el LogRE.txt y LogPP.txt
- makeLog(mensaje): Guarda en los archivos destinados el mensaje ingresado.

Por parte del PP:

- Iniciar: Análogo al RE solo que con un puerto distinto y con la **premisa** de que se verifica si está online el RE. Caso contrario, el server notifica al usuario y no inicia.
- Detener: Análogo a RE.
- Registrar Usuario con PP: Lo hace mediante las funciones que validan en PP los datos (valida_reg_PP(rutIngresado)) , y luego se utilizan "llamadores" (también conocidos como *callbacks*) , que a partir de la petición del cliente, hacen las veces de cliente con el servidor RE, para

asi realizar el registro en RE con los datos tales como rut,nombre,comuna,partido político.

- Registrar Partido Político: Primero se valida que el usuario exista en los registros electorales en servRE.txt, si es así, entonces se procede a realizar el ingreso en un partido político (que solo puede ser "UDI","PPD" ó "PC") con registra_user_existente_RE_en_PP(rutIngresado, partidoIngresado).
- Desinscribir usuario de Partido Político: Se hace directamente en ServPP con el método borraRegistroPP(rutIngresado), el cual abre servPP.txt y va agregando en un vector, todos los registros excepto el que se identifica que se quiere borrar (usando como llave el rut). Luego se vacía el vector en el nuevo archivo servPP.txt, es decir, se escribe todo de nuevo .Una vez realizado esto, retorna un entero, determinando si se borro UDI, PPD ó PC. Cabe destacar que anterior al borrado se realiza una validación de si efectivamente el usuario existe (usando un testigo o flag), por lo que si no existe en servPP.txt, entonces el archivo no es alterado en ninguna forma.

2.2.2 Cliente

Como el cliente será quien haga uso de las funcionalidades del servidor, esta aplicación contendrá varias vistas (aunque en 1 sola clase):

- Vista principal: Posee las 4 funcionalidades, las cuales son:
  - Inscribirse en los registros electorales como nuevo usuario
  - Inscribirse en los registros electorales eligiendo un partido político
  - Inscribirse en un partido político
  - Desinscribirse de un partido político
  - Carga Masiva:

- El sistema permite al usuario cargar un archivo con diferentes instrucciones y datos por línea, para “automatizar” los procesos, es decir, el sistema le permite al usuario mediante esta carga masiva, hacer “en masa” lo que sería ingresar usuario, partido político, borrar partido, de forma manual.

Si bien, es la aplicación servidor la que prestará servicios (funcionalidades) al cliente, éste también tendrá ciertas funcionalidades internas, como lo es el caso de notificar al usuario según sea la situación.

CAPÍTULO 3. ANÁLISIS DE RENDIMIENTO

En este capítulo, por falta de tiempo, no hay un estudio detallado del rendimiento, pero si se puede comentar, que tomando un archivo de carga masiva y usando 2 clientes en un mismo computador (localhost), con 2000 registros cada archivo, funciona sin problemas de latencia ni negación de servicio o bloqueo, es decir, está cubierta la tolerancia a esas fallas, además de las ya indicadas, respecto a la validación de que si un servidor deja de funcionar, el sistema “no se cae”, pero si, cuando requiere algún servicio de un servidor OFFLINE, notifica al usuario de la imposibilidad de la realización de esta acción.

CAPÍTULO 4. CONCLUSIONES

A lo largo del desarrollo de la presente experiencia, el desarrollador se ha encontrado con distintas situaciones de distintos niveles de dificultad, por lo que se pueden hacer las siguientes conclusiones:

- Los objetivos generales han sido cumplidos a cabalidad. Se ha logrado una aplicación cliente-servidor que incluso fue probada en forma distribuida y logró completar los requerimientos básicos expuestos en el enunciado de la presente experiencia.
- No se puede mencionar lo mismo de los objetivos particulares. Independiente de la razón que sea, no se logró una estructura completamente modular y organizada del programa, por lo que su mantención sería levemente más dificultosa.
- Si bien el análisis de rendimiento no se pudo extrapolar por la dificultad encontrada para realizar las pruebas, es claramente obvio que, dado los tiempos presentes en dicho análisis, el mayor costo está asociado a la comunicación. Si se agregan más clientes, debiese existir un incremento de rendimiento hasta un punto óptimo, el cual una vez superado, presentará una merma importante en el rendimiento dada la comunicación necesaria y el procesamiento concurrente.
- Existe una ineficiencia en términos de que para cada validación y borrado, se recorre de forma “linear” todo el archivo según sea el caso. A pesar de esto, el sistema no funciona con latencias asociadas.

La experiencia ha sido de gran apoyo cognoscitivo para el desarrollador en el área de programación distribuida.

CAPÍTULO 5. REFERENCIAS

Referencias a documentos de Internet:

1. Apuntes de la asignatura encontrados enhttp://www.usachvirtual.cl

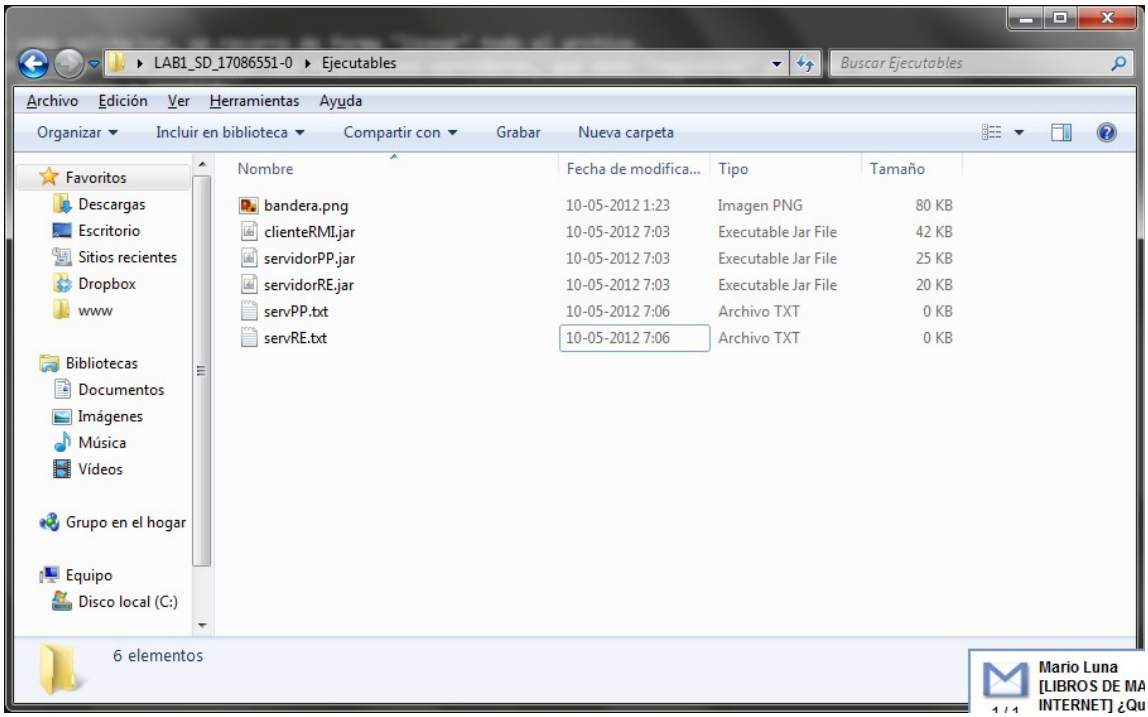
CAPÍTULO 6. ANEXO: MANUAL DE USUARIO

6.1 REQUERIMIENTOS DEL SISTEMA

Tanto para la aplicación servidor como la aplicación cliente, es necesario que los computadores tengan instalado *Java Development Kit* (JDK). Según el sistema operativo del computador, JDK se puede descargar de la página web: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Además, sin considerar el espacio disponible para los archivos de respaldo y consistencia (que dependerán del crecimiento del sistema), es necesario un mínimo de 50 KB para la aplicación servidor y unos 42 KB para el cliente.

Se requiere que en la carpeta donde estén:



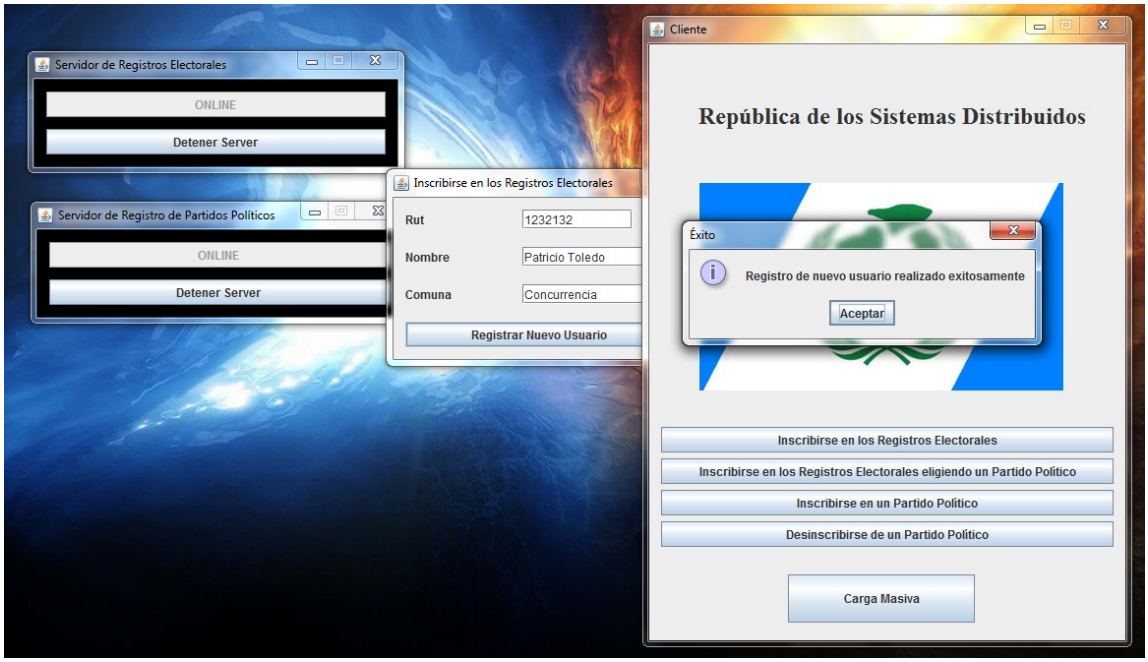
Se haga doble click (asumiendo entorno Windows, en linux puede ser por comando) en servidorPP.jar, servidorRE.jar y clienteRMI.jar para poder inicializarlos.

En el caso de los servidores, se deben encender ambos (cuidando de iniciar servRE antes de servPP, ya que sino aparece la notificación siguiente):

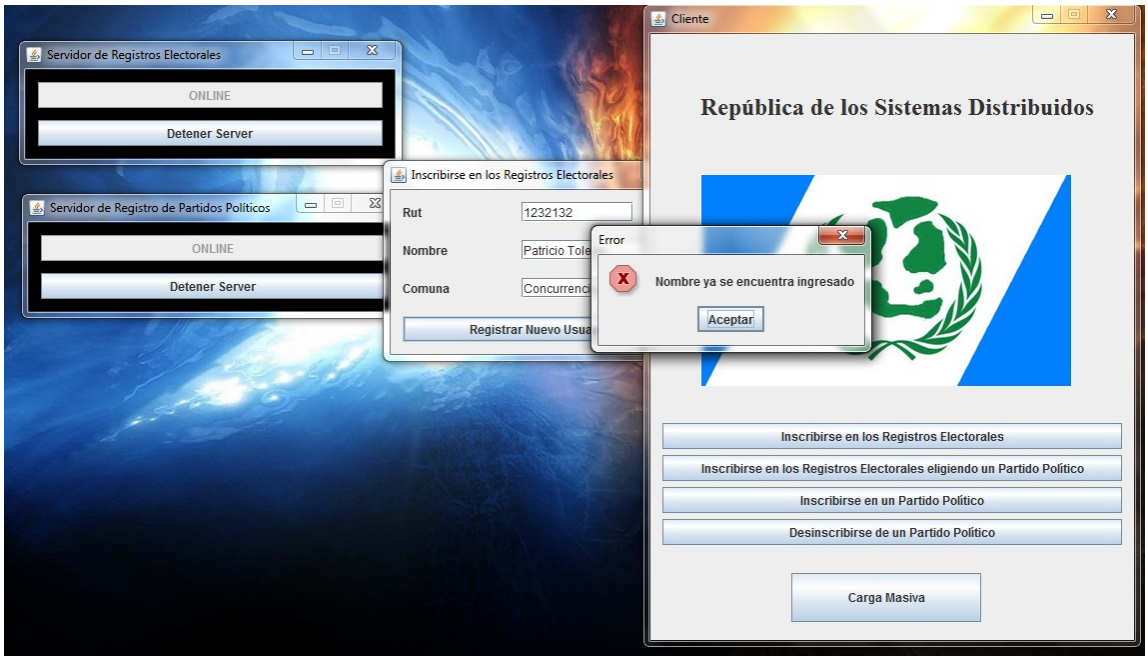
Por lo tanto debería quedar de la siguiente forma:

Luego en el cliente se pueden proceder a realizar las distintas acciones:

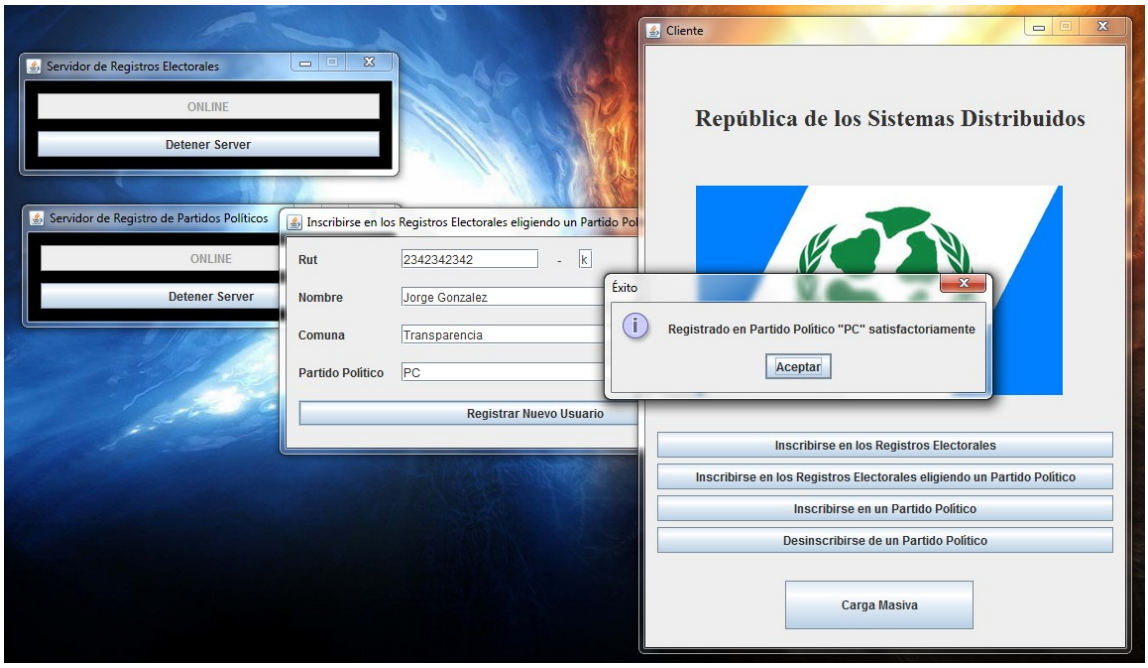
1.- Inscribirse en los Registros Electorales



Si se intenta de nuevo, ya el sistema reconoce el usuario:

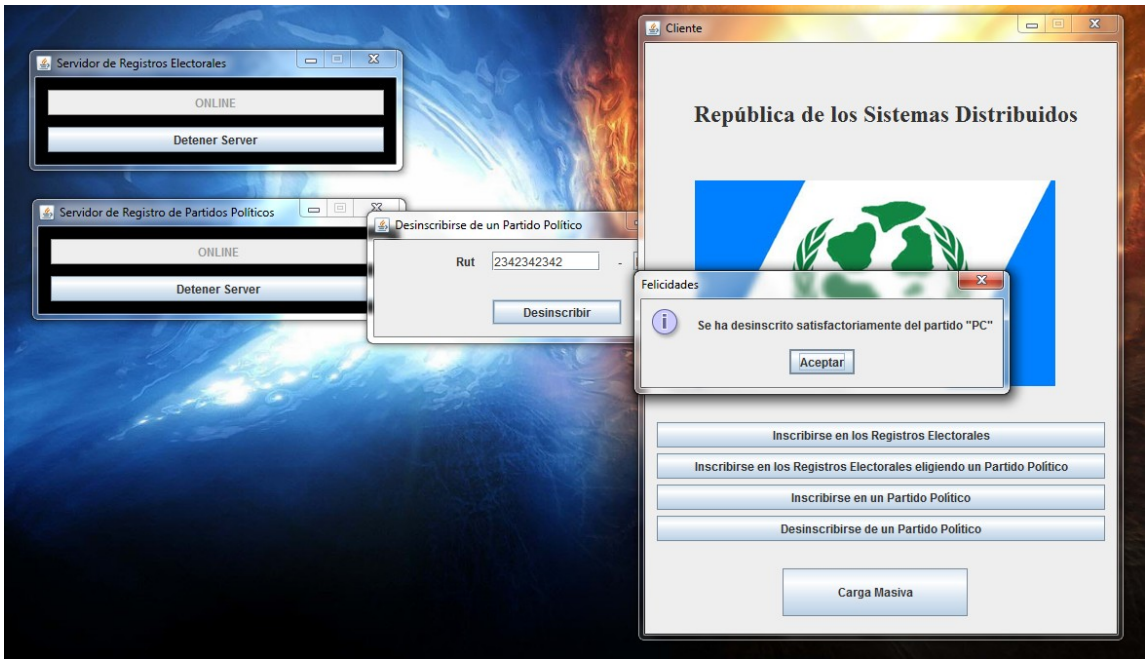


2.- Inscribirse en los Registros Electorales eligiendo un Partido Político



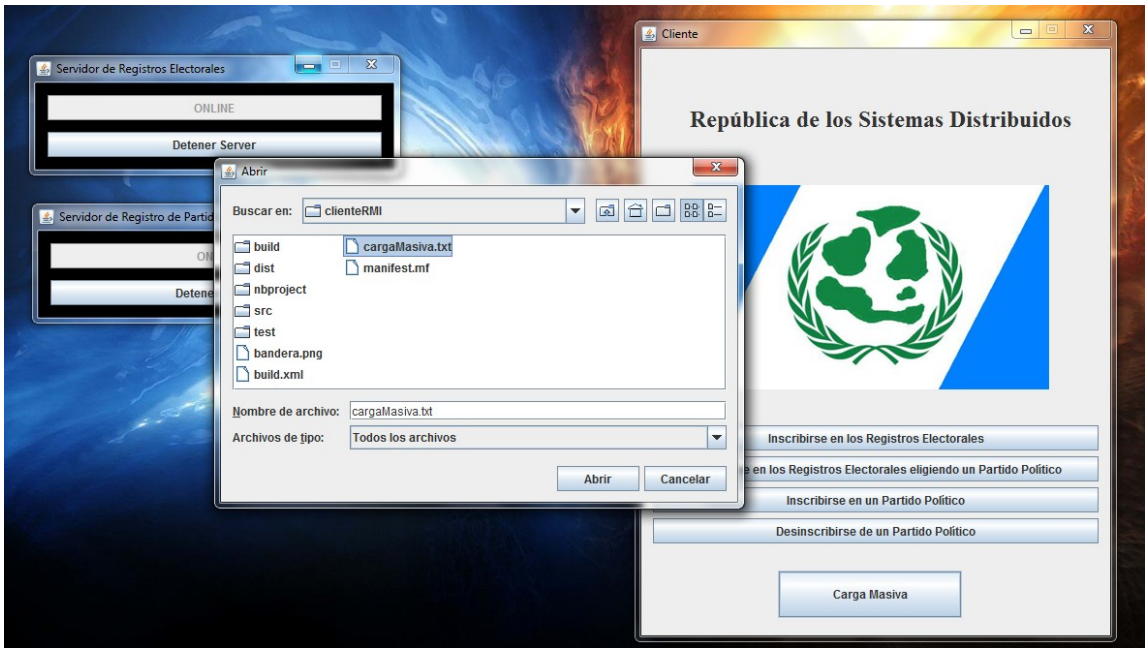
3.- Inscribirse en un Partido Político

4.- Desinscribirse de un Partido Político



5.- Carga Masiva

Primero se selecciona el archivo



Y luego el sistema automáticamente hace los cambios, y escribe en los logs correspondientes. (transparente al usuario)

FIN